Universidad del País Vasco
Euskal Herriko Unibertsitatea

Facultad de Informática

Grado de Ingeniería Informática

▪ Gradu Amaierako Lana ▪

Konputagailuen Ingenieritza

Paketeak kudeatzeko biltegiratze-sistema automatizatua.

Eneko Bermejo Larrazabal

Iraila 2017

Tutoreak:

Elena Lazkano eta Txelo Ruiz

informatika fakultatea
facultad de informática

Eskerrak

Unibertsitateko bost urte hauek pasata, garai honi bukaera emateko denbora iritsi egin da. Bost urte hauetan asko izan dira informatikaren inguruan ikasitakoak eta nola ez, informatikatik at dauden hainbat gauzak ere ikasi egin ditugunak.

Lehenik eta behin, eskerrak eman nahi dizkiet Txelo Ruiz eta Elena Lazkanori, nire buruan neukan proiektu hau errealitate egitea lagundu dutelako, eta nola ez ibilbide osoan zehar bere laguntza eskaini didatelako.

Nola ez, nire guraso eta familiari, nire bizitza osoan zehar eta bereziki ikasketak aurrera eramateko eman dizkidaten erraztasunengatik. Hauetatik, aipamen bereziak, nire aitari, Jesus, proiektuan zehar eskainitako laguntzagatik. Proiektuko bideoa sortzen emandako laguntzagatik, Joneri eskerrak eman nahi dizkiot.

Urte guzti hauetan unibertsitateko bizitza osatu duten gainerako pertsona guztiei ere eskerrak eman nahi dizkiet:Jon Gonzalez, Aitor, Urtzi, Alejandro, Mikel, Olatz eta Odei. Bereziki, Sergio Vega, Jon Viedma eta Asier Santos aipatu nahiko nituzke. Hauek, lau urte hauetan, graduan zehar eskaini didaten laguntzaz aparte, lagunak izan direlako, ibilbideko une oro nire ondoan egonda.

Laburpena

Proiektu honetan, informatikak dituen bi atal nagusiak lotzea saiatu egin da, software eta hardware munduak. Beti ere, gure adarra Konputagailuen Ingenieritza denez, foko gehiena hardware atalan jarri da. Izan ere, gaurko munduan, gero eta gehiago zabaltzen ari den “Internet Of Things” teknologien bitartez, software eta hardware munduak gero eta hurbilago aurkitzen dira.

Ondorioz, gure proiektuan, bi munduko osagaiak elkartu egin ditugu. Alde batetik, erabiltzaileekiko elkarrekintza modu dinamiko batean egin ahal izateko World Wide Web munduan erabiltzen diren hainbat teknologia erabili egin ditugu. Bestetik, Arduino plaken mundua eta hauekin lan egiten dituzten moduluak aztertu egin dira. Bi mundu hauek elkartuz, paketeak automatikoki biltegiratzeko sistema bat garatu da. Honetan garatu den web aplikazioaren bitartez, aginduak bidali egin zaizkio biltegiratze-sistema kontrolatzen duen Arduinoari; honek, pakete berriak gorde edo gordeta dauden paketeak atera ditzan. Horrela, sisteman erregistratuta dauden erabiltzaileek, beraientzat edo sistemako beste erabiltzaile batentzako paketeak gorde ahal izango dituzte.

Gaien Aurkibidea

ERANSKINAK

Irudien aurkibidea

Taulen aurkibidea

Kodearen aurkibidea

1. Aurkezpena

1.1 Sarrera

Gaur egun, gero eta gehiago , *Internet of Things* eta robotika mundua garrantzia handia lortzen ari dira gure bizitzan. Izan ere, normala da, gure bizitzan zehar horrelako gailuekin topo egitea gure etxean, lanean, kalean, ospitaleetan…etab. Beraz, hau kontuan izanda, gure bizitza eta langileen bizitza erraztu egingo duen sistema bat garatu nahi izan da.

Hau egiteko, guk diseinatu eta garatuko dugun sistema bat eraikiko dugu. Hau osatzeko, merkatuan eskuragarri dagoen Arduino plataforma erabili egingo da, guk eskuragarri ditugun teknologiekin egin daitezkeen sistema konplexuak aztertuz. Zer nolako sistema egin dugu? Arduinok zer nolako mugak ditu?

1.2 Motibazioa

Jarraian azalduko den proiektua guk pentsatutako eta diseinatutako proiektua izanik, hasieratik izan dugun motibazioa handia izan da. Honek suposatzen du, hainbatetan aurkitzen diren arazoen aurrean zailtasun handiak egotea baina, bestetik, proiektu baten nora norakoak pentsatzea eta hau garatu ondoren emaitzak ikusteak ematen duen poza handia da ere.

Proiektu hau egiteko motibazioa, hainbatetan guri gertatzen zaizkigun hainbat egoerak ekiditzeko asmoz pentsatu egin da. Izan ere, ohikoa den moduan, pakete bat etxera ailegatzen denean etxean ez egotea edo lagun bati zeozer eman behar eta berarekin ezin geratzea, egoera horiei nolabaiteko erantzun bat bilatu nahi izan da proiektu honen bitartez.

Bestalde, ezin gara ahaztu ikasketen arlotik datorren motibazioa. Proiektu honetan zehar Arduino plataformaren inguruan informazio bilatu, aztertu eta garatu dugu. Gainera, web teknologietan ditugun gaitasunak ere zabaltzea lortu dugu.

1.3 Proiektuaren deskribapena

Sarreran esan dugun bezala, dokumentu honetan azaltzen den proiektuari biltegiratze automatikoko biltegia deitu diogu. Modu orokor batean azaltzearren, gure produktua sisteman erregistratuta dauden erabiltzaileei, sisteman paketeak gordetzeko eta bere izenean dauzkaten paketeak ateratzeko aukera emango die. Horretarako, hainbat gailu erabili egingo dira, bai hardware atalerako eta baita software atalerako ere. Gure produktua bi zatitan banatu daiteke; aginduak bidaliko dituen atala eta agindu horiek asetuko dituen atala.

- Agindu atala: Erabiltzaileak gure sistemarekin elkarrekintza egingo duen atala da. Hau web aplikazio baten bitartez osatuta egongo da. Horrela erabiltzaileek aukeratu ahal izango dute pakete berri bat sartu edo jadanik gordeta dagoen pakete bat atera nahi duten.
- Atal eragilea: Gure biltegia kontrolatu egingo dugu bi Arduino plaka erabilita. Biltegiak modu automatiko batean funtziona dezan, hardware sistema hau hainbat sentsore/eragingailuez osatuta egongo da. Hau bi ataletan banatuko da, biltegia bera, eta biltegia kanpoarekin konektatu egingo duen plataforma mugikorra. Hauek, web aplikaziotik datozen aginduak asetuko dituzte, paketeak gordetzeko eta ateratzeko beharrezko prozesuak eginez. Atal hau, proiektuaren atal garrantsitzuena izango da.

Ikusi ditugun bi atalaz aparte, biltegia fisikoki eraiki egin beharko dugu, bertan biltegia kontrolatzeko erabiliko dugun beso robotikoa, plataforma mugikorra, biltegiratze-eremuak, Arduinoak eta beharrezko sentsoreak jarri ahal izateko.

2. Plangintza

Atal honetan, proiektua aurrera eramateko egin beharrekoen ezaugarriak eta azalpenak emango dira. Lehenik eta behin egindako proiektuaren irismena azaltzen da, honekin batera zer nolako lanen deskonposaketa eta definitu diren atazak azalduz. Bestetik, denboraren kudeaketa eramateko egindako kronograma eta denbora estimazioak daude. Bukatzeko, helburuak zehazten dituen kalitate plana, proiektua aurrera eramateko erabiliko den lan-metodologia, tutoreekin izandako komunikazio plana eta arriskuen plana daude.

2.1 Helburuak

- Helburu nagusiak
    - Biltegiratze-sistema automatizatu bat garatzea
        - Biltegia fisikoki eraiki.
        - Arduino bitartez biltegia kontrolatu, bai beso robotikoa, baita plataforma mugikorra ere.
        - Web aplikazio baten bitartez, biltegiari aginduak bidali, erabiltzaileen eskakizunen arabera.
    - Proiektuaren garapena biltzen duen memoria idaztea.
    - Proiektuaren defentsarako aurkezpena.
- Bigarren mailako helburuak
    - Gaitasunak lortzea
        - Arduino ingurunea ezagutzea.
        - Web aplikazio ingurunea ezagutzea.
        - Ikasketa autonomoa.

2.2 Irismena

Ondoren, proiektu bera eta honen bitartez egindako produktuaren irismena azalduko ditugu. Hasteko produktuaren ezaugarriak eta helburuak definituko ditugu, gerora produktu hau egiteko prozesuaren (proiektua) ezaugarriak eta pausuak azalduz.

2.2.1 Produktua

Gaur egun, robotikaren garapena dela eta, prozedura automatizatzeko eta errazteko aukerak sortu dira. Garatu diren teknologien artean, beso robotikoak gehien zabaldu direnen artean aurkitu ditzakegu. Hainbat sektoretan erabiltzen dira; adibidez, produkzio kateetako hainbat lanetan, pertsonak ordezkatu dituzte, edota biltegi oso ezberdinen antolaketetan erabiltzen dira. Gure inguruneta Farmaceutica Guipuzcoanaren kasua aurkitu dezakegu, non biltegi automatizatuak erabiltzen dituzten. Hala ere, gaur egun inplementatuta dauden beste sistema batzuk 3. kapituluan sakonago aztertuko ditugu.

Azken adibide hau oinarritzat hartuta, garatuko dugun produktuaren nondik norakoak aztertuko ditugu. Hain zuzen ere, proiektuak zentro komertzial edo hainbat kokalekutan paketeak elkar

trukatzeko edo jasotzeko (biltzeko/antolatzeko) agertu diren plataformen antzeko bat garatzea dauka helburu. Hauek, erabiltzaileak bidali, edota jaso, nahi dituzten paketeak gordetzeko erabiltzen dira. Behin testuingurua definituta dagoelarik, proiektua zehazki definitzera pasatuko gara.

Guk garatu nahi dugun proiektuak, unibertsitateko fakultatean edota enpresa batean, lankideen artean edota online eskatutako paketeak elkartrukatzeko eta gordetzeko sistema bat garatzea dauka helburu. Sistema hau modu zentralizatu batean kontrolatuko litzateke web aplikazio baten bitartez. Web aplikazioaren bidez, erabiltzaileak sisteman bere izenean gordeta dituzten paketeak ikusi eta atera ahal izango ditu. Aldi berean, lankide batentzak pakete bat gorde nahi badute posible izango dute. Gainera, banatzaile enpresentzat ere, posible izango litzateke sistema honetan paketeak uztea, bezero baten izenean.

Web aplikazio honek erabiltzaileengandik jaso dituen aginduak biltegiratze-sistemari bidaliko dizkio. Sisteman hainbat biltegi egon daitezke, bakoitza bere sistema fisikoarekin. Sistema fisiko honek, agindua jasotzerakoan, atera nahi den paketea identifikatu eta zerbitzatuko dio erabiltzaileari. Hau egiteko, biltegiaren beso robotikoak paketea hartuko du, plataforma mugikor baten gainean utziz. Plataforma mugikor honek paketea kanporatu egingo du. Plataforma mugikorra biltegitik eta biltegira paketeak sartzeko/ateratzeko erabiliko da.

Bestetik, erabiltzaileek bere izenean dauden paketeak ateratzeaz gain, paketeak beraien izenean edo beste erabiltzaile baten izenean gorde ahal izango dituzte. Kasu honetan, prozesua alderantziz egingo litzateke. Erabiltzaileak paketea plataforman utziko du eta plataformak paketea biltegian sartzerakoan, beso robotikoak paketeari dagokion lekuan gordeko du.

Ikusten den moduan, sistemak hainbat biltegi eskatzen ditu eta gure denbora eta baliabide mugak direla eta, ezinezkoa daukagu horrelako sistema garatzea. Beraz, sistema nola funtzionatuko lukeen simulatzeko, biltegiratze-sistema baten prototipo bat garatzea erabaki dugu, etorkizun batean, sistema osoa nola funtzionatuko duen ikusteko.

Ondorioz , gure Gradu Amaierako Proiektuaren helburu nagusia objektuak (paketeak) biltzeko eta zerbitzatzeko biltegi oso bat garatzea da, etorkizuneko sistema simulatuko duena. Beraz, gure plataformak bi funtzionalitate nagusi izango ditu. Ematen zaizkion paketeak hartu eta biltegian gorde modu antolatu batean, eta bestetik, erabiltzaileak sistemari gordeta dauzkan paketeen artean bat eskatzean hau zerbitzatzea. Erabiltzaile eta biltegiaren arteko elkarrekintzak web aplikazio baten bitartez egingo dira. Esan dugun moduan, proiektu honek sistemaren prototipo bat garatzea dauka helburu eta beraz, bai pakete erreal, bai biltegi/beso robotiko handiekin egitea ezinezko izango da, proiektuan existitzen diren mugak direla medio. Hau dela eta, prototipoa eskala txikiago batean egitea erabaki da, gure eskura genituen baliabideak erabiliz.

Gure prototipoak bete beharreko funtzioak aurrera eramateko, produktua hiru atal ezberdinetan banatzea erabaki dugu. Honako hauek izango dira:

- Alde batetik, erabiltzaileek erabiliko duten web aplikazioa. Honetan 3 erabiltzaile mota egongo dira, erabiltzaile arrunta, banatzaileak (enpresa banatzaileek erabiliko dutena) eta administratzaileak. Web aplikazio honen bitartez, erabiltzaileek biltegiari aginduak bidaliko dizkiote.
- Bestetik, biltegia bera daukagu. Kasu honetan, egurrezko biltegia izango da, non beso robotikoa kokatuko den. Beso robotikoa kontrolatzeko Arduino Mega plaka erabiliko da. Honek, beso robotikoa kontrolatzeaz gain, paketeen administrazioa, web aplikazioarekin komunikazioa eta ondoren ikusiko dugun plataforma mugikorraren kontrola eramango du. Beraz, honek web aplikaziotik datozen aginduak asetzeko beharrezko prozesuak aurrera eramango ditu, biltegian egin beharreko guztiak kontrolatuz.

- Bukatzeko, plataforma mugikorra izango dugu. Hau biltegiko paketeak ateratzeko eta biltegira paketeak sartzeko erabiliko den plataforma izango da. Hau, esan dezakegu, biltegia kanpoko munduarekin konektatzen duen plataforma dela, non erabiltzaileek paketeak utzi edo hartu egingo dituzten. Plataforma honen kontrola, Arduino UNO plaka baten bitartez egingo da. Plataformaren kontrolaz aparte, paketeen identifikazioa egingo du ere. Plataforma mugikor honek, bere prozesuak aurrera eramango ditu, Arduino Megatik aginduak ailegatzen direnean.

Lehen azaldu dugun moduan, gure produktuak bi funtzionalitate nagusi izango ditu. Hurrengo lerroetan bi funtzionalitate hauek nola prozesatuko diren azaltzen saiatuko gara.

- Lehenik eta behin, erabiltzaileek sisteman pakete berriak sartzeko prozesua daukagu. Prozesua hasiko da erabiltzaileak web aplikazioan pakete berri bat gorde nahi duela zehazten duenean. Orduan, wifi socket[1] bitartez, web aplikazioak agindua bidaliko dio biltegia kontrolatzen duen Arduino Megari. Honenbeste, Arduino Megak plataforma mugikorra kontrolatzen duen Arduino UNO-ri pakete bat sartzeko agindua bidaliko dio. Agindua jaso ondoren, plataforma mugikorra irekiko da eta, irekita dagoenean, erabiltzaileak sartu nahi duen paketea plataforma mugikorraren gainean utziko du. Behin paketea bertan utzi dela, Arduino UNO-k paketea identifikatuko du eta plataforma itxi egingo da, paketea biltegiaren barrura sartuz. Paketea sistemaren barruan dagoela, beso robotikoak paketea plataformatik hartu eta dagokion biltegiratze-eremuan utziko du. Biltegiratze-eremu hauen kontrola, lehen esan dugun moduan Arduino Megak eramango du. Honekin prozesua bukatutzat emango dugu. Orduan, berriro ere, wifi bitartez, web aplikazioari prozesua amaitu dela esango zaio eta honek erabiltzaileari abisatuko dio.
- Bigarren funtzionalitatea, erabiltzaileak bere izenean gordeta dauzkan paketeak sistemari eskatzea izango da. Erabiltzaileak, web aplikazioaren bitartez, atera nahi duen paketea aukeratzen duenean, biltegiari (Arduino Megari) atera nahi den paketea zein den esango zaio wifi socket bitartez. Beraz, Arduino Megak pakete hau zein biltegiratze-espazioan dagoen zehaztuko du eta beso robotikoari hau hartzeko agindua bidaliko dio. Beso robotikoak paketea hartu ondoren, plataforma mugikorraren gainean utziko du. Orduan, Arduino Megak Arduino UNO-ri paketea ateratzeko aginduko dio. Ondorioz, plataforma mugikorra ireki egingo da. Plataforma irekita dagoela, erabiltzaileak paketea hartuko du eta plataforma berriro itxi egingo da. Prozesua behin amaitu dela, Arduino Megak abisatuko dio web aplikazioari, honek erabiltzaileari jakinaren gainean jartzeko.

Honenbestez, hiru elementu hauek osatzen duten osagaiak, modu sakonago batean aztertuko ditugu:

- Beso robotikoa eta biltegia: Lynxmotion etxearen AL5D beso robotikoa kontrolatuko da. Honen kontrola, Arduino Mega plaka baten bitartez egitea erabaki da. Beso robotikoak modu automatiko batean paketeak biltegiratze-eremu eta plataforma mugikorretik hartu beharko ditu. Beso robotikoa kontrolatuko duen plakak, plataforma mugikorra kontrolatuko duen Arduino UNO-rekin komunikatu beharko da (HC-05 Bluetooth moduluen bitartez), plataforma ireki eta ixteko. Gainera, biltegiari aginduak bidaliko dion web aplikazioarekin komunikatu beharko da. Komunikazio hau wifi socket baten bitartez egingo da. Biltegia egurrezkoa izango da eta barruan beso robotikoa, biltegiratze-espazioak eta plataforma mugikorra izango du.

[1] *Socket* bat sare-zerbitzuak erabiltzeko komunikazio-puntu bat da. Gure kasuan, sare berdinera konektatuta dauden (Arduino eta Apache zerbitzaria) bi elementuen arteko komunikazio-kanal bat sortzeko erabiliko dugu.

- Plataforma Mugikorra: Honek, pi posizio izango ditu, bat itxita eta bestea irekita. Bi posizio hauen kontrola Arduino UNO baten bidez egingo da. Arduinoari konektatuta dagoen serbomotor baten bitartez kontrolatuko da ireki eta itxiera, distantzia-sentsore batzuekin plataforma itxita edo irekita dagoen zehaztuz. Plataforma hau biltegitik edo biltegira paketeak sartzeko/ateratzeko erabiliko da, erabiltzaileak paketeak hartu/utzi ahal izateko. Plataforma honek paketeen identifikazioa egingo du, RFID teknologia erabiliz (RFID-RC522 modulua). Erabiliko ditugun pakete guztiek RFID etiketa bat izan beharko dute. Plataforma hau Arduino Megaren aginduen zain egongo da, plataforma noiz ireki edo itxi behar duen jakiteko. Komunikazio hau, lehen esan dugun moduan, HC-05 Bluetooth moduluaren bidez egitea pentsatu da.
- Web aplikazioa: Biltegia eta erabiltzaileen arteko elkarrekintza bermatzeko erabiliko den web aplikazioa. Kodea garatuko da egiten diren ekintza guztiak monitorizatu ahal izateko. Honetan, erabiltzaile bakoitzak gordeta dituen paketeak ikusi edo berri bat sartzeko aukera izango du. Bai gordeta dagoen pakete bat ateratzeko edota pakete berri bat sartzeko, aplikazioak agindu bat bidaliko dio gure biltegi-sistemari (Arduino Megari) wifi socket baten bitartez.

Kontuan izan behar dugu proiektu honen muga nagusiena denbora dela, eta beraz, lortu daitekeen proiektu bikaintasuna honen araberakoa izango da. Ondorioz, denbora gehiago izanda edota etorkizunean hobetu nahi izanez gero, funtzionalitate gehiago edo daudenak hobetu ahal izango litzateke.

2.2.2 Proiektua

Proiektuaren azkenengo helburua 2.2 atalan definitutako produktua aurrera eramatea eta bukatzea izango litzateke. Beraz, proiektua arrakastatsua izan dadin, plangintza ahal den neurrian bete beharko da.

2.2.3 Mugak

Beti ere kontuan izan behar dugu gure proiektuak denbora muga bezala 300 ordu dituela. Gradu amaierako proiektu honek 12 kreditu baititu. Beraz, planifikazioa eta irismena definitzerakoan, hau kontuan hartuko dugu, gure eskuetatik haratago dagoen proiektu bat ez definitzeko.

Kontuan izan behar dugu daukagun denbora, produktuaren garapenenaz aparte, beste atalei ere eskaini beharko diegula. Adibidez, memoria idazteari, izan ere, memoria idazteak proiektuaren %30 inguruko denbora hartuko baitigu.

Beraz, argi izan behar dugu, planifikatzen hasterakoan , gure helburua 300 orduko proiektu bat egitea dela, eta beraz, planifikatzen dugun proiektua ordu kopuru horren inguruan egon behar dela. Hala ere, baliteke proiektuan ordu gehiago sartu behar izatea, eta horregatik denbora maximo muga bat jartze arren, 370 orduko estimazio posiblea maximoa jarriko dugu.

2.3 Lanaren deskonposaketa egitura

Proiektua hainbat atazatan banatu behar da, proiektuaren arrakasta ziurtatzeko eta probetxuzko antolamendu bat izan dezagun. Ondorioz, proiektua bost ataletan banatzea erabaki da. Lehendik, kudeaketa atala izango dugu. Honetan hiru azpiatal definitu ditugu: proiektuaren plangintza, denboraren kudeaketarako beharrezkoa den jarraipen eta kontrola eta, azkenik, tutoreekin egindako kudeaketa bilerak.

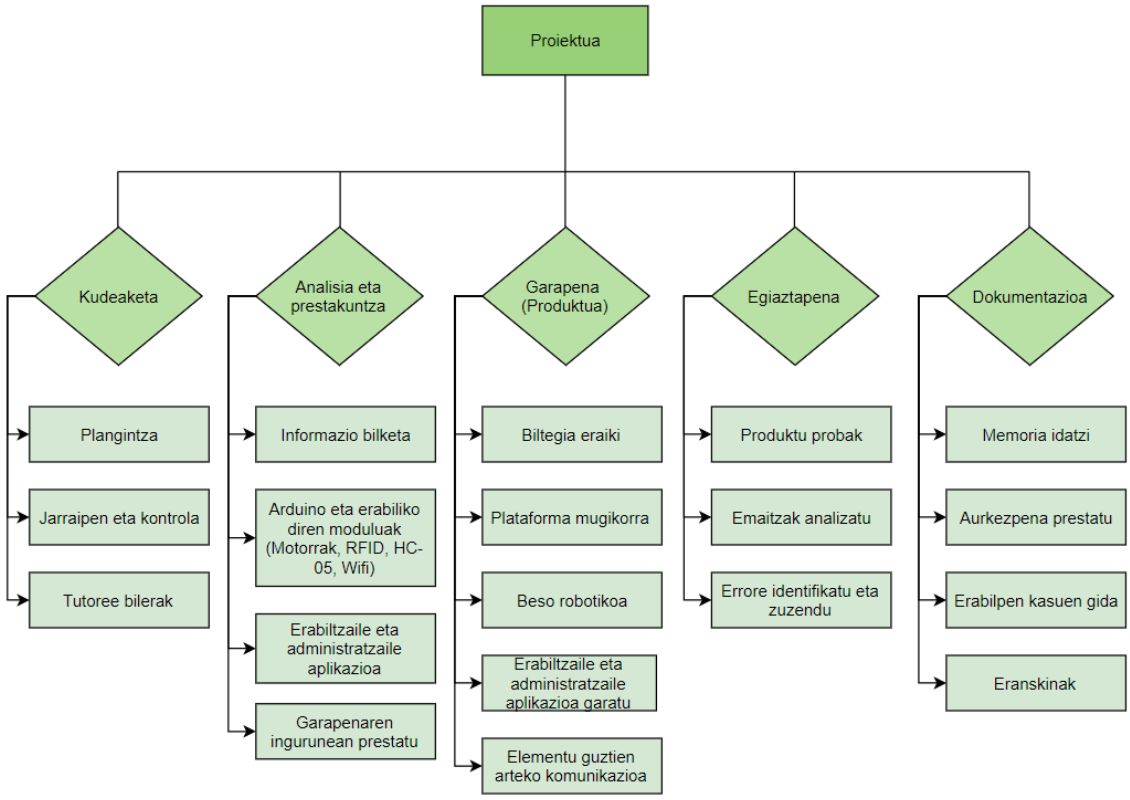
Bestetik, produktuaren garapenarekin hasi baino lehen, beharrezkoak diren ezagutza eta informazioa lortu behar ditugu. Horretarako gure bigarren atala daukagu, analisia eta ezagutzak.

Atal honetan, proiektuarekin hasi baino lehen, egin behar dugun informazio bilketa (biltegia eraikitzeko materialaz, plataforma nola egin, biltegiaren diseinua) aurkituko dugu. Honetaz aparte, sistema kontrolatzeko erabiliko diren Arduino eta hauen moduluen funtzionamendu eta erabiltzaile/administratzaile aplikazioak garatzeko beharrezko teknologiak eta tekniken inguruko prestakuntza daukagu. Bukatzeko, garapen inguruneen prestakuntza aurkituko dugu.

Prestakuntzaren ondorioz, proiektuaren muinera pasatuko gara, hau da, produktuaren garapenera. Honetan, biltegia eraiki behar dugu, plataforma mugikorra eraiki eta kontrolatu (Arduino UNO), beso robotikoaren kontrola eta paketeen administrazioa garatu (Arduino Mega) eta erabiltzaile/administratzaileek sistemarekin elkar eragin ahal izateko aplikazioa garatu beharko dugu. Azkenik, nahiz eta aurreko atalak garatzeko beharrezkoa izan atal guztien arteko elkar komunikazioa inplementatu beharko dugu.

Garapenarekin bukatu ondoren, egindako lanaren egiaztapenaren txanda izango da. Hemen hiru pakete definitu ditugu; lehendik, garatutako produktuaren proba ezberdinak egin beharko dira. Jarraian, hauek aztertu eta analizatuko ditugu, hortaz, honetan aurkitutako erroreak identifikatu eta zuzendu, berriro ere garapen adarrera bueltatuz. Kasu honetan, garapen eta egiaztapen bi atalen artean ziklo bat egongo da, produktuari eta proiektuari itxiera eman arte.

Azkenik, aurreko atal guztiak egiten joan ahala, egindako lana dokumentatuz joan beharko dugu. Honetarako, dokumentazioaren atala definitu dugu. Hemen, memoria idaztea, erabilpen kasuen gida, eranskinak prestatzea eta, azkenik, proiektuaren aurkezpena egiteko behar ditugun materialen prestakuntza dauzkagu. Guzti honen erreferentzi grafikoa 2.1 irudian daukagu.



2.1. Irudia: LDE

2.4 Atazak eta ezaugarriak

A1. Kudeaketa

- A1.1 Plangintza
  - A1.1.1 Irismena definitu
  - A1.1.2 LDE zehaztu
  - A1.1.3 Atazak definitu
  - A1.1.4 Emangarriak identifikatu
  - A1.1.5 Denboraren kudeaketa
    - A1.1.5.1 Kronograma
    - A1.1.5.2 Dedikazio estimazioa
  - A1.1.6 Kalitate plana burutu
  - A1.1.7 Arrisku plana burutu
  - A1.1.8 Komunikazio plana zehaztu
  - A1.1.9 Eskuraketen plana burutu
  - A1.1.10 Lan metodologia definitu
- A1.2 Jarraipen eta kontrola
  - A1.2.1 Ataza guztien jarraipena
  - A1.2.2 Desbideraketak kalkulatu
  - A 1.2.3 Desbideraketen arrazoiak eta justifikazioa
- A1.3 Tutoreekin bilerak
  - A1.3.1 Konstituzio bilera
  - A1.3.2 Kontrol bilera
  - A1.3.3 Itxiera bilera

A2. Analisia eta prestakuntza

- A2.1 Informazio bilketa
  - A2.1.1 Biltegiaren eraikuntza
  - A2.1.2 Biltegi materialak
  - A2.1.3 Plataforma eraikuntza
  - A2.1.4 Beso robotikoa
- A2.2 Arduino eta moduluak
  - A2.2.1 Arduino eta serbomotorrak

A2.2.2 Arduino eta RFID

A2.2.3 Arduino eta HC-05 bluetooth moduluak

A2.2.4 Arduino eta wifi

A2.2.5 Arduino eta distantzia-sentsoreak

A2.3 Erabiltzaile eta administradore aplikazioa

A2.3.1 Aplikazioa garatzeko plataforma eta lengoaia

A2.3.2 Biltegi sistemarekin komunikazioa

A2.3.3 Erabiltzaileari eta administradoreari aurkeztuko zaion interfazea

A2.4 Garapenaren ingurunea prestatu

A2.4.1 Zirkuitu elektriko guztiak prestatu

A2.4.2 Arduino eta moduluen arteko konexioak egin

A2.4.3 Prestatu erabiltzaile eta administradore aplikazioak garatzeko ingurunea (XAMPP)

A2.4.4 Prestatu web aplikazioak, arduinoarekin komunikazioa egiteko garatuko diren programen ingurunea (NetB)

A2.4.5 Biltegia eraikitzeko espazioa prestatu

A3. Garapena

A3.1 Biltegia eraiki

A3.2 Plataforma mugikorra (Arduino Uno)

A3.3 Beso robotikoaren kontrola eta paketeen administrazioa (Arduino Mega)

A3.4 Erabiltzaile eta administratzaile aplikazioa garatu.

A3.5 Elementu guztien arteko komunikazioa

A4. Egiaztapena

A4.1 Produktuaren probak

A4.1.1 Erabiltzaile batek pakete bat sartzea

A4.1.2 Erabiltzaile batek gordeta daukan pakete bat atera

A4.1.3 Sistema osoak kontrolatzen ez diren egoeren aurreko erantzuna

A4.1.4 Erabiltzaile berri baten erregistroa

A4.2 Emaitzak aztertu

A4.3 Errore identifikatu eta zuzendu

A5. Dokumentazioa

A5.1 Memoria idatzi

A5.2 Eranskinak

A5.3 Erabilpen kasu baten gida

A5.4 Aurkezpena prestatu

2.5 Emangarrien identifikazioa

Proiektuak lau emangarri nagusi ditu: Produktua, memoria, aurkezpena eta erabilpen kasuaren gida. Hauek banaka aztertzera pasatuko gara orain.

2.5.1 Produktua

Alde batetik, garatu den biltegia, barnean dituen plataforma, beso robotikoa eta moduluak modu egokian kokatuta eta konektatuta; bestetik, gure sistemak ondo funtzionatzeko garatu egin diren kontrol programa eta erabiltzaile/administratzaileentzat garatutako aplikazioak entregatu behar ditugu. Produktua defentsaren ostean entregatuko da, hala ere, prest egon beharko da irailaren 3a baino lehenago.

2.5.2 Memoria

Dokumentu hau, proiektua aurrera eramateko egindako planifikazioa eta honen kudeaketaz gain, proiektua nola garatu eta aurrera eraman den azaltzen duen txostena da. Honen egitura EHU Informatikako Fakultateak eskaintzen duen txantiloia erabilita osatu da. Dokumentu hau ADDI plataformara igo behar da irailaren 3a baino lehen.

2.5.3 Aurkezpena

Dokumentu hau proiektuan zehar egindakoa tribunalaren aurrean azaltzeko erabilgarria izan daitekeen dokumentua izan beharko da. Honetan modu argi batean egindakoaren laburpen bat agertu beharko da, prozesuan zehar egindako pausuak azalduz. Honen formatua aurkezpena izan beharko da eta beraz, graduan zehar aurkezpenak egituratzeko eta osatzeko irakasleek emandako aholkuak jarraituz egingo da. Proiektuaren defentsa iraileko 18-21 bitartean burutuko da, beraz, bukatuta egon beharko da egun batzuk lehenago.

2.5.4 Erabilpen kasuaren gida

Memorian produktuaren garapena eta funtzionamendua agertzen diren arren, komenigarria ikusi da gida bat idaztea, edonork gure proiektua erabili nahi badu, nola funtzionatzen duen jakin dezan. Hau honela izanda, dokumentu hau memoriarekin batera entregatuko da.

2.6 Denboraren plangintza

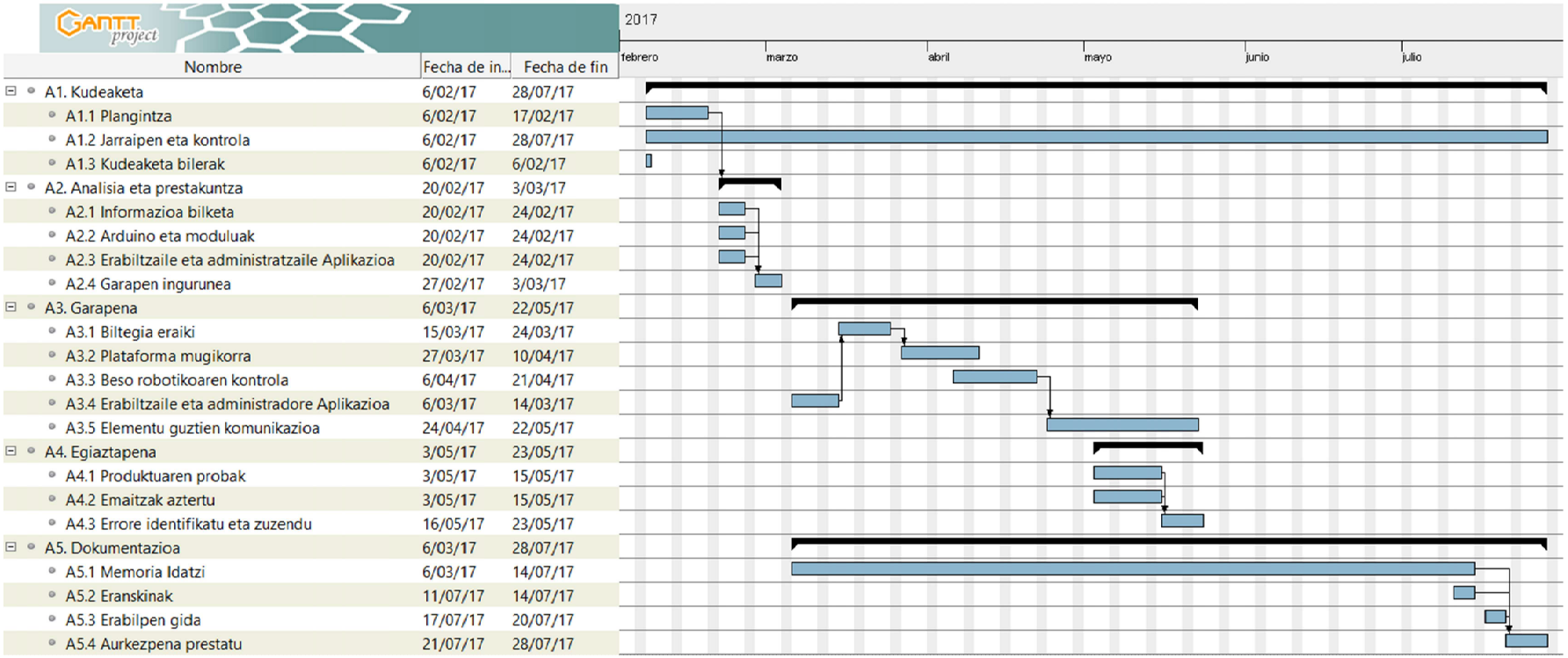
2.6.1 Kronograma

Lehenik eta behin, lehenengo kudeaketa bilera bat egongo da tutoreekin proiektuaren planteamendua azaltzeko. Hau egin baino lehenago, plangintzaren zati bat, irismena behintzat, landu beharko dugu, irakasleari ideia landuta aurkezteko. Bilerak periodikoki egitea planifikatu da, tutoreari proiektuaren ibilbidearen berri emateko eta hau ondo doala ziurtatzeko. Bestalde, kronograman ikusi daitekeen moduan, jarraipena eta kontrola proiektuaren iraupen osoan zehar eraman beharko da.

Hasteko, plangintza garatu beharko dugu. Plangintza honen barruan informazio bilketa txiki bat egin beharko dugu, planifikazio egoki bat egiteko. Pare bat aste igaro ondoren, plangintza bukatuta dugula, informazio bilketa eta prestakuntzarekin hasiko gara. Informazioa bildu beharko dugu eta garapenerako lan-ingurua prestatzen hasi beharko gara, beharrezko framework-ak eta lan esparrua prestatzen. Hau, aste batean egitea espero da.

Gerora garapenarekin sartuko gara. Proiektuaren atal luzeena izango da. Honetan produktuaren garapena eta egin beharreko probak kontuan izanda, honetan hilabete eta erdi inbertituko dugula estimatu dugu. Garapena eta probak hainbat atazatan banatu dira, bakoitzak bere estimazio izanda. Garapena egiterako momentuan, 2.2 irudian ikusi daitekeen moduan, 3.1 atazatik hasi gara, hau da, biltegi sistema garatzen. Behin, biltegiratze-sistema garatuta dugula, web aplikazioa garatzen hasiko gara.

Beste aldetik dokumentazioa daukagu. Memoria garapenarekin batera hasiko gara idazten, garatzen duguna egiaztatuz joan ahala memorian jasoz. Beraz, memoria osatzen egongo gara garapenak irauten duen bitartean. Behin memoria eta garapena bukatutzat ematen ditugula beste aste eta erdi gehiago emango dugu erabilpen kasuak idazten eta defentsarako behar dugun materiala prestatzen. Kronograma ikusgai dago 2.2 irudian.



Nombre	Fecha de in...	Fecha de fin
A1. Kudeaketa	6/02/17	28/07/17
A1.1 Plangintza	6/02/17	17/02/17
A1.2 Jarraipen eta kontrola	6/02/17	28/07/17
A1.3 Kudeaketa bilerak	6/02/17	6/02/17
A2. Analisia eta prestakuntza	20/02/17	3/03/17
A2.1 Informazioa bilketa	20/02/17	24/02/17
A2.2 Arduino eta moduluak	20/02/17	24/02/17
A2.3 Erabiltzaile eta administratzaile Aplikazioa	20/02/17	24/02/17
A2.4 Garapen ingurunea	27/02/17	3/03/17
A3. Garapena	6/03/17	22/05/17
A3.1 Biltegia eraiki	15/03/17	24/03/17
A3.2 Plataforma mugikorra	27/03/17	10/04/17
A3.3 Beso robotikoaren kontrola	6/04/17	21/04/17
A3.4 Erabiltzaile eta administradore Aplikazioa	6/03/17	14/03/17
A3.5 Elementu guztien komunikazioa	24/04/17	22/05/17
A4. Egiaztapena	3/05/17	23/05/17
A4.1 Produktuaren probak	3/05/17	15/05/17
A4.2 Emaitzak aztertu	3/05/17	15/05/17
A4.3 Errore identifikatu eta zuzendu	16/05/17	23/05/17
A5. Dokumentazioa	6/03/17	28/07/17
A5.1 Memoria Idatzi	6/03/17	14/07/17
A5.2 Eranskinak	11/07/17	14/07/17
A5.3 Erabilpen gida	17/07/17	20/07/17
A5.4 Aurkezpena prestatu	21/07/17	28/07/17

2.2. Irudia: Gantt Diagrama

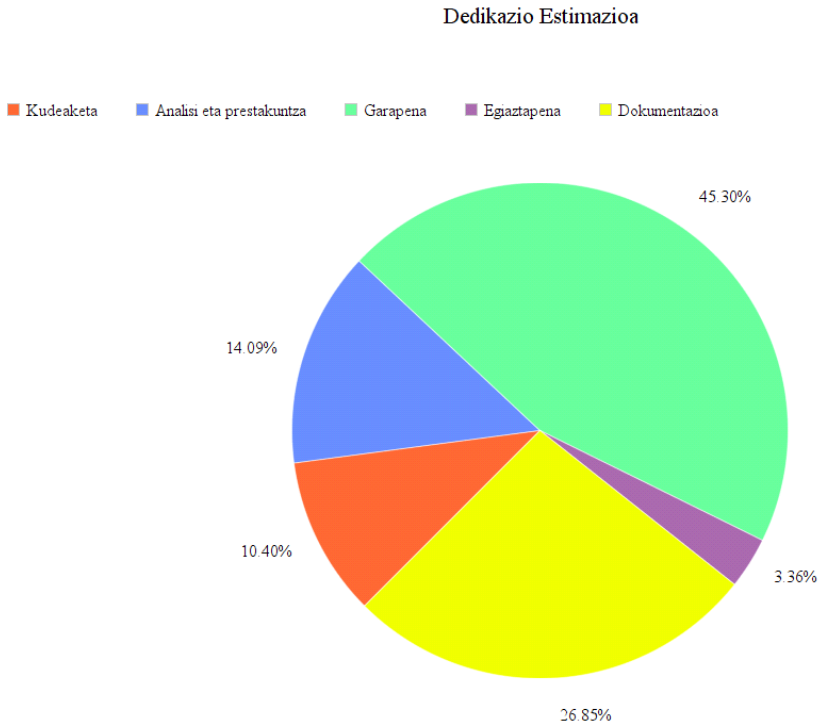
2.6.2 Dedikazio estimazioa

2.1 taulan, LDE-an definitu ditugun atazak egiteko kalkulatu diren dedikazioaren estimazioak ikus daitezke. Informazioa modu grafiko batean ikusi daiteke, 2.3 irudiko sektore-diagraman.

Dedikazioa estimatzerakoan, argi ikusten den moduan, garapena eta dokumentazioa dira pisu handien duten lan-paketeak. Hala ere, nahiz eta planifikazioari, jarraipenari, informazio bilketari eta probei denbora gutxiago eskaini, hauek ondo burutzea ezinbestekoa izango da, desbiderapen handiak gertatu ez daitezen. Ikusten dugun moduan, guztira 348 ordu estimatu dira. Desbiderapen txikiak aurreikusten direnez, egutegian hainbat egun/ordu utzi dira koltxoi moduan, agertzen diren arazoak konpondu ahal izateko.

Lan paketea	Estimazioa(ordutan)
A1.Kudeaketa	31,00
A1.1 Plangintza	20,00
A1.2 Jarraipen eta kontrola	9,00
A1.3 Tutoreekin bilerak	2,00
A2. Analisia eta Prestakuntza	42,00
A2.1 Informazioa bilketa	8,00
A2.2 Arduino eta moduluak	15,00
A2.3 Erabitzaile eta administradore aplikazioak	10,00
A2.4 Garapenaren ingurunea	9,00
A3. Garapena	150,00
A3.1 Biltegia eraiki	10,00
A3.2 Platafoma mugikorra	45,00
A3.3 Beso Robotikoa kontrola	55,00
A3.4 Erabiltzaile eta administradore aplikazioa garatu	30,00
A3.5 Elementu guztien arteko komunikazioa	10,00
A4. Egiaztapena	10,00
A4.1 Produktuaren probak	3,00
A4.2 Emaitzak aztertu	1,00
A4.3 Errore identifikatu eta zuzendu	6,00
A5. Dokumentazioa	115,00
A5.1 Memoria idatzi	75,00
A5.2 Eranskinak	15,00
A5.3 Erabilpen kasua	15,00
A5.4 Aurkezpen Prestatu	10,00
TOTALA	**348,00**

2.1. Taula: Dedikazioaren estimazioa



2.3. Irudia: Dedikazio estimazioa portzentaian

2.6.3 Lan-karga

Estimazioan eta diagraman ikus daitekeen moduan proiektu luzea izango da honakoa. Beraz, izango dugun lan-karga ez da berdina izango beti. Proiektuaren hasieran, otsailean eta martxoan izango dugun lan karga ez da hainbestekoa izango, informazio bilketa eta proiektuari hasiera ematen egongo garelako. Bestetik, behin garapenarekin hasten garenean bai apirilean, bai maiatzean, esfortzu nagusiena kontzentratuko da. Behin proiektuaren muina eginda daukagula, maiatza bukaera,ekaina eta uztail hasieran gauzak bukatzen joango gara.

2.7 Kalitate Plana

2.7.1 Kalitatearen adierazleak

Proiektuaren kalitate maila finkatu ahal izateko, hasteko proiektuak bete beharko duen oinarrizko maila zehaztu egin beharko dugu. Behin oinarrizko maila garatuta, daukagun denboraren arabera, kalitate handiago bat lortzeko aukera izango dugu hobekuntza batzuk inplementatuz.

2.7.1.1 Adierazle kuantitatiboak

- Plangintzaren jarraipena: Proiektua garatuz joan ahala, posible den heinean ziurtatu beharko da plangintzan definitutakoa betetzen dela. Plangintza on bat definitzeak geroago arazo asko ekiditu dakiguke garapenean murgiltzen garenean. Beraz, plangintzaren gidoiari jarraitu beharko diogu. Plangintza betetzen dugula ziurtatzeko jarraipen eta kontrola erabiliko ditugu.

- Errekurtso minimoak erabili: Proiektuaren aurrekontua ahalik eta txikiena egitea izango da gure helburua. Horretarako erosi behar diren elementuak minimoa izatea erabaki da, beharrezko elementu gehienak aurretik egin diren proiektuetatik edota fakultatean eskuragarri dauden elementuetatik hartuz. Biltegia eraikitzerakoan, erabiliko diren materialak birziklatuak izaten saiatuko da.

2.7.1.2 Adierazle kualitatiboak

Oinarrizko maila, irismenean definitu ditugun ezaugarriekin bat datoz.

1. Biltegia eraikitzea

   Proiektuaren oinarria biltegia da. Biltegi honen barruan gure biltegiratze-sistemaren atal gehienak aurkituko ditugu. Alde batetik, egurrezko egitura tinko bat eraiki beharko dugu, hormaren zati bat atera egingo dela kontuan izanda (plataforma mugikorra). Kontuan izan beharko dugu ere, barruan plataforma mugikorra, beso robotikoa eta biltegiratze-eremuak egon behar direla, bakoitzak bere espazio duelarik. Bukatzeko sistema osatuko duten hardware elementu guztiak jartzeko lekua ere izan beharko dugu.

2. Plataforma Mugikorra (Arduino UNO)

   Biltegitik eta biltegira paketeak sartu eta ateratzeko diseinatutako plataformak modu automatiko batean funtzionatu beharko du. Honetarako plataforma serbomotor baten bidez kanpora eta barrura sartu beharko da, irteera/sarrera hau distantzia-sentsore batzuen bitartez kontrolatuz. Guzti hau Arduino UNO baten bidez kontrolatu beharko da. Gainera, beso robotikoa kontrolatzen duen Arduino Megarekin komunikatu beharko da HC-05 Bluetooth modulua erabiliz. Komunikazio honen bitartez plataformak jakingo du noiz ireki eta noiz itxi beharko den. Paketeak identifikatu ahal izateko RFID teknologia bitartez paketeak izango dituzten Mifare etiketak irakurriko ditu.

3. Beso Robotikoa (Arduino Mega)

   Beso robotikoa gure biltegiaren ardatza izango da. Honek automatikoki sartzen diren paketeak eta atera nahi diren paketeak hartu eta utzi egin beharko ditu, modu antolatu batean. Beso robotikoa kontrolatzen duen Arduino Megak, web aplikaziotik wifi bitartez, aginduak jasoko ditu. Hauei jarraituz, pakete bat biltegiratu egingo du edota jadanik biltegiratuta dagoen pakete bat atera. Horretarako, ezinbestekoa izango da beso robotikoa eta plataformaren arteko Bluetooth bitartezko komunikazioa, HC-05 moduluak erabiliz. Gainera, biltegian dauden paketeen administrazioa eramango du, okupatuta dauden biltegiratze-eremuak eta libre daudenak kontrolatuz.

4. Erabiltzaile eta administratzaile web aplikazioa

   Web aplikazioak interfaze baten bitartez erabiltzaileari gure biltegiratze-sistemarekin eragiteko aukera eman beharko dio. Horretarako, erabiltzaile erregistratu batek gordeta dituen paketeak ikusi ahal izango ditu, edozein pakete ateraz edota berarentzako edo beste erabiltzaile batentzako pakete berri bat gordez. Bestetik, banatzailea den erabiltzaileak sisteman erregistratuta dagoen edozein erabiltzaileri pakete bat utzi ahal izango dio. Azkenik, administratzaileak gordeta dauden pakete guztiak ikusteaz gain, sisteman erregistratuta dauden erabiltzaileen administrazioa eramango du, edozein erabiltzaile ezabatu dezakelarik. Beraz, erabiltzaile mota ezberdinek biltegiarekin izan dezaketen elkarrekintza hau aurrera eramateko, aplikazioa beso robotikoarekin wifi socket bitartez komunikatu beharko da.

5. Sistemaren fidegarritasuna

   Sistema osoa fidegarritasun oinarrizko maila batera ailegatu beharko da. Kontuan izanda prototipo bat garatu egingo dela, ezinezkoa izango da produktu erreal batek izango duen fidegarritasun batera ailegatzea. Hala ere, saiatuko da gure prototipoak fidegarritasun maila nahiko bat izatea, non bete behar dituen funtzioak modu egoki batean betetzen dituen.

6. Kodearen ulergarritasuna:

   Informatika proiektu bat garatzen denean, beti presente izan behar den ezaugarri bat da argitasuna. Hain zuzen ere, kodea inplementatzerakoan, kontuan izan behar dugu, kode hori beste pertsona batzuek erabiliko edota ulertu beharko dutela. Beraz, ezinbestekoa da gure kodea ulergarria izatea. Horretarako, kodea iruzkinekin osatu beharko dugu. Honetaz aparte, komenigarria den bezala, aldagaiei jarriko zaizkien izenak egiten dutenaren erreferentzia izan behar dira. Gainera, behin kodea exekutatzen denean, prozesuan zehar monitorizaziorako mezu batzuk inprimatzea komeni da, prozesuari jarraitu ahal izateko.

Hauek izango dira bete beharko direnak, gure oinarrizko mailara ailegatu ahal izateko. Izan ere, hauek beharrezkoak baitira proiektua arrakastaz bukatu ahal izateko. Hala ere, kalitate gehigarri moduan beste hainbat faktore definitu dira. Hauek, denboraren arabera egingo dira, eta kalitatea areagotzeko balio izango digute.

- Web aplikazioaren itxura: Web aplikazioak oinarrizko garapena izango du, hau da, bere funtzionalitatea betetzeko beharrezkoak diren elementuez osatuta egongo da. Denbora soberan izanda, honen interfazearen itxura hobetu ahal izango da. Horrela web aplikazioren kalitateak gora egingo luke.
- Biltegiaren kontrol manuala: Adiministratzailei funtzionalitate bat gehitu. Honen bitartez, biltegiratze-sistemako elementuak manualki kontrolatu ahal izango ditu. Kontrol hau, joystic baten bitartez gauzatuko da. Kontrol hau hobeto egin ahal izateko, sistemari kamara bat gehitzea ere ondo legoke, barruan gertatzen dena monitorizatzeko.
- Biltegia eta web aplikazioaren arteko komunikazioa: Diseinuan, komunikazioan errore bat gertatzen bada, aurreikusten den bakarra bi aldeek, bai web aplikazioak eta baita Arduino Megak ere, errorea detektatzea da, sistema egoera egonkor batera administratzaileak eraman beharrez. Baina, erroreak detektatzeaz gain, sistemak egoera egonkor batera modu automatiko batean itzultzea inplementatu ahalko da. Adibidez, komunikazioa galtzen bada pakete bat plataforma mugikorretik sartzen den momentuan, pakete hori kanporatu ahal izateko aukera inplementatzea aurreikusi da. Beraz, komunikazio erroreak identifikatzean sistemak egoera egonkor batera iristeko egin beharrekoak identifikatzea eta egitea izango da helburua. Hau denbora izan ezkero, inplementatuko den lehenengo funtzionalitatea izango da.

2.7.2 Kalitatea ziurtatzea

Kalitate planean definitutako oinarrizko maila betetzen dela ziurtatu beharko dugu. Horretarako proiektu guztian zehar jarraipen eta kontrolara eskainitako orduak erabiliko dira, egiaztatzeko kalitate plana betetzen goazela.

2.7.3 Kalitatearen kontrola

Etengabe, proiektua bizitza zikloan zehar, egiaztatu egingo da oinarrizko maila betetzen den edo ez. Horretarako, definitu egin dugun oinarrizko maila bete den edo ez zehazteko, egiaztapen batzuk egin beharko ditugu. Bestetik, behin oinarrizko maila bete egin dugula, kontuan izan beharko dugu kalitatea gehitzen duten hainbat funtzionalitate bete ditugun edo ez eta zein kalitaterekin bukatu dugun proiektua.

2.8 Lan metodologia

Proiektua hainbat ziklo/atal ezberdinetan banatu dugu. Hasteko, egin nahi denaren ideia garatu behar da eta horretarako informazio bilketa sakon bat egin beharko da.

Behin gure proiektua garatzeko behar diren teknologiak definituta, plangintza bat burutu egin da. Honetan proiektuaren helburuak eta hauek lortzeko jarraitu behar den metodologia eta orduak irudikatuz.

Plangintzarekin bukatu ahala, garapenarekin hasiko gara. Batzuetan informazioa bilatu egin beharko dugu, beste kasu batzuetan tutoreekin bilerak egin beharko ditugu arazo/zalantzak argitzeko. Gainera, inbertitutako orduak kontrolatu beharko ditugu desbiderapenak ekiditeko. Gerora memorian egindakoa irudikatzeko, eguneroko lana bukatzean, honetan inbertitutako orduak eta egindako lanaren deskripzio baten erregistro bat eramango da.

Oinarrizko mailara ailegatzean, proba sakonak egingo dira produktuaren kalitatea ziurtatzeko. Dena dela, nahiz eta proba finalak garrantzitsuenak izan, garapenarekin aurrera egin ahala probak egingo dira, egindakoaren funtzionamendua egiaztatzeko. Beraz, esan dezakegu probak eta garapena modu paraleloan egingo direla.

Behin kalitate plana bete egin dela, memoria idaztera pasatuko gara. Proiektua egiten joan ahala memoriako zatiak joan gara idazten eta osatzen, hala ere, azkenean guztia osatu, elkartu, txukundu eta antolatu egin beharko da.

Azkenik, egin behar ditugun erabilpen gida eta defentsarako materiala prestatuko ditugu.

Halaber, astero garatu egin diren kode edo dokumentuak Drive, online biltegiratze sisteman, gordeko dira, segurtasun kopia gisa izateko.

2.9 Parte interesatuak

Proiektuaren interesatu nagusia proiektuaren egilea bera da. Proiektuaren erantzukizuna harena da, eta harena proiektu arrakastatsu bat garatzea.

Beste ikuspuntu batetik, proiektuaren zuzendariak parte interesatuak izango dira ere. Gure tutoreak Txelo Ruiz eta Elena Lazkano dira. Hauen ardura proiektuaren egilea gidatzea eta beharrezko zuzenketak proposatzea da.

Azkenik, proiektua ebaluatu eta aztertuko duten epaimahaikideak interesatuak izango dira ere. Hauek, proiektuaren kalitate maila zehaztuko dutenak direlako. Epaimahai hau, konputagailu ingenieritza arloko irakasleaz osatuta egongo da.

2.9.1 Komunikazio plana

Proiektua garatzerakoan agertuko diren zalantza eta arazoak argitzeko proiektuaren tutoreekin harremanetan jarriko gara. Horretarako, EHUko posta elektronikoa erabiliko dugu. Arazo edo

zalantza txikia bada zuzenean posta-elektronikoaren bidez argituko da, ordea, zalantza edo arazoa handiak badira bilera bat egiteko data adostuko da.

Bilera guztietan hitz egin denaren erregistroa egingo da, bilera aktetan jasoz. Hauek, C eranskinean aurkitu ditzakegu.

2.10 Eskuraketen kudeaketa

2.10.1 Software baliabideak

Programatzeko bereziki bi plataforma erabiliko dira: Arduino IDE[2] eta NetBeans[3]. Bestetik, gure web aplikazioa exekutatzeko beharrezkoa diren Apache zerbitzaria eta MySQL datu baseak lortu egingo ditugu XAMPP instalazio paketetik.

Gure proiektua garatzeko erabili egin den sistema-eragilea Windows izan da; hala eta guztiz ere erabili egin diren ingurune guztiek bai Windows eta baita Ubuntun funtzionatzen dute, beraz, posible izango litzateke beste sistema-eragile batean gure proiektua exekutatzea.

Memoria idazteko beste hainbat software erabili ditugu. Memoria idazteko Microsoft officek eskaintzen dituen Word eta Excel herramintak erabili egin dira, dokumentu honek jarraitu beharrezko egiturak eta itxurak mantendu ahal izateko. Plangintzan ere, Gantt diagrama osatzeko GanttProject softwarea erabili egin da. Azkenik, hainbat grafiko egituratzeko online eskuragarri dagoen draw.io herraminta erabili da.

2.10.2 Hardware baliabideak

Proiektua garatu ahal izateko hainbat material eskuratu behar izan ditugu.

Hasteko, biltegia eraikitzeko erabili diren tresnak eta materialak eskuratu ditugu. Ahal den neurrian, materialak eta tresnak etxetik eskuratu dira.

Bestetik, biltegiratze-sistema automatizatzeko behar diren bi Arduino plakak, serbomotorra, distantzia-sentsoreak, RFID modulua, wifi shield-a, bi HC-05 moduluak eta Lynxmotion etxeko AL5D beso robotikoa proiektuaren zuzendaria den Elena Lazkanoren bidez eskuratuak izan dira. Kontuan izan behar da, gure helburuen artean aurrekontua ahal den neurrian txikia izatea dela, eta beraz, saiatu gara fakultatean eskuragarri diren materialak erabiltzen.

Proiektuko sistema osoa garatzeko Windows 10 duen PC eramangarri bat erabili da.

2.11 Arriskuen plana

Proiektua ondo joan dadin ezinbestekoa da egon daitezkeen arriskuak aurreikustea eta hauek gertatuz gero konponbide bat planifikatuta izatea. Arrisku guztiek ez dute eragin berdina izango proiektuarengan, ezta gertatzeko probabilitate berdina ere. Ondorioz, arriskuak ondo identifikatzea beharrezkoa da.

Arriskuak bi taldetan multzokatu ditugu. Produktuarekin zerikusia daukaten arriskuak, hau da, biltegiratze-sistemaren funtzionamendua arriskuan jartzen dutenak eta beste aldetik, proiektuaren garapena arriskuan jartzen dutenak daude.

---

[2] *Arduino IDE* Arduino plakak programatzeko erabili egiten den garapen ingurunea da.

[3] NetBeans garapen ingurune libre bat da. Moduluen bitartez bere ezaugarriak zabaldu daitezke. Gure C++ programak garatzeko erabili den garapen ingurunea da.

2.11.1 Produktua

- Serbomotorrak funtzionatzeari uztea

  Produktuko hainbat ataletan serbomotorrak erabiltzen dira. Hain zuzen ere, plataforma mugikorrean eta beso robotikoan erabiltzen dira. Hauek funtzionatzeari uzten badiote, sistema guztia bertan behera geratuko lirateke. Arazoa, funtzionatzen ez duen serbomotor berri batengatik aldatuz konponduko litzateke.

  Probabilitate: Txikia

  Eragin: Handia

- Distantzia-sentsoreak ez funtzionatzea

  Distantzia-sentsoreak funtzionatuko ez balu, plataforma mugikorraren mugimendua ez zen izango kontrolatua, eta beraz, plataforma apurtu edo funtzionamendu okerra izango luke. Honek, paketeak apurtu edo plataforma erabiltzen dituen errail eta serbomotorraren hauzketa erakarriko luke. Distantzia-sentsorea aldatu beharko genuke.

  Probabilitatea: Txikia

  Eragina: Ertaina

- RFID sentsoreak ez funtzionatzea

  RFID sentsoreak funtzionatzen ez badu, paketeak ezingo genituzke identifikatu. Hau gertatzen bada, konexioak ondo eginda daudela eta plakaren led-ak piztuta dauden ala ez begiratu beharko dugu. Oraindikan ere funtzionatzen ez badu, proba programa batekin proba egin beharko genuke. Honen emaitzak aztertuz, moduluak funtzionatzeari utzi dion ala gure kodearen arazoa den zehaztuko genuke.

  Probabilitatea: Txikia

  Eragina: Handia

- HC-05 moduluak ez funtzionatzea

  Modulu hauek funtzionatzen ez badute, plataforma eta beso robotikoaren arteko komunikazioa galdu egingo da eta beraz, sistema osoak ez du ondo funtzionatuko. Hasteko, modulua ondo konektatuta daugoela begiratu behar dugu. Konexioak egiztatu ondoren, arazoa jarraitzen badu, prestatuta daukagun proba programa exekutatu beharko dugu. Honen emaitzak aztertuta software arazo baten aurrean gauden edo hardware arazo baten aurrean gauden zehaztuko dugu.

  Probabilitatea: Txikia

  Eragina: Handia

- Wifi konexioa

  Beso robotikoa kontrolatzen duen Arduinoaren eta web aplikazioaren arteko komunikazioa galtzen bada, sistema osoaren automatizazioa eta erabilgarritasuna galdu egingo da. Komunikazioa hiru faktorerengatik galdu daiteke:

  a. Wifi sarea ez dago ikusgai: Sarea ikusgai ez badago, sarea administratzen duen routerrak arazoren bat daukala ondorioztatu dezakegu. Korrontetik deskonektatu delako edo modu egokian funtzionatzen ez duelako. Kasu honetan, routerraren konexioak ondo daudela begiratu edo, bestela, routerraren konfigurazioa aztertzea komeni da.

b. Sarea badago, hau da, routerrak modu egokian funtzionatzen badu, baina gure Arduino/Ordenagailua ezin bada bertara konektatu: Kasu honetan, arazoren bat daukagu, bai konexioaren konfigurazioarekin, bai routerrari bidalitako kredentzialekin edota routerrak dituen mugekin.

c. Bi osagaiak sarera konektatuta badaude, baina beraien arteko komunikazioa ezinezkoa da: Hau da, ping bat bidaltzen badugu ez da beste aldera ailegatuko. Kasu honetan, komunikazioan zeozer ez doa ondo. Routerrak ez dituelako paketeak ondo bideratzen edo bidean paketeak galdu egiten direlako. Hau gertatu daiteke komunikazio sareko elementuren batean botila-lepoa efektua gertatzen delako.

Behin sareko egiaztapen hauek egin ditugula eta argi daukagunean arazoa konexioarekin zerikusia ez duela, errorea gure softwarean dagoela ondorioztatuko dugu.

Probabilitatea: Txikia

Eragina: Handia

- Web aplikazioa exekutatzen duen zerbitzariak ez funtzionatzea.

  Erabiltzaileak ezin badu web aplikazioa ireki, sistema eta erabiltzailearen arteko interakzioa galduko da. Honek gure sistemak ez funtzionatzea erakarriko du. Arazo honek iturri ezberdinak izan ditzake, adibidez, Apache zerbitzariak ez funtzionatzea, inplementatutako web orrialdeak ez funtzionatzea, datu basearekin dagoen komunikazioa galdu izana, etab. Horretarako, XAMPP instalazio paketeak daukan administradore panelaren bitartez ikusi ahal izango dugu gure zerbitzariaren egoera.

  Probabilitatea: Ertaina

  Eragina: Ertaina

2.11.2 Proiektua

- Denbora falta

  Baliteke proiektua bukatzeko edota oinarrizko kalitatera ailegatzeko denbora gabe geratzea. Honek proiektuaren entrega eta beraz, defentsa atzeratzeko beharra ekarriko luke ere. Gainera, ondorioztatu ahal izango genuke egindako planifikazioa ez dela egokia izan, bai erabiltzen ari garen teknologiak berriak direlako edota atal batzuetan planifikatutako denbora baino gehiago inbertitu behar izan dugulako.

  Probabilitatea: Txikia

  Eragin: Handia

- Egindako lana galtzea

  Gerta daiteke egindako lana galtzea, proiektua garatzen ari garen ordenagailua apurtu daitekelako, edota gorde egiten dugun online biltegiratze plataformarekin arazoren bat eduki dezakegulako. Honek ondorio larriak izango lituzke. Beraz, honen aurrean, hartutako neurriak, bai produktua bai memoria leku batean baino gehiagotan gordetzea izango da, hau da, babes-kopiak edukitzea. Horretarako, kopia bat proiektua garatzen ari den ordenagailuan gordeko da eta beste bat Drive online biltegiratze plataforman.

  Probabilitatea: Txikia

  Eragin: Handia

3. Erabilitako teknologiak

3.1 Aurrekariak

Gaur egun, biltegiratze-sistema automatikoko hainbat adibide ditugu. Alde batetik, industrian, gero eta gehiago, teknikak inplementatzen ari dira, biltegira ailegatzen diren materialak edo paleak automatikoki antolatu eta biltegiratzeko. Hauetan, beso robotikoez aparte, uhal garraiatzaileak erabiltzen dira ere, paketeak biltegira sartzeko eta biltegitik ateratzeko.

Beso robotikoak garatzen dituen eta hainbat prozesuak robotizatu eta automatizatu duen enpresa bat KUKA (https://www.kuka.com/) da, adibidez. KUKA nahiz eta XX. mendean sortu, urteetan zehar bere produkzioa bilakatzen joan da, 1970 hamarkadatik aurrera robotika munduan zentratuz. Ondorioz, munduko lehen robot industriala sortu zuten, FAMULUS izenekoa. Hemendik aurrera, KUKA automatizazio munduko liderretariko bat bihurtu egin da. Ez bakarrik hainbat funtzionalitate betetzen dituzten beso robotiko ezberdinak garatzen, baizik eta prozesu industrial handiko automatizazioan ere erreferente bihurtu dira.

ULMA enpresa biltegi automatizatu hauen erreferentea da ere. Honek biltegiak automatizatzeko hainbat erantzun eskuragarri dituzte. Adibidez, Unit Load biltegiratze-sistema daukate. Hauek paletak modu automatiko batean gordetzeko erabili egiten dira, biltegiratze- sisteman zehar dauden armairu batzuetan. Hau enpresen eskaerak prestatzeko eta ailegatzen diren elementuak biltegian modu automatiko batean gordetzea ahalbidetzen dute. Biltegiratze-sistema honetan objektuak uhal garraiatzaileetan eta armairuko pisu ezberdinetara ailegatzeko transgoragailu batzuen bitartez mugitzen dira. Adibidez, ULMA (http://www.ulmahandling.com/es) enpresa honek automatizatu egin du Euskal Herriko Bernardo Ecenarro (BESA), sektore industrialerako pintura eta estaldurak egiten duen enpresak Azkoitin daukan biltegia. Biltegi honetan, salgaiak modu automatiko batean garraiatzeko eta hauen biltegiratze-sistema eta eskaera eraketa automatizatuak daude. Prozesuan zehar sortu diren produktuak zuzenean automatikoki biltegian gorde egiten dira eta automatikoki dagokien biltegiratze-eremuetara garraiatuak izaten dira. Honekin BESA enpresak modu esponentzialean haren errendimendua handitu du, erabiltzaileei beraien eskaeren entregatzea 24 ordutan ahalbidetuz (http://www.ulmahandling.com/es/casos-de-exito/automatizacion-logistica-besa).

Biltegiratze-sistema automatizatuen munduan, Amazon enpresak aurkeztu zuen KIVA sistema erreferentea da ere. KIVA lurretik doazen robot laranja batzuk dira. Hauek biltegi guztian zehar dauden biltegi/armairuak alde batetik bestera mugitzeaz arduratzen dira. Armairu hauetan Amazonek dituen hainbat produktu gordeta egongo dira. Beraz, erabiltzaile baten eskaera osatzen ari den langileak ez ditu produktuak biltegi osoan zehar hartzen joan behar, baizik eta KIVA robotak eskaera osatzeko produktuak aurkitzen diren armairuak langilearen lan-estazioara garraituko ditu. Behin langileak armairutik nahi duena hartzean, KIVA robotak armairua dagokion lekura bueltatuko du. Honek eskaera egituratzerako bataz-besteko denbora 90 minutu izatetik 15 minutu izatera bilakatu du. Biltegiaren antolamendu egokia eta KIVA sistemak produktibitatearen errendimendu altua izatea ekarri du. (https://www.cnet.com/es/noticias/conoce-al-robot-kiva-el-hacendoso-empleado-de-amazon/).

Orain arte ikusi dugun moduan, paketeen identifikazioa modu ezberdinetan egiten da. Adibidez KIVA sisteman langileak izango dira produktuak biltegira ailegatzen direnean armairuetan eskuz sartuko dutenak eta beraz, hauen esku geratzen da sisteman sartzen ari diren produktuak modu egokian identifikatzea. Bestetik, ikusi dugun BESA enpresaren biltegian, zuzenean garapen prozesutik produktuak biltegira sartu egingo dira, sistemak produktuak non gorde dakielarik.

Irakurri ahal izan dugun moduan, paketeak identifikatzeko modurik erabilgarrienak RFID edo QR kode teknologiak dira.

Prozesu industrialak alde batera utzita, kontsumitzaileek erabiltzen dituzten beste biltegiratze-sistemak aurkitu ditugu. Adibidez, internet bidez egindako erosketak biltegi batean uzteko aukerak gero eta gehiago hedatzen ari dira. Askotan gertatzen den moduan, paketeak etxera ailegatzen direnean, etxean hauek hartzeko ez dago inor eta ondorioz, geroago korreo enpresa horren egoitzara joan behar gara, gure paketea hartzera. Hori ekiditeko asmoz, hainbat lekutan, gehienbat zentro komertzialetan, inplementatzen ari dira biltegi-sistema automatizatuak (3.1 irudia). Hauetan erabiltzaileak diru kantitate baten truke biltegi-espazio bat alokatzen du. Horrela, online erosketa bat egitean biltegiratze-espazio horren helbidea emanda, paketea bidaltzean, banatzaileak biltegiratze-sisteman sartuko du, paketearen jabeak berak nahi duenean hartu dezan.

Horrelako sistemak Euskal Herriko hainbat lekutan aurkitzen dira, Hapiik (http://www.hapiick.com/recibir-compras-online/) enpresari esker. Enpresa honek bere zerbitzuak eskaintzen ditu Gasteizen, Bilbon, Donostian, Arrasaten eta Euskal Herriko beste hainbat hiritan. Hapiik enpresaren web orrialdearen bitartez, sisteman erregistratu egiten zara eta hauek dituzten biltegi guztietatik bat aukeratu ondoren, leku horretako biltegi-sistemako eremu bat esleitzen dizute. Momentu horretatik aurrera, online erositako paketeak biltegi horretara bidaltzeko eskatu dezakezu, enpresak eskuragarri jarritako helbidea erabiliz. Behin banatzaileak paketea bertan utzi ondoren, erabiltzaileari SMS bat bidaliko zaio, paketea ailegatu dela abisatzeko eta gordeta dagoen armairua irekitzeko kodea bidaliz. Erabiltzaileak nahi duenean pakete horren bila joan daiteke.

Sistema hauek guk inplementatuko dugun sistemaren antzekoak dira. Hau da, lehenik eta behin, paketea sartu edo aterako duen erabiltzailea identifikatu egin beharko da. Behin identifikatuta, erabiltzaile horri enpresak esleitutako armairua automatikoki ireki egingo da eta paketea eskuz hartuko du. Ikastetxeko armairuak bezala funtzionatzen dute, kasu honetan giltza fisiko bat izan beharrean, pantaila baten bitartez dagokion ezkutuko kodea sartu ondoren, modu automatizatu batean ireki eta ixten direnak. Horrelako zerbitzuak eskaintzen dituen beste enpresa baten izena My Pick Box da (http://lanzadera.es/proyecto/my-pick-box). Honek ikusi dugun Hapiik enpresaren funtzionamendu berdina dauka, baina kasu honetan Espainar estatura zabaltzen dituzte beraien zerbitzuak.

3.1. Irudia: Hapiik enpresaren biltegia

Laburbilduz, gure proiektua aztertu ditugun industriako biltegiratze automatizatuak eta online erosketak jasotzeko sistemak elkartzen saiatuko da. Horrela erabiltzaileak bere paketea

hartzeko fisikoki armairu batetik hartu beharrean, zuzenean sistemak pakete hori aterako dio, biltegiratze-sistemak izango dituen biltegiratze-espazioak izkutuan geratuz. Hainbat lekutan janari makina-saltzaileak egiten duten moduan.

3.2 Erabilitako teknologiak

Jarraian, gure produktua aurrera eramateko erabili ditugun teknologiak eta moduluak aztertuko ditugu, hauen funtzionamendua azalduz. Gainera, elementu bakoitzak aukeratzearen arrazoiak ere aztertuko ditugu.

3.2.1 Hardwarea

3.2.1.1 Arduino

Arduino mota ezberdinak egon arren, guztiek oinarri berdina dute. Hardware libreko plataforma da, mikrokontrolagailu eta plaka batez osatua. Honetaz aparte, Arduino proiektuak softwarea garatzeko ingurune prestatu bat izatea eta garapen-komunitate handi bat izatea bere arrakastaren faktoreak izan dira. Honekin batera, gaur egun hain modan jarri den “Do it yourself” etikaren adierazle garrantzitsuenetariko bat da.

Arduino programatzeko garaian, programazio lengoaia eta ingurune espezifikoak erabiltzen dira. Gure Arduinoa programatzeko Arduino IDE-a erabili egiten dugu. Hau beste bi plataformetatik eratorria da. Alde batetik Processing plataforma erabiltzen da. Hau Javan oinarritutako programazio lengoia eta garapen ingurune batez osatuta dago. Honen garapena 2001-ean hasi zen MIT-n. Processing ez dago pentsatuta programatzailentzako, baizik eta ikus-entzunezko diseinatzailentzako. Honekin grafikoak eta figura geometrikoak egin ahal baitira.

Bestetik Wiring daukagu. Wiring programazio lengoaia batez, garapen ingurune batez eta mikrokontrolagailu batez osatutako plataforma da. Wiring-ek Processing oinarri bezala dauka. Wiring plataforma bitartez, mikrokontrolagailuak programatzea eta elektronika prototipoak garatzea ahalbidetzen digu. Honen IDE-a Javaz idatzita dago baina C/C++ GCC konpiladoreak ditu. Honekin garatzaileari elektronikarekin eta sarrera/irteera lanak errazten die. Gainera, Arduinon gertatzen den moduan Sketch [4] egitura setup eta loop funtzioetan banatu egiten da.

Beraz, hauek integratuta aurkitzen dira Arduinok web-an eskuragarri daukan IDE-an. Wiring plataformatik C/C++ konpiladoreak hartuz eta Processing-etik garapen ingurunea hartuz. Abantaila moduan, Arduinon behin programatzen denean, txartelak guztiz autonomoak dira.

Arduinoarekin proiektu oso ezberdinak egin daitezke. Ezaugarri ezberdineko proiektu oso sinple edo konplexuak sortu eta programatu daitezke. Nahiz eta merkatuan aurkitu daitezkeen beste mikrokontrolgailuekin konparatuta, graduan erabilitako PICH 24-arekin konparatuta adibidez, prozesadore eta memoria ahalmen txikiagoa izan, besteak baino merkeagoa da. Bestalde, Arduinorekin bateragarri diren eragingailu eta sentsore katalogoa oso zabala da. Hauekin batera, Arduinok hainbat modulo eskuragarri jartzen ditu, normalean komunikazio ahalmena handitzeko helburuarekin. GPS antena bat, Bluethooth konektibitatea gehitzeko edo wifi konektibitaterako modulu bat. Aitzitik, komunikazio mundutik aparte, beste eragingailu batzuk kontrolatzeko moduluak daude, adibidez DC motorrak kontrolatzeko. Arduinok eskuragarri jartzen dituen modulu ofizialei shield-ak deitzen zaie. Modulu hauek Arduino plakaren gainean jartzen dira, dorre forma hartuz.

[4] Arduino programak *Sketch* deitzen dira eta .ino extentsioa daukate. Hauek setup() eta loop() funtzioez osatuta daude beti.

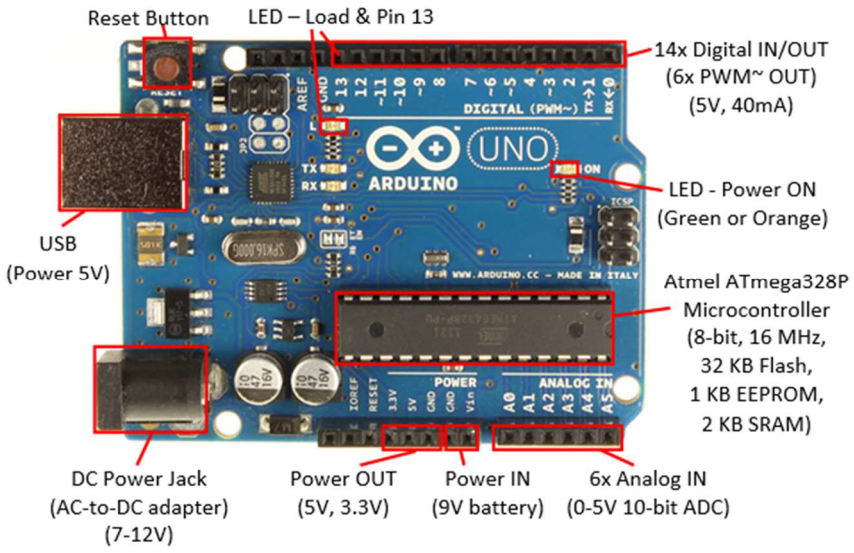
Modulu ezberdin asko egotea eta Arduino komunitatea oso handia izatea bi faktore oso garrantsitzuak izan dira Arduinoa aukeratzerako momentuan. Alde batetik, modulu katologo hain zabala izateak gure proiektua diseinatzerakoan aukera gehiago aztertzea egin digu, gure proiektua aberastuz. Bestetik, modulu hauek inplementatzerakoan, online lortu izan dugun laguntza eta dagoen informazio mordoa garapena asko erraztu du. Gainera, Arduinok hainbat modulu elektronikan asko sartu gabe kontrolatzea ahalbidetzen du, eskuragarri dituen liburutegiak erabiliz. Kontrajarriz, egia da, Arduinok ahalmen txikia izateak mugak jartzen dituela, baina gure proiektuan egin nahi duguna egiteko nahikoa da.

Lehen esan bezala, hainbat arduino mota egon arren, guk erabili egingo ditugun bi Arduino ezberdinen ezaugarriak aztertzera pasatuko gara orain.

Alde batetik, plataforma mugikorra kontrolatuko duen Arduinoa daukagu. Kasu honetan, Arduinoak kontrolatu behar dituen modulu eta eragingailuak asko ez direnez eta gainera kontrolatzeko errazak direnez Arduino plaka generikoa erabiliko dugu. Hau da, Arduino UNO rev 3.

Arduino UNO plakaren ezaugarriak aztertuko ditugu jarraian. Alde batetik, 14 sarrera/irteera digitalak ditu, hauetatik 6 PWM irteera bezala erabili daitezkeenak. PWM[5] hauek ezinbestekoak izango dira ditugun serbomotorrak kontrolatzeko. Bestetik, 6 sarrera analogiko ditu. Honetaz aparte, plakak beste 6 pin ditu zirkuituak konektatzeko. Hauek 2 GND, bi elikadura pin (bat 3,3 V eta bestea 5V), reset pin bat eta elikadurarako beste pin bat dira.

Azkenik, beste bi konektore daude. Bat USB konektorea izango da, Arduinoa ordenagailura konektatzeko erabiliko dena. Honek bi funtzio ditu, plaka programatzea eta plaka elikatzea. Hala ere, elikatzeko erabili daitekeen beste konektore bat dauka, plakak tentsio handiagoak behar dituenean erabiliko dena. Honek, 7-12V artean elikatzen duen kanpo-elikadura batera konektatzeko balio digu (3.2 irudia).

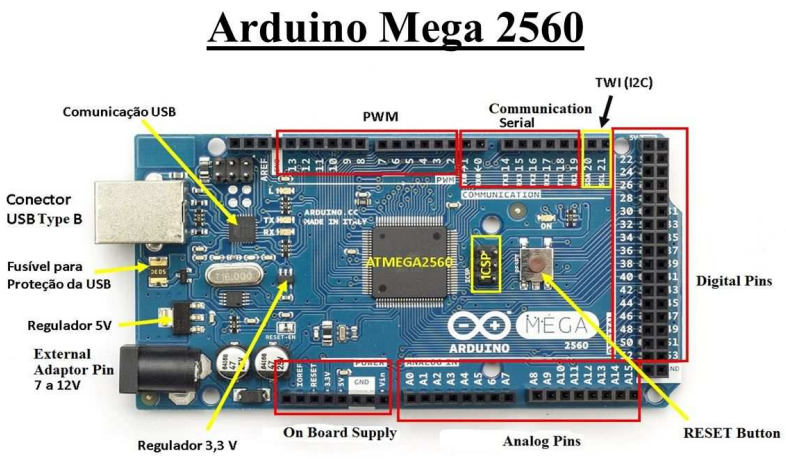


3.2. Irudia: Arduino UNO

Beso robotikoa kontrolatzeko, aldiz, Arduino Mega 2560 hautatu da. Hau ATmega2560 mikrokontrolagailuan oinarritutako plaka da. Honek 54 sarrera/irteera pin digital ditu, honetako 15 PWM irteera moduan erabili daitezkeenak. Bestalde, 16 sarrera analogiko ditu, 4 UART eta reset botoi bat. Azkenik, beste bi konektore ditu. Bat plaka ordenagailura konektatzeko USB-a

[5] *PWM* Pulse Width Modulation edo pultsu zabalera bidezko modulazioa.

da. Honen bitartez, plaka programatu eta elikatu (5V) daiteke. Beste modu batean elikatzeko Jack konektore bat dauka, plaka (7-12V) elikatu ahal izateko (3.3 irudia).



3.3. Irudia: Arduino Mega 2560

Helburu bakoitzerako, Arduino ezberdin bat aukeratzeko arrazoia memoria da. Arduino UNO-k aurrera eraman beharreko prozesuek memoria erabilpen txikia behar dute. Bakarrik identifikazioaz eta prozesu mekanikoaz arduratzen delako. Horregatik, Arduino UNO-k dituen 32 KB-ak nahikoak dira. Hala eta guztiz ere, espero da Arduino UNO-rentzako garatutako programa memoria ia osoa erabiltzea eta pin digital guztiak erabiltzea.

Berriz, Arduino Megak aurrera eraman behar dituen prozesu kantitatea handiago izateaz gainera, aurrera eraman beharreko prozesu horiek Arduino UNO-k aurrera eraman behar dituenak baino memoria eta prozesamendu ahalmen handiagoa eskatzen dute. Gainera memoria soberan izateak gure produktuaren eskalagarritasuna bermantzen du, etorkizun batean gure biltegira funtzionalitate gehiago gehitzea ahalbidetzen duelako. Ondorioz, Arduino Megak daukan 256 KB memoria eta prozesamendu ahalmen handiagok direnez hau erabiltzea erabaki da.

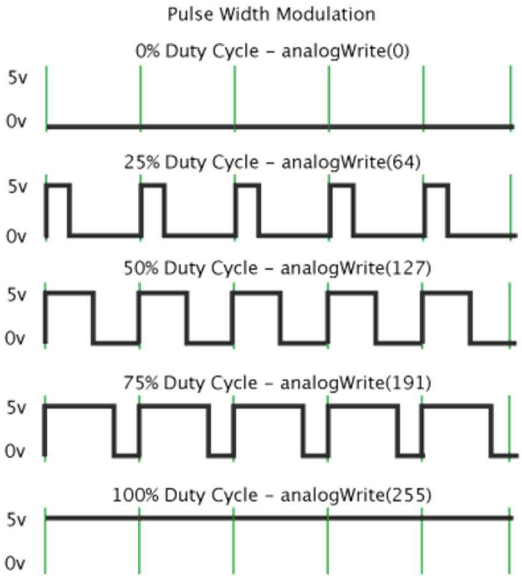
3.2.1.2 Serbomotorrak

Serbomotor guztiek egitura berdina daukate. Hiru kable izango dituzte, kontrolerako bat, elikatzeko beste bat eta GND-a. Normalean serbomotorrak elikatzeko 4-6V-eko tentsioa eskatzen dute, hala ere, motaren araberakoa izango da.

Kontrola PWM (pultsu zabalera bidezko modulazioa) seinalearen bitartez egingo da. PWM pultsu digitalen bidez zirkuitu analogikoak kontrolatzeko teknika da. Erabilpen nagusia gailu elektrikoei hornitzen zaien potentzia maila kontrolatzea da. Hau lortu egiten da, tentsioa eta korrontearen balio medioa, iturrian maila altua eta bajua abiadura haundiz aldatuz. Zenbat eta seinalea maila altuan gehiago egon, orduan eta potentzia handiagoa aplikatuko da kargan. Lan- zikloa (Duty-Cicle) seinale digitalaren pultsuaren zabalera eta ziklo osoaren arteko erlazioa da (3.4 irudia).

$$D = t/T$$

D = Lan-zikloa; t = Seinalearen tarte positiboa; T = Seinalearen periodoa

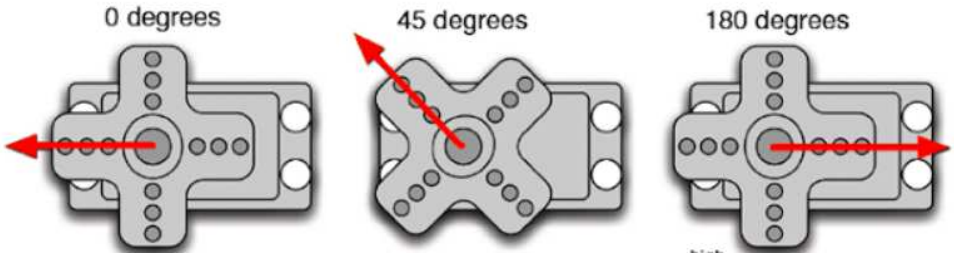


3.4. Irudia: Lan-ziklo ezberdinen adibidea

Gure kasuan, serbomotorren kontrolerako Arduino eskuragarri duen `<Servo>` liburutegia erabili dugu. Normalean, hainbat mikrokontrolagailutan motorraren ezaugarriak jakin behar dira PWM pultsua ondo modulatu ahal izateko, aldiz, Arduino liburutegia berak egiten du eta beraz, erabiltzaileak bakarrik motorraren posizioa definitu beharko du. Serbomotorrek hartzen duten balio tartea 0-180° da.

Bi motako serbomotorrak erabili egingo dira proiektu honetan. Serbomotorrak bi modutan kontrolatu daitezke:

- Abiadura adieraziz, etengabeko biraketa egingo dutenak. Ezaugarri hontako serbomotor bat erabiliko da, gure plataforma mugikorra ireki eta ixteko. Kasu honetan, motorrari adieraziko zaiona biraketa-abiadura izango da, motor ardatza abiadura horretan biratuz. Ardatzak bi norabidetara biratuko du. 90° inguruko balioa serbomotorra geldi egoteko balioa da. Aldiz, erdiko balio honetatik gorako balioak edo beherako balioak norabide baterako biraketa zehaztuko dute.
- Adierazten zaien posizioa hartzen duten serbomotorrak daude. Hauen ardatzek posizio jakin bat hartu egingo dute balio bakoitzaren arabera, eta hor mantenduko dira balio berri bat bidaltzen zaien arte. Hauen balioak ere 0-180 balioen artean aurkitzen dira. Kasu honetan balio bakoitza gradu bati dagokio. 3.5 irudian ikusi daiteke serbomotor baten rotazioa. Erabiliko den ALD5 beso robotikoa horrelako 6 serbomotorrez osatuta dago.

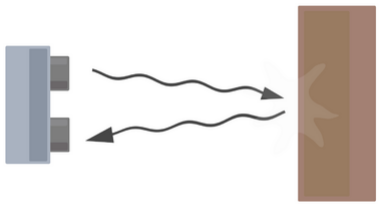


3.5. Irudia: Serbomotorren rotazioa adibidea

3.2.1.3 Distantzia-sentsoreak

Distantzia-sentsoreak, sentsore aktiboak dira. Erabili diren distantzia-sentsoreak ultrasoinuen bitartez funtzionatzen dute. Hauek distantzia txikietan aurrean daukaten objektua zenbateko distantziara dagoen zehazteko erabiltzen dira. Objektua zenbateko distantzia dagoen jakiteko distantzia-sentsore hauek *Time of Fligth* (ToF) teknika erabiltzen dute.

Teknika honen bitartez sentsoreak ultrasoinu uhin batzuk bidaliko ditu, bidaltzerakoan tenporizadore bat martxan jarriz. Seinalearen oihartzuna jasotzean, pasatu den denbora proportzionala da objektua aurkitzen den distantziaren bikoitzara, 3.6 irudian ikusi daiteken moduan.



3.6. Irudia: Distantzia-sentsore funtzionamendua

Honenbestez, distantzia kalkulatzeko hainbat faktore izan behar ditugu kontuan. Ingurune optimo batean, hau da, 20ºC, 50% hezetasun eta itsaso mailako presio atmosferikoa izanda, soinuak 343 m/s -ko abiadura dauka. Unitate aldaketarekin emaitza hau lortzen dugu:

$$343\frac{m}{s} \times 100\frac{cm}{m} \times \frac{1}{10^6}\frac{s}{\mu s} = \frac{1}{29.2}\frac{cm}{\mu s}$$

Hau da, soinuak zentimetro bat zeharkatzeko 29,2 mikrosegundo behar ditu. Beraz, lortu dezakegu distantzia pultsuaren emisio eta jasotzeren arteko denbora oinarri bezala hartuz:

$$Distantzia(cm) = \frac{Denbora(\mu s)}{29.2 \times 2}$$

Denbora hau izango da distantzia-sentsoretik lortuko dugun erantzuna. Denbora zati bi egitearen arrazoia sinplea da. Sentsoreak lortuko duen denbora pultsuak objektura eta objektutik sentsorera behar izan duena da, ondorioz distantzia horren erdia izango da distantzia erreala.

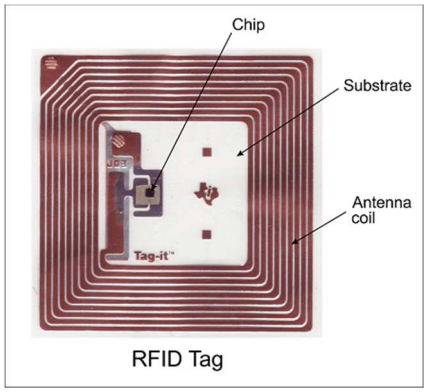
Proiektu honetan URM37 v4.0 bi distantzia-sentsoreak erabili dira. 3.7 irudian ikusi dezakegun moduan, distantzia-sentsore hauek igorle/hartzaile sistemak bananduta dauzkate. Horrelako distantzia-sentsoreetan atal batek ultsaoinuak bidaliko ditu eta besteak honen oihartzunaren zain geratuko da.

3.7. Irudia: URM37

3.2.1.4 RFID irakurle modulua

Gaur egun *RFID (*Radio Frequency IDentification) etiketak gero eta gehiago erabili egiten ari dira QR kodeekin batera, lehen erabili egiten ziren barra-kode edo objektuak identifikatzeko metodoak ordezkatuz. RFID etiketak identifikazio kodea eta batzuetan informazio gehiago dituzten memoriazko chip oso txikiak dira. QR kodeak erabiltzea aztertu zen, baina hauek eskatzen duten prozesamendu ahalmena (kamara erabilpena) Arduinoak daukan ahalmenetik at geratzen da. Horregatik RFID teknologia erabiltzea aukeratu da.

Etiketa hauek duten abantail nagusiena irrati frekuentzien bidez identifikatu egiten direla da, hau da, ez direla ez kable ez kontaturik behar RFID irakurleak etiketak irakur ditzan. Bestalde, etiketak identifikatzeko erabiltzen den irrati frekuentzia berak, etiketak elikatu egiten ditu, eta ondorioz, etiketak ez du elikagairik behar. RFID etiketa baten adibidea daukagu 3.8 irudian.

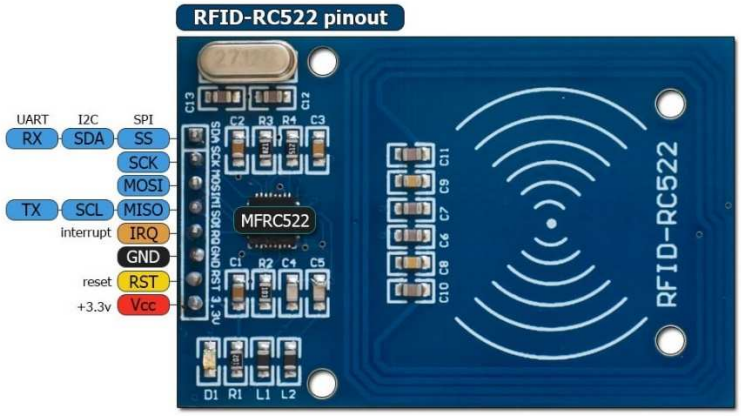


3.8. Irudia: RFID tag adibidea

Normalean RFID etiketek irudian ikusi daitekeen diseinua daukate. Guk erabiliko ditugun etiketak NXP enpresak garatutako Mifare Classics 1KB motakoak dira. Hauek munduan zehar gehien zabalduta daudenak dira. Erabiliko ditugunek 1 KB EEPROM memoria daukate informazioa garraiatzeko (guk erabiliko ez duguna) eta, bestetik, etiketa bakoitza identifikatu ahal izateko 4Byte-ko id-a (biltegiratze-sistemak paketeak identifikatzeko erabiliko duena) daukate. Proiektuan lehen esandako Mifare Classics 1KB motako etiketa itsasgarri batzuk erabiliko ditugu, paketeetan itsatsi ahal izateko.

Arduinok RFID etiketak irakurri ahal izateko 3.9 irudian daukagun RC522 modulua erabiliko dugu. Modulu hau RFID etiketak irakurri eta idazteko balio digun MFRC522 txipan oinarrituta dago. Moduluak SPI[6] interfazea erabiliko du gure Arduinoarekin komunikatzeko. Horretarako gure moduluak dituen pin-ak modu jakin batean konektatu beharko ditugu plakara (3.1 taula), SPI interfazea erabili ahal izateko. Kasu honetan SPI interfaze honen kontrola, modulu hau erabiltzeko erabiliko dugun liburutegiaren barruan egingo da. Erabiliko den liburutegia `<MFRC522>` izango da, geroago 5. kapituluan sakonago aztertuko duguna.

[6] *SPI* (Serial Peripheral Interface) sistema elektronikoen arteko komunikazio protokolo bat da. Komunikazio sinkronoa da. Sinkronizazioa eta datu transmisioa 4 seinaleen bidez egiten da. SCLK (sinkronizaziorako pultsua), MOSI (nagusiaren datu irteera eta morroiaren datu sarrera), MISO (nagusiaren datu sarrera eta morroiaren datu irteera) eta SS (morroia aukeratzeko).



3.9. Irudia: RFID-RC522 modulua

PIN	Arduino UNO
GND	GND
VCC	3.3
RST	9
SDA	10
SCK	13
MOSI	11
MISO	12
IRQ	–

3.1. Taula: Rfid RC522 pin-ak

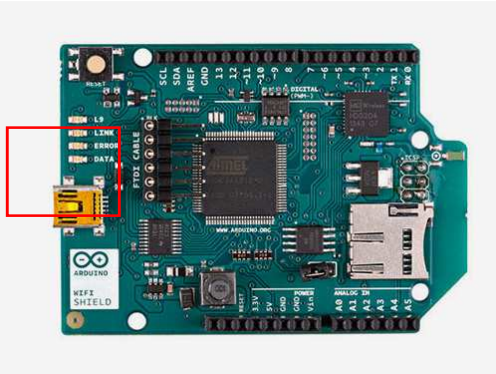
3.2.1.5 Wifi shield

Gure Arduino Mega wifi-ra konektatu ahal izateko Arduinok eskaintzen duen shield ofiziala erabiliko dugu. Hau gure proiektuarentzako ezinbestekoa da, wifi bitartez ematen baita Arduino Mega eta web aplikazioaren arteko komunikazioa. Shield honen bitartez, Arduino Mega konektatu ahal izango da 802.11b/g protokoloa daukaten sareetara. Shield honi esker, web aplikaziorekin sortuko den TCP socket-a sortu ahal izango dugu. Moduluaren kontrolerako, Arduinok liburutegi bat eskuragarri dauka.

Arduinoaren eta modulu hauen arteko komunikazioa SPI bus-a erabiliz egingo da. Shield-a Arduinoa konektatzerakoan zuzenean funtzionatuko luke (Plug and play), beraz, ez ditugu pin bereziak erabili behar. RFID moduluarekin gertatzen den moduan, moduluaren kontrol guztia `<Wifi>` liburutegiaren barnean egingo da. Garapenean liburutegi honek eskuragarri dituen funtzioak erabili egin dira bakarrik, shield-aren barruko funtzionamendua modu sakonean aztertu gabe. 3.10 irudian shield-aren irudia daukagu.

Moduluak hainbat LED erabili egiten ditu, erabiltzaileari moduluaren egoeraren berri emateko.

- L9 (horia) : 9 pin digitalera konektatuta dagoena.
- LINK (berdea) Sarera konektatuta dagoela adierazteko.
- ERROR (gorria) : Sarera konektatzerakoan errore bat gertatu dela adierazteko.
- DATA (urdina) : Datuak transmizioa adierazteko.



3.10. Irudia: Wifi Shield

3.2.1.6 HC-05 moduluak

Gure proiektuan bi Arduino plaka komunikatzeko Bluetooth teknologia erabiltzea erabaki da. Erabiltzen ari garen Arduino plakek ez dute Bluetooth bidezko komunikazio egiteko ahalmena. Ondorioz, bi Arduinoen arteko komunikazioa posible izateko kanpo-modulu bat gehitu behar diegu. Merkatuan modelo asko daude horretarako eta beraz, dauden moduluen arteko azterketa eta aukeraketa bat egin behar izan dugu. Gure kasuan HC-05 moduluak erabiltzea erabaki dugu. Aukeraketaren arrazoiak bi izan dira: lehenengoa gure proiektuaren beharretarako nahikoak direlako. Bestetik, beraien erabilpena oso zabalduta dago Arduino munduan, eta, honenbestez, laguntza aurkitzea erraza izango da (3.11 irudia).

Bluetooth haririk gabeko komunikazio teknologia da. Abantaila garrantzitsuen artean aurkitu dezakegu bere kostu energetiko txikia eta gailuak parekatzeko daukan sistema. Komunikazio abiadura nahiko onak lortu daitezke, hala ere, nahiz eta 100 metroko tartea dituzten gailuak egon, Bluetooth teknologia erabiltzen duten gailuen desabantaila nagusiena konexio distantziak nahiko txikiak izatea da.

Halaber, daukan parekatze-sistemak erabilpena asko errazten du, moduluak piztu bezain laster automatikoki konektatzen baitira. Dispositibo bakoitzak izen batez identifikatu egiten da eta beraz dispositiboak bilatzea erraza da. Dispositibo batekin konektatu ahal izateko honekin parekatzeko eskaera egin eta pasahitzak elkar trukatuz lortzen da. Behin lehengo konexioa ezarri denean, serie-lerro komunikazioa egin daiteke. Gainera, lehenengo konexiotik aurrera dispositiboen parekatzea zuzenean egingo da.

Gure kasuan Bluetooth modulu bakoitzak komunikaziorako funtzio ezberdina bete behar du, batak nagusi bezala jokatuz eta besteak morroi bezala. HC-05 moduluak konfigurazioaren bitartez bi modu hauek ahalbidetzen ditu. Honetaz gain, bi moduluak pizten diren bakoitzean hauek parekatzea lortu behar dugu. Hau ere konfigurazioaren bitartez lortu dezakegu.

Modulua erabiltzeko atal zailena hasierako konfigurazioa da. Arduino plakatik moduluen konfigurazioa aldatzeko, moduluak *AT komando moduan* piztu behar dira. Horretarako moduluaren KEY pin-a gure Arduino plakaren pin digital batekin tentsio altuarekin elikatu behar da, jarraian modulua piztuz. Behin modulua piztu dela eta *AT komando moduan* dagoela, TX eta RX pin-ak erabiliz serie-lerro bidez komandoak bidaliko dizkiogu moduluari beharrezko konfigurazioa definitzeko. Konfigurazioan zehar erabili diren komandoak 3.2 taulan daude.

Komandoa	Funtzioa
AT	UART konexioa egiaztatu
AT+ADDR	Moduluaren Bluetooth helbidea eskatu
AT+NAME	Moduluaren izena eskatu/ezarri
AT+RNAME	Eskuragarri dauden Bluetooth moduluen izena eskatu
AT+ROLE	Moduluaren funtzionamendu mota ezarri (Nagusi edo morroi)
AT+CMODE	Konexio mota ezarri/eskatu
AT+BIND	Parekatzeko Bluetooth helbidea eskatu/ezarri

3.2. Taula: AT komando taula

Behin modulua konfiguratuta daukagula, hemendik aurrera KEY pin-a konektatu gabe erabili egingo dugu modulua. Horrela modulua piztean beste moduluarekin parekatuko dira eta RX eta TX pin-en bitartezko komunikazio normala erabili ahal izango dugu.

3.11. Irudia: HC-05 modulua

Komunikaziora sartzen bagara azpimarratu behar da Bluetooth bidezko komunikazioa Arduino UNO-n aurrera eramateko Arduinok eskuragarri duen `<SoftwareSerial>` liburutegia erabili dela. Arduino UNO-k serie-lerro bidezko komunikazio baterako gaitasuna du eta hau erabili egiten da, USB konektorean. Arduinoa USB-z ordenagailura konektatuta egongo da bai elikatzeko eta baita Arduinoaren prozesuak ordenagailutik kontrolatu ahal izateko. Horregatik, serie-lerro bidezko komunikazioa software bitartez gauzatu behar dugu `<SoftwareSerial>`[7] liburutegia erabiliz, Arduino UNO-ren bi pin digitalak gure komunikazioaren TX eta RX bezala bilakatuz.

Arduino Megan berriz, hardwarean serie-lerro komunikaziorako hardware kanal bat baino gehiago inplementatuta etortzen direnez, ez dugu `<SoftwareSerial>` bitartezko objektu bat sortu behar, zuzenean Arduinok eskuragarri daukan `Serial` erabili ahal dugu.

3.2.1.7 EEPROM memoria

Electrically-Erasable-Programmable Read Only (EEPROM) hainbat alditan idatzi eta ezabatu daitekeen ROM familiako memoria da. Memoria mota hau idazteko, ezabatzeko eta birprogramatzeko elektrizitatea erabiltzen da. Arduino plaka guztiek EEPROM memoria bat

[7] `SoftwareSerial` Arduinok 0 eta 1 pin-etan duen UART-ez gain, pin digitalak erabilita seriezko komunikazioa ahalbidetzen duen liburutegia da. Honek software bidez erreplikatu egiten du hardware bidezko komunikazioaren ezaugarriak.

daukate. Hauen bitartez, informazioa gorde daiteke epe luzerako nahiz eta plakak itzali. Gure proiektuan Arduino Megan aurkitzen den EEPROM memoria erabili egingo dugu, dauden paketeen informazioa epe luzerako gorde ahal izateko. Arduino Megak 4KB-eko EEPROM memoria dauka.

EEPROM memoria honetan ekintzak egiteko Arduinok `<EEPROM>` liburutegia eskuragarri dauka. Honen bitartez zehazten den memoria helbidean bytez byte irakurri eta idatzi daiteke. Gure kasuan byte balioak irakurri beharrean geroago gure garapenean ikusiko dugun moduan (6.2.1 atalan), osokoak eta karaktere-kateak idatzi eta irakurtzea interesatzen zaigu. Horretarako Arduino Foruan (https://playground.arduino.cc/Code/EepromUtil) `EepromUtil` liburutegia aurkitu dugu. Liburutegi honetan inplementatuta dauden funtzio erraz batzuen bitartez EEPROM memorian osokoak eta karaktere-kateak irakurri eta idatzi daitezke. Honek lan asko aurreztu digu, osokoak eta karakete-kateak idazteko egin beharreko bytez byte-ko tratamendua liburutegiak egiten duelako.

Adibidez, `EepromUtil::eeprom_write_int(int addr, int value)` funtzioa erabili dugu. Honen bitartez, guk nahi dugun osoko balioa idatzi dezakegu zehaztutako memoriaren helbide batean. Kontuan izan behar dugu osoko balioak Arduinon 2 byte hartzen dituela eta ondorioz bi helbide okupatu egingo ditu memorian.

Bestetik, karaktere-kateak irakurtzeko `EepromUtil::eeprom_read_string(int addr, char* buffer, int bufSize)`funtzioa daukagu. Honekin guk definitzen dugun helbidetik aurrera gorde egin dugun karaktere-katea irakurri egingo da , guk zehaztutako buffer batean gordez.

Honek izugarrizko abantailak ekarri dizkigu biltegiratze-sistemako paketeen administrazioa egiterako momentuan.

3.2.2 Softwarea

Atal honetan, web aplikazioa garatzeko erabili ditugun teknologiak aztertuko ditugu.

3.2.2.1 XAMPP

XAMPP (X(Sistema-eragilea) Apache MariaDB PHP Pearl) software libreko hainbat teknologiez osatutako instalazio pakete bat da. Hau, Apache zerbitzariaren dohako banaketa da, non Apache zerbitzariaz aparte, datu-baseak kudeatzeko erabiltzen den MariaDB (MySQL-tik eratorria) dauka.

Tresna hau web garatzaile eta programadoreei zuzenduta dago. Honen bitartez, ordenagailu pertsonal batean edota ingurune kontrolatu batean, modu erraz batean, web zerbitzari bat eraiki daiteke. Honek ahalbidetzen die garatzaileei modu seguru batean beraien produktuak lokalki egiaztatzen. Hau erabili da gure proiektuko web aplikazioa garatzeko. XAMPP-ek dituen osagaiak konfiguratzeko eta aktibatzeko, eskuragarri daukan administratzaile panela erabili egingo da.

Honen Windows plataformarako bertsioa instalatu eta erabili da, WAMPP izenekoa.

3.2.2.2 Apache

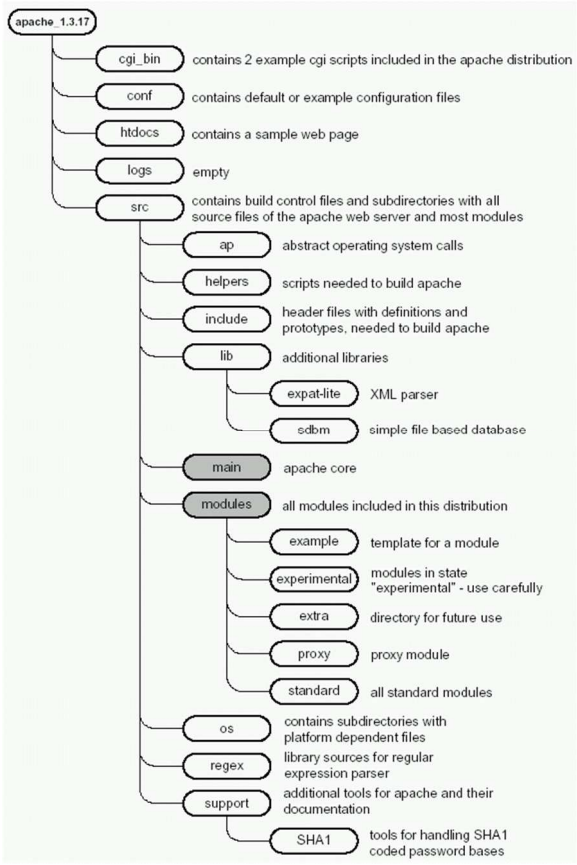
Apache kode irekiko HTTP web zerbitzaria da. World Wide Web-en (WWW) gehien erabiltzen den web zerbitzaria da. Honek dituen abantailak honako hauek dira:

- Modularra.
- Kode irekikoa.
- Plataforma anitzekoa.

- Hedagarria.
- Web zerbitzari erabilena.

Apache web zerbitzarien egitura modularra da. Zerbitzariak nukleo modulu batzuk ditu, bere funtzionamendurako ezinbestekoak. Baina hauetatik aparte, hainbat funtzionalitate gehitu daitezke, komunitateak garatutako beste moduluak erabiliz. Adibidez, PHP lengoaia interpretatu ahal izateko modulu bat dago.

Honetaz aparte, Apachek daukan fitxategi-sistema 3.12 irudian ikus dezakeguna da. Gure kasuan, web arakatzaile batetik eskuragarri utzi nahi ditugun web orrialdeak Apachek daukan *htdocs* karpetaren barruan utzi beharko ditugu. Hau baita Apache zerbitzariak eskuragarri uzten duen karpeta.



3.12. Irudia: Apache fitxategi sistema

3.2.2.3 MariaDB

Datu baseen gestiorako softwarea da. Kode irekiko datu-baseen kudeaketarako software ospetsuena da. MySQL-tik Oracle enpresa erosi ondoren garatu zen kode irekiko bertsioa da MariaDB. MariaDB oso erabilia izaten da web aplikaziotan. Honen ospeak PHP scripting lengoaiarekin zerikusia dauka, honen bitartez MariaDB erabiltzea asko errazten baitu.

3.2.2.4 PHP

PHP kode irekiko scripting lengoaia da, oso erabilia web garapenean. Hau zuzenean HTML kodean jartzen da, HTML kode estatikoari nolabaiteko dinamismo bat emanez. Beraz, zerbitzariak PHP kodea interpretatu eta HTML kode estatiko moduan txertatuko du orrialdean.

Apache zerbitzari aldean exekutatzen den scripting lengoaia da. Hau da, zerbitzariak exekutatu egingo du eta lortutako emaitzak orrialdearen HTML kodean jarriko du, web-arakatzaileari HTML orrialde bat bidaliz. Ondorioz, web-arakatzaileak jasoko duen orrialdea HTML orrialde sinple bat izango da, zerbitzarian gertatu denaz ezer jakin gabe. Javascript[8]-ekin gertatzen den kontrakoa. Kasu horretan scripting-a web-arakatzaileak exekutatu egingo du, erabiltzaile aldean.

3.2.2.5 AJAX

Asynchronous JavaScript And XML da teknika honen izena. Hau web garapenean erabili egiten da, aplikazio interaktiboak lortu ahal izateko. Aplikazio hauek erabiltzaile aldean (web-arakatzailean) exekutatu egiten dira, atzeko planoan. Teknika honek zerbitzariarekin modu asinkrono batean komunikatzea ahalbidetzen du. Horrela, kargatuta daukagun orrialdean aldaketa dinamikoak egin daitezke, orrialdea berriro kargatu gabe, zerbitzaritik lortu den informazio berria aurkeztuz.

Normalean, AJAX JavaScript erabiliz inplementatu egiten da. AJAX teknika erabiltzeko hainbat modu daude, adibidez, `XMLHttpRequest` objektua erabili daiteke. Honen bitartez, orrialdea web-arakatzailean ikusten den bitartean, web-arakatzaileak erabiltzaileak galdetutako informazioa eskatuko dio zerbitzariari. Honek XML formatua daukan erantzun bat bidaliko dio objektuari eta honek zuzenean web-arakatzailean kargatuta dagoen orrialdean jasotako erantzuna bistaratuko du, garatzaileak definitutako orrialdearen esparru jakin batean, JavaScript eta DOM[9] erabiliz.

[8] *JavaScript* interpretatutako programazio lengoia da. Objektuetara bideratutako programazio lengoia da eta web orrialdeen garapenean erabili egiten da. Gehinbat erabiltzaile-aldean inplementatu egiten da, web-arakatzailean dinamismoa eta interfazea hobetu ahal izateko.

[9] *DOM* (ingelesez, *Document Object Model*, hau da, Dokumentuaren Objektu Eredua) W3Cek kudeatutako API bat da. DOM-ak objektu multzo zehatz bat eskaintzen du HTML eta XML dokumentuak erakusteko eta objektu haiek nola erabiltzeko eta aldatzeko arauak ezartzen ditu. Horrela, ECMAScript (JavaScript) bezalako lengoaiekin posiblea da HTML edo XML dokumentu baten edukiak, egitura eta formatoa dinamikoki aldatzea.

4. Sistemaren funtzionalitateen analisia

4.1 Funtzionalitateen analisia

Gure sistemaren funtzionalitateak analizatzerako orduan, kontuan hartu ditugu jada existitzen diren antzeko biltegiratze-sistemak. Existitzen diren sistemek funtzionalitate asko dituzte baina gure denbora eta baliabide mugak direla eta gure irismenetik asko aldentzen dira. Horregatik, aurkitutako funtzionalitate guztien artean sailkapen bat egitea erabaki da: egingo direnak, hobekuntzak izan daitezkeenak eta etorkizunerako ideia bezala geratuko direnak.

Oinarrizko lerroa:

- Erabiltzaileek paketeak biltegiratu eta atera: Erabiltzaileak paketeak sartu edo bere izenean dauden paketeak atera ahal izango ditu sistematik.
- Banatzaileek paketeak sartu: Sistemako erabiltzailentzako paketeak sartu ahal izango dituzte.
- Administratzaileak paketeak atera eta sartzeaz gain, sistemaren erabiltzaileen gestioa egingo du: Administratzaileak pakete guztiak atera ahal izango ditu eta erabiltzaileak sistematik kendu.

Kontuan izan behar da paketeak biltegiratze-sisteman sartzeko plataforma mugikorreko leku eta posizio jakin batean utzi egin beharko direla, gure sistemak dituen muga mekanikoak direla medio. Gainera biltegiratze-sisteman eta web aplikazioan paketeak identifikatzeko pakete bakoitzak daukan Mifare etiketaren id-a erabili egingo da.

Funtzionalitate gehigarriak:

- Administratzaileen funtzionalitate gehigarria: Sistema eskuz kontrolatu ahal izatea joystick baten bitartez.
- Pakete bat ateratzeko eskatzean eta erabiltzaileak paketea hartzen ez duenean, paketea sistemara berriro sartu, zegoen tokian gorde eta sistema aurreko egoerara eramateko ahalmena.
- Sistemaren komunikazioan errore bat egonda, sistema egoera egonkor batean geratzeko ahalmena.
- Web aplikazioren interfazea hobetu.

Irismenetik kanpo:

- Erabiltzaileei sisteman gertatzen denaren berri eman email edo SMS bitartez.
- Tamaina ezberdineko paketeak hartu eta gordetzeko ahalmena.
- Erabiltzaileak paketea edozein modutan utzita ere, identifikatzeko eta beso robotikoak hartzeko gai izatea.
- Erabiltzaileek agindu bat baino gehiago egiteko eskatu eta sistema guztiak asetzeko gai izatea.

4.2 Biltegia

Biltegiaren funtzioa berez erreza da, bere funtzioa modu egoki batean bete ahal izateko beharrezkoa dena gauzatzea, aldiz, ez da hain erraza. Bi funtzio nagusi ditu:

- Alde batetik, paketeak biltegiratuko dituena da eta beraz, hauek mugituko dituen beso robotikoa barruan sartzeko lekua izan behar du. Gainera, paketeak gordetzeko definitu diren eremuak beso robotikoarentzako eskuragarri egon behar dira, paketeak modu erraz batean hartu eta utzi ahal izateko.
- Bestetik, paketeak biltegitik sartu eta ateratzeko plataforma mugikorra bertan egongo da. Hau modu egonkor batean mugitu behar da, ireki eta ixten denean paketeak modu egoki batean garraitu ahal izateko .

4.3 Hardwarea

Hardwareari begira, hainbat modulu ditugu (URM37 distantzia-sentsoreak, RFDI-RC522, HC-05, ALD5 beso robotikoa eta wifi shield-a) eta hauek kontrolatzeko beharrezko diren Arduino UNO eta Arduino Mega.

Arduinoek hardware-osagaiak kontrolatuko dituzte, hau da, biltegia funtzionatzeko ezinbestekoak dira. Arduino Megak, web aplikazioak bidaliko dituen aginduak interpretatu, eta egin beharrekoak aurrera eraman beharko ditu. Horrez gain, beso robotikoaren mugimenduak, biltegiratze-eremuen administrazioa eta plataforma ireki eta ixteko eskatuko dio beste arduinoari HC-05 Bluetooth trasmisorearen bitartez.

Beraz, lehen aipatu bezala, Arduino UNO Bluetooth moduluaren bitartez Arduino Megak bidaltzen dizkion aginduen zain geratuko da. Hau plataformaren mugimenduaz (serbomotorra eta distantzia-sentsoreak) eta paketeen identifikazioz (RFID-RC522 modulua) arduratuko da.

3. kapituluan azaldu dugun moduan bi Arduino hauek beraien eginkizunak aurrera eramateko hainbat osagai/modulu beharko dituzte:

Arduino Mega

- Wifi shield: Arduino Megak wifi konexioa izateko beharrezkoa den modulua. Honek arduinoren gainean jarriko da. Web aplikazioarekin komunikazioa aurrera eramateko behar den wifi socket-a osatzeko beharrezkoa. Modulu honen kontrolarako, Arduinok eskaintzen duen SPI teknologia beharrezkoa da.
- Lynxmotion ALD5 beso robotikoa: 6 serbomotorrez osatutako egitura. Honen bitartez paketeak mugitu ahal izango dira biltegiaren barruan. Honetarako Arduino Megaren 6 PWM pin beharko dira.
- HC-05 modulua: Bi arduinoen arteko komunikazioa gauzatzeko beharrezkoa.
- EEPROM: Arduino Megak eskuragarri daukan EEPROM memoria erabiltzen da biltegiratze-sisteman gordeta dauden paketeen informazioa epe luzerako gorde ahal izateko.

Arduino UNO

- Serbomotorra (HSR-1425CR): Plataformak izango dituen errailak erabiliko ditu, plataforma mugikorra mugitzeko. Honen kontrolerako PWM pin bat beharrezkoa da.
- URM37 v4.0 bi distantzia-sentsore: Plataformaren irekiera eta itxiera kontrolatzeko erabiliko direnak. Bakoitzak bi S/I digitaleko pin behar ditu.

- HC-05: Modulu honek bi Arduinoen arteko Bluetooth bitarteko komunikazioa posible egingo du.
- RFID-RC522 modulua: Paketeak identifikatzeko erabiliko diren RFID etiketak irakurtzeko behar dugun RFID irakurlea. Modulu honen kontrola SPI busaren bitartez egingo da. Hau egiteko berariazko 4 pin erabili beharko ditugu.

Ondorioz, Arduino UNO-k dituen S/I digital guztiak okupatuta izango ditugu, eta beraz sistemari zeozer gehiago gehitu behar bazaio Arduino Megara konektatuko da, honek pin-ak soberan ditu eta.

4.4 Softwarea

Hardwarearekin gertatzen den moduan kasu honetan ere bi atal nagusi bereiztu ditzazkegu: Web aplikazioa eta bi Arduinotan inplementatu beharrezko aplikazioak.

4.4.1 Web Aplikazioa

Web aplikazioaren kasuan ahalik eta errendimendu handiena eta errore tratamendu optimoena lortu behar da. Kontuan hartu beharra dago web aplikazioa erabiltzailearekin elkarrekintza egingo duen atala dela, eta beraz, ahalik eta interfaze sinple, irisgarri eta erabilgarriena egiten saiatu beharko gara. Izan ere, argi utzi beharko da web aplikazioaren funtzionalitate eta erabilgarritasuna bilatzen dela, itxura kalitate-gehigarri moduan definituz.

Web-aren funtzioak erabiltzaileen erregistroa eta saioak kontrolatzea, erabiltzaileek egin ahal dituzten ekintzak aurkeztea eta erabiltzaileak aukeratzen dituen ekintza horiek Arduinoari transmititzea da.

Beraz, Web aplikazio honen bi funtzio mota bereiztu ditzazkegu:

- Biltegiratze-sistemarekin erlazio zuzena daukatenak:

  Hauek paketeak sartzeko eta ateratzeko funtzioak dira. Hauek aurrera eramateko Arduinorekin komunikazioa behar dute. Paketeen identifikazioa egiteko pakete bakoitzak daukan Mifare etiketaren id erabili egingo da.

- Aplikazioaren administraziorako beharrezko funtzioak:

  Funtzio hauek aplikazioaren funtzionamendurako beharrezkoak dira. Kasu honetan, ez dute biltegiratze-sistemarekin erlazio zuzena, baina beharrezkoak dira sistemaren antolamendurako. Hauek erabiltzaileak erregistratu, erabiltzaileak administratu, eta erabiltzaile bakoitzak biltegiratze-sistemarekin egin ahal dituzten funtzioak kontrolatzeko beharrezkoak dira.

Ondorioz, dauden erabiltzaile mota guztiak azaldu eta horietako bakoitzak eskuragarri dituen funtzioak banan-banan aztertuko ditugu. Azaldu beharra dago web aplikazioak erabiltzaile mota ezberdinak izango dituenez, bakoitzak eskuragarri dituen funtzionalitateak atzitu baino lehen sesioa hasi egin beharko dute web aplikazioan, geroago ikusiko ditugun kredentzialak (email, pasahitza) erabilita. Sesioa ireki ondoren ikusiko ditugun funtzionalitateak eskuragarri izango dituzte.

4.4.1.1 Paketeak ateratzea

Funtzio honen helburua erabiltzaileak biltegiratze-sistemaren barruan duen pakete guztien artean berak aukeratzen duen paketea ateratzeko behar den informazioa transmititzea izango

da. Transmititu beharreko informazioa Arduinoari bidaliko zaio wifi socketaren bitartez. Argitu beharra dago aukera hau erabiltzaile arruntak eta administratzaileak baino ez dutela izango.

Hortaz, bidali beharreko informazioa aztertu beharra daukagu. Kontuan izan behar dugu, gure sistemaren barruan, pakete guztiek RFID etiketa bat izango dutela. Paketeak identifikatzeko etiketa hauen id-a erabiliko da eta beraz, aplikazioak Arduinoari bidali beharko dion informazioa atera nahi den paketearen id-a izango da.

Hori dela eta, atera nahi den paketearen id-a lortzeko, hurrengo metodoa aukeratu da:

- Erabiltzaileak paketeren bat ateratzea erabakitzean, erabiltzaileak paketearen id-a zuzenean sartu behar izatea desabantaila ugariak ditu (erabiltzaileak id-a apuntatu edo memorizatu beharra, erabiltzaileak id-a ez jakitea, etab). Metodo hau ez da ez erosoa, ezta egokia ere.

  Halatan, erabaki da erabiltzaileari sisteman bere izenean dituen pakete guztien zerrenda bat aurkeztea. Honela, pakete bakoitzaren informazio orokorra ikusi ahal izango du erabiltzaileak (banatzailea, id-a, deskripzioa) eta bakoitza aukeratzeko botoi bat izango du. Beraz, erabiltzaileak egin beharko duen bakarra atera nahi duen paketeari dagokion botoia sakatzea izango da. Behin paketea aukeratuta, sistema Arduinori id hau transmititzeaz arduratuko da, biltegiratze-sistemak id hori daukan paketea ateraz.

4.4.1.2 Paketeak sartzea

Kontuan izan behar dugu funtzio hau erabiltzaile arrunt eta banatzaileek eskuragarri izango dutela.

Aurreko kasuaren antzekoa izango litzateke baina kontrako funtzioa betez. Honetan erabiltzaileak sartu nahi duen paketearen ezaugarri batzuk formulario baten bitartez bete beharko du, id-a izan ezik, paketea sisteman modu egokian gordetzeko. Kasu honetan web aplikazioak paketea sartzeko eskatu beharko dio Arduino Megari. Ondoren, paketea jadanik gordeta dagoelarik biltegiratze-sistemak erabiltzaileak sartutako paketearen id bidaliko dio web aplikazioari, lehenik erabiltzaileak bete duen informazioarekin batera paketea sisteman erregistratuz. Beraz, ikusi ahal den moduan paketeen identifikazio prozesutik erabiltzailea isolatzen saiatu da, funtzionamendua erabiltzailearentzat ahalik eta errazen suerta dadin.

Azaldu dugun moduan, id-az gain, paketeek beste informazio atal batzuk dauzkate eta hauek eskuratu beharko ditugu, erabiltzaileentzat erosoa den modu batean. Kasu honetan, modu errazena formulario baten bitartez egitea da. Beraz, formularioaren atal bakoitza, erabiltzaileak bete beharko du beharrezko informazioarekin.

- Banatzailea: Erabiltzaile arrunt nahiz banatzaile izanda, banatzaile informazioa beteta agertuko zaio erabiltzaileari. Informazio hau aplikazioan saioa hasterakoan gordetzen den informazioarekin beteko da eta beraz, erabiltzaileak ez ditu bere datuak bete beharko. Gainera ezingo da informazioa aldatu, arazoak ekiditeko.
- Jasotzailea: Jasotzailearen informazioa betetzeko, erabiltzaileari sisteman erregistratuta dauden erabiltzaileen zerrenda bat agertuko zaio eta hauen artean aukeratu beharko du.
- Deskribapena: Erabiltzaileak paketeak barruan daukanaren deskribapen txiki bat idatzi beharko du.

Behin hau lortuta, Arduinorekin transmisioa hasiko da, pakete bat sartzeko eskatuz. Paketea barruan eta gordeta dagoela, Arduinoak paketearen id-a transmitituko dio aplikazioari eta paketea aplikazioak duen datu basean gordeko da.

4.4.1.3 Erabiltzaileen administrazioa

Administratzaileak baino ezingo du funtzio hau bete. Aurreko bi funtzioetan ez bezala, hau aurrera eramateko ez da inongo komunikaziorik behar aplikazio eta Arduinoaren artean, erabiltzaileen administrazioa web aplikazioaren esku geratzen delako, Arduino sistematik guztiz isolatuta.

Funtzioaren helburua dauden erabiltzaileen zerrenda ikustea izango litzateke, eta beharrezkoa balitz, erabiltzaile bat ezabatzea. Erabiltzaile bat ezabatu ahal izateko, ezin du sisteman paketerik bere izenean izan, bestela pakete horiek lehenago atera beharko lirateke sistematik, gero erabiltzailea ezabatu ahal izateko.

Kasu honetan, administratzailearen lana errazteko, informazio hau lortzeko eta ikustaratzeko zerrenda bat aurkeztuko zaio erabiltzaile guztiekin.

4.4.1.4 Erabiltzaileen erregistroa

Erabiltzaile berri bat sartzeko aukera egongo da. Aplikazioan sartzerako momentuan bi aukera izango ditugu: sartu (`LogIn`) eta erregistratu (`SignUp`). Jadanik erabiltzaileak bagara, web aplikazioan sartzen bagara aurretik ikusi ditugun funtzioak egin ahal izatera pasatuko gara. Oraindik erabiltzaileak ez bagara, erregistratzeko aukera izango dugu. Erregistratzeko momentuan bi aukera izango ditugu:

- Erabiltzaile arrunta: Kasu honetan erabiltzaile berri bat sortzeko beharrezko informazioa eskatuko zaio erabiltzaileari formulario baten bitartez. Hurrengo informazioa eskatuko zaio: Izena, abizena, bigarren abizena (aukera), email, mugikor zenbakia, pasahitza, pasahitza errepikatu, pasahitza berreskuratzeko galdera eta erantzuna.
- Banatzailea: Kasu honetan informazioa sartzera pasatu baino lehen, administratzaileak aurretik emandako kode bat sartu beharko du. Hau banatzailea egiaztatzeko erabiliko da. Behin kodea sartuta, erabiltzaile kasuaren formulario bera betetzera pasatuko gara. Hala ere, kasu honetan email-a aurrez definituta egongo da. Egiaztatzeko kode hau, prototipo honetan zuzenean administratzaileak datu-basean eskuz sartutako kode batzuk izango dira. Etorkizunean kode hauen banaketa ondo inplementatzea espero da. Hau da, kodeak bakarrik etorkizuneko funtzionamendua simulatzeko balio digute.

Beraz, informazioa formulario baten bitartez eskatuko zaio erabiltzaileari. Sartzen duen informazioa aplikazioak egiaztatuko du, behintzat sartzen den informazioa egitura ona daukala. Hau behin eginda erabiltzailea erregistratu egingo da.

4.4.2 Biltegiratze-sistema

Kasu honetan bi Arduino izango ditugu eta beraz bakoitzak dituen funtzioak ezberdinak izango dira, nahiz eta funtzio orokor baterako lan egiten ari. Funtzio orokorrak web aplikazioan ikusi ditugunak izango dira, hau da, paketeak ateratzea eta sartzea. Arduino aplikazioek ez dute zuzeneko harremanik izango erabiltzailearekin, hala ere, web aplikazioak eskatzen dituen aginduak bete beharko ditu. Ondorioz, modu orokor batean web aplikazioa Arduino batekin edo sistema bakarrarekin hitz egiten ari dela pentsatuko du, barruan gertatzen dena jakin gabe.

4.4.2.1 Arduino Mega

Arduino Mega izango da web aplikazioarekin komunikazioa mantenduko duena. Hori dela eta, web aplikaziotik etengabe datozen aginduen zain egongo da. Hala ere, nahiz eta Arduinoak ez duen bere kabuz erabaki berririk hartu behar, ez da guztiz web aplikazioaren menpean egongo.

Kontuan izan behar dugu sistema fisikoaren berri Arduinoak izango duela eta, beraz, Arduinoa izango dela sistemaren funtzionamendu egokiaz arduratuko dena. Arduinoari iristen zaion informazioa bete behar ez izatea gertatu dakioke, hardwarrean muga bat dela eta, adibidez pakete gehiago sartu ahal ez izatea. Kasu hauetan, Arduinoak erantzun zuzen bat eman beharko du sistemaren integritatea bermatu ahal izateko. Beraz, esan dezakegu Arduinoa lehenik eta behin sistemaren integritateaz arduratu behar dela eta bigarren plano batean, erabiltzailearengandik datozen aginduak betetzeaz.

Erabiltzailearengandik datozen aginduak bete ahal izateko, paketeak atera eta sartu alegia, Arduino aplikazioak bi zati garrantzitsu izango ditu: Mezua tratatzea eta jasotako mezuaren aginduak aurrera eramatea.

Modu orokor batean Arduino aplikazioak jarraituko duen eskema hurrengoa da:

- Mezua jaso eta prozesatu
- Jasotako agindua bete daitekeen edo ez aztertu
- Aurrera eramaterik badago, agindua bete.

Egin beharreko funtzio nagusienak bi izango dira, paketeak sartu edo atera. Baina hauek betetzeko aplikazioak hainbat zeregin izango ditu. Alde batetik, web aplikazioarekin komunikazioa, beste Arduino UNO-rekin komunikazioa , beso robotikoaren kontrola eta biltegiratze-sisteman dauden paketeen kontrola. Guzti hauek bi funtzio nagusiak betetzeko erabiliko dira.

4.4.2.1.1 Paketeak sartu

- Lehenik eta behin, sisteman pakete berri batentzako lekuren bat dagoen edo ez egiaztatuko du Arduino Megak.
- Arduino Megak Arduino UNO-ri Bluetooth bitartez pakete bat sartzeko agindua bidali.
- Arduino UNO-tik dena ondo joan denaren mezua jaso, eta honekin, sartu berri den paketearen id-a bidaliko da. Arduino Megak paketearen id bidali web aplikazioari.
- Arduino Megak gordeko den paketeari zein posizio dagokion erabaki eta sisteman erregistratuko da.
- Beso robotikoak plataformatik paketea hartuko du, eta aurretik zehaztu den posizioan utziko da.
- Dena bukatzean, bukatu egin den mezua bidaliko zaio web aplikazioari.

Kontuan izan behar da prozesu guzti hau betetzen den bitartean hainbat mezu bidaliak izango direla web aplikaziora, honek prozesua jarraitu ahal izateko.

4.4.2.1.2 Paketeak atera

- Lehenik eta behin, web aplikaziotik agindua jasoko da, eta atera nahi den paketearen id. Hauekin paketea existitzen dela eta honen kokapena egiaztatuko da.
- Ondoren, beso robotikoa biltegiratuta dagoen eremura joango da, paketea hartu eta plataforman utziko du.
- Hau eginda, Arduino Megak Bluetooth bidez plataforma irekitzeko aginduko dio Arduino UNO-ri.
- Arduino UNO-k bere betebeharra bukatzean, Arduino Megak “eginda” mezu bat jasoko du.

- Arduino Megak, web aplikazioari jakinaraziko dio dena ondo joan dela wifi socketaren bitartez

4.4.2.2 Arduino UNO

Arduino Megari web aplikazioarekin gertatzen zaion moduan, Arduino UNO-ri gertatuko zaio Arduino Megarekin, hau da, itxoiten egongo da Arduino Megarengandik agindu bat heldu arte. Hau ere, Arduino Megaren antzera, dena ondo funtzionatzen duenaren arduraduna izango da sistemaren plataformaren atalean. Beraz, lehen esan dugun moduan, lehenik eta behin plataformaren integritatea mantentzea izango da bere funtzioa eta gerora Arduino Megarengandik datozen aginduak betetzea.

Arduino Megarekin gertatzen den moduan, honek ere bi funtzio nagusientzako lana egingo du, paketeak ateratzea eta sartzea. Hauek aurrera eramateko, zeregin sinple batzuk izango ditu: plataforma ireki eta ixteko serbomotorra alde batera edo bestera akzionatu, itxiera eta irekiera hau kontrolatzeko distantzia-sentsoreen irakurketak interpretatu, paketeak identifikatzeko beraien RFID etiketak irakurtzeko beharrezkoa den RFID modulua kudeatu eta, azkenik, Arduino Megarekin komunikazioa mantendu HC-05 moduluaren bitartez.

Osagai hauek guztiek posible egingo dute jarraian aztertuko ditugun bi funtzio nagusiak betetzea. Kontuan izan behar da plataforma mugikorrak paketeak sartzean edo ateratzean erabiltzaileak paketeak sartu/hartzeko 10 segundu izango dituela, bestela errore bezala tratatuko da.

4.4.2.2.1 Paketea sartzea

- Arduino Megatik pakete bat sartzeko agindua heltzen denean, hardware guztien egiaztapena egin.
- Plataforma ireki (serbomotorrak eta distantzia-sentsoreak erabiliz).
- Plataforma irekita mantendu erabiltzaileak paketea jartzen duen arte. Hau RFID moduluaren bitartez egingo da, plataforman pakete bat uzterakoan RFID etiketa irakur ahal izango duelako.
- Erabiltzaileak paketea utzi duela egiaztatu ondoren, plataforma itxi egin behar da.
- Bukatzeko, plataforma itxi denean, pakete berriaren id-a bidaliko zaio Arduino Megari Bluetooth bitartez.

4.4.2.2.2 Paketea ateratzea

- Arduino Megatik pakete bat ateratzeko agindua heltzen denean, hardware osoaren egiaztapena egin. Ateratzeko aginduarekin batera atera nahi den paketeren id-a bidaliko da.
- Egiaztatu atera nahi den paketearen id-a eta beso robotikoak plataforman utzi duenaren id-a berdinak direla RFID moduluaren bitartez.
- Behin egiaztatuta, plataforma ireki egingo da.
- Irekita utzi erabiltzaileak paketea hartzen duen arte. Hau RFID moduluaren bitartez jakingo dugu, izan ere, paketeak duen etiketa irakurtzeari uzten dionean Arduinoak jakingo du paketea jada bertan ez dagoela.
- Erabiltzaileak paketea hartu duenean, plataforma itxiko da.
- Bluetooth bitartez Arduino Megari bukatu dela jakinarazi mezu batekin.

4.5 Erabilpen kasuak

Gure sistemaren funtzionalitateak analizatu ondoren, honako hauek dira identifikatu ditugun erabiltzaile motak eta bakoitzak izango dituen erabilpen kasuak:

Erabiltzaile arrunta (4.1 irudia):

- Paketeak sartu: Berarentzako pakete bat gorde dezake, edota sistemaren beste erabiltzaile batentzako pakete bat utzi.
- Paketeak atera: Erabiltzaileak bere izenean; berak utzi dituelako edota beste norbaitek utzi dizkiolako, dituen paketeak ateratzeko ahalmena izango du.

Banatzailea (4.2 irudia):

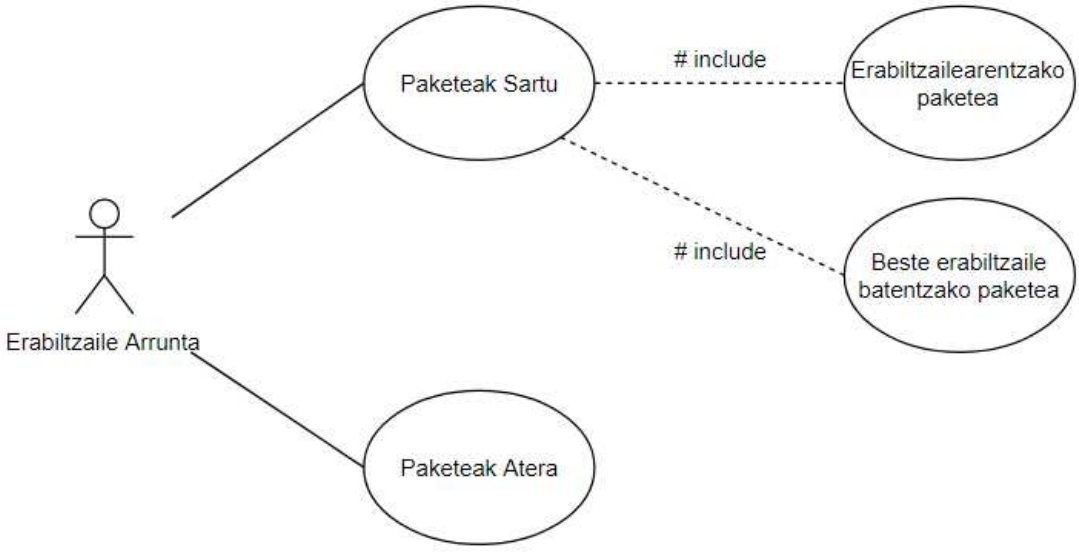
- Paketeak sartu: Banatzaileak paketeak sartu ahal izango ditu soilik, sistemaren erabiltzaile baten izenean. Erabiltzaile mota honekin gaur egungo banatzaile enpresen jarrera simulatu nahi izan da.

Administratzailea (4.3 irudia):

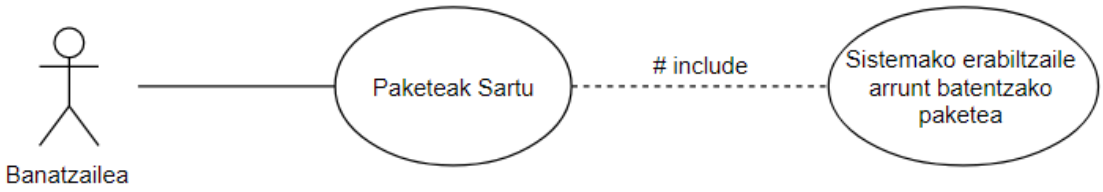
- Paketeak atera: Honek sisteman biltegiratuta dauden pakete guztiak atera ahal izango ditu.
- Erabiltzaileen administrazioa: Erabiltzaile zerrenda izango du eta behar izango balitz erabiltzaileren bat ezaba lezake.

Erregistratu gabeko erabiltzailea (4.4 irudia):

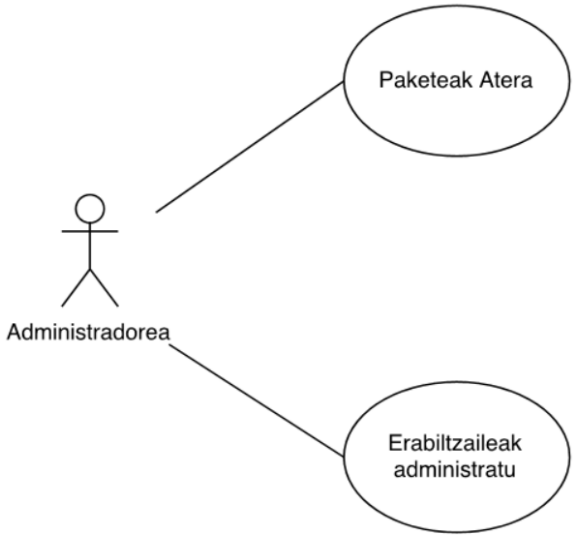
- Erregistratu: Sisteman erregistratuta ez dagoen norbait, erregistratu daiteke. Erregistratzeko bi modu daude, erabiltzaile arrunt bezala erregistratzea, edota banatzaile moduan. Banatzailearen kasuan egiaztatze prozesu batetik pasa beharko lirateke erregistroa egin baino lehen.
- Sartu: Sisteman erregistratuta dagoen edonork, email-a eta pasahitza jarriz lehen ikusi ditugun erabiltzaile motaren bat izatera pasatuko litzateke. Hemen ere erabiltzaile batek pasahitza ahaztu badu, hau aldatu ahal izango du segurtasun galdera bati erantzuten.



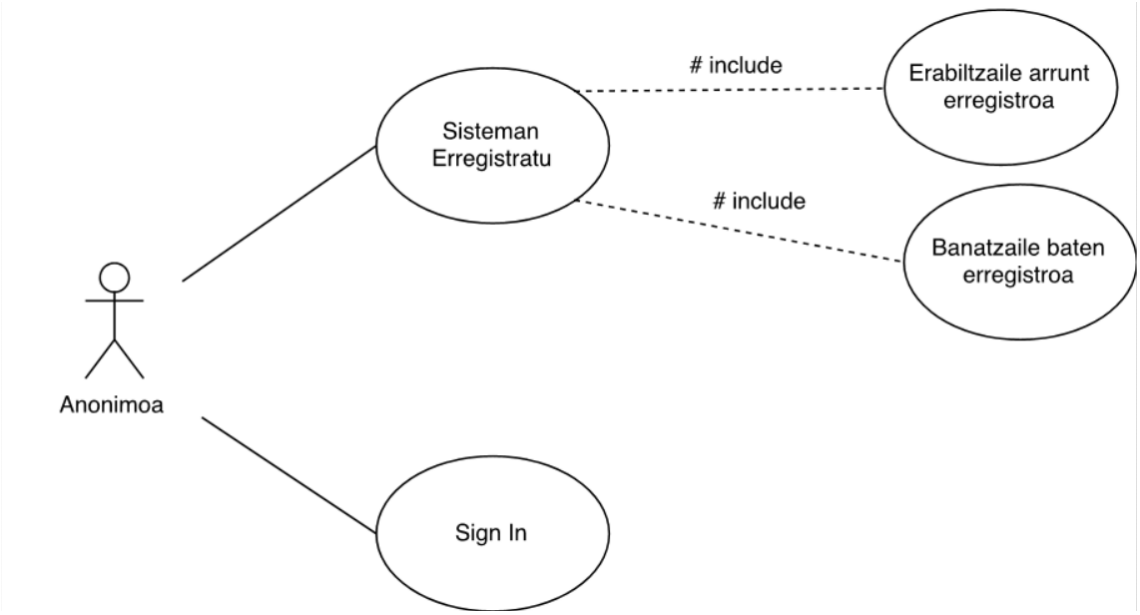
4.1 Irudia: Erabiltzaile arrunt-en erabilpen-kasuen diagrama



4.2 Irudia: Banatzaileen erabilpen-kasuen diagrama



4.3 Irudia: Administradoreen erabilpen-kasuen diagrama



4.4 Irudia: Erabiltzaile anonimoen erabilpen-kasuen diagrama

5. Hardwarearen diseinua eta garapena

5.1 Hardwarearen diseinua

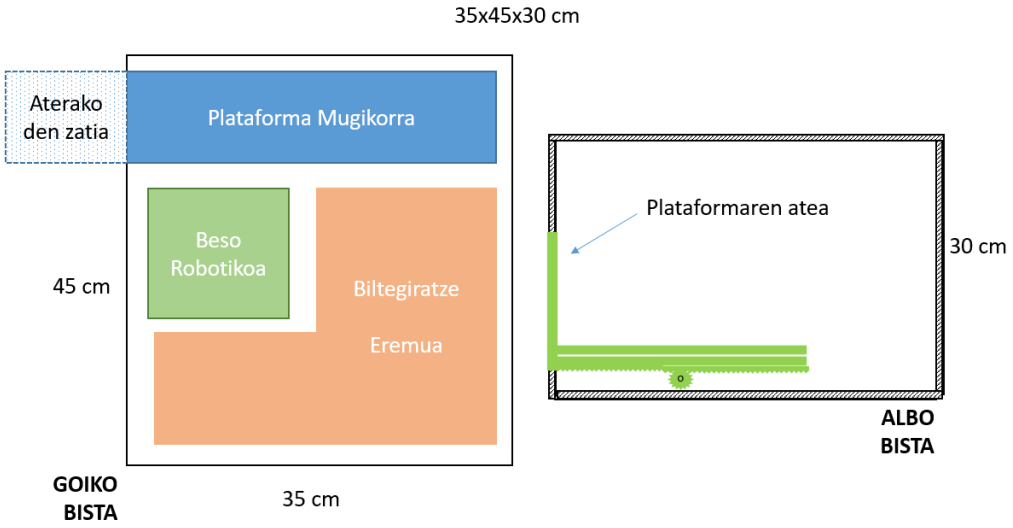
4. kapituluan ikusi dugun moduan, web aplikazioaz aparte, biltegiratze-sistema daukagu. Hau da, gure hardwarea. Kasu honetan, hardwarea gure biltegia izango litzateke, fisikoki paketeak gordetzen diren espazioa. Sistemak dituen funtzioak aurrera eramateko, 4. kapituluan ikusi dugun moduan, bi Arduino eta hainbat sentsore eta eragingailu erabili dira. Kapitulu honetan, biltegiaren diseinua eta gure hardwarearen diseinua ikusiko ditugu. Hasteko, biltegiaren egitura fisikoa aztertuko dugu, geroago gure hardwarearen muina diren bai Arduino Mega eta Arduino UNO izango dituzten konexio eskemak aztertuz.

5.1.1 Biltegia

Biltegiaren diseinuarekin hasi baino lehen, argitu beharra dago ez ditugula proiektu mekaniko bat diseinatzeko behar diren gaitasunak, eta beraz, egin den diseinua ditugun konpetentziekin hoberen egin daitekeena izango da. Egin diren diseinuak oso sinpleak izan dira, garatzerako momentuan gida orokor moduan erabilgarria suertatu daitezen.

Biltegia 35x45x30 cm izango ditu. Oinarria eta kutxaren paretak egurrezkoak izango dira. Biltegiak ez du sabairik izango, barruko elementuak eskuragarri egoteko. Biltegiaren espazioa hiru zonaldetan banatuko dugu, 5.1 irudian ikusi daiteken moduan:

- Biltegiratze-eremua
- Beso robotikoa
- Plataforma mugikorra



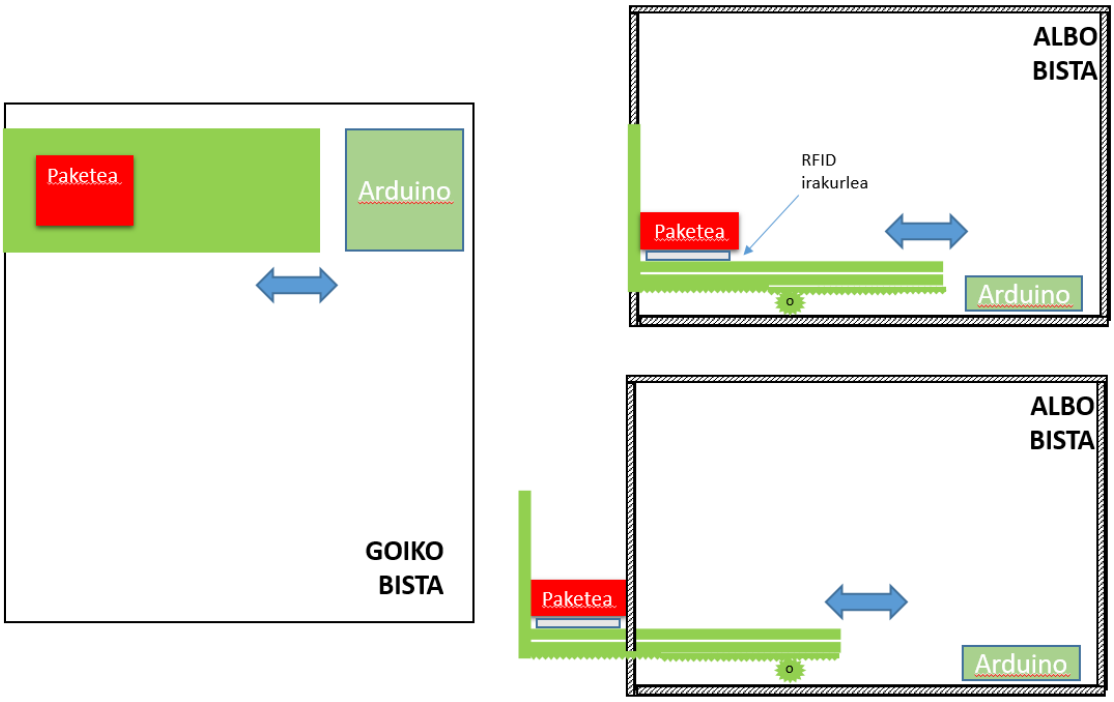
5.1. Irudia: Biltegiaren egitura

5.1.1.1 Paketeak biltegira sartzeko/ateratzeko plataforma mugikorra

Plataforma honen funtzionalitatea, paketeak kutxatik atera eta kutxara sartzea izango da. Ondorioz, plataforma mugikorrak bi helburu izango ditu:

- Erabiltzaileak objektu bat barrura sartu nahi duenean, plataforma atera egingo da. Erabiltzaileak paketea bertan utzi duenean plataforma barrura sartuz.
- Erabiltzaileak barruko pakete bat atera nahi duenean beso robotikoa paketea gordeta dagoen eremutik hartu eta plataformaren gainean utziko du. Orduan, plataforma ireki egingo da, erabiltzaileak paketea hartuz. Behin erabiltzaileak paketea hartu duela, plataforma itxi egingo da.

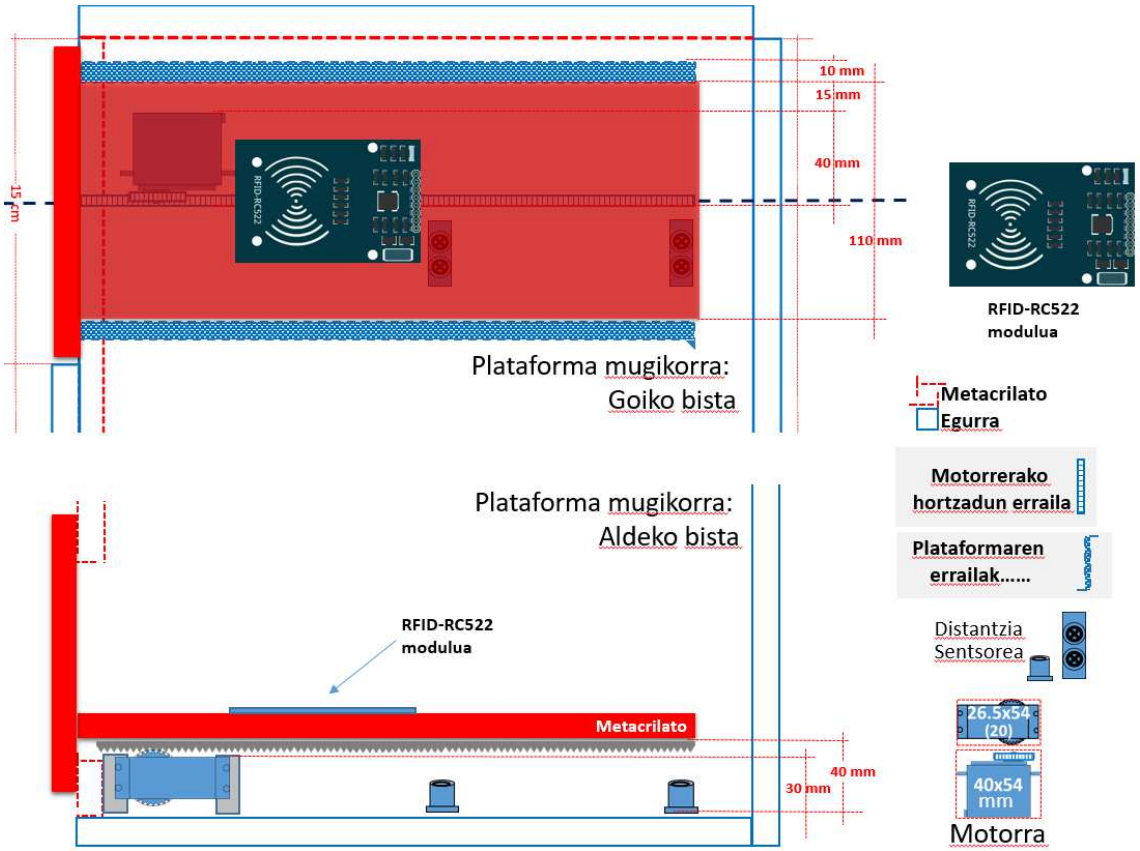
Honetarako diseinatu dugun plataforma 5.2 irudian ikusi daiteke.



5.2. Irudia: Plataforma mugikorraren diseinu orokorra

Plataformak metakrilatozko egitura izango du. Plataforma mugiarazteko errail batzuk eta serbomotor bat erabiliko ditugu. Beheko aldean, irudian ikusten den moduan, errail hortzadun bat jarriko da. Hauek serbomotorraren buruarekin bat egingo dute. Ondorioz, serbomotorra ezker edo eskuin mugitzen denean plataforma ere alde horretara mugituko da, plataforma kanpora edo barrura mugituz. Serbomotorra finko egongo da plataformaren erdian, beraz, mugituko dena ardatza izango da.

Plataforma kanpoan edo barruan dagoen kontrolatzeko distantzia-sentsore parea izango ditugu. Distantzia-sentsore bakoitza mutur batean jarriko dugu. Honela plataforma irekitzen denean, barruko distantzia-sentsorearen irakurketa aldatuko da, plataformaren ireki posiziora ailegatu dela ohartaraziz. Berdin gertatuko da plataforma ixterakoan, kasu honetan atean aurkitzen den sentsorearekin. 5.3 irudian diseinu zehatza daukagu.



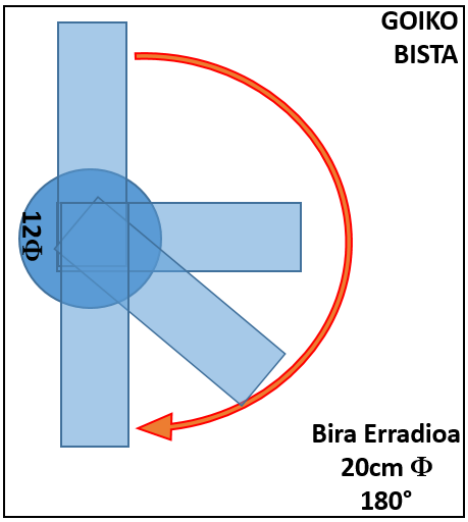
5.3. Irudia: Plataforma mugikorraren diseinu zehatza

Honetaz aparte, plataforma mugikorrak paketeak identifikatzeko modu bat izan beharko du, gerora biltegiak modu egoki batean biltegiratu ditzan paketeak. Horretarako 3. kapituluan aztertu dugun RFID-RC522 modulua erabiliko dugu. Hau 5.3 irudian ikusi daiteken moduan, plataformaren mutur batean jarriko da, paketeak horren gainean jarri ahal izateko. Pakete guztiek Mifare etiketa bat itsatsita izan beharko dute, RFID moduluak hauek irakur ditzan.

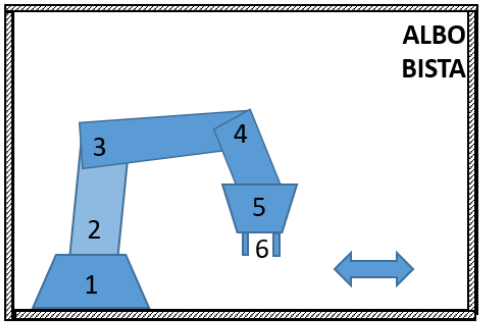
Plataforma mugikorra osatzen duten elementu guzti hauek 5.3 atalan ikusiko dugun moduan Arduino UNO baten bidez kontrolatuko dira.

5.1.1.2 Beso robotikoa

Beso robotikoak balio digu erabiltzaileak sartzen dituen paketeak plataformatik hartzeko eta dagokien biltegiratze-eremuan uzteko, eta erabiltzaileak atera nahi dituen paketeak biltegiratze-eremutik plataformara eramateko, plataformak paketeak kanporatu ditzan. Funtzionalitatea, beraz, argi dago, besoak paketeak ordenatu, biltegiratu eta sisteman zehar mugitzeko da. 5.4\5.5\5.6 irudietan beso robotikoaren ezaugarriak ikus daitezke. Berriro aipatzearren, 3. kapituluan esan dugun moduan, erabilitako beso robotikoa Lynxmotion etxeko ALD5 modeloa izan da.

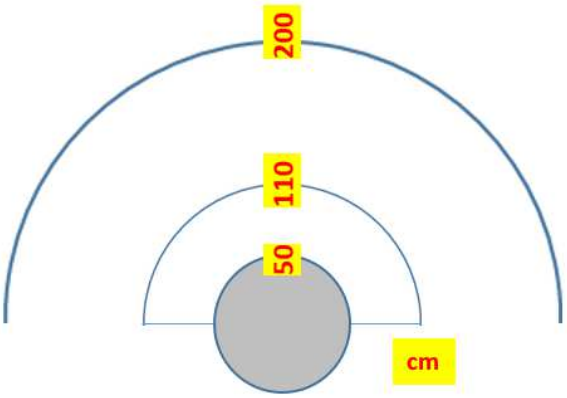


a) XY atzipen eremua



b) Besoaren askatasun graduak

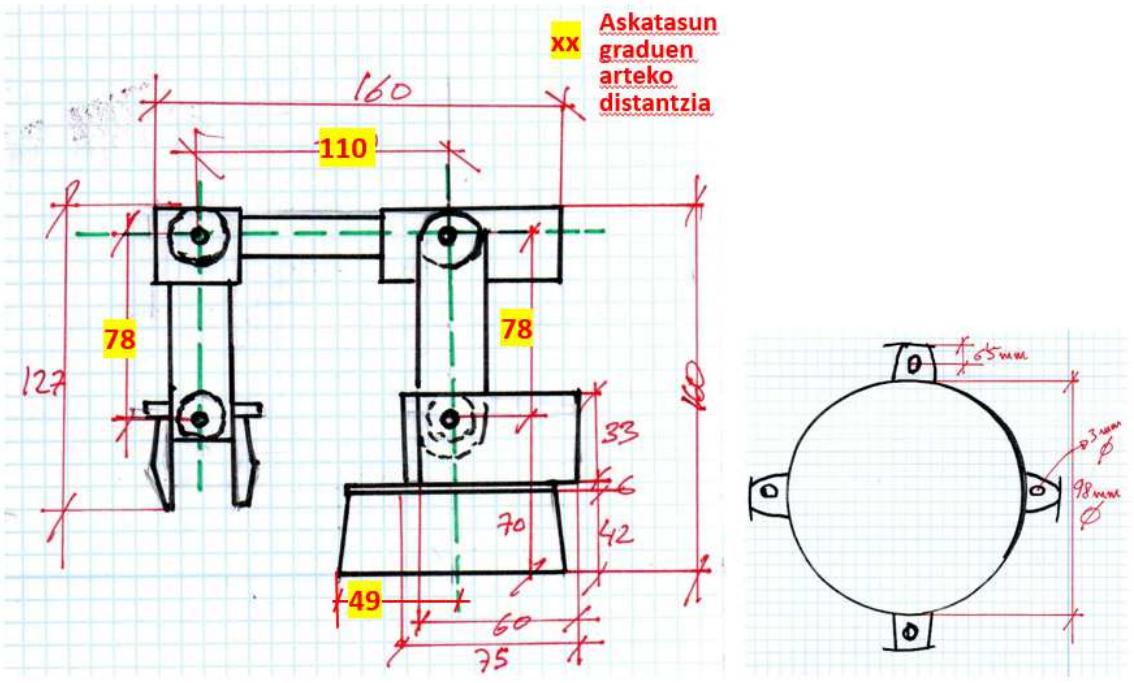
5.4. Irudia: : Beso robotikoaren egitura eta diseinu orokorrak



5.5. Irudia: Beso robotikoaren atzipen mugak

Beso robotikoak sei askatasun gradu ditu, bakoitza serbomotor batez kontrolatuta:

- Lehenengoa beso robotikoaren oinarrian aurkitzen da. Serbomotor honek besoaren ezker eskuineko mugikortasuna ahalbidetzen du. Besoak serbomotorraren muga berdina du, hau da, 180° mugikortasuna dauka.
- Bigarren eta hirugarren serbomotorrek besoaren gora eta beherako mugimenduak ahalbidetzen dituzte.
- Laugarren serbomotorrak paketeak hartzeko erabiliko dugun pintzaren goi beherako mugikortasuna kontrolatuko du.
- Bostgarren serbomotorrak besoak paketeak hartzeko duen pintzaren errotazioa zirkularra kontrolatzen du.
- Seigarren serbomotorrak pintza itxi eta irekitzeko mugimenduaz arduratzen da, paketeak hartzeko.

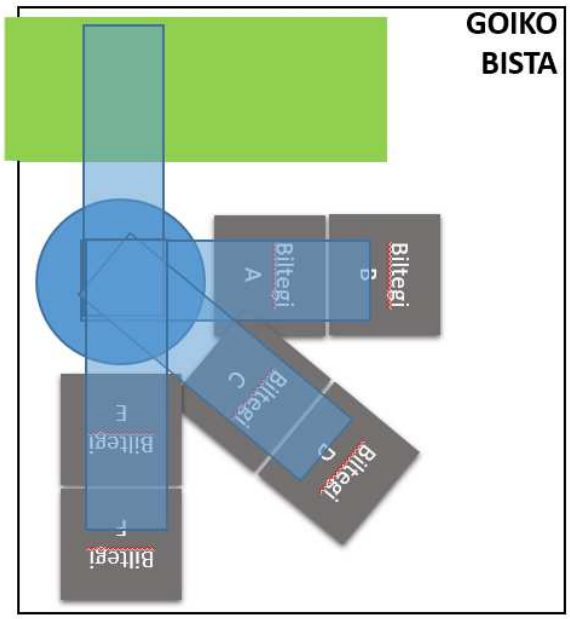


5.6. Irudia:Beso robotikoaren neurriak

Besoa kontrolatzeko Arduino Mega plaka erabili egingo da. Honen bitartez, besoaren serbomotorrak kontrolatuko dira, egin behar dituen mugimendu segidak automatizatuz. Beso robotikoa gure biltegi-sistemaren ardatza da.

5.1.1.3 Biltegiratze-eremua

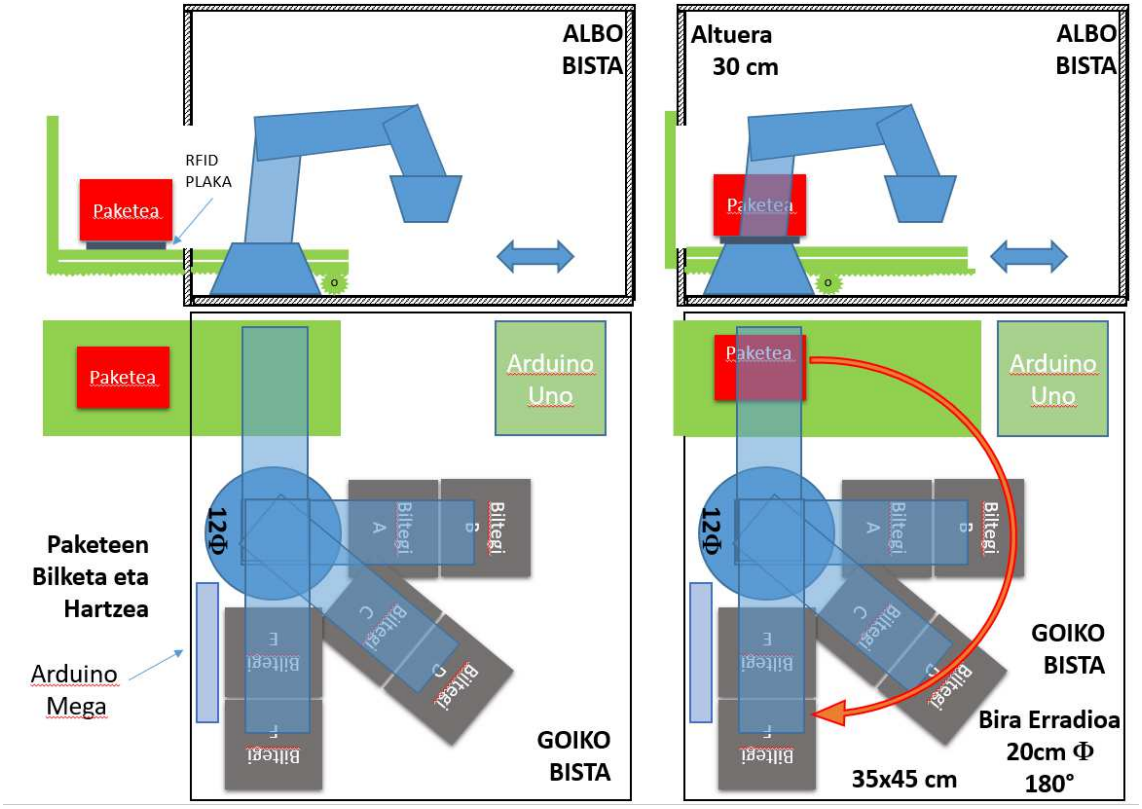
Biltegiratze-eremua biltegian geratzen zaigun espazioaren banaketa izango da. Kasu honetan, kontuan izan behar dugu beso robotikoaren atzipen eremua, baita adibide moduan erabiliko ditugun kutxen tamaina ere. Hau kontuan izanda, daukagun espazioa zatituko dugu hainbat eremutan, bakoitzean pakete baterako espazioa izanda.



5.7. Irudia: Biltegiratze eremua

Biltegiratzearen antolamendua eta funtzionamendua 5.7 irudian daukaguna izango da. Paketeak ailegatu ahala posizioak okupatuz joango dira. Adibidez, irudian lehenengo paketea A eremuan utziko da, bigarrena B eremuan, etab. Begi bistan biltegiratze honek ez du izango inongo antolamendurik; hala ere, beso robotikoak paketeak gorde ahala, software bitartez Arduino Megak hauek non gorde dituen eta baita hauen informazioa gordeko du ere. Paketeei buruz gordeko duen informazioa honako hau izango da: RFID zenbakia, hau da, geroago paketearen id-a deituko dioguna.

Ondorioz, sistema osoaren diseinua 5.8 irudian adierazita datorrena izango da.



5.8. Irudia: Sistema osoaren diseinua

5.1.2 Arduino Mega

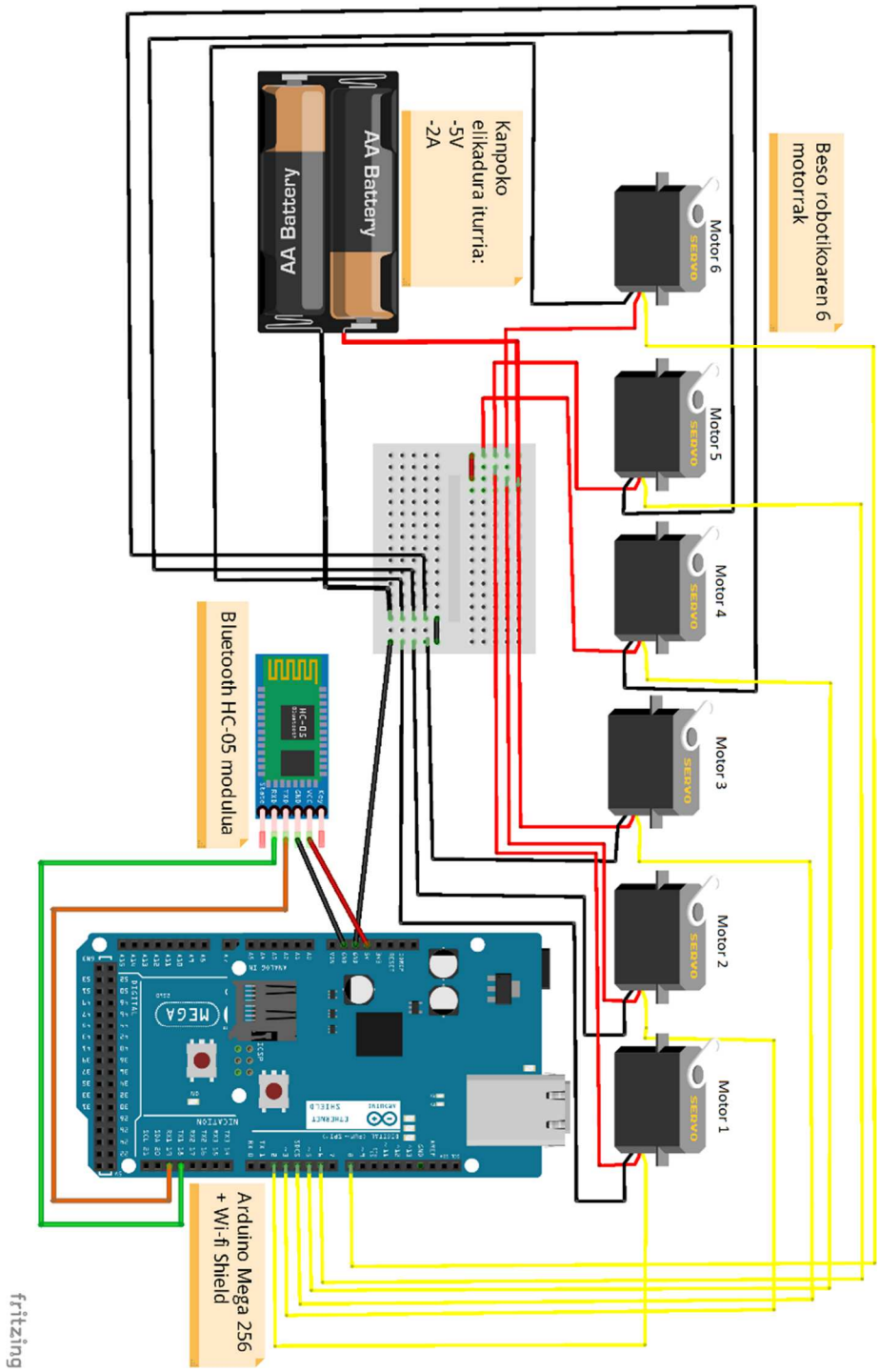
Gure biltegiaren hardware muina da. Honetara hainbat osagai konektatuko dira eta beraz, konexio hauek nola egingo diren zehaztea ezinbestekoa dugu. Arduino Mega honek web aplikazioarekin beharrezkoa den wifi komunikaziorako shield-a izango du, gainera beso robotikoa kontrolatu beharko du eta azkenik beste Arduinoarekin (plataforma mugikorrarekin) komunikaziorako beharrezkoa den HC-05 bluetooth modulua izango du.

Osagai batzuk hasieratik garbi genekien nola konektatuko genituen. Adibidez, wifi shield-a Arduinoaren gainean kokatu behar da modu jakin batean. Baina beste osagai guztiak nola konektatu behar diren gure esku geratu da, eta beraz, zein pin aukeratu ditugun adierazteko 5.1 taula daukagu.

Pin zenbakia	**Funtzioa**
2	Beso robotikoa(Motor1)
3	Beso robotikoa(Motor2)
4	Beso robotikoa(Motor3)
5	Beso robotikoa(Motor4)
6	Beso robotikoa(Motor5)
8	Beso robotikoa(Motor6)
Rx1	Bluetooth serie lerroa(Tx)
Tx1	Bluetooth serie lerroa (Rx)

5.1. Taula: Arduino Mega pin konexioak

Gure kasuan osagai guztiak zuzenean konektatuko dira Arduinoaren pin-etara, ez dugu beste zirkuitu konplexuagorik erabili behar izan. Hala ere, egindako konexioak modu grafikoan ikusi ahal izateko 5.9 irudia daukagu. Honetan, beharrezko osagai guztien konexioak aurkeztea lortu da eta gainera, garapena egiterako garaian, ideia nahiko garbi bat izateko balio izan digu. Bestalde, ikusi dezakegu ere zer nolako elikadura iturriak erabili ditugun zirkuituaren osagai ezberdinak elikatzeko. HC-05 modulua elikatzeko Arduinok daukan 5V-eko pin-a erabiliko da, aldiz, beso robotikoa elikatzeko kanpo elikadura bat erabiliko da.



5.9. Irudia: Arduino Mega konexio eskema

Bukatzeko, Arduino Mega gure produktuaren zentroa izango da, web apikazioa eta biltegiratze-sistema elkartzen baititu. Beraz, aurrera eman behar duen komunikazioaren paradigma hobetu ulertzearren, 5.10 irudia daukagu.

5.10. Irudia: Komunikazio eskema orokorra

5.1.3 Arduino UNO

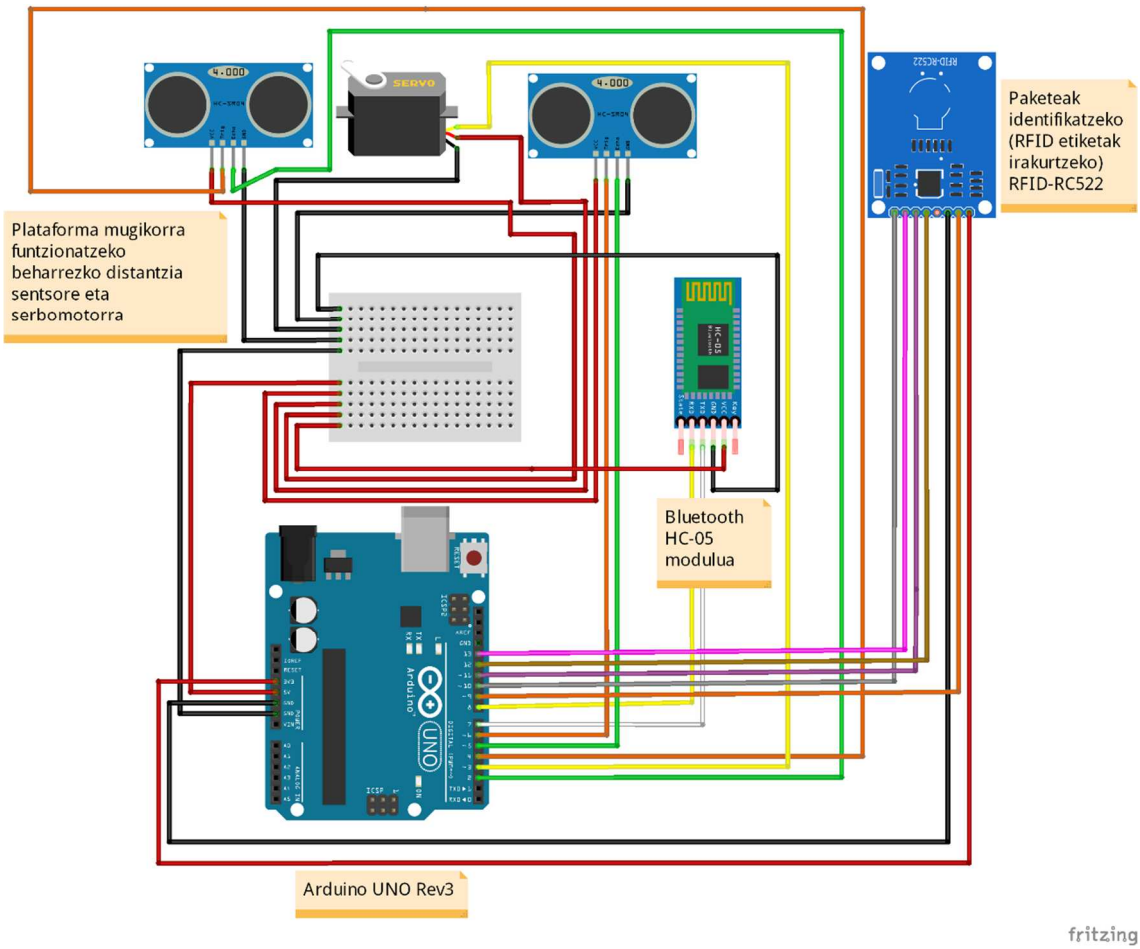
Arduino UNO plataforma mugikorra eta paketeen identifikazioaz arduratuko den hardware osagaia da. Beraz, plataforma mugikorra kontrolatuko du, non hainbat sentsore eta eragingailu dituen konektatuta. Hain zuzen ere, Arduino UNO-k serbomotor bat, bi distantzia-sentsore, RFID-RC522 irakurle bat eta HC-05 bluetooth modulu bat izango ditu konektatuta.

Kasu honetan, RFID-RC522 modulua alde batera utzita, beste osagai guztiak guk nahi ditugun pin digital ezberdinetan konektatu ditzazkegu. RFID moduluaren kasuan SPI busera konektatu behar dugunez, horretarako Arduino UNO-k definituta dituen pin-ak erabili beharko ditugu. 5.2 taulan ikusi dezakegu egindako konexio guztien eskema. Arduino UNO-k dituen S/I pin digital guztiak erabiltzen ditugu.

Pin zenbakia	Funtzioa
2	Distantzia-sentsore 1 echo pin
3	PWM pin Serbomotorraren kontrola
4	Distantzia-sentsore 1 trig pin
5	Distantzia-sentsore 2 echo pin
6	Distantzia-sentsore 2 trig pin
7	Bluetooth serie lerroa (Tx)
8	Bluetooth serie lerroa (Rx)
9	RST (RFID readear)
10	SDA (RFID readear)
11	MOSI (RFID readear)
12	MISO (RFID readear)
13	SCK (RFID readear)

5.2. Taula: Arduino UNO pin konexioak

Beraz, behin konexioak zeintzuk diren aztertu eta gero, hauek modu grafiko batean ikustera pasatuko gara 5.11 irudian. Honetan ere, elikadura konexioak agertuko dira, zirkuituaren parte diren osagai ezberdinek behar duten elikadura nabarmenduz. Eskema hau behin betikoa da, eta beraz, inplementazio garaian erreferente moduan erabiliko dugu.



5.11. Irudia: Arduino UNO konexio eskema

5.11 irudian ikusten dugun moduan bai serbomotorra eta baita bi distantzia-sentsoreak ere Arduino UNO-k ematen digun 5V pinarekin elikatuko dira. Berriz, RFID-RC522 modulua 3,3 V-ekin funtzionatuko du.

5.2 Hardwarearen garapena

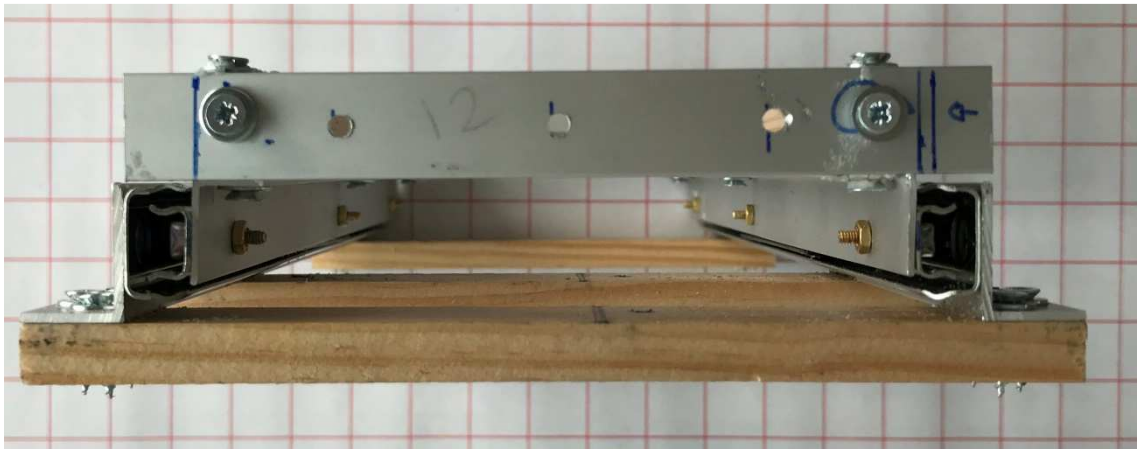
Bi Arduinoen softwarea garatu ahala hardware berria gehitzen joan gara. Jarraitu den prozesua nahiko erraza izan da, aurretikan egindako diseinua egokia zelako. Garapena osagaiz osagai egin da, osagaiak modu isolatuan probatu eta geroago proba integratuak eginez. Dena den, Arduino eta moduluak kontrolatzeko inplementatutako softwarea hurrengo kapituluan landuko dugu.

Hardwarearen garapenari buruz hitz egiten badugu 5.1.1 atalan aztertu dugun biltegiaren eraikuntzaz hitz egin beharko dugu. Biltegia eraikitzean, bi helburu izan ditugu beti. Biltegia ahalik eta sinpleen egin eta bete behar zituen funtzioak betetzen zituela egiaztatu. Beraz, ideia horrekin diseinuak egin genituen, eraikitzerako momentuan ezaugarri guztiak ezartzeko lekua soberan izanda. Gainera, garapena errazteko erabili diren materialak metakrilatoa eta egurra izan dira, hauen maneiua erraza baita.

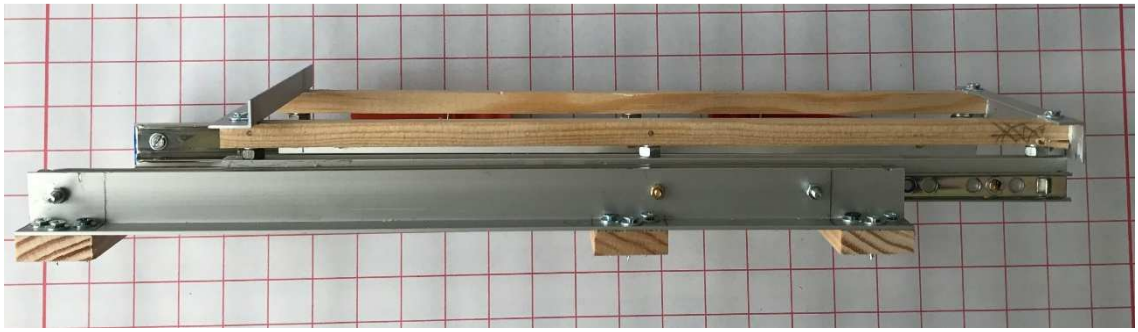
Hasteko biltegiaren egitura eraiki dugu lau egur xaflekin, biltegiaren bizkarezurra sortuz. Bestalde, behin egitura geneukala honen oinarrian silikonazko hankatxo moduko osagai batzuk jarri ditugu, egitura lurrean uzterakoan egonkorragoa izan dadin. Oinarrian beso robotikoa finkatu dugu. Honekin biltegiaren egitura orokorra lortu dugu.

Plataforma mugikorrari dagokionez, nahiz eta egindako diseinua erabilgarria suertatu, esan beharra dago hau egiterako momentuan arazo batzuk izan ditugula. Hau osatzeko, lehenik eta

behin plataformaren oinarriak egituratu dira. Horretarako errail batzuk prestatu egin dira beheko eta goiko atalen artean, plataforma ireki eta itxi ahal izateko.



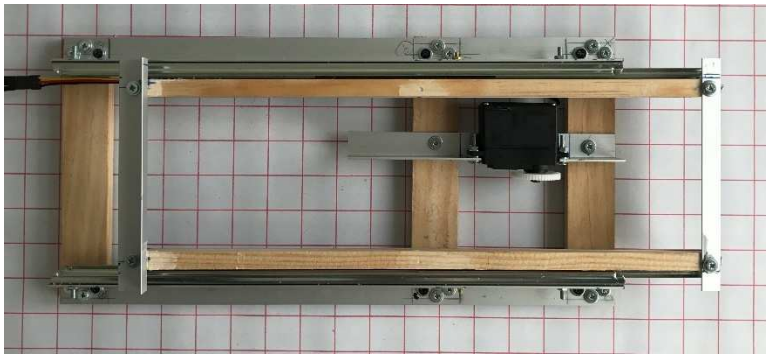
5.12. Irudia: Plataforma mugikorraren egitura albo batetik ikusita



5.13. Irudia: Plataforma mugikorraren egitura orokorra

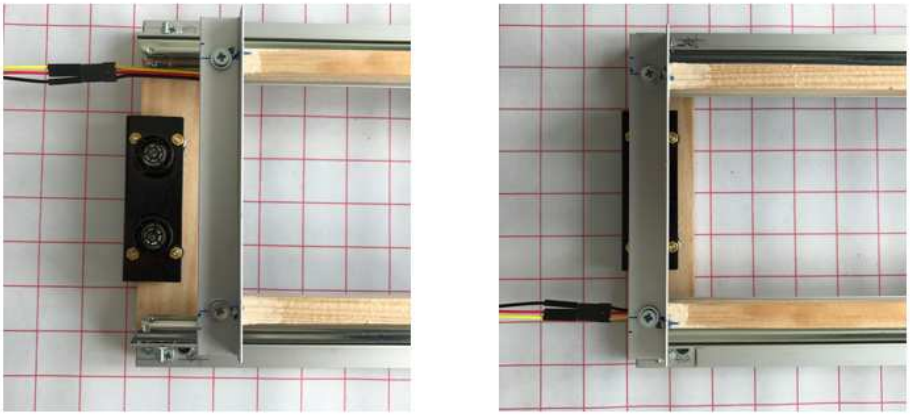
5.12 eta 5.13 irudietan ikusten den moduan behekaldean hiru egur bloke jarri dira oinarri bezala. Hauek izango dira biltegiaren oinarriarekin finkatuko direnak. Bestetik, plataforma irekitzeko eta ixteko egindako mekanismoa ere ikusi daiteke. Erabili den mekanismoa normalean etxeko altzariek erabiltzen duten mekanismoaren antzekoa da. Horrela plataforma mugikorraren goiko atala izango da mugituko dena, oinarria geldirik mantentzen delarik.

Behin plataformaren oinarria eta mugimenduak garatuta dauzkagula, mugimendua eragingo duten serbomotorra eta errailak jarri ditugu, 5.14 irudian ikusi daitekeen moduan.



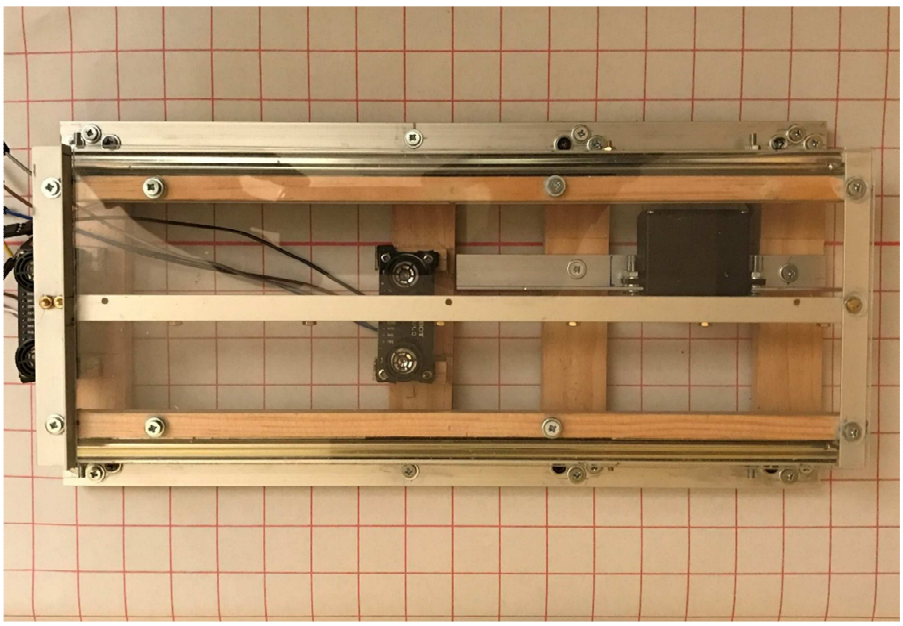
5.14. Irudia: Plataforma mugikorraren motorra

Honekin batera plataformak izango dituen mugimendu mugetan bi distantzia-sentsoreak jarriko ditugu. 5.15 irudian ikusi dezakegu distantzia-sentsore baten kokapena.



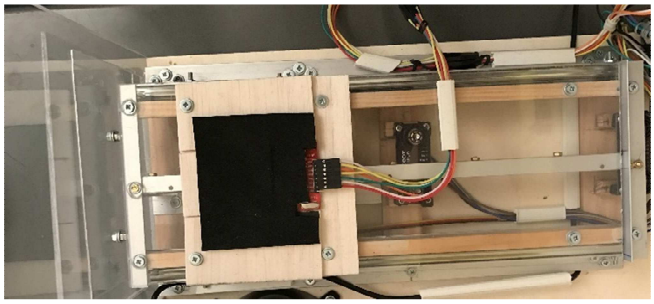
5.15. Irudia: Distantzia-sentsoreak plataforman

Plataforma osoaren egitura eta itxura 5.16 irudian daukagu. Honetan RFID irakurlea jartzea falta da oraindik. 5.16 irudian ikusi dezakegu nola metakrilatezko xafla bat jarri dugun goialdean, paketeak utzi ahal izateko. Gainera, erdialdean metalezko xafla bat ikusi dezakegu, plataforma erditik zeharkatzen duena. Hau, serbomotorrak eragingo duen erraila da.



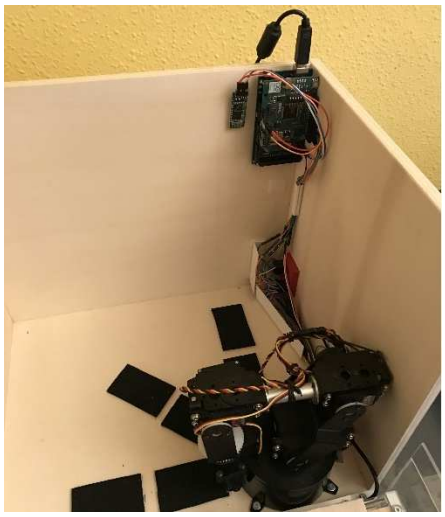
5.16. Irudia: Plataforma mugikorra guztiz osatuta

Azkenik plataforman paketeak utziko diren eremua definitu eta RFID-RC522 modulua finkatu dugu. Beso robotikoari paketeak hartzea errazteko eta RFID irakurlea babesteko material ez labaingarria jarri zaio, 5.17 irudian ikusi dezakegun moduan.



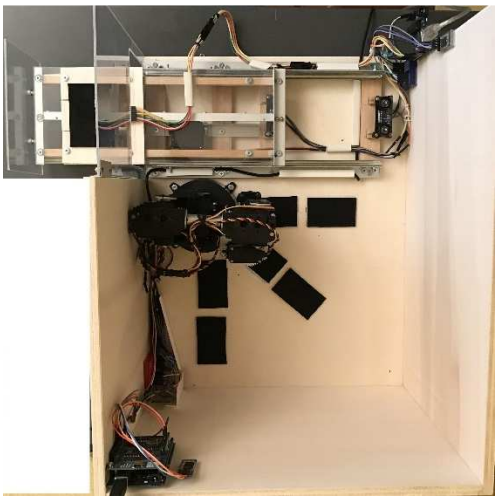
5.17. Irudia: Plataforma mugikorra RFID-RC522 moduluarekin

Azkenik Arduino UNO eta Arduino Mega biltegiaren hormetan finkatuko dira. Hormetan ezartzea erabaki da biltegiaren oinarria libre utzi ahal izateko; bai plataformaren eta baita beso robotikoaren mugimenduekin interferentziarik izan ez dezaten. 5.18 irudian ikusi daitekeen moduan, biltegiratze-eremuak zehaztu egin dira eta plataforman erabili egin den material ez labaingarria jarri zaio honetako bakoitzari.



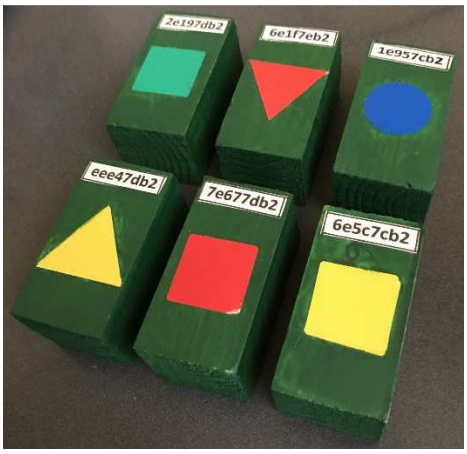
5.18. Irudia: Arduino Mega solairura itsatsita

Biltegi osoaren egitura eta itxura modu orokor batean ikusteko 5.19 irudia daukagu. Honetan biltegia osatzen duten elementu ezberdinak ikusi ditzakegu (Arduino UNO, plataforma mugikorra, beso robotikoa, biltegiratze-eremuak).



5.19. Irudia: Biltegiratze-sistema osoa

Biltegia osatuta, sisteman erabili egingo diren paketeak eraiki egin ditugu bakoitzari atzeko aldean Mifare etiketa bat itsatsiz (5.20 irudia). Bestalde, pakete bakoitzak besteengandik ezberdindu ahal izateko forma geometriko eta kolore ezberdineko pegatinak erabili dira. Halaber, pegatinekin batera pakete bakoitzari bere RFID etiketaren id-a jarri zaie, biltegiratze-sistemaren funtzionamendua egiaztatzean, erabiltzailearentzako paketeak identifikatzea errazago suertatatzeko.



5.20. Irudia: Erabilitako paketeak

Bukatzeko nabarmendu beharra dago, biltegia eraikitzerako momentuan helburu nagusia funtzionalitatea izan dela eta, ondorioz, itxura bigarren planoan geratu da.

6. Biltegiaren software inplementazioa

Kapitulu honetan, gure biltegiratze-sistemarako inplementatutako softwarea aztertuko dugu. 5. kapituluan, garatutako biltegia eta gure biltegiratze-sistema osatzen duten elementuen arteko konexioak ikusi ditugu. Orain, elementu bakoitzak dituen funtzioak betetzeko garatutako softwarea aztertuko dugu.

6.1 Plataforma mugikorra (Arduino UNO)

Plataforma mugikorraren funtzionamenduarekin hasiko gara. Hau 5. kapituluan ikusi dugun moduan Arduino UNO baten bidez kontrolatu egingo dugu. Plataforma mugikorrak paketeak gure biltegiratze-sistematik sartu/atera egiteaz gain, paketeen identifikazioa egiteko balio digu ere. Bi funtzio hauek betetzeko, garatu dugun kodea aztertuko dugu.

6.1.1 Distantzia-sentsoreak

4. kapituluan aipatu dugun moduan, Arduino UNO-k, plataforma itxita edo irekita dagoen jakiteko, distantzia-sentsoreak erabiliko ditu. Hauen kontrola errazteko, klase bat inplementatu dugu. Kasu honetan, serbomotorrekin gertatzen ez den bezala, Arduinok ez digu liburutegirik eskaintzen eta horregatik erabaki dugu guk bat inplementatzea.

Nabarmendu beharra dago, gure proiektuan erabili diren distantzia-sentsoreak URM37 v4 direla, eta beraz, baliteke beste distantzia-sentsore batzuentzako klasea erabilgarria ez suertatzea.

Distantzia-sentsoreen funtzionamendua argi izan behar dugu, klasea inplementatzen hasi baino lehen. Horregatik 3.2.1.3 atalan erabilitako ultrasoinu distantzia-sentsoreen funtzionamendua aztertu dugu. Erabilitako distantzia-sentsoreetan aztertu dugun moduan, ultrasoinu uhina igorriko duen atala daukagu (trig) eta bestetik bidalitako seinalearen oihartzunaren zain geratuko den entzulea daukagu (echo). Bidali den seinalea sentsorearen aurrean topatzen duen objektuarekin topatuko da, errebotatu, eta berriro distantzia-sentsorera bueltatuko da. Seinalea distantzia-sentsorera ailegatzerakoan, echo-k antzemango du. Honek ultrasoinuaren abiadura eta seinalea joan etorrian jardutako denbora kontuan hartuz, seinaleak topatu egin duen objektua zer distantziara dagoen kalkulatuko du.

Beraz, funtzionamendua aztertuz, argi dago `DistanceSensor` klasearen propietate pribatuak distantzia-sentsoreak dituen igorle/entzule atalak izango direla, non batak seinalea bidaliko duen (trig) eta bestea seinalea antzemateaz arduratuko den (echo). Propietate pribatuaz gain, klasean hainbat metodo publiko inplementatu dira. Alde batetik, oinarrizkoak diren eraikitzailea eta bi propietate pribatuak aldatzeko set funtzioak ditugu. Ondoren, aurrean aurkitzen den objektua zein distantziatara dagoen kalkulatzen duen `DistanceSensorCalculate()` metodo daukagu. Klasearen definizioa 6.1 kodean ageri da.

```
/* Klase hau URM37 V4.0 distantzi sentsoreak erabiltzeko inplementatuta dago*/
class DistanceSensor{
private:
        /*Pin parametroak*/
        int trigPin;
    int echoPin;

public:
        /*metodo publikoak*/
        DistanceSensor();
        DistanceSensor(int pin_trigPin, int pin_echoPin);
```

```
        void Set_trigPin(int newtrigPin);
        void Set_echoPin(int newechoPin);
        long DistanceSensorCalculate();

} ;
```

6.1. Kodea: DistanceSensor.h

Definizioa alde batera utzita jarraian 6.2 kodean metodo hauen inplementazioa daukagu.

```
/* Eraikitzailea*/
DistanceSensor::DistanceSensor(){

}

/* Konfiguragarri den eraikitzailea, honen bitartez erabiliko diren pin-ak definitu
ahal izango dira*/
DistanceSensor::DistanceSensor(int pin_trigPin,int pin_echoPin){

trigPin = pin_trigPin;
echoPin = pin_echoPin;

pinMode(trigPin, OUTPUT);
digitalWrite(trigPin,LOW);
pinMode(echoPin, INPUT);
}
/*Distantzia-sentsorearen parean dagoen objektua zenbat distantziara aurkitzen den
zehazteko metodoa.
 --return: long distance: Objektua aurkitzen den distantzia(cm).
 */
long DistanceSensor::DistanceSensorCalculate(){
        // Bidali ultrasoinua.
        digitalWrite(trigPin, LOW);
    digitalWrite(trigPin, HIGH);
        // Ultrasoinua antzematen denean, kalkulatu bidali denetik pasatako
denbora(ms)
    unsigned long LowLevelTime = pulseIn(echoPin, LOW);
        if(LowLevelTime>=45000)                 // the reading is invalid.
    {
      return 500;
    }
    else{
    return (LowLevelTime /58);   // 58us bakoitzak = 1cm
        }
}
/* Metodo hau erabili trig pin-a aldatzeko*/
void DistanceSensor::Set_trigPin(int newtrigPin){
        trigPin = newtrigPin;
        pinMode(trigPin, OUTPUT);
        digitalWrite(trigPin,LOW);
}
/* Metodo hau erabili echo-ren pin aldatzeko*/
void DistanceSensor::Set_echoPin(int newechoPin){
        echoPin = newechoPin;
        pinMode(echoPin, INPUT);
}
```

6.2. Kodea: DistanceSensor.cpp

6.2. kodean ikusi dezakegun moduan hainbat metodo definitu ditugu. Eraikitzaileak eta set metodoak alde batera utzita, klase honetatik garrantzitsuena den `DistanceSensorCalculate()` funtzioa aztertuko dugu. Honek distantzia-sentsoreak aurrean daukan objektua zein distantziara dagoen kalkulatuko baitu.

Horretarako, ikusi dezakegu nola, ultrasoinu seinalea bidali ahal izateko, trig pin-ak LOW eta HIGH tentsioetara jarri beharko dugun. Ondoren, echo pin-a seinalearen bueltaren zain geratuko da, seinalea bidali denetik, jasotzen den arte pasatzen den denbora kalkulatuz. Behin seinalea jaso ondoren, funtzioak objektua zenbat cm-etara dagoen kalkulatuko du, lortzen duen balioa bueltatuz. Funtzioak balioa cm-etan bueltatu dezan, kontuan izan behar dugu soinuak daukan abiadura. Soinuak 29us-etan 1cm aurreratzen du. Beraz, guk lortu dugun denbora, seinaleak

objektura arte eta bueltan eman duen denbora denez, zein distantziara dagoen kalkulatzeko denbora/2*29us kalkulatu beharko dugu.

6.1.2 Serbomotorren kontrola

Serbomotorraren kontrolerako inplementatu dugun klasea aztertu baino lehenago, erabili dugun serbomotor motaren funtzionamendua aztertuko dugu. 3.2.1.2 atalan ikusi dugun moduan serbomotorrek bi motako erabilera onartu egiten dute: abiadura bidezkoa, non serbomotorraren ardatzak norabide batean edo bestean errotatzeko abiadura definitu egiten den eta posizio bidezko kontrola, non serbomotorraren ardatzak definitu egiten den posizio jakin bat hartzen duen. Gure Arduino UNO plataforma mugikorraren mugimendurako abiadura bidezko serbomotor bat behar du, eta beraz, honen kontrolarako klase bat definitu dugu.

Abiadura bidezko serbomotorrek stop posizio bat daukate, normalean 90° balioaren inguruan dagoena. Stop posizio horretatik gora dauden balioak, 180° baliora arte, serbomotorra norabide batean mugitzea eragingo dute. Balioa gero eta handiagoa, orduan eta serbomotorra mugituko den abiadura handiagoa izango da. Berriz, stop posizioa baino balio txikiago bat ezartzen badugu, kontrako norabidean mugituko da. Zehazten den balioa gero eta txikiagoa bada orduan eta abiadura handiagoan mugitzen delarik. Abiadura handienak, muturretan lortuko dira, 0° balioarekin norabide batera eta beste norabidean 180° balioarekin.

Laburbilduz, gure klaseak kontrolatu beharko duena, serbomotorraren ardatzaren mugimenduaren abiadura eta noranzkoa izango dira.

Arduino IDE-ak serbomotorrak kontrolatzeko liburutegi bat eskuragarri dauka. Liburutegia, `<Servo>` da eta honetan serbomotorraren kontrolerako beharrezko diren funtzioak definituta daude. Honenbeste, gure kodean serbomotorra objektu modura inizializatu daiteke, `Attach()` funtzioaren bitartez motorra zein PWM pin-era konektatuta dagoen zehaztuz. Gainera, klaseak hainbat metodo definituta ekartzen ditu, adibidez, guk erabili egin dugun `write()` funtzioak serbomotorrari zuzenean 0-180 tarteko balioak emateko balio digu. Klase hau eskuragarri izatea oso garrantsitzua da garapenaren ikuspuntutik, hardwarearekiko lortzen den abstrakzio maila handia delako. Adibidez, bi serbomotor guztiz ezberdinak, modu berean hasieratu eta kontrolatu daitezkeelako.

Hortaz, behin gure klasearen oinarriak aztertu ditugula, guk definitutako `ContinuousMotor` klasea aztertuko dugu. Gure plataformaren serbomotorrak egin beharreko mugimenduak finkoak dira. Beraz, mugimenduak aurretikan definituta ditugunez, hauen erabilpena errazteko klase bat definitzea erabaki dugu, Arduinok eskuragarri daukan `<Servo>` liburutegia oinarri bezala hartuz. Honetarako, klasearen propietate pribatuak eta hauek manipulatzeko erabiliko ditugun metodo publikoak definituko ditugu `ContinuosMotor` klasean. Honen inplementazioa 6.3 kodean ikusi dezakegu.

```
#include "Arduino.h"
#include <Servo.h>
/* Klase hau abiadura zehaztutako serbomotorrak kontrolatzeko erabiltzen da. KOntuan
izan abiadurak eta stop balioak serbomotorraren arabera ezberdinak izango dira
Ondorioz, kalibratu egin beharko ditugu erabili baino lehen. Hau gure plataforma
mugiarazten duen serbomotorra kontrolatzeko erabiliko dugu. */

class ContinuosMotor{
private:
        /*Pin parametroak*/
        int PwnMotorR;
        /*Serbomotorraren ezaugarriak*/
        int SpeedMotorR;
        int StopMotorR;
        /**/
        Servo ServoR;

public:
        /* Serbomotorrak erabiltzeko definitutako metodo publikoak*/
```

```
        ContinuosMotor(void);
        ContinuosMotor(int pin_PwnMotorR, int v_SpeedMotorR, int v_StopMotorR);
        void SetSpeedR(int speedR);
        void SetStopR (int stopR);
        int GetSpeedR();
        int GetStopR ();
        void AttatchR();
        void SetPwnMotorR (int PwnR);
        void MoveOpen();
        void MoveClose();
        int stopM();
} ;
```

6.3. Kodea: ContinuousMotor.h fitxategia

Gure klaseak hainbat propietate pribatu izango ditu. Serbomotor objektua eta hau kontrolatzeko erabiliko den Arduino UNO-ren PWM pin-az aparte, motorraren funtzionamendurako beharrezkoak diren `Speed` eta `Stop` balioak ditugu ere. Azkenengo biek, serbomotorrak zer abiaduratan, bai aurreraka bai atzeraka, mugitu behar den eta zein baliotan serbomotorrak stop posizioan (geldirik) egongo den definituko dute.

Bestetik metodo publikoak ditugu. Hauetan eraikitzailea, serboa PWM pinekin konektatzeko erabiliko dugun `AttachR()` eta elementu pribatuak lortzeko/aldatzeko erabiliko diren set/get metodoak ditugu. Hauek 6.4 kodean dauden azalpenenekin ulergarriak direnez ez ditugu modu sakon batean aztertuko. Berriz, `MoveClose()`, `MoveOpen()` eta `stopM()` metodoak sakonago aztertuko ditugu, hauek baitira serbomotorraren mugimenduak kontrolatuko dituztenak. Horretarako 6.4 kodean metodoen inplementazioa aztertuko dugu.

```
#include “Arduino.h”
#include “ContinuosMotor.h”
#include <Servo.h>
#include “DistanceSensor.h”


#define _PwnMotorR 3
#define _SpeedMotorR 45
#define _StopMotorR 95


/* KOnfigurazio estandarra duen serbomotor objektu baten instantzia sortzeko erabiliko
dugun eraikitzailea. Hasieratuko diren balioak
 serbomotor pin,speed balioa eta  stop balioak izango dira */
ContinuosMotor::ContinuosMotor(void){

        PwnMotorR = _PwnMotorR;
        SpeedMotorR = _SpeedMotorR;
        StopMotorR = _StopMotorR;
}

/* Parametro moduan pasatako balioak objektu berri baten instatzia sortzeko
eraikitzailea */
ContinuosMotor::ContinuosMotor(int pin_PwnMotorR, int v_SpeedMotorR, int
v_StopMotorR){

        PwnMotorR = pin_PwnMotorR;
        SpeedMotorR = v_SpeedMotorR;
        StopMotorR = v_StopMotorR;
}
/* Metodo hau Arduino plakako pin-a eta serbomotorrari esleitu zaion pin-ak
konektatzeko erabiliko da*/
void ContinuosMotor::AttatchR(){
        ServoR.attach(PwnMotorR);
}
/* Serbomotorrak rotatuko duen abiadura zehazteko balio digun metodoa.
        --In: int speedR-> Abiadura balioa
        */
void ContinuosMotor::SetSpeedR(int speedR){

        int constrainedSpeedR = constrain(speedR,0,180-StopMotorR);
        SpeedMotorR = constrainedSpeedR;
}
```

```
/* Serbomotorrak gelditzen den balioa zehazteko metodoa.
        --In: int stopR-> the value in which the servomotor stops.
        */
void ContinuosMotor::SetStopR(int stopR){

        int constrainedStopR = constrain(stopR,0,180);
        StopMotorR = constrainedStopR;

}
/* Serbomotorra kontrolatzeko erabiltzen den pin-a aldatzeko metodoa.
        --In: int PwnR-> Pin.
        */
void ContinuosMotor::SetPwnMotorR (int PwnR){

        ServoR.detach();
        PwnMotorR = PwnR;
        ServoR.attach(PwnMotorR);
}
/* Metodo hau erabili serbomotorra definituta daukan abiadura lortzeko.
        --return: int PwnR-> Abiadura balioa.
        */
int ContinuosMotor::GetSpeedR(){return SpeedMotorR;}
/* Metodo hau erabili serbomotorra gelditzeko definitutako balioa lortzeko.
        --return: int PwnR-> stop balioa.
        */
int ContinuosMotor::GetStopR(){return StopMotorR;}

/* Plataforma mugikorra irekitzeko beharrezko abiadura agintzeko serbomotorrari*/
void ContinuosMotor::MoveOpen(){
        ServoR.write(StopMotorR+SpeedMotorR);
}
/*Serbomotorrak plataforma ixteko*/
void ContinuosMotor::MoveClose(){
        ServoR.write(StopMotorR-SpeedMotorR);
}
/* Serbomotorra gelditzeko metodoa
 --return : 0-> Dena ondo joan da
                        -1-> Motorrei aginduak ematean erroreak*/
int ContinuosMotor::stopM(){

        ServoR.write(StopMotorR);
        if (ServoR.read() == StopMotorR){
                Serial.println(" Motorra gelditu da");
                return 0;
        }else{
                Serial.println("Zeozer gaizki joan da motorrak gelditzean");
                return 1;
        }
}
```

6.4. Kodea: ContinuousMotor.cpp

Bai `MoveOpen()` bai `MoveClose()` egitura berdinak dauzkate. Biak serbomotorra mugiarazteko erabiliko ditugun funtzioak dira. Ikusten dugun moduan, bata `Stop` balio horri gehitzen dio `Speed` balioa eta bestea alderantziz, alde batera edo bestera abiadura berdinean mugitu ahal izateko. Bata plataforma irekitzeko balio digu, `MoveOpen()`, alegia. Bestea berriz, plataforma ixteko.

Azkenik, `stopM()` funtzioa motorra geldiarazteko funtzioa izango da. Honetan, serbomotorran stop balioa idatzi ondoren, egiaztapen bat egingo da. Horrela kontrolatu ahal izango dugu, prozesuan zeozer gaziki joan den edo ez motorrak gelditzean; hardwarearen integritatea bermatuz.

Nabarmendu arren, ikusi daiteke nola, motorrari aginduak bidaltzeko Arduino `<Servo>` liburutegiak eskuragarri duen `write()` funtzioa erabili dugun. Bestalde, `AttachR()` funtzioan, liburutegiak motorra eta Arduinoaren pin-a konektatzeko eskaintzen digun `attach()` funtzioa erabili egin da.

Argitzearren, esatea, definitu egin diren `SpeedMotor` (45) eta `StopMotor` (95) balioak ez direla estandarrak, baizik eta gure aplikaziora moldatzen diren balioak aukeratu ditugula. `Stop` balioaren kasuan, erabiltzen ari garen serbomotorraren arabera aldatu egingo da, lehen esan dugun moduan `Stop` balioa 90 inguruan egongo da, baina ez dauka balio zehatzik. Ondorioz, klase

hau erabiltzen hasi baino lehen ziurtatu behar dugu gure serbomotorraren gelditze balioa zein den.

6.1.3 RFID-RC522 (rfid-master)

Gure RFID-RC522 modulua sisteman dauden paketeak identifikatzeko erabiliko dugu. Honek pakete bakoitzak eramango duen Mifare Classics etiketak irakurriko ditu, 3.2.1.4 atalan ikusi dugun moduan. Horretarako, modulua kontrolatzeko eta etiketak irakurtzeko, jadanik definituta dagoen liburutegi bat erabiltzea erabaki dugu. Liburutegi hau GitHub-en: https://github.com/miguelbalboa/rfid eskuragarri dago. Kode honek ez dauka lizentziarik eta beraz, guk nahi dugun moduan erabiltzeko aske gara. Liburutegia erabili ahal izateko Arduinoaren SPI busa erabiltzeko beharrezkoa den <SPI> liburutegia ere behar da.

Liburutegia oso zabala da eta guztia azaltzea oso astuna izango litzateke. Gainera, proiektuan bakarrik liburutegiak funtzionamendu errazerako eskuragarri dituen hiru funtzio erabili ditugu, moduluaren funtzionamendu sakonera sartu gabe. Funtzio hauekin, moduluak egiten dituen lanetatik guztiz isolatuko gara eta ondorioz, erabilpena asko erraztu da. Jarraian hiru metodo hauek aztertzera pasatuko gara, egiten dutena gainetik azalduz. Hiru metodo hauekin batera, irakurriko ditugun etiketen id-a string batean hamaseitarrez gordetzeko inplementatu dugun funtzioa ere aztertuko dugu.

```
/**
 * Initializes the MFRC522 chip.
 */
void MFRC522::PCD_Init() {
	// Set the chipSelectPin as digital output, do not select the slave yet
	pinMode(_chipSelectPin, OUTPUT);
	digitalWrite(_chipSelectPin, HIGH);
	// Set the resetPowerDownPin as digital output, do not reset or power down.
	pinMode(_resetPowerDownPin, OUTPUT);

	if (digitalRead(_resetPowerDownPin) == LOW) {	//The MFRC522 chip is in
power down mode.
		digitalWrite(_resetPowerDownPin, HIGH);		// Exit power down
mode. This triggers a hard reset.
		// Section 8.8.2 in the datasheet says the oscillator start-up time is
the start up time of the crystal + 37,74µs. Let us be generous: 50ms.
		delay(50);
	}
	else { // Perform a soft reset
		PCD_Reset();
	}
	// When communicating with a PICC we need a timeout if something goes wrong.
	// f_timer = 13.56 MHz / (2*TPreScaler+1) where TPreScaler =
[TPrescaler_Hi:TPrescaler_Lo].
	// TPrescaler_Hi are the four low bits in TModeReg. TPrescaler_Lo is
TPrescalerReg.
	PCD_WriteRegister(TModeReg, 0x80);		// TAuto=1; timer starts
automatically at the end of the transmission in all communication modes at all speeds
	PCD_WriteRegister(TPrescalerReg, 0xA9);		// TPreScaler =
TModeReg[3..0]:TPrescalerReg, ie 0x0A9 = 169 => f_timer=40kHz, ie a timer period of
25µs.
	PCD_WriteRegister(TReloadRegH, 0x03);		// Reload timer with 0x3E8
= 1000, ie 25ms before timeout.
	PCD_WriteRegister(TReloadRegL, 0xE8);

	PCD_WriteRegister(TxASKReg, 0x40);		// Default 0x00. Force a 100 % ASK
modulation independent of the ModGsPReg register setting
	PCD_WriteRegister(ModeReg, 0x3D);		// Default 0x3F. Set the preset
value for the CRC coprocessor for the CalcCRC command to 0x6363 (ISO 14443-3 part
6.2.4)
	PCD_AntennaOn();						// Enable the
antenna driver pins TX1 and TX2 (they were disabled by the reset)
} // End PCD_Init()

/**
 * Returns true if a PICC responds to PICC_CMD_REQA.
```

```
 * Only “new” cards in state IDLE are invited. Sleeping cards in state HALT are
ignored.
 *
 * @return bool
 */
bool MFRC522::PICC_IsNewCardPresent() {
        byte bufferATQA[2];
        byte bufferSize = sizeof(bufferATQA);
        MFRC522::StatusCode result = PICC_RequestA(bufferATQA, &bufferSize);
        return (result == STATUS_OK || result == STATUS_COLLISION);
} // End PICC_IsNewCardPresent()

/**
 * Simple wrapper around PICC_Select.
 * Returns true if a UID could be read.
 * Remember to call PICC_IsNewCardPresent(), PICC_RequestA() or PICC_WakeupA() first.
 * The read UID is available in the class variable uid.
 *
 * @return bool
 */
bool MFRC522::PICC_ReadCardSerial() {
        MFRC522::StatusCode result = PICC_Select(&uid);
        return (result == STATUS_OK);
} // End

/**
 * Used to exit the PCD from its authenticated state.
 * Remember to call this function after communicating with an authenticated PICC -
otherwise no new communications can start.
 */
void MFRC522::PCD_StopCrypto1() {
        // Clear MFCrypto1On bit
        PCD_ClearRegisterBitMask(Status2Reg, 0x08); // Status2Reg[7..0] bits are:
TempSensClear I2CForceHS reserved reserved MFCrypto1On ModemState[2:0]
} // End PCD_StopCrypto1()
```

6.5. Kodea: MFRC522 liburutegiaren funtzioak

Kodean ikusi ditzakegu rfid-master liburutegitik erabili ditugun funtzio ezberdinak. Hauek aztertzera pasatuko gara:

- `PCD_Init()`: Modulua hasieratzeko erabiliko den metodoa. Barruan pin-ak hasieratzeaz gain, komunikaziorako ezaugarriak definituko dira eta azkenik etiketak irakurriko dituen antena piztuko da.
- `PICC_IsNewCardPresent()`: True bueltatuko du etiketaren bat ready egoeran aurkitzen badu. Hau da, etiketa bat hurbiltzen denean ready egoeran egongo da eta ondoren moduluak detektatu egingo du. Behin etiketa detektatzean, ready egoteari utziko dio. Ondorioz, ez da berriro detektatuko, ez badugu etiketa ready egoerara egotera behartzen. Hau lortu egingo dugu, berriro ere `PICC_IsNewCardPresent()` funtzioari deitzen, honela aurkitzen dituen etiketak ready egoerara bueltatuko ditu eta beraz, hurrengo eskaera batean hau detektatu egin dezakegu.
- `PICC_ReadCardSerial()`: Detektatu den etiketaren identifikadorea (id) irakurri egingo du, eta id-arentzako RFID klaseak daukan `uid` bufferran gordeko du. `Uid` hau byte-ez osatutako taula bat da. Erabiltzen ditugun etiketek dituzten id-ak 4 bytez osatuta daude.
- `PCD_StopCrypto1()`: Modulua etiketarekin egindako komunikazio guztiaren ondoren, komunikazioa ixteko eta berri bat finkatzeko aukera izateko erabiliko den funtzioa. Modulua eta etiketaren arteko komunikazioa enkriptatua dago. Honekin komunikazio enkriptatuta bukatutzat emango da eta komunikazio berri baterako prest geratuko gara. Gogoratzea RFID etiketekin gertatuko diren komunikazioak klasea hasieratzean definitzen den gako baten bidez enkriptatuko dira.

Azkenik, etiketa bat `PICC_ReadCardSerial()` funtzioaren bitartez irakurtzen dugunean, ikusi dugun moduan, id-a `uid` izeneko byte taula batean gorde egiten da. Gure aplikaziorako id-a

Arduinoa eskuragarri daukan `String` motako objektu batean gordetzea erabaki dugu. Gainera, byte balioak hamaseitarrez adieraziko ditugu. Beraz, gure id-a byte taula batean gordetzetik `String` batera bilakatzeko 6.6 kodean ikusi dezakegun `printHex()` funtzioa definitu dugu.

```
/* Gorde RFID reader-ak irakurritako id-a String batean modu hexadecimalean
--In: byte buffer -> Rfid reader irakurritako id gordetzen duen byte array-a
      byte buffersize -> Byte array horren luzeera
--return: String id -> Irakurri berri den paketearen id-a modu hexadecimalean*/

String printHex(byte *buffer, byte bufferSize) {
  String id;
  for (byte i = 0; i < bufferSize; i++) {
    id=id+String(buffer[i], HEX);
  }
  return id;
}
```

6.6. Kodea: printHex funtzioa

Bilakaera horretarako `PICC_ReadCardSerial()` funtzioak id-a gordetzen duen bufferraren byte guztiak banaka desplazatu beharko ditugu. Byte bakoitzaren balio bitarra balio hamaseitarrera pasatuz eta lortzen den balioa `String`-era sartuz. Lehen esan dugun moduan, paketeak identifikatzeko erabiliko ditugun etiketak, Mifare Classics 1K motatakoak dira. Hauek 4 byte-ko id-ak izango dituzte eta beraz, hamaseitarrez adierazteko 8 karaktereko `String`-a beharko dugu.

6.1.4 Plataformaren itxiera eta irekiera

5. kapituluan ikusi dugun moduan, plataforma mugikorrak modu automatizatu batean ireki eta ixteko ahalmena izan behar du. Horretarako 6.1.1 eta 6.1.2 ataletan ikusi ditugun distantzia-sentsoreak eta serbomotorra kontrolatzeko bi klaseak erabili egingo dira.

Itxiera eta irekiera prozesu bakoitzerako, funtzio bana definitu egin dira. Gainera, plataforma irekita edo itxita dagoen jakiteko erabiliko dugun funtzio lagungarri bat definitu da ere. Hauek aztertzera pasatuko gara orain, hauen inplementazioa 6.7 kodean dagoelarik.

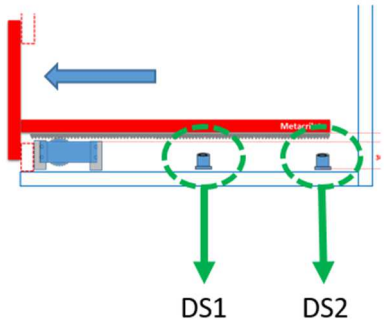
```
/* Plataforma irekita edo itxita dagoen egiaztatzeko funtzioa
--return : False-> Plataforma itxita badago
           True -> Plataforma irekira badago*/
int isOpen(){
  D1=DS1.DistanceSensorCalculate();
  D2=DS2.DistanceSensorCalculate();
  Serial.println(D1);
  Serial.println(D2);
  // Distantzia sentzoreen irakurketen arabera plataformaren egoera ezarri
  if(D2<10 && D1<10){
    Serial.println("Plataforma itxita dago");
    return 0;
  }
  else if(D1>10 && D2>10){
    Serial.println("Plataforma irekita dago");
    return 1;
  }
  else{
    Serial.println("Erdian geratu da");
    return 2;
  }
}

/*Plataforma mugikorra irekitzeko egin beharreko prozesuak
 -- return: 0-> dena ondo joan da
            1-> Itxi baino lehen egiaztatu egin da plataforma jadanik irekita dagoela
*/
int openGate(){
  Serial.println(“Plataforma irekitzen hasiko da”);
  closed=false;
  // Egaiztatu jadnaik plataforma irekita ez dagoela
  if (isOpen()==false){
```

```
    // plataforma irekitzen hasi
    motor.MoveOpen();
    while(!closed){
      // irekitzen mantendu sentzoreak irekita dagoela esan ez duten arte
      D1=DS1.DistanceSensorCalculate();
      if(D1>15){
        closed=true;
      }
    }
    // behin irekita dagoela motorrak gelditu
    motor.stopM();
    return 0;
  }else{
    Serial.println("Plataforma jadanik irekita dago");
    return 1;
  }
}

/*Plataforma mugikorra ixteko egin beharreko prozesuak
 -- return: 0-> dena ondo joan da
            1-> Itxi baino lehen egiaztatu egin da plataforma jadanik itxita dagoela
*/
int closeGate(){
   Serial.println("Plataforma ixten hasiko da");
  opened=false;
  // Egaiztatu plataforma ez dagoela jadanik itxita
  if (isOpen()== true){
    // Plataforma ixten hasi
    motor.MoveClose();
    while(!opened){
      // Etengabe begiratzen egon sensorea, honen irakurketen arabera
      // jakingo dezakegulako plataforma noiz itxi egin den
      D2=DS2.DistanceSensorCalculate();
      if(D2<15){
          if (isOpen()== true){
            continue;
          }else{
            opened=true;
          }
      }
    }
    // Behin itxita dagoela motorra gelditu
    motor.stopM();
    return 0;
  }else{
    Serial.println(" The gate is already close");
    return 1;
  }
}
```

6.7. Kodea: Plataforma ireki eta ixteko funtzioak

Beraz, 6.7. kodean agertzen diren funtzioak banaka aztertzera pasatuko gara:

- `IsOpen()`: Gure plataformak dituen bi distantzia-sentsoreek, `DS1` eta `DS2` (6.1 Irudia), egiten dituzten irakurketen arabera, plataforma itxita edo irekita dagoen jakingo dugu. Irakurketa hauek 6.1.1 atalan ikusi dugun `DistanceSensorCalculate()` funtzioaren bitartez egingo dira. `DS2` eta `DS1` sentsoreek, 10 cm baino gutxiagora objektu bat detektatzen badute, plataforma itxita dagoela adieraziko du eta, beraz, *0* balioa bueltatuko du. `DS1` eta `DS2` distantzia-sentsoreen irakurketa 10 cm baino gehiago bada, plataforma irekita dagoela esan nahi du eta *1* balioa bueltatuko du. Aldiz, aurreko bi kasuetako batean ez bagaude, plataforma ez itxita ez irekita dagoela adieraziko du, erdiko egoera batean dagoela ondorioztatuz. Orduan, funtzioak 2 balioa bueltatuko du. Kontuan izan 10 cm horien balioak aurretik parametrizatu ditugula, distantzia-sentsore bakoitzaren eta plataformaren artean dauden distantziak kontuan izanda.



6.1. Irudia: DS1 eta DS2 posizioak

- `openGate()`: Funtzio honek plataforma irekitzeko balio digu. Funtzionamendua oso erraza da. Lehenik eta behin, aurretikan azaldu dugun `isOpen()` funtzioarekin egiaztatzen dugu plataforma oraindikan itxita dagoela. Ondoren, 6.1.2 atalean serbomotorra aztertzean ikusi dugun `MoveOpen()` funtzioari deituko zaio. Honekin plataforma irekitzen hasiko da. Plataforma irekitzen jarraituko da, `DS1` sentsoren irakurketetan 10 cm baino gehiago antzematen direnean, alegia, plataforma ireki dela antzeman arte. Orduan serbomotorra geldituko da `stopM()` funtzioari deituz.
- `closeGate()`: Funtzio honetan `openGate()` funtzioan gertatzen denaren antzekoa gertatuko da. Kasu hoentan `isOpen()` funtzioarekin plataforma irekita dagoela egiaztatuko dugu. Ondoren, plataforma ixteko serbomotorren (`ContinuosuMotor`) klasean definitutako `MoveClose()` funtzioari deituko zaio. Serbomotorrak plataforma ixten egongo da `DS2` sentsoreak plataforma itxi dela antzematen duen arte, hau da, `DS2` sentsoren irakurketan 10 cm baino gutxiago detekta bitartean. Orduan, serbomotorra gelditu egingo da.

Beraz, 3 funtzio hauen laguntzarekin, plataforma itxi eta irekitzeko prozesua definitu dugu. Behin plataformaren mugimenduak definitu ditugula, mugimendu hauekin batera gertatzen den paketeen identifikazioak nola funtzionatzen duen aztertzera pasatuko gara.

6.1.5 Biltegian paketeak sartzea eta ateratzea

Orain arte azaldu dugun moduan, gure plataforma mugikorra, paketeak biltegitik ateratzeaz eta sartzeaz arduratuko da. Behin plataforma mugikorraren funtzionamendu mekanikoa, hau da, plataforma nola ireki eta ixten den aztertu dugula, gure plataformaren funtzionaltasun osoa aztertzera pasatuko gara.

Plataforma ireki eta ixteaz aparte, bere gainean uzten diren paketeen identifikazioaz arduratuko da. Ondorioz, plataformaren mugimendua eta paketeen identifikazioa elkartzen dituzten funtzioak aztertzera pasatuko gara.

Funtzio hauek egituratzeko 6.1.4 eta 6.1.3 ataletan ikusi ditugun funtzioak erabili ditugu.

```
/* timer bat, itxoin dezan sartzen diren milisegunduak
--In : long interval -> Itxoin behar diren milisegunduak
*/
void timer(long interval){
  unsigned long currentMillis,previousMillis;
  currentMillis = millis();
  previousMillis = millis();
  while (currentMillis-previousMillis<interval){
      currentMillis=millis();
  }
}
```

6.8. Kodea: timer() funtzioa

- `timer()`: 6.8. kodean definituta ikusi dezakegu `timer()` funtzioaren kodea. Parametro bezala ematen zaion milisegundo kopurua itxarongo du. Horretarako Arduino-k eskuragarri daukan `millis()` funtzioa erabili dugu. Honek Arduino piztu denetik pasatu diren milisegundoak bueltatuko dizkigu. Beraz, etengabe hasieran hartu ditugun milisegundoak eta oraingoen arteko kenketa egiten badugu, igarotako denbora kalkulatu ahal izango dugu. Behin, parametro moduan pasatako milisegundoak pasa direla antzematen denean funtziotik aterako gara.

```
/* Pakete bat sartzeko beharrezkoa den
 prozesu aurrera eramaten du.
 -- In : String Id -> Sartzen den paketearen id gordetzeko*/
int PlataformInsert(String &id)  1
  {
    finish=false;
    Serial.println(“Pakete bat insertatxera goaz”);
    // Ireki plataforma erabiltzaileak pakete bat utzi ahal izateko
 2  openGate();
    long current;
    long previous;
    long dif=0;
    while (!finish){
       previous=millis();
       current=millis();
 3     while(dif<10000){
        if(rfid.PICC_IsNewCardPresent()){
          break;
        }
        dif=current-previous;
        current=millis();
        }
      // Ixoin pakete bat usten den arte
 4    if (dif < 10000){
        id="";
        rfid.PICC_ReadCardSerial();
        // Paketea usterakoan id gorde
        id=printHex(rfid.uid.uidByte,rfid.uid.size);
        Serial.println(id);
        rfid.PCD_StopCrypto1();
        Serial.println(“Paketea utzi da plataforman”);
        rfid.PICC_IsNewCardPresent();
        // Itxoin 3 segundu paketea kendu ez dela egiaztatzeko
 5      timer(3000);
        Serial.println(“Hiru segundu pasa dira”);
        if (rfid.PICC_IsNewCardPresent()){
          rfid.PICC_ReadCardSerial();
          // egiaztatu lehen irakurri den id-a eta orain irakurri dena berdinak direla
 6        if (id==(printHex(rfid.uid.uidByte,rfid.uid.size))){
             Serial.println(“Plataforma ixtera goaz”);
             // Plataforma itxi
             closeGate();
             rfid.PCD_StopCrypto1();
             finish=true;
          }
          else{
             Serial.println(“Lehen jarritako paketea eta oraingoak ezberdinak dira”);
             rfid.PICC_IsNewCardPresent();
             dif=0;
          }
        }else{
          Serial.println(“Paketea kendu da plataformatik, berriro jarri”);
          dif=0;
        }
 7    }else{
        Serial.println("Plataforman ez da paketerik utzi");
        closeGate();
        rfid.PCD_StopCrypto1();
        return 1;
      }
    }
  }
```

6.9. Kodea: PlataformInsert() funtzioa

6.9. kodean ikusi dezakegun `PlataformInsert()` funtzioa biltegian pakete bat sartzeko jarraitu beharreko prozesua zehazten da.

1. Funtzioari gure Arduino UNO-k paketeen id-arentzako globalki definituta daukan `String`-a pasatuko zaio parametro bezala. Honetan sartu berri den paketearen id-a gorde egingo da. Kontuan izan funtzioak osoko balio bat bueltatuko duela, bukatu daitekeen egoerak ezberdinak direlako. Bi egoera definitu dira. Prozesua ondo badoa funtzioak 0 balioa bueltatuko du, berriz, erabiltzaileak paketerik sartzen ez badu funtzioak 1 balioa bueltatuko du.
2. Hasteko, plataforma ireki egingo da, lehen ikusi dugun `openGate()` funtzioa erabiliz (6.7 kodea).
3. Behin, plataforma irekita dagoela, sistemak erabiltzaileak paketea plataformaren gainean jartzearen zain geratuko da begizta baten bitartez. Gehienez sistemak erabiltzailearen zain 10 segundo igaroko ditu. Sistemak etengabe pakete bat dagoen edo ez egiaztatzen egongo da `PICC_IsNewCardPresent()`funtzioaren bitartez.
4. 10 segundu horien barnean pakete bat antzematen denean, 6.1.3 atalan aztertu dugun `PICC_IsNewCardPresent()` funtzioaren bitartez, paketearen id irakurriko da, `PICC_ReadCardSerial()` funtzioa erabiliz. Paketearen id irakurrita, id-a gordeko da `id` aldagaian, `printHex()` funtzioaren bitartez.
5. Paketea lehenengo aldiz antzeman ondoren, `timer()` funtzioa erabiliz, 3 segundo itxarongo du sistemak. 3 segundo horiek pasata, egiaztatuko da aurretikan detektatutako paketea oraindikan plataformaren gainean dagoela. Segurtasun neurri hau, erabiltzaileak paketez konfunditu egin bada, edota paketea plataformatik kentzen badu eragin ditzakeen erroreak ekiditeko hartu da. Honela paketea kendu edo beste pakete bat utzi bada, paketerik identifikatu ez balu bezala egingo du eta berriro ere pakete berri baten zain geratuko da, prozesu guzti hau errepikatuz eta 10 segunduko denboragailua 0-ra jarriz.
6. Erabiltzaileak utzitako paketea, 3 segundoz plataforman mantendu bada, plataforma ixtera pasatuko gara `closeGate()` funtzioa erabiliz. Plataforma behin itxita dagoela, `PlataformInsert()` funtzioa ondo amaituko da, 0 balioa bueltatuz.
7. Aldiz, pakete bat uzteko 10 segunduak igaro badira erabiltzaileak paketerik utzi gabe, plataforma itxiko da `closeGate()` funtzioa erabiliz eta errore moduan bukatuko da funtzioa, 1 balioa bueltatuz.

`PlataformInsert()` funtzioarekin gure ArduinoUNO-k dituen bi helburuak betetzen dira. Bata, erabiltzaileak sartu nahi duen paketea fisikoki biltegiaren barruan sartzea; eta bigarren, sartu den paketea identifikatzea, identifikadorea `id` aldagaian gordez.

```
/* Behin paketea plataforma mugikorrean dagoela, pakete bat ateratzeko beharrezkoa den
 prozesu guztia aurrera eramaten du. Erabiltzaileak ez badu paketea hartzen berriro
ere barrura
 sartuko da denbora bat pasa eta gero
 -- In : String Id -> Atera nahi den paketearen id
 -- return : 0 dena ondo joan da
             1 Atera nahi den paketea eta plataforman dagoen paketea ez dira berdinak
             2 Erabiltzaileak ez du paketea hartu
*/
int PlataformExtract(String &id){  1,2
    String aux;
    finish=false;
    while (!finish){
      // Irakurri plataforman utzi den paketearen rfid tag
 3    if (rfid.PICC_IsNewCardPresent()){
        rfid.PICC_ReadCardSerial();
        aux=printHex(rfid.uid.uidByte,rfid.uid.size);
```

```
        // Plataforma gainean dagoen paketearen id eta atera nahi denaren id berdina
ez badira
 4      if(id!=aux){
          return 1;
          Serial.println("Jasotako id eta plataforman dagoen paketea ez dira
berdinak");
        }
        rfid.PCD_StopCrypto1();
        finish=true;
        Serial.println("Paketea plataforman utzi da");
      }
    }
    rfid.PICC_IsNewCardPresent();
    timer(1000);
    // Ireki plataforma paketea kanpora ateratzeko
 5  openGate();
    long current=millis();
    long previous=millis();
    long dif=0;
    // 10 segundu pasa edota paketea kentzen bada plataformatik itxaron
    // Hau rfid tag irakurri ahal den edo ez funtzioarekin egiaztatuko da.
 6  while(rfid.PICC_IsNewCardPresent()&& dif<10000){
      rfid.PICC_IsNewCardPresent();
      dif=current-previous;
      current=millis();
    }
    // denbora ez bada pasa paketea kendu da aurretik
 7  if (dif<10000){
      Serial.println("Erabiltzaileak paketea hartu du");
      Serial.println("Plataforma ixtera goaz");
      // Plataforma itxi
      closeGate();
      return 0;
    }else{// denbora pasa da eta paketea ez da kendu
      Serial.println("Paketea ez da hartu");
      Serial.println("Plataforma ixtera goaz");
      //Plataforma itxi
      closeGate();
      return 2;
    }
}
```

6.10. Kodea: PlataformExtract() funtzioa

6.10. kodean `PlataformExtract()` funtzioa daukagu. `PlataformInsert()` funtzioaren antzekoa da, baina kontrakoa da bere helburua. Pakete bat biltegitik ateratzea. Prozesua pausuz pausu aztertuko dugu.

1. Honetan ere, paketearen id-a gordeko duen `String`-a erreferentzia moduan pasatuko zaio funtzioari. Kasu honetan, helburua ez da `id` horretan paketearen identifikadorea gordetzea baizik eta atera nahi den paketearen id-a aldagai horretatik hartzea.
2. Kontuan izan funtzioak osoko balio bat bueltatuko duela. Osoko balio hau geroago ikusiko dugun bezala, plataforma hainbat egoeratan geratu daitekeelako definitu da.
3. Plataforma itxita egoera da abiapuntua. Aurrerago ikusiko dugun bezala, pakete bat ateratzerako momentuan biltegitik atera nahi den paketea plataformaren gainean utziko du eta, ondoren, funtzioari deitu ahala atera nahi den paketea plataforman egongo da. Beraz, plataformak egingo duen lehenengo gauza plataformaren gainean utzi den paketearen id-a irakurri izango da.
4. Paketearen id-a irakurri eta gero, konparatu egingo da parametro moduan pasa den `id` aldagaian dagoen balioarekin. Berdinak badira, jarri den paketearen id-a eta sistemak atera nahi duen paketearen id-a berdinak direla ondorioztatu daiteke. Berdinak ez badira, orduan errore bat egongo da atera nahi den paketearekin eta funtzioak 1 balioa bueltatuko du.

5. Paketea eta pasatako id-ak berdinak badira, paketea ateratzera pasatuko gara. Horretarako plataforma ireki behar da `openGate()` funtzioa erabiliz.
6. Behin plataforma irekita dagoela, sistemak 10 segundo itxarongo ditu gehienez, paketea lehenago hartzen ez bada. Denbora hau, erabiltzaileari paketea hartzeko ematen zaion denbora izango da.
7. 10 segundo hauek pasatzen direnean, paketea hartu den edo ez egiaztatuko da, paketearen id-a irakurri daitekeen aztertuz. Paketea hartu bada, plataforma itxi egingo da eta 0 balioa bueltatuko du, dena ondo joan dela adieraziz. Aldiz, paketea ez bada hartu, plataforma itxi egingo da paketea oraindikan gainean daukala eta 2 balioa bueltatuko du, Arduino Megari paketea ez dela hartu adieraziz.

Honekin plataforma mugikorrak eta, ondorioz, Arduino UNO-k bete behar dituen funtzionalitateak aztertzen bukatu dugu. Hala ere, diseinuan ikusi ahal izan dugun moduan, plataforma mugikorraren funtzionamendua, biltegia kontrolatzen duen Arduino Megaren esku geratuko da. Hau da, nahiz eta kontrola eta funtzionamenduak Arduino UNO-k eraman, Arduino Megak modu orokorrean kontrolatzen du biltegia, funtzioak noiz exekutatu behar diren eskatuz Arduino UNO-ri.

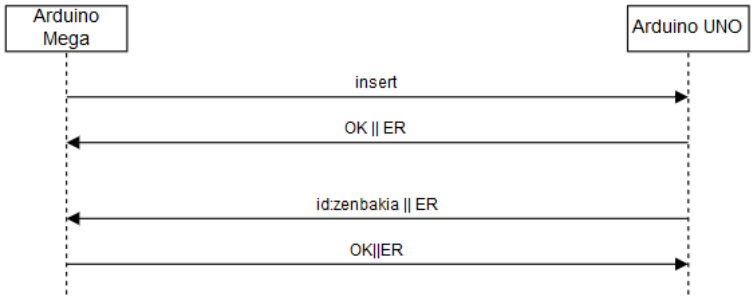
Ondorioz, bukatzeko Arduino Mega eta Arduino UNO-ren arteko komunikazioa aztertu behar dugu, plataforma mugikorraren funtzionamendu orokorra ulertu ahal izateko.

6.1.6 Biltegia (Arduino Mega) eta plataforma mugikorraren (Arduino UNO) komunikazioa

6.2. eta 6.3. irudietan ikusi dezakegu bi Arduinok jarraituko duten protokoloaren definizioa. Hasteko plataforma mugikorrak pakete bat ateratzeko definitutako protokoloa aztertuko dugu (6.2. irudia). Horretarako komunikazioaren komandoak aztertuko ditugu.

- `insert`: Arduino Megak (Biltegia) erabiltzaile batek pakete bat sartu nahi duela jaso duenean, `insert` agindua bidaliko dio Arduino UNO-ri (Plataforma mugikorra). Arduino UNO-k agindua ondo jaso badu, `OK` karaktere-katea bidaliko dio Arduino Mega-ri eta `PlataformInsert()` (6.1.5 atala) funtzioari deituko dio, pakete bat sartzeko beharrezko prozesua hasiz. Berriz, komandoa ez bada ondo ailegatu, `ER` karaktere-katea bidaliko zaio Arduino Megari, prozesua bertan behera geratuz.
- `id || ER`: Behin, `PlataformInsert()` funtzioa ondo bukatu delarik (gogoratu erabiltzaileak 10 segundu dituela paketea plataforman uzteko), funtzioak sartu berri den paketearen id-a `id` aldagaian gordeko du. Jarraian id balioa bidali egingo dio Arduino UNO-k Arduino Megari. Bidaliko den komandoa id izango da eta argumentu bezala id bera bidaliko da, `id:zenbakia` formatua jarraituz. Honek id-a jaso,egiaztatu eta zuzena bada `OK` erantzungo dio Arduino UNO-ri, aldiz, erroreren bat badago `ER` bidaliko dio. Errore baten aurrean berriro ere paketea atera egingo da erabiltzaileak hartu dezan. Horretarako `PlataformExtract()` funtzioari deituko zaio.

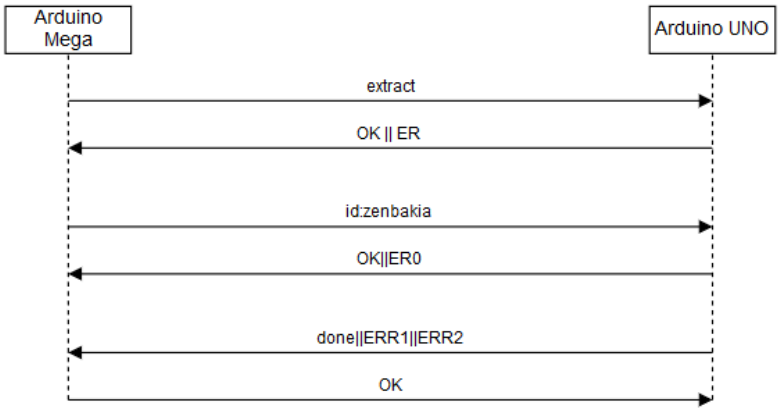
  Aldiz, erabiltzaileak paketeak plataforman uzteko dituen 10 segundu horietan paketerik uzten ez badu, Arduino Megari `ER` mezua bidaliko zaio. Honi Arduino Megak `OK` mezuarekin erantzungo dio.



6.2. Irudia: Bi Arduinoen arteko komunikazioa pakete bat sartzeko

Bestetik, pakete bat atera nahi denean jarraitu beharreko protokoloa hau izango da (6.3. irudia):

- `extract`: Arduino Megak (Biltegia) erabiltzaile batek pakete bat atera nahi duela jaso duenean, `extract` agindua bidaliko dio Arduino UNO-ri (Plataforma Mugikorra). Agindua ondo jaso bada `OK` karaktere-katea bidaliko dio Arduino Megari. Aldiz, komandoa ez bada ondo ailegatzen `ER` karaktere-katea bidaliko zaio Arduino Megari, prozesua bertan behera geratuz.
- `id`: Arduino Megak atera nahi den paketearen id-a bidaliko dio Arduino UNO-ri. Honek id-a ondo jasotzen badu `OK` bidaliko dio. Gaizki jasotzen badu berriz, `ER0` mezua bidaliko dio. Id-a ondo jaso bada, `PlataformExtract()` funtzioari deituko dio pakete bat ateratzeko beharrezko prozesua hasiz.
- `done`: Jaso den id-a eta plataformaren gainean dagoen paketeak daukan id-a berdinak badira, orduan paketea ateratzera pasatuko gara. Id-ak berdinak ez badira `ERR1` mezua bidaliko dio Arduino Megari prozesua bertan behera utziz. Plataforma irekita dagoenean 10 segudu emango zaizkio erabiltzaileari paketea hartzeko. Paketea hartzen badu, plataforma itxi eta `done` komandoa bidaliko zaio Arduino Megari. Aldiz, paketea hartzen ez bada, berriro ere barrura sartuko da paketea eta `ERR2` mezua bidaliko zaio Arduino Megari. Arduino Megak aurreko hiru mezu posibleak jasotzean `OK` erantzungo dio Arduino UNO-ri, komunikazioa bukatuz.



6.3. Irudia: Bi arduinoen arteko komunikazioa pakete bat ateratzeko

3.2.1.6 atalan ikusi dugun moduan Bluetooth HC-05 moduluen bitartezko komunikazioa serie-lerroko komunikazio sinple baten antzekoa da, eta horretarako, komunikazioa posible izateko `SoftwareSerial BTserial` objektu bat sortu dugu, 5.1.3 ataleko Arduino UNO-ren diseinuan ikusi ditugun bi pin digitalak erabiliz.

Beraz, hau behin azalduta, komunikaziorako 6.11. kodean definitu ditugun funtzioak aztertzera pasatuko gara.

```
/* Bilatu ea jaso den mezuako agindua, aurretikan definituta dauden komando array-an
dagoen edo ez. Badago zein posiziotan
aurkitzen den bueltatu
--In : String buf -> Jaso den mezua
       String komandoak-> Aginduak definituta dituen string array-a
-- return : i -> Aurktiu bada ze posiziotan dagoen bueltatu
            -1 -> Ez da agindu bat
*/
int search_command(String buf, String *komandoak)
{
  for(int i=0; i < ARRAYSIZE; i++){
    if(buf == komandoak[i])
      return i;
  }
  return -1;
}
/* Bluetooth bitartesz datozen mezuak irakurtzeko erabili egingo dugun funtzioa
-- IN : String buf -> Mezua gorde egingo den bufferra
    int kop    -> Espero diren byte kopurua
*/
void btserialread(String &buf,int kop){
  buf="";
  while (!recv){
    if (BTserial.available()>kop){
      while (BTserial.available()){
        character = BTserial.read();
        buf = buf + String(character);
      }
      recv=true;
    }
  }
  recv=false;
}
/* Bluettoh komunikaziotik id jaso behar denean, paketearen id jaso den mezutik
ateratzeko erabiliko dugun funtzioa
-- In : String buf -> Id-a atera nahi den mezua
--return : String  -> Paketearen id.*/
String ateraparametroa(String buf){
  int index=buf.indexOf(":");
  return (buf.substring(index+1,11));
}
/*Id jasotzen denean bluetooth komunikaziotik, jasotzen den mezu horretatik bidali den
agindua atera
 --In: String buf-> agindua atera nahi den mezuaren buffera
 --return: String -> Jaso egin den agindua
 */
String aterakomandoa(String buf){
  int index=buf.indexOf(":");
  if (index>0){
    return (buf.substring(0,index));
  }else{
    return buf;
  }
}
/* Bluetooth bidez bidali behar den Komandoa egituratzeko funtzioa
 --In: String Komandoak -> Komando posible guztiak gordetzen duen String array-a
       int agindua -> Komando array horretatik bidali nahi den aginduaren posizioa
       String id-> Parametro bezala bidali behar bada paketearen id-a
 -- reutrn: String eran -> Komandoa egituratua
 */
String komandoaezarri(String *komandoak,int agindua,String &id){
  String eran;
  if (agindua==3){
    eran=KOMANDOAK[agindua];
  }else{
    eran=KOMANDOAK[2];
    eran=eran+":"+id;
  }
    return eran;
}
```

6.11. Kodea: Komunikaziorako funtzio lagungarriak

- `Search_command()`: Protokoloan definitu egin ditugun komando guztiak, `String` taula batean definituta ditugu. Orduan Arduino Megatik jasotako mezua zein den jakiteko, definituta dugun komando zerrendan bilatuko dugu (`insert` eta `extract`), bertan definitu ditugun komandoak jasotako mezuarekin konparatuz. Jasotako mezua komandoen artean aurkitzen bada, komando horrek gure `String` taulan daukan posizioa bueltatuko du. Horrela, jasotako mezua zein den jakingo dugu. Gainera, komandoak taulan daukaten posizio hauek eta Arduino UNO bete behar dituen bi funtzioen identifikadoreen(`INSERT` eta `EXTRACT`) arteko konmunztadura zuzena da. Ailegatu den mezua protokoloan definituta ez dagoen mezua bada, errore kode bat bueltatuko du.
- `Btserialread()`: Bluetooth komunikaziotik datozen mezuak irakurtzeko definitu dugun funtzioa. Honek `BTserial` serie lerrotik mezuak karakterez-karaketere irakurriko ditu, mezuak gordetzeko bufferrean sartuz.
- `Ateraparametroa()` eta `aterakomandoa()`: Id komandoa ailegatzen denean, id komandoa eta id zenbakia den argumentua banatzeko erabiliko diren funtzioak.
- `Komandoezarri()`: Bluetooth bitartez bidali behar diren komandoak bidaliko den buferrean ezartzeko erabili dugun funtzioa.

Behin komunikaziorako lagungarriak diren funtzio hauek ikusi ditugula, kodea aztertzera pasatuko gara. Hasteko Arduino UNO-ren begizta nagusia agindu berri baten zain egongo da beti, 6.12. kodean ikusi dezakegun moduan.

```
#include <SPI.h>
#include <MFRC522.h>
#include <SoftwareSerial.h>
#include "DistanceSensor.h"
#include "ContinuosMotor.h"

// Definitu erabili egingo dire definizio guztiak
#define SS_PIN 10
#define RST_PIN 9
#define ARRAYSIZE 3
#define MAX_BUF 15
// Plataforma mugikorrak izango dituen bi egoerak
#define INSERT 0
#define EXTRACT 1
// Hasieratu Arduino UNO-k erabiliko dituen objektu guztiak
SoftwareSerial BTserial(7,8); // Bluetooth komunikaziorako
// RFID-RC522 modulua erabiltzeko
MFRC522 rfid(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;
// Bi distantzia sentsoreak eta serbomotorra
DistanceSensor DS1(4,2);
DistanceSensor DS2(6,5);
ContinuosMotor motor;

// Definitu aldagai guzitak
String KOMANDOAK [] = {"insert","extract","id","done","ER"};
long D1,D2;
int kasua,ondo,index,egoera;
String id,eran,buf;
byte nuidPICC[3];
boolean finish=false;
boolean opened,closed;
char erantzuna[MAX_BUF];
boolean recv=false;
char character;

void setup()
{
  motor.AttatchR();// Serbomotorra konektatu Arduinoari
  Serial.begin(9600);//PC-ra konektatuta badago, prozesuaren mezua bidaltzkeo konexioa
ezarri 9600 baud-etara
  BTserial.begin(9600);// Bluetooth komunikazioa ezarri 9600 baud-etara
  SPI.begin();           // Hasieratu SPI bus
  rfid.PCD_Init();       // Hasieratu RFID-RC522 modulau
```

```
  // RFID bere komunikazioetarako erabiliko enkriptatzeko erabiliko duen key-a
definitu
  for (byte i = 0; i < 6; i++)
    {  key.keyByte[i] = 0xFF;
    }
  closeGate();
}
void loop()
{
    // Begizta nagusian itxaron Arduino Megatik agindu berri bat ailegatzen den arte
    if (BTserial.available()>4){
      while (BTserial.available()){
        character = BTserial.read();
        buf = buf + String(character);
      }
      recv=false;
      Serial.println("Jasotako agindua "+buf);
      // Bilatu jasotako mezua aurretik definitutako aginduen artean dagoen
      kasua=search_command(buf,KOMANDOAK);
      // Jasotako aginduaren arabera
      switch(kasua){
```

6.12. Kodea: Arduino UNO-ren begizta nagusiaren hasiera

6.12 kodean hasieran ikusi dezakegu Arduino UNO-n egiten diren hasieraketak eta erabili diren liburutegiak. Hauek kodearekin ulergarriak direnez ez ditugu sakonean aztertuko.

Arduinoaren `setup` funtzioan, Arduinoak erabiliko dituen modulu ezberdinen hasieraketak egiten dira.

Begizta nagusian etengabe Arduino Megatik mezu baten zain geratuko da gure Arduino UNO-k. Mezu bat jasotzen denean Bluetooth komunikaziotik, aurretikan definituta dauden bi komandoen artean (`insert` eta `extract`) dagoen edo ez aztertuko da, aurretikan ikusi dugun `search_command()` funtzioarekin. Ondorioz, funtzio horren emaitzaren arabera hiru kasu ezberdin definitu ditugu.

```
case INSERT:
         id="";
         // Ok bidalito Arduino MEgari koamndoa
         // ondo jaso da eta.
  1      BTserial.write("OK");
         Serial.println("OK send");
         // Paketeak sartzeko funtzioa. Paketearen rfid etiketaren
         // id balioa sartu id aldagaian.
         Serial.println("Sisteman pakete bat sartzera doa");
  2      egoera=PlataformInsert(id);
         if (egoera==0){
             komandoaezarri(KOMANDOAK,2,id).toCharArray(erantzuna,MAX_BUF);
         Serial.println("Jasotako paketearen id bidalita: "+
komandoaezarri(KOMANDOAK,2,id));
           //id-a bidali Arduino Mega-ri
  3      BTserial.write(erantzuna);
        //Arduino Megatik OK mezua itxaron eta jasotakoa interpretatu
  4      btserialread(buf,1);
           if(buf=="OK"){
            //Dena ondo antera bada
              recv=true;
              buf="";
              Serial.println(id+":daukan paketea ondo sartu da biltegian");
           }else if(buf=="ER"){
            //ERR mezua bidaltzen badu sartutako paketea aterako da, erabiltzaileak
hartu dezan
              PlataformExtract(id);
              recv=true;
              buf="";
              Serial.println("Zeozer gaizki joan da");    //TODO
           }
  5      }else{
         komandoaezarri(KOMANDOAK,4,id).toCharArray(erantzuna,MAX_BUF);
         BTserial.write(erantzuna);
```

```
        //Arduino Megatik OK mezua itxaron eta jasotakoa interpretatu
        btserialread(buf,1);
          if(buf=="OK"){
           //Dena ondo antera bada
             recv=true;
             buf="";
             Serial.println("Ez da paketerik sartu biltegian");
          }else if(buf=="ER"){
             recv=true;
             buf="";
             Serial.println("Zeozer gaizki joan da");
          }
        }
        break;
```

6.13. Kodea: Insert agindua jasotzearen kasua

Arduino Megatik `insert` komandoa jaso bada 6.13 kodean ikusi daitekeen kasua definitu dugu.

1. Agindua ondo jaso dela bidali egingo zaio Arduino Megari bluetooth bitartez, `OK` mezua bidaliz.
2. Ondoren, lehen ikusi dugun `PlataformInsert()` funtzioari deituko zaio, pakete berria biltegira sartzeko prozesua egiteko. Aurretik ikusi dugun moduan (6.1.5 atala) funtzio honek osoko balio bat bueltatuko du bukatu den egoera adierazteko. Bi egora ezberdinetan bukatu daiteke.
3. `PlataformInsert()` funtzioa ondo bukatu bada, hau da, 0 balioa bueltatu badu, erabiltzaileak paketea sartu duela adieraziko du. Jarraian, sartu den paketearen id-a bidaltzeko, komandoa egituratu eta bidaliko da, Arduino Megaren erantzunaren zain geratuz.
4. Erantzuna jaso eta gero, jasotako mezuaren arabera bi egoera eman daitezke.
   - `OK` jaso bada, dena hasierako egoerara bueltatuko da eta prozesua amaitutzat emango da. Hau da, plataforma mugikorra paketea ondo sartu du.
   - `ER` jaso bada, paketearekin arazoren bat dagoela adieraziko du. Ondoren, `PlataformExtract()` funtzioari deituko zaio, erabiltzaileak erroredun paketea hartzeko. Honekin prozesua amaitutzat emango da eta komando berri baten zain egotera pasako da plataforma.
5. Aldiz, `PlatormInsert()` funtzioak 1 balioa bueltatzen badu, erabiltzaileak sistemak emandako denboran plataforman paketerik utzi ez duela adieraziko du. Beraz, Arduino Megari `ER` mezua bidali egingo zaio eta honen `OK` mezuaren zain geratuko gara. Hau ailegatzerakoan prozesua amaitutzat emango da eta agindu berri baten zain egotera pasatuko da Arduino UNO-k.

```
case EXTRACT:
        // Agindua ondo jaso dela, Arduino Megari OK bidali
  1     BTserial.write("OK");
        Serial.println("OK send");
        buf="";
        recv=false;
        // jaso atera behar den paketearen id-a
  2     btserialread(buf,10);
        Serial.println("Jasotako id -> "+buf);
        //agindutik id-a ateratzeko
        id=ateraparametroa(buf);
        Serial.println("Atera behar den paketearen id "+id);
  3     if(search_command(aterakomandoa(buf),KOMANDOAK)== 2){
           // ondo jaso bada id-a OK bidali
          BTserial.write("OK");
          Serial.println(id+" paketea ateratzera pasatuko gara");
          //Atera behar den paketea
          ondo=PlataformExtract(id);
```

```
4          switch(ondo){
           //Paketea ondo atera bada, eta erabiltzaileak hartu badu
I           case 0:
             komandoaezarri(KOMANDOAK,3,id).toCharArray(erantzuna,MAX_BUF);
             BTserial.write(erantzuna);
             recv=false;
             btserialread(buf,1);
             if(buf==”OK”){
               Serial.println(“Dena OK”);
               buf=””;
             }else{
                  Serial.println(“Sistema hasierako egoerara bueltatu da”);
                   buf=””;
             }
             break;
             //Plataforman biltegiak jarritako paketea ez da jarri behar zena eta beraz
ez da atera
             // Done mezua bidali beharrean ER1 mezua bidali
II          case 1:
             BTserial.write(“ERR1”);
             Serial.println(“Plataforman daogen paketea ez da izan behar dena”);
             recv=false;
             btserialread(buf,1);
             if(buf==”OK”){
               Serial.println(“Plataforma bere hasierako egoerara bueltatuko da “);
               buf=””;
             }else{
                   Serial.println(“Sistema hasierako egoerara bueltatu da”);
                    buf=””;
             }
             break;
             // Atera da paketea, baina erabiltzaileak ez du hartu, Done agindua bidali
beharrean
             // ER2 mezua bidali
III         case 2:
             BTserial.write(“ERR2”);
             Serial.println(“Erabiltzaileak ez du paketea hartu”);
             recv=false;
             btserialread(buf,1);
             if(buf==”OK”){
               Serial.println(“Plataforma bere hasierako egoerara bueltatuko da”);
               buf=””;
             }else{
                   Serial.println(“Sistema hasierako egoerara bueltatu da”);
                    buf=””;
             }
             break;
            }
        }else{
        // Jasotako id komandoa ez da jaso behar dena, ER0 errorea bidali OK izan
beharrean
        BTserial.write(“ERO”);
        Serial.println(“Ez da jaso id-a formatu honean”);
        }
        break;
```

6.14. Kodea: Extract agindua jasotzearen kasua

Arduino Megatik `extract` koamandoa jaso bada 6.14 kodean ikusi daitekeen kasua definitu dugu.

1. Hasiera berdina izango da. Hasteko `OK` mezua bidali egingo zaio Arduino Megari agindua ondo jaso dela abisatzeko.
2. Prozesuarekin hasi baino lehen, Arduino Mega biltegitik atera nahi den paketearen id-a bidaltzearen zain geratuko gara. Behin jasotzen dugunean, jasotako mezu honetatik komandoa eta parametroak banatzera pasatuko gara, lehen ikusi ditugun `ateraparametroa()` eta `aterakomandoa()` funtzioak erabiliz. Behin biak ditugula, jaso den komandoa (`id`) behar zena dela egiaztatu egingo da. `Id` komandoa ez bada zuzena, `ER0` karaktere-katea bidaliko zaio Arduino Megari eta prozesua bertan behera geratuko da.
3. Id-a ondo jasotzen bada, paketea ateratzera pasatuko gara, lehen azaldu dugun `PlataformExtract()` funtzioa erabiliz.

4. Lehen esan dugun moduan `PlataformExtract()` funtzioak osoko balio bat bueltatuko du, prozesuak zer nolako egoeran bukatu den zehazteko. Aurreikusi diren hiru egoerak aztertuko ditugu:

    I. Paketea ondo atera da eta erabiltzaileak paketea hartu badu, funtzioak 0 balioa bueltatuko du. Ondoren, `done` komandoa bidaliko zaio Arduino Megari eta honetatik `OK` mezua jasotzean, prozesua bukatuko da, hasierako egoerara bueltatuz.

    II. Jasotako id-a eta plataformaren gainean jarritako paketeak id desberdina du. Orduan `ERR1` errore mezua bidaliko zaio Arduino Megari eta honen `OK` erantzuna jasotzerakoan prozesua bukatutzat emango da.

    III. Paketea ondo atera bada, baina erabiltzaileak ez du paketea hartu aurretik ikusi dugun 10 segunduko denbora tartean. Kasu honetan, `ERR2` errore mezua bidaliko zaio Arduino Megari eta Arduino Megak mezua jaso ondoren bidaliko duen `OK` mezua jasotzean, prozesua amaitutzat emango da.

Berriz, jaso den lehengo mezua ez badago espero diren aginduen artean, hau da, `insert` edo `extract` aginduak ez badira, `default` kasuan gaude.

```
default:
BTserial.write("ER");
Serial.println("Ez da komando jakin bat ailegatu");
buf="";
break;
```

6.15. Kodea: default kasua

Jasotako komandoa ezezaguna denez, `ER` mezua bidaliko zaio Arduino Megari eta agindu berri baten zain geratuko da Arduino UNO (Ikusi 6.15. kodea).

6.2 Arduino Mega (Biltegia)

Arduino Megak gure sistemaren hardware atalaren kontrol orokorra eramango du. Dokumentu honetan zehar azaldu dugun moduan Arduino Mega honek lau funtzio nagusi ditu:

- Biltegian dauden eta sartuko diren paketeen administrazioa.
- Paketeak biltegian zehar mugitzeko erabiliko den beso robotikoaren kontrola.
- Arduino UNO-k kontrolatzen duen plataforma mugikorrari paketeak biltegian sartzeko edo biltegitik ateratzeko aginduak bidaltzea.
- Azkenik, erabiltzaileak erabiliko duen web aplikazioarekin komunikazioa mantendu, honetatik egin beharreko aginduak jasoz.

Beraz, lau funtzio hauek nola inplementatu diren ikusiko dugu. Hasteko, biltegiaren funtzionamendua errazteko, egituratu egin diren bi klaseetatik hasiko gara, geroago sistema osoa nola inplementatu den ulergarriago izan dadin. Aztertuko ditugun bi klaseak azaldu ditugun lehenengo bi funtzioak betetzeko egituratu dira.

6.2.1 Paketeen erregistroa (Biltegiko paketeen adiministrazioa)

Web aplikazioaren garapena aztertzen dugunean (7.2.1.2 atala), ikusiko dugu nola paketeen erregistroa datu-base baten bidez egin den. Kasu honetan, web aplikazioan ez bezala, Arduinok memoria ahalmen gutxi dauka eta, beraz, datu-base bat inplementatzea proiektu honen esparrutik at geratzen da. Ondorioz, klase baten bidez, sisteman dauden eta egon daitezkeen

paketeen kontrola eramatea erabaki da. Horretarako ikusiko dugun moduan bi taulen bidez egitea erabaki da.

```
// Kasu honetan biltegiratzeko 6 espazio daudenenz.
#define space 6

class Biltegia{
private:
	/*Klaseko ezaugarri pribatuak. Kasu honetan zuzenean inizializatu egiten dira
	, inizializatzeko metodoak definitu ahal izango ziren*/

	/* Biltegia definitzeko bi array erabili egiten dira. Lehenengo inUse[]
momentu horretan
	erabilita dauden biltegiratze espazioak definitzen ditu.
		- True: Espazioa erabilita izaten ari data.
		- False: Espazioa erabilgarria da.
	*/
	boolean inUse[space]={false,false,false,false,false,false};
	/*Bigarrean berriz gordeta dauden paketeen id-a gordetzen dute.Kontuan izan,
nahi eta posizio
	batean id bat idatzita egon, inUse[] array-a izango da erabilgarria den edo ez
adieraziko diguna.
	*/
	String packages[space]={"","","","","",""};
	const int HasMemory=12;
public:

	Biltegia();
	boolean emptySpace();
	int savedPackage(String rfid);
	int savePackage(String rfid);
	int removePackage(String rfid);
	String valuePackage(int i);
	boolean valueinUse(int i);
} ;
```

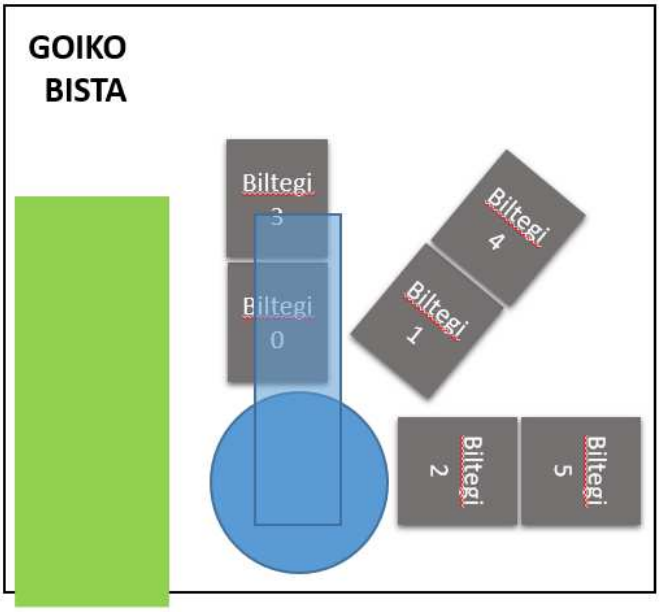
6.16. Kodea: Biltegia.h

6.16 kodean gure `Biltegia` klasea definituta daukagu. Klaseak bi taula izango ditu propietate pribatu gisa, paketeen administrazioa eramateko.

- `inUse[]`: Taularen luzera biltegiak daukan biltegiratze-espazio maximoa izango da. Gure proiektuan 6 biltegiratze-espazio posible definitu egin dira. Beraz, biltegiratze maximoa 6 pakete izango da. `inUse[]` taulak biltegiratze-espazio hauek erabilita edo libre dauden kontrolatuko duen egitura da. Horretarako, balio boolearrak erabiltzea erabaki da.

  6.4 irudian ikusi dezakegun moduan taula honetako posizio bakoitzak biltegiko espazio bati erreferentzia egingo dio. Ondorioz, posizio batean `false` balioa badago, espazioa erabilgarria dela adieraziko du. `True` balioa badauka, berriz, espazioa jadanik beteta dagoela adieraziko du.

  Adibidez, `inUse[4]` posizioan `true` balioa badauka, biltegiaren 4. posizioan jadanik pakete bat gordeta dagoela ondorioztatuko dugu.

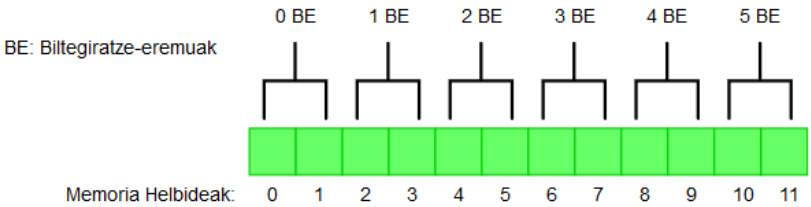


6.4. Irudia: Biltegi espazioen eskema

- `packages[]`: Kasu honetan karaktere-katez osatutako taula bat daukagu. Honen luzera `inUse[]`-ekin gertatzen den moduan biltegiak fisikoki duen biltegiratze-espazio maximoarekin zuzenean erlazionatuta dago. Honetan, biltegiratze-espazio bakoitzean gordeta dagoen paketearen id zenbakia gorde egingo da. Beraz, `packages[]` eta `inUse[]` taulek erlazio zuzena izango dute. `InUse[]`-k ezarriko du, espazio bakoitzean paketerik gordeta dagoen edo ez eta, aldi berean, paketeak gordeta dauden espazioetan paketearen id-a `packages[]` taulan gordeko dira. Beraz, `inUse[]`-eren kasuan jarritako adibidearekin jarraituz, `inUse[4]` posizioan `true` balioa badago, 4. biltegiratze-eremuan pakete bat gordeta dagoela adieraziko du. Ondorioz, `packages[4]` posizioan pakete horren id-a egongo da.

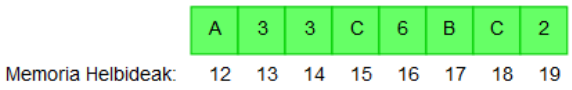
Kontuan izan behar da, pakete bat ateratzen denean, `packages[]` taula honetan gordeta zegoen paketearen id-a ez dela ezabatuko, posizioak erabilita edo ez dauden zehazten duena `inUse[]` taula izango baita. Beraz, pakete berri bat posizio horretan gordetzean, `packages[]` dagoen balio zaharra berridatziko da. Hau da, ez da beharrezkoa pakete bat ateratzean, `packages[]` posizio horretan dagoen balioa berriz hasieratzea.

Bestetik paketeen informazioa epe luzerako gorde ahal izateko, eta Arduinoa itzaltzean gordeta dauden paketeen informazioa ez galtzeko, Arduino Megak daukan EEPROM memorian informazioa gordetzea erabaki da. Beraz, ikusi ditugun `inUse[]` eta `packages[]` taulen egituren kopia bat EEPROM memorian izatea erabaki da. Horretarako `inUse[]` tauleko balioak osoko moduan gorde dugu, 0-a `false` adieraziz eta 1 balioa `true` izanda. 6.5 irudian ikusi daiteke memorian `inUse[]` taularen parekoa. Kontuan izan Arduinon osokoak bi byte osatzen dutela.



6.5. Irudia: EEPROM memorian inUse[]

Bestetik memoriaren 12. posiziotik aurrera paketearen id-ak gordeko ditugu. 12. posizio horretatik hasi behar garela klaseak propietate pribatu bezala daukan `HasMemory` aldagaia zehaztuko du. Hau biltegiratze-eremu maximoaren balioarekin zuzenki erlazionatuta dago. Lehengo 12 posizioak `inUse[]` katea gordetzeko erabili egingo dira, biltegiratze-eremu bakoitza gordetzeko 2 byte behar baitira (osokoak= 2byte). Beraz, 12. helbidetik aurrera memoria posizio bakoitzean karaketere bat gorde egingo da. Lehenengo posizioan paketerik gordeta badago, 6.6 irudian paketearen id-a nola gordeko zen ikusi dezakegu. Beraz, bigarren posizioa 20. helbidean hasiko da, hirugarren posizioa 28. helbidean, etab.



6.6. Irudia: packages() taularen lehenengo biltegiratze-eremua EEPROM-en

Azkenik, memorian egin behar diren bilaketak azkarragoak izan daitezen 120 eta 121. posizioetan osoko balio batean momentu bakoitzean gordeta dauden paketeen kopurua gorde egingo da. 120. helbidea erabiltzea erabaki da, helbide hauek ez baitira erabiltzen.

Behin `Biltegia` klasea osatzen duten bi propietate pribatuak aztertu ditugula, klasean definitu ditugun metodo publikoak azalduko ditugu. (6.17. kodea)

```
/* Biltegi klasearen inizializazioa. Kasu honetan 6 eremutaz hitz egingo dugunez, 6
balio izango
dute array bakoitza, balio bat eremu bakoitzeko. Lehen esan dugun moduan, aurretikan
hasieratuta
dugu, hala ere, hemen modu ezberdin batean hasieratu ahal da*/
Biltegia::Biltegia(void){
}
/* Biltegian pakete berri batentzako lekurik dagoen edo ez jakiteko funtzioa.
        --return:True espazioa badago
                         False espazioa ez badago
*/
boolean Biltegia::emptySpace(){
        for (int i=0;i<space;i++){
                if (inUse[i]==false){
                        return true;
                }
        }
        return false;
}
/* Parametro moduan pasatako rfid sisteman gordeta dagoen edo ez esango digun funtzioa
        -- in : String rfid (Paketearen identifikadorea)
        -- return : i    -> Paketea sisteman gordeta badago, gordeta dagoen posizioa
bueltatuko da
                                -1   -> Paketea sisteman ez badago
*/
int Biltegia::savedPackage(String rfid){

        for (int i=0;i<space;i++){
                if (packages[i]==rfid){
                        if (inUse[i]==true){
                                return i;
                        }
                        else{
                                return -1;
                        }
                }
        }
        return -1;
}
/* Parametro moduan pasatako rfid (paketea), gorde egingo du sisteman. Sistema, libre
aurkitzen duen
lehenengo espazioan gorde egingo du paketea beti.
        -- in : String rfid (Paketearen identifikadorea)
        -- return: i -> Paketea gorde egin den array posizioa.
```

```
                        -1 -> Rfid horrekin jadanik pakete bat gordeta badago edo
sisteman espazio gehiago ez badago.
*/
int Biltegia::savePackage(String rfid){
       char aux[8];
       int pos;
       int gordeta;
       int index=savedPackage(rfid);
       if (index < 0){
         if (emptySpace()){
           for (int i=0;i<space;i++){
             if (inUse[i]==false){
               packages[i]=rfid;
               inUse[i]=true;
              // zenbat pakete gordeta dauden memorian idatzi
               EepromUtil::eeprom_read_int((HasMemory * 10),&gordeta);
               gordeta++;
               EepromUtil::eeprom_write_int((HasMemory * 10),gordeta);
               // zein posizioan gorde den zehaztu memorian
               EepromUtil::eeprom_write_int(i*2,1);
               rfid.toCharArray(aux,9);
               pos=HasMemory+(8*i);
               // gorde den paketearen id-a gorde memorian
               EepromUtil::eeprom_write_string(pos,aux);
              return i;
             }
           }
         }else{
           return -1;
         }
       }else{
         return -1;
       }
}
/* Sisteman gordeta dagoen pakete bat atera egingo du, hau da, biltegitik ezabatu
egingo du
    Horretarako bakarrik inUse[] array-a aldatu egingo da, hau baita erabilgarritasuna
adierazten
       duen array-a
       -- in : String rfid (Paketearen identifikadorea)
       -- return: index -> Borratu den paketea aurkitzen zen posizioa bueltatuko da
                           -1   -> rfid horrekin ez dago paketerik gordeta*/
int Biltegia::removePackage(String rfid){
       int index=savedPackage(rfid);
       int gordeta;
       if (index >= 0){
              inUse[index]=false;
              // gordeta dauden paketeen kopurua dekrementatu
              EepromUtil::eeprom_read_int((HasMemory * 10),&gordeta);
              gordeta--;
              EepromUtil::eeprom_write_int((HasMemory * 10),gordeta);
              // posizioa libre bezala ezarri
              EepromUtil::eeprom_write_int(index*2,0);
              return index;
       }else{
              return -1;
       }
}
/* Emandako posizioan, packages array-ean dagoen balioa bueltatuko digun
       -- in: i -> Lortu nahi den balioaren posizioa
       -- return: packages[i] -> arrayaren posizioan horretan dagoen balioa
bueltatuko du.
*/
String Biltegia::valuePackage(int i){
       return packages[i];
}
/* Emandako posizioan, packages array-ean dagoen balioa bueltatuko digun
       -- in: i -> Lortu nahi den balioaren posizioa
       -- return: inUse[i] -> arrayaren posizioan horretan dagoen balioa bueltatuko
du.
*/
boolean Biltegia::valueinUse(int i){
       return inUse[i];
}
```

6.17. Kodea: Biltegia.cpp

Klase gehienetan ikusten diren eraikitzailea eta balioak lortzeko diren get (`valuePackage()` eta `valueinUse()`) metodoak alde batera utzita, biltegian paketeak erregistratzeko eta ezabatzeko inplementatu ditugun metoak aztertzera pasatuko gara.

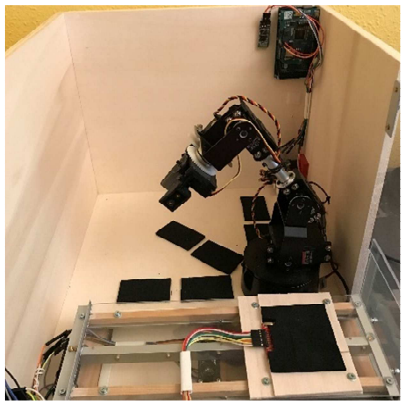
- `emptySpace()`: Lehenik eta behin, biltegiratzeko espazio librea dagoen edo ez aztertzen duen funtzioa daukagu. Honek `false` bueltatuko digu biltegia beteta dagoenean. Biltegiratze espazioren bat libre dagoenean, berriz, `true` bueltatuko du.
- `savedPackage()`: Pakete bat biltegiratuta dagoen edo ez jakiteko balio digun funtzioa. Parametro moduan pasatzen den id-a biltegian bilatzen du. Aurkitzen badu, hau gordeta dagoen biltegiratze-eremuaren posizioa bueltatuko du. Gordeta ez badago, berriz, *-1* balioa bueltatuko du.
- `savePackage()`: Pakete bat sisteman erregistratzeko (gordetzeko) erabiliko den funtzioa da. Honetan, parametro moduan pasatzen den id-a gordeko da biltegian. Horretarako lehenik eta behin, jadanik id horrekin paketerik gordeta dagoen edo ez egiaztatuko da, `savedpackage()` funtzioaren bitartez. Id hori gordeta ez dagoela egiaztatzean, biltegian pakete berri baterako espazioa dagoen edo ez aztertuko da, `emptyspace()` funtzioren bitartez. Pakete berri baterako espazioa badago, paketea sisteman erregistratuko da.

  Paketeak sisteman erregistratzeko, libre dagoen lehenengo eremuan gordeko da beti. Beraz, libre dagoen lehenengo espazioan, `inUse[posizioa]=true` balioa ezarriko da, eta honekin batera posizio horri EEPROM memorian dagokion helbidean 1 balioa ezarriko da. Jarraian `packages[posizioa]`-n id-a gordeko da eta honekin batera posizio horri dagokion EEPROM memoriaren helbidean id-a gorde egingo da. Bestalde, gordeta dauden paketeen kopurua irakurri eta bategaz inkrementatu egingo da, dagokion posizioan balio berria gordez. Funtzioak paketea gorde den posizioa bueltatuko du.
- `removePackage()`: Gordeta dagoen pakete bat sistematik ezabatzeko erabiliko den funtzioa. Ezabatu behar den paketearen id-a, parametro moduan pasatuko zaio metodo honi. Beraz, paketea gordeta dagoen ala ez jakiteko, `savedPackage()` funtzioa erabaliko da. Gordeta badago, funtzioak zein posiziotan gordeta dagoen bueltatuko digu eta beraz, posizio horretako `inUse[posizioa] false`-*ra* ezarriko da eta honekin batera biltegiratze-espazio horri dagokion EEPROM memoriako posizioan 0-ko balioa ezarriko da. Bestalde, paketeen kopurua dekrementatu egingo da eta dagokion EEPROM memoriaren eremuan gorde.

Beraz, honekin gure hardware sistemak gordeta dituen paketeen administrazioa nola kudeatzen duen ikusi dugu. Jarraian, paketeak sisteman zehar mugiarazteko erabiliko dugun beso robotikoa kontrolatzeko definitutako klasea azalduko dugu.

6.2.2 Beso robotikoa

Hardware diseinua aztertzean biltegian paketeak alde batetik bestera mugitzeko, beso robotikoa erabiliko dugula zehaztu dugu (ikusi 5.1.1.2 atala). Beso robotikoaren funtzio nagusia paketeak plataforma mugikorretik biltegiratze-espazioetara eta, alderantziz, mugitzea da. Beso robotikoaren mugimenduak finkoak eta errepikagarriak direnez, hauek aurretik definitzea erabaki da, prozesu bakoitzerako mugimendu segida batzuk definituz; hauekin beso robotikoarentzat klase bat sortuz: `BesoRobotikoa`. Definizioa eta API-a 8.18 kodean ikus daiteke (6.7 irudia).



6.7. Irudia: Beso robotikoaren irudia

Serbomotorren erabilpena errazteko Arduino-k eskuragarri daukan `<Servo>` liburutegia erabili dugu.

```
#include “Arduino.h”
#include <Servo.h>
/* Gure sistemak biltegiratzeko sistemak 6 espazio ditu. 0-tik 5-era joaten direnak.*/
class BesoRobotikoa{
private:
        /*Pin Parameters*/
        int PwnMotor1;
        int PwnMotor2;
        int PwnMotor3;
        int PwnMotor4;
        int PwnMotor5;
        int PwnMotor6;
        /*Servo objektuak*/
        Servo Servo1;
        Servo Servo2;
        Servo Servo3;
        Servo Servo4;
        Servo Servo5;
        Servo Servo6;
        /* Mugimendu bakoitza egiteko itxaron behar den denbora*/
        long StopTime;
public:

        BesoRobotikoa(void);
        void Attach();
        void StartPosition();
        void PrincipalPosition();
        void PlataformPosition();
        void PlataformCatch(boolean take);
        void DownPackages(int position,boolean take);
        void UpPackages(int position,boolean take);
        int SavePackage(int position);
        int ExtractPackage(int position);
        void timer(long interval);
        int calculatePosition(int position);
        int takePackage(boolean take);
} ;
```

6.18. Kodea: BesoRobotikoa.h

Propietate pribatuak hiru multzotan banatu ditzakegu:

- `PwmMotorX`: Beso robotikoa osatzen duten sei serbomotorrak kontrolatzeko erabiliko diren PWM pin-ak.
- `ServoX`: Beso robotikoaren sei serbomotorrak kontrolatzeko erabiliko diren `Servo` objektuak. Serbomotorrek daukaten zenbakitzea 5.1.1.2 ataleko 5.4 irudian dagoenarekin zuzenki erlazionatuta dago.

- `StopTime`: Beso robotikoari posizio berri bat hartzeko agindu ondoren, posizioa ondo hartzeko itxaron behar den batazbesteko denbora definitzen du.

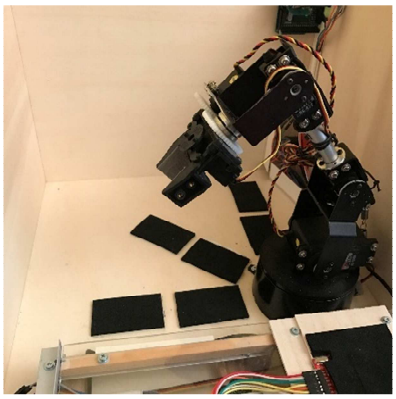
6.19 kodean `BesoRobotikoa` klasearentzat definitutako metodo publikoak ikus daitezke.

```
#include “BesoRobotikoa.h”
#define _PwnServo1 2
#define _PwnServo2 3
#define _PwnServo3 4
#define _PwnServo4 5
#define _PwnServo5 6
#define _PwnServo6 8
/* Beso robotiko hasieratzeko behar den konstruktorea. Kasu honetan pin-ak eta
StopTime jadanik
definitutta ditugu*/
BesoRobotikoa::BesoRobotikoa(void) {
        PwnMotor1=_PwnServo1;
        PwnMotor2=_PwnServo2;
        PwnMotor3=_PwnServo3;
        PwnMotor4=_PwnServo4;
        PwnMotor5=_PwnServo5;
        PwnMotor6=_PwnServo6;

        StopTime=2;
}
/* Serbo bakoitza, kontrol pinarekin konektatzeko balio digun metodoa. Metodo hau ondo
funtzionatzeko,
Arduino Scketch-aren setup() atalan jarri beharko da*/
void BesoRobotikoa::Attach(){
        Servo1.attach(PwnMotor1);
        Servo2.attach(PwnMotor2);
        Servo3.attach(PwnMotor3);
        Servo4.attach(PwnMotor4);
        Servo5.attach(PwnMotor5);
        Servo6.attach(PwnMotor6);
}
/* Beso robotikoa hasieratzeko erabili egingo dugun posizioa, hau bakarrik erabili
egingo da behin,
programaren hasieran*/
void BesoRobotikoa::StartPosition(){
        Servo1.write(95);
        Servo2.write(90);
        Servo3.write(70);
        Servo4.write(0);
        Servo6.write(0);
        timer(StopTime);
}
/* Beso robotikoa mugimendu bat egin eta gero hasierako posiziora bueltatzeko
erabiliko duen metodoa.
Kontuan hartu honetan 6. serboa ez dela erabiltzen. Servo6 paketea hartzeaz arduratuko
da, beraz, hau ez da ukitu behar,
balitekelako pakete bat izatea eta botatzea*/
void BesoRobotikoa::PrincipalPosition(){
        Servo1.write(95);
        Servo2.write(90);
        Servo3.write(70);
        Servo4.write(0);
        timer(StopTime);
}
/* Plataforma mugikorretik pakete bat hartzeko beso robotikoa hartu behar duen
posizioa*/
void BesoRobotikoa::PlataformPosition(){
        Servo1.write(10);
        Servo2.write(90);
        Servo3.write(70);
        Servo4.write(0);
        timer(StopTime);
}
```

6.19. Kodea: Biltegia.cpp lehengo atala

Inplementatu diren metodoak luzeak direnez kodea hainbat zatitan banatzea erabaki da, (6.19 – 6.22 kodeak).

- `BesoRobotikoa()`: Klasearen eraikitzailea. Honetan, erabiliko diren PWM pin-ak definitu eta `StopTime` propietatearen balioa zehazten da, kasu honetan, 2 segundutan definitu dugu.
- `Attach()`:Honek serbomotorren eta Arduino Megaren arteko konexioa ezartzeko balioko digu. Erabili diren pin-ak propietate pribatu gisa dauden `PwmMotorX` dira.
- `StartPosition()`: Arduino piztean, beso robotikoak hartuko duen posizioa. Honetan serbomotor guztiek izango dute hasierako balioa, 6.8 irudian ikusi dezakegun moduan. Metodo hau bakarrik sistema hasieratzean erabiliko da. Hortik aurrera hasierako posiziora bueltatzeko `PrincipalPosition()` metodoari deituko zaio.



6.8. Irudia: Beso robotikoa StartPosition()

- `PrincipalPosition()`: Beso robotikoa paketeak mugitzen hasten denean, beso robotikoa edozein posizioan dagoela hasierako posiziora bueltatzeko erabiliko den metodoa. `StartPosition()` metodoak eta metodo honek daukaten desberdintasun bakarra, seigarren serbomotorran egongo da. Seigarren motor hau paketeak hartu eta usteaz arduratuko da eta, beraz, daukan balioa ez da aldatu behar.

  Adibidez, imagina dezagun pakete bat hartzera goazela, paketea bere posiziotik hartzen dugu eta beste posizio batean utzi baino lehen, hasierako posiziotik pasatu behar garela. `StartPosition()` metodoari deitzen badiogu seigarren serbomotorrari 0 balioa jarriko zaio eta, ondorioz, mugitzen ari garen paketea jauzi egingo da. Beraz, garrantsitzua da seigarren serbomotorrak daukan balioa errespetatzea.

- `PlataformPosition()`: Plataforma mugikorretik paketeak hartu edo plataforma mugikorrean paketak utzi aurretik hartu beharreko posizioa.

`BesoRobotiko` klasean inplementatu ditugun metodo publikoak aztertzen jarraituko dugu (6.20 kodean).

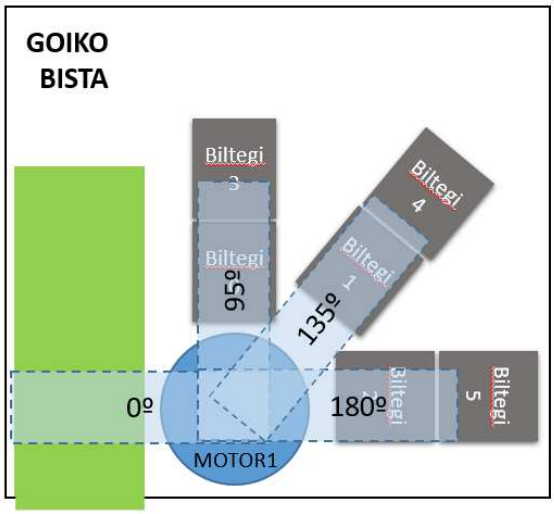
```
/* Beso robotikoa pakete bat hartzeko edo usteko behar duen balioa kalkulatu. Kasu
honetan, honetaz
arduratuko den servoa servo6 izango da.
       -- In: bool take -> True: Pakete bat hartu.
                          False:Pakete bat utzi.
       -- return: angelua -> 90: Pinzak itxi eta paketea hartu.
                             0: Pinzak ireki eta paketea utzi.*/
int BesoRobotikoa::takePackage(boolean take){
       if (take){
              return 90;
       }else{
              return 0;
       }
}
/* Plataforma robotikoa plataformatik pakete bat utzi edo hartzeko egin beharreko
mugimendu segida.
       -- In : bool take -> True: Pakete bat hartu behar da plataformatik.
                         False: Beso robotikoa daukan paketea plataforma utzi behar da.
*/
```

```
void BesoRobotikoa::PlataformCatch(boolean take){
	int tak;
		tak=takePackage(take);
		timer(StopTime);
		Servo2.write(80);
		Servo3.write(80);
		timer(StopTime);
		Servo3.write(90);
		timer(StopTime);
		Servo3.write(98);
		timer(StopTime);
		Servo6.write(tak);
		timer(StopTime);
		if (!take){
			Servo3.write(80);
			timer(StopTime);
		}
		PrincipalPosition();
		timer(StopTime);
}

void BesoRobotikoa::timer(long StopTime){
/* timer bat itxoin dezan sartzen diren segunduak*/
	unsigned long currentMillis,previousMillis;
	long interval=1000;
	currentMillis = millis();
	previousMillis = millis();
	interval=interval*StopTime; // segundu hoiek milisegunduetara pasa
	while (currentMillis-previousMillis<interval){
		currentMillis=millis();
	}
}
/* Biltegiko eremu bakoitzeko beso robotikoa hartu behar duen x ardatzarekiko angelua
kalkulatzen duen funtzioa. Servo1 motorrentzako balioa.
	-- In: int position -> Biltegiko espazio bat.
	-- return: angelua -> Posizioaren arabera servo1 hartu beharreko angelu balioa
			-1	-> Posizio hori existitzen ez bada.
*/
int BesoRobotikoa::calculatePosition(int position){
	if ((position== 0) || (position==3)){
		return 95;
	}
	else if ((position== 1) || (position==4)){
		return 135;
	}
	else if ((position== 2) || (position==5)){
		return 180;
	}else{
		Serial.println("Ez da baliozko posizioa");
		return (-1);
	}
}
```

6.20. Kodea:Biltegia.cpp bigarren atala

- `takePackage()`: Funtzio honen bitartez pakete bat utzi edo hartzeko, seigarren serbomotorrari eman behar zaion balioa kalkulatuko da. Paketea hartu edo utzi behar den jakiteko, parametro moduan pasatuko zaio `take` aldagai boolearra.
- `PlataformCatch()`: `PlataformPosition()`-ek adierazitako hasierako posizioan gaudela, plataforma mugikorretik pakete bat hartzeko edo uzteko erabiliko dugun funtzioa da. Paketea hartu edo utzi behar den, `take` parametro boolearraren bitartez adieraziko da. Matxardak (seigarren serbomotorra) `takePackage()` funtzioaren bitartez paketea hartu edo utzi behar duten jakingo du. Beso robotikoa gutxinaka gutxinaka plataforma mugikorrera hurbilduko da. Plataformaren paketeen eremura ailegatzen denean, paketea hartu/utzi egingo du `takePackage(take)` funtzioak bueltatutakoaren arabera eta, ondoren, `PrincipalPositon()`-era bueltatuko da.
- `timer()`: `StopTime` aldagiak zehazten duen denbora itxaroten duen metodoa.

- `calculatePosition()`: Esan dugun moduan, beso robotikoak paketeak biltegiratze-eremu ezberdinetara mugituko ditu. Biltegiratze-espazioaren arabera, oinarrizko motorrak (lehenengo serbomotorra) izan beharko dituen balioak ezberdinak izango dira. Oinarrizko motor honek izango dituen posizio ezberdinak 6.9 irudian ikusi daitezke. Funtzio honek, parametro moduan pasatzen zaion posizioaren arabera, lehenengo serbomotorrak hartu beharreko balioa bueltatuko du. 6.9 irudian ikusten dugun moduan, lehenengo serbomotorrak izango dituen balioak hiru biltegiratze-eremu taldeetan banatu daitezke.



6.9. Irudia: CalculatePosition() balio posibleak

Ondoren, beso robotikoaren hurrengo kode zatia ikusgai daukagu.

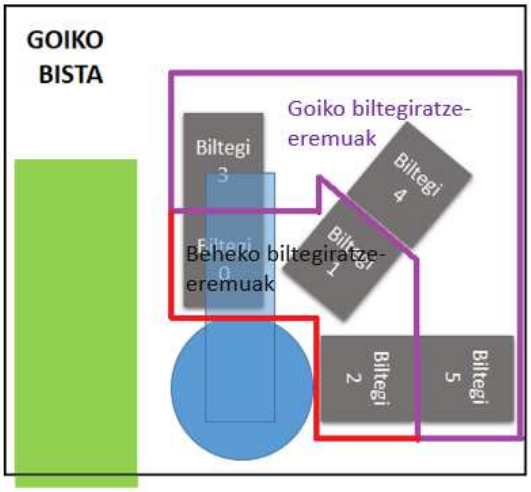
```
/* Beso robotikoa hurbilen dauden biltegiratzeko eremuetan pakete bat hartzeko edo
uzteko egin beharreko
mugimendu segidak

       In: int position -> Non utzi edo hartu behar den paketea, biltegiko zein
posizioan
              bool take       -> True: Pakete bat hartu behar da .
                                     False: Beso robotikoa daukan paketea utzi
behar da.
*/
void BesoRobotikoa::DownPackages(int position,boolean take){
       int pos= calculatePosition(position);
       int tak= takePackage(take);
       if (pos>0){
       timer(StopTime);
       Servo1.write(pos); // Eremu ezberdinerako aldatu behar da.
       Serial.println(Servo1.read());
       timer(StopTime);
       Servo3.write(90);
       Servo4.write(10);
       timer(StopTime);
       Servo3.write(110);
       timer(StopTime);
       Servo3.write(120);
       timer(StopTime);
       Servo3.write(130);
       timer(StopTime);
       Servo3.write(135);
       Servo4.write(25);
       timer(StopTime);
       Servo6.write(tak); // paketea hartu edo utzi behar da.
       timer(StopTime);
       if (take){
              Servo3.write(70);
              timer(StopTime);
       }else{
              Servo4.write(40);
              timer(StopTime);
```

```
                Servo3.write(70);
                timer(StopTime);
        }
        PrincipalPosition();
        }
        else{
                Serial.println("Emandako posizioa ez da existitzen");
        }
}
/* Beso robotikoa hurrunago dauden biltegiratzeko eremuetan pakete bat hartzeko edo
usteko egin beharreko
mugimendu segidak

        In: int position -> Non utzi edo hartu behar den paketea, biltegiko zein
posizioan
                bool take  -> True: Pakete bat hartu behar da .
                                        False: Beso robotikoa daukan paketea utzi
behar da.
*/
void BesoRobotikoa::UpPackages(int position,boolean take){
        int pos= calculatePosition(position);
        int tak= takePackage(take);
        if (pos>0){
        timer(StopTime);
        Servo1.write(pos); // Eremu ezberdinerako aldatu behar da.
        timer(StopTime);
        Servo2.write(80);
        Servo4.write(10);
        timer(StopTime);
        Servo2.write(70);
        timer(StopTime);
        Servo2.write(60);
        timer(StopTime);
        Servo2.write(55);
        Servo3.write(60);
        timer(StopTime);
        Servo2.write(50);
        Servo3.write(55);
        timer(StopTime);
        Servo2.write(46);
        Servo3.write(45);
        timer(StopTime);
        Servo6.write(tak); // paketea hartu edo utzi behar da.
        timer(StopTime);
        if (take){
                Servo3.write(70);
                timer(StopTime);
        }else{
                Servo4.write(40);
                timer(StopTime);
                Servo3.write(70);
                timer(StopTime);
        }
        PrincipalPosition();
        }
        else{
                Serial.println("Emandako posizioa ez da existitzen");
        }
}
```

6.21. Kodea: BesoRobotikoa.cpp-ren hirugarren atala

6.21. kodean ageri dira `BesoRobotikoa` klasean inplementatuta dauden beste bi metodo publikoak. Hauek biltegiratze-espazioetatik paketeak hartzeko/uzteko egin beharreko mugimendu segidak definitzen dituztenak dira. Biltegiratze-espazioak bi taldetan banatu ditugu. Beheko atalean aurkitzen diren biltegiratze-espazioak eta goian dauden biltegiratze-eremuak. Bi taldetan banatzea erabaki dugu, talde bakoitzaren biltegiratze-eremuetara egin beharreko hurbilpena berdinak direlako. Talde bakoitzeko kasu partikular bakoitzean, aldatzen den balio bakarra, lehen `calculatePosition()` funtzioan ikusi dugun lehenengo serbomotorraren balioa izango da. Talde ezberdinak modu grafikoan ikusteko 6.10 irudia daukagu.

- `DownPackages()`: Beheko taldean (0,1,2 biltegiratze-eremuak) aurkitzen diren paketeak, hartu/uzteko egin beharreko mugimendu-segidak prozesatzen dituen funtzioa da. Funtzioaren hasieran, `calculatePosition(position)` funtzioaren bitartez, lehengo serbomotorrak hartu beharreko posizioa zehaztuko da, biltegiratze-eremuaren arabera. Hasieran ere, `takePackage(take)` funtzioen bitartez paketea biltegiratze-eremuan utzi edo hartu behar dugun zehaztuko da. Behin egin beharrekoa zehaztuta, beso robotikoa mugimendu-segidak egitera pasako da, prozesua aurrera eramanez.
- `UpPackages()`: Goiko eremuan (3,4,5 biltegiratze-eremuak) kokatzen diren biltegiratze-eremuetatik paketeak hartzeko edo uzteko erabiliko den funtzioa. `DownPackages()` funtzioaren funtzionamendu berdina izango du, kasu honetan beso robotikoak egingo dituen mugimendu-segida ezberdinak dira, goiko posizioetara ailegatu ahal izateko.



6.10. Irudia: Goiko eta beheko bilteritza eremuen definizioak

Azkenik, orain arte ikusi ditugun funtzio guztiak elkartzen dituzten bi funtzioak aztertuko ditugu. Hauei, paketea zein posiziotik hartu edo utzi nahi den zehaztuz, prozesu guztia egiteaz arduratuko diren funtzioak dira. Hauek dira, Arduino Megak erabiliko dituen funtzioak. Bi funtzio hauen inplementazioa 6.22. kodean ikusi dezakegu.

```
* Pakete bat sisteman sartzean gordetzeko egin beharrekoa egiten duen metodoa.
Aurretik ikusi ditugun metodoak erabili
egiten ditu. Hau da, erabiltzaileak erabili egingo dutena.

-- In : int position -> Paketea gorde behar den biltegiratze espazioa
-- return: 0 -> Paketea ondo gorde bada
                -1 -> Parametro bidez pasatako posizioa ez bada exisititzen,
errorea*/
int BesoRobotikoa::SavePackage(int position){

        if (0<= position && position<3){
                PlataformPosition();
                PlataformCatch(true);
                DownPackages(position,false);
                return 0;
        }else if (3<= position && position<6){
                PlataformPosition();
                PlataformCatch(true);
                UpPackages(position,false);
                return 0;
        }else{
                Serial.println("Emandako posizioa ez da existitzen");
                return(-1);
        }

}
```

```
/*  Sisteman gordeta dagoen Pakete bat ateratzeko egin beharrekoa mugimenduak egiten
duen metodoa. Aurretik klasean zehar
definitu ditugun metodoak erabili egiten ditu. Hau da erabiltzaileak erabili egingo
duten metodoa.

-- In : int position -> Paketea gorde behar den biltegiratze espazioa
-- return: 0 -> Paketea ondo gorde bada
                -1 -> Parametro bidez pasatako posizioa ez bada exisititzen,
errorea*/
int BesoRobotikoa::ExtractPackage(int position){
        if (0<= position && position<3){
                DownPackages(position,true);
                PlataformPosition();
                PlataformCatch(false);
                return 0;
        }else if (3<= position && position<6){
                UpPackages(position,true);
                PlataformPosition();
                PlataformCatch(false);
                return 0;
        }else{
                Serial.println("Emandako posizioa ez da existitzen");
                return(-1);
        }
}
```

6.22. Kodea: BesoRobotikoa.cpp-ren laugarren atala

- `SavePackage()`: Beso robotikoak plataforma mugikorretik sartu den pakete bat hortik hartzeko eta dakogion biltegiratze-espazion uzteko egin behar dituen mugimendu-segidak definitzen ditu. Funtzio honi paketea gorde behar den biltegiratze-eremuaren posizioa pasatzen zaio. Funtzioaren exekuzioan bi kasu ezberdintzen dira. Lehen ikusi dugun moduan, paketea gorde behar den biltegiratze-eremua beheko posizioen artean badago, `DownPacakges()` funtzioari deituko zaio. Aldiz, paketea goiko posizioko biltegiratze-eremu batean gorde egin behar bada, `UpPacakges()` funtzioari deituko zaio.

- `ExtractPackage()`: Beso robotikoak sisteman gordeta dagoen pakete bat ateratzeko egin beharreko mugimendu-segidak definitzen ditu. Parametro moduan atera nahi den paketearen biltegiratze-eremua lortzen du. Honen bitartez, biltegiratze-eremu horretan aurkitzen den paketea hartu eta plataforma mugikorraren gainean utziko du, plataforma mugikorrak paketea atera ahal izateko. Berriro ere, `SavePackage()` funtzioan gertatzen den moduan, goiko eta beheko biltegiratze-espazioen arteko ezberdintasun argia ikusi daiteke.

Honekin, `BesoRobotiko` klasea azaltzeaz bukatu dugu. Bai Arduino UNO-n baita Arduino Megan ere biltegiak paketeak gordetzeko eta ateratzeko definitu ditugun hainbat klase eta metodo aztertu ditugu orain arte.

Jarraian, biltegiak paketeak sartzeko eta ateratzeko egin beharreko prozesu orokorra kontrolatzeko, inplementatu dugun `ServiceFunctions` liburutegia aztertuko dugu. Liburutegi honek, orain arte paketeen maneiuarentzako inplementatu diren bai `Biltegia` klasea baita `BesoRobotiko` klasea ere elkarlanean jarriko ditu. Liburutegi honetan ere plataforma mugikorra kontrolatzeko Arduino Megaren aldeko Bluetooth komunikazioa aztertuko dugu ere.

6.2.3 Biltegian paketeak sartzeko eta ateratzeko prozesu orokorra: ServiceFunctions liburutegia

Diseinuan azaldu dugun moduan, Arduino Mega web aplikazioarekin komunikatuko dena izango da. Arduino Megak helburu moduan komunikazio horretatik datozen aginduak asetzea dauka. Agindu hauetatik bi funtzio nagusi ondorioztatu daitezke. Bat erabiltzaileak sartu nahi duen paketeak gordetzea eta bestea, erabiltzaileak gordeta dauden paketeak ateratzea. Ondorioz,

horretarako aurrera eraman behar diren prozesuak, Arduino Megak modu kontrolatu eta antolatu batean prozesatu beharko ditu.

Arduino Megak prozesua kontrolatu ahal izateko, `ServiceFunctions` izeneko liburutegia sortu dugu. Honetan, paketeak sartu eta ateratzeko beharrezkoak diren prozesu guztiak (orain arte 6. kapituluan ikusi ditugun funtzioak) elkartzen dituzten bi funtzio definituta daude. Hauek `InsertPackage()` eta `ExtractPackage()` dira.

Bi funtzioek duten egitura eta funtzionamendua berdina izango da. Beraien funtzionamendua egoeren bitartez kontrolatu egingo da. Hasierako egoeratik, egoeraz egoera prozesatzen joango da funtzioa, azkenengora ailegatu arte, prozesuan errorerik egon ez bada. Funtzioak ez ditu zuzenean egoera guztiak prozesatuko, baizik eta funtzioari egoera bat prozesatzeko esango zaio. Egoera hori bukatzean hurrengo egoera zein den bueltatuko digu funtzioak. Errore bat badago, prozesua bertan behera utziko da, funtzioak errore kode bat bueltatuko du eta eskaera berri baten zain geratuko da sistema, prozesuak hasierako egoerara bueltatuz.

Beraz, etengabe egoera bakoitzak prozesatzeko funtzioari deitzen egon beharko gara. Horrela prozesuaren pausu bakoitza kontrolatu ahal izango dugu. Egoera egitura honek egoeren arteko askatasuna bermatzen du, prozesu guztia modularra bilakatuz. Modulu bakoitza beste guztietatik independentea izateak kodearen ulergarritasuna bermatzeaz aparte, erroreen kontrola errazten du ere. Hauen automatak B eranskinean aurkitu ditzakegu, B.1 eta B.2 irudietan.

Bi funtzio hauek aurretikan ikusi ditugun `Biltegia` eta `BesoRobotiko` klaseak erabiltzen dituzte. Gainera, plataforma mugikorraren funtzionamendua kontrolatzeko, Arduino UNO-rekin Bluetooth bidezko komunikazioa inplementatuta dago, plataformari aginduak bidali ahal izateko.

Bluetooth komunikazioa 6.1.6 atalan ikusi dugun protokoloaren bitartez gauzatuko da. Komunikazio hau posible izateko lagungarri diren funtzio batzuk definitu ditugu, `ServiceFunctions` liburutegian. Hauek 6.1.6 atalean Arduino UNO-ren Bluetooth komunikazioan ikusi ditugun berdinak dira eta, beraz, azalpen gehiagoren beharra ez dago. Hala ere, funtzio hauen inplementazioa 6.23. kodean ikus daiteke.

```
/* Bluetooth bitartesz datozen mezuak irakurtzeko erabili egingo dugun funtzioa
-- IN : String buf -> Mezua gorde egingo den bufferra
              int kop    -> Espero diren byte kopurua
*/
void Serial1read(String &buf,int kop){
  boolean recv=false;
  buf="";
  char character;
  while (!recv){
    if (Serial1.available()>kop){
      while (Serial1.available()){
        character = Serial1.read();
        buf = buf + String(character);
      }
      recv=true;
    }
  }
  recv=false;
}
/* Bluetooth bidez sartu nahi den paketearen id bidali nahi denean, egituratu behar
den
komandoa sortzeko erabiliko dugun funtzioa
--In : String id -> Sartu nahi den paketearen id-a
--return : String eran -> id:id osatutako komandoa
*/
String komandoaezarri(String &id){
       String eran;
       eran="id:"+id;
    return eran;
}
```

```
/* Bluettoh komunikaziotik id jaso behar denean, paketearen id jaso den mezutik
ateratzeko erabiliko dugun funtzioa
-- In : String buf -> Id-a atera nahi den mezua
--return : String  -> Paketearen id.
*/
String ateraparametroa(String buf){
  int index=buf.indexOf(":");
  return (buf.substring(index+1,11));
}
/*Id jasotzen denean bluetooth komunikaziotik, jasotzen den mezu horretatik bidali den
agindua atera
 --In: String buf-> agindua atera nahi den mezuaren buffera
 --return: String -> Jaso egin den agindua
 */
String aterakomandoa(String buf){
  int index=buf.indexOf(":");
  if (index>0){
    return (buf.substring(0,index));
  }else{
    return buf;
  }
}
```

6.23. Kodea: ServiceFunctions funtzio lagungarriak

Ikusi dezakegun moduan, 3.2.1.6 atalan ikusi dugun moduan Arduino UNO-n ez bezala honetan Bluetooth bitartezko komunikazioa egiteko Arduino Megak eskuragarri daukan `Serial1` hardware kanala erabili egingo dugu, software bitartezko serie-lerrorik erabili behar gabe.

Beraz, behin funtzio lagungarriak ikusi ditugula, bi funtzio nagusien funtzionamendua aztertzera pasatuko gara. Lehenik eta behin, erabili egingo diren egoeren definizioa egingo dugu, geroago beraien arteko erlazioa eta trantsizioak kodearen bitartez aztertuz.

6.2.3.1 InsertPackage()

Pakete berri bat sisteman sartzeko prozesuarekin hasiko gara. Horretarako, prozesu guztia hainbat egoeratan banatu dugu. Egoera bakoitzak pakete bat sisteman erregistratzeko eta gordetzeko beharrezko den prozesuaren atal bat beteko du. Definitu egin ditugun egoerak hauek dira (B.1 irudia):

- ST_OPEN: Pakete bat sartzerako momentuan, egin beharreko lehenengo gauza plataforma mugikorra irekitzea da; erabiltzaileak sartu nahi duen paketea bertan utzi ahal izateko. Hau egin baino lehenago, `Biltegia` klasearen bitartez, biltegian pakete berri baterako lekurik dagoen edo ez egiaztatuko da. Behin egiaztapen hori eginda, Bluetooth komunikazioren bitartez, `insert` komandoa bidali egingo zaio Arduino UNO-ri 6.1.6 atalan aztertu dugun komunikazio protokoloa jarraituz. Honekin plataforma irekitzera pasatuko da.
- ST_CLOSE: Plataforma mugikorra irekita dagoela, erabiltzaileak pakete bat gainean utzi beharko du. Erabiltzaileak paketea uzteko 10 segundu izango ditu. Orduan, erabiltzaileak paketea plataforman uzten badu plataforma itxi egingo da eta Bluetooth bitartez sartu den paketearen id-a bidaliko da. Beraz, egoera honetan, gure Arduino Mega sartu berri den paketearen id-a jasotzearen zain geratuko da. Hau jasotzerakoan, id-ak egitura egokia daukala eta id horrekin beste paketerik gordeta ez dagoela egiaztatuko da. Aldiz, 10 segundu horietan paketerik ez bada uzten, Arduino Megak Arduino UNO-tik errore mezua jasoko du, prozesua bertan behera utziz eta prozesua errore kode batekin bukatuz.
- ST_REGISTER: Paketea `Biltegia` klasean erregistratzera pasako gara eta honekin paketea gorde behar den biltegiratze-eremua lortuko dugu.

- ST_SAVE: Behin paketea biltegian erregistratu dela, eta bere biltegiratze-eremua aukeratu dela, beso robotikoak plataforma mugikorretik biltegiratze-espaziora mugituko du paketea.
- ST_DONE: Behin paketea dagokion biltegiratze-espazioan dagoela, prozesua bukatutzat emango dugu.

Beraz, hau izango da, Arduino Megak jarraitu beharreko prozesua. Orain, egoera bakoitzean azaldu dena, kodean nola inplementatu den aztertzera pasatuko gara.

Hasteko `InsertPackage()` hartzen dituen parametroak eta ST_OPEN den lehengo egoera aztertuko dugu. Horretarako inplementatu den kodea 6.24. kodean ageri da.

```
#define ST_OPEN       0
#define ST_CLOSE      1
#define ST_REGISTER   2
#define ST_SAVE       3
#define ST_DONE       4

/* Pakete bat sartzeko egin beharreko prozesu guztiak egingo dituen funtzioa. Funtzioa
egoera bitartez funtzionatuko du,
hau da, egoeraz egoera joango da, egin beharrekoak eginez. Prozesua egiten denaren
kontrola funtzioari deitzen dion programa izan beharko du.

--IN : int uneko-> Funtzioa dei honetan aurrera eraman behar duen egoera.
          int aurreko-> Funtzioak aurretikan betetako egoeraren identifikadorea
          String id-> Paketearen id gordetzeko erabili egingo den String aldagaia
          BesoRobotikoa besoa -> Beso robotikoa kontrolatzen duen klaseko objetua.
Honek beso robotikoari deituko baitio.
          Biltegia biltegia -> Biltegiaren okupazioa eramango duen Biltegiko biltegi
objektua

-- return : int hurrengoegoera -> Prozesuaren hurrengo egoeraren identifikadorea.
*/
int InsertPackage(int uneko,int aurreko, String &id, int &position,BesoRobotikoa
besoa, Biltegia &biltegia ){
String jaso,agindua;
int index;
       // Jaso den uneko egoeraren araberaren identifikadorearen arabera prozesu
ezberdinak definitu
       switch (uneko){
              // Plataforma mugikorra ireki behar da, paketea sisteman sartzeko
              case ST_OPEN:
                     if (aurreko== ST_OPEN){
                            // biltegia pakete berri baterako espazioa badauka
                            if (biltegia.emptySpace()){
                                   //Plataforma mugikorra kontrolatzen duen Arduino
UNO rekin Bluetooth komunikazioa hasi
                                   Serial1.write("insert");
                     // Keep reading from HC-05 and send to Arduino Serial Monitor
                                   Serial1read(jaso,1);
                                   // OK jaso bada  egoera honen prozesua bukatu
                                   if(jaso=="OK"){
                                   Serial.println("Dena ondo doa");
                            // TODO all the communication with the other board
                            Serial.println("Dena ondo joan da egoeran:  ST_OPEN\n");
                                          return ST_CLOSE;
                                   }else{
                            Serial.println("Jakin ez den erantzuna lortu da");
                                          return -2;
                                   }
                            }else{
                     Serial.println("Ez dago lekurik pakete berri batentzako");
                                   return -3;
                            }
                     }else{
                             Serial.println("Aurreko egoera ez da zuzena");
                             return -1;
                     }
              break;
```

6.24. Kodea: InsertPackage ST_OPEN egoera

`InsertPackage()` funtzioak 6.24. irudian ikusi daiteken moduan hainbat parametro jasotzen ditu.

- `Int Uneko`: Funtzioari deitzerakoan bete beharreko egoeraren identifikadorea.
- `Int Aurreko`: Uneko egoera baino lehenago egindako egoeraren identifikadorea.
- `String &id`: Gorde berri den paketearen `id` gordetzeko aldagaia.
- `Int &Posizioa`: Paketea zein biltegiratze-eremutan gorde behar den adieraziko duen aldagaia.
- `BesoRobotikoa besoa`: Beso robotikoa kontrolatzen duen `BesoRobotiko` klasearen objektua.
- `Biltegia &biltegia`: Biltegiaren paketeen kontrola eramango duen `Biltegia` klasearen objektua.

Kontuan izan behar dugu funtzioak osoko balio bat bueltatuko duela exekuzio bakoitzaren ostean. Exekuzio bakoitzean, egoera baten prozesua beteko da eta bueltatuko den balioa, dena ondo joan bada noski, hurrengo egoeraren identifikadorea izango da. Erroreren bat gertatuz gero, prozesuak errore kode bat bueltatuko du. Errore kode bakoitzaren esanahia 6.2.4 atalan wifi komunikazio protokoloaren erroreak ikustean aztertuko ditugu, hauek zuzenki erlazionatuta baitaude.

Beraz, behin ikusita zer nolako parametroak hartzen dituen funtzioak, egoera bakoitzean egiten dena ikustera pasatuko gara. Lehen esan dugun moduan, ST_OPEN egoerarekin hasiko gara. Honen inplementazioa 6.24. kodean ikusi daiteke. Bestalde pakete bat biltegian sartzeko definitu den Bluetooth komunikazioaren protokoloa 6.2 irudian daukagu.

- Hasteko, biltegian pakete berri batentzako espaziorik dagoen edo ez egiaztatuko da, `Biltegia` klaseko `emptySpace()` funtzioaren bitartez.
- Espazioa badago, plataforma mugikorrari `insert` komandoa bidaliko zaio Bluetooth moduluaren bitartez. Kasu honetan 5. kapituluan ikusi dugun moduan, Bluetooth komunikazioa Arduino Megak eskuragarri daukan serie-lerroko `Serial1` objektuaren bitartez egingo da. Plataforma mugikorrak agindua jaso ondoren irekitzen hasiko da, `OK` erantzuna bidaliz. `OK` hau Arduino Megak jasoko du eta hurrengo egoerarako prest egongo gara, ST_CLOSE egoeraren balioa bueltatuz.

6.25. kodean hurrengo egoerak izango direnen ST_CLOSE eta ST_REGISTER-en inplementazioak dauzkagu.

```
case ST_CLOSE:
   if (aurreko== ST_OPEN){
   // Bluetooth komunikaziotik sartu den paketaren ID jaso
        Serial1read(jaso,10);
        Serial.println(jaso);
       // jaso denatik id eta agindua atera
        agindua=aterakomandoa(jaso);
        id=ateraparametroa(jaso);
        if (agindua=="id"){
            if (id.length()== 8 && biltegia.savedPackage(id)<0){
              Serial.print("Sartu den pakete berriaren id: ");
              Serial.println(id);
            // Ondo jaso basda ID OK bidali ArduinoUNO-ri
              Serial1.write("OK");
             // Id horrekin jadanik paketeren bat gordeta dagoen edo ez egiaztatu
             // Honekin ArduinoUnorekin komunikazioa bukatutzat emango da
              Serial.println("Dena ondo joan da egoeran: ST_CLOSE");
              return ST_REGISTER;
            }else{
              Serial1.write("ER");
              delay(5000);
              Serial.println("id-a ez da ondo jaso\n");
```

```
            return -3;
          }
       }else{
           Serial.println("Ez da paketerik sartu barrura");
           Serial1.write("OK");
           return -4;
       }
  }else{
       Serial.println("Aurreko egoera ez da zuzena\n");
       return -1;
  }

              break;
case ST_REGISTER:
              // Paketean sisteman erregistratzera pasatuko gara, hau da , biltegi
klasean sartzera
              if (aurreko== ST_CLOSE){
// Aurreko egoeran jasotako id hori biltegian gordeko dugu, gorde behar den
espazioaren identifikadorea lortuz(position)
                     position=biltegia.savePackage(id);
                     if (position>=0){
                            Serial.println("Dena ondo joan da egoeran:
ST_REGISTER\n");
                            return ST_SAVE;
                     }else{
                            Serial.println("Errorea gertatu da paketea sartzean");
                            return -3;

                     }
              }else{
                      Serial.println("Aurreko egoera ez da zuzena");
                      return -1;
              }
       break;
```

6.25. Kodea: InsertPackage ST_CLOSE eta ST_REGISTER egoerak

Beraz, 6.25. kodean ikusi ahal dugun moduan ST_CLOSE egoeran emango diren pausuak hurrengoak dira:

- Lehenik eta behin, aurreko egoeratik gatozela egiaztatuko dugu. Horrela aurreko egoera ondo joan dela egiaztatuko dugu.
- Ondorioz, aurretikan ST_OPEN egoeran hasi dugun komunikazio horrekin jarraituko dugu. Kasu honetan, 6.2. irudian definitutako protokoloan definitu dugun moduan, plataforman erabiltzaileak utzitako paketearen id-aren zain geratuko gara.
    - Behin erabiltzaileak plataforman paketea utzi duela eta plataforma itxi dela, Arduino UNO-k Arduino Megari bidaliko dio id-a. Mezua jasotzean, bitan banatuko dugu mezua, `id` komandoa eta id bera den karaktere-katea. Identifikadore beraren egitura eta `id` komandoa ondo jaso direla egiaztatuko da, informazioa ondo ailegatu dela frogatzeko. Informazioa ondo ailegatu dela egiaztatzean, plataformari `OK` erantzuna bidaliko zaio. Jasotako id horrekin, biltegian beste paketerik gordeta ez dagoela egiaztatuko dugu. Horretarako `Biltegia` klaseko `savedpackage()` funtzioa erabiliko dugu. Aurreko egiaztapenetako baten bat gaizki badoa `ER` mezua bidaliko zaio Arduino UNO-ri eta prozesua errorearekin bukatuko da. Kontuan izan, Arduino UNO-k `ER` mezua jasotzean sartu berri den paketea aterako duela `PlataformExtract()` funtzioa erabiliz, erabiltzaileak paketea hartu dezan.
    - Aldiz, erabiltzaileak plataforman paketerik utzi ez badu eta horretarako ematen zaien 10 segunduko tartea bukatu bada, Arduino UNO-k `ER` mezua bidaliko dio Arduino Megari. Hau jasotzerakoan OK mezuarekin erantzungo dio Arduino UNO-ri eta prozesua -4 errore kodearekin bukatuko da.

- Egiaztapen guztiak ondo joan badira, prozesua bukatutzat emango da eta hurrengo egoera den ST_REGISTER egoeraren identifikadorea bueltatuko da.

Hurrengo egoera den ST_REGISTER-ek (6.25. kodea) hurrengo prozesuari jarraitzen dio:

- Hasteko, berriro ere, aurreko egoera egiaztatuko da.
- Ondoren, sartu berri den paketearen id-a biltegian erregistratzera pasatuko gara. Horretarako `Biltegia` klaseko `savePackage()` funtzioa erabiliko dugu. Honen bitartez, paketeari biltegiratze-eremu jakin bat atxikituko zaio, `position` aldagaian gordeko dena. Funtzioa bukatzen denean paketea biltegian erregistratuta geratuko da. Erregistroa egin eta gero, hurrengo egoerako identifikadorea bueltatuko da, kasu honetan ST_SAVE egoerarena.

Azkenengo bi egoeren, ST_SAVE eta ST_DONE egoeren inplementazioa 6.26. kodean ageri da.

```
case ST_SAVE:
                // Fisikoki paketea hartu plataformatik eta aurreko egoeran id gorde
egin den posizioari dagokion eremuan utzi paketea
        if (aurreko== ST_REGISTER){
                // Beso robotikoari agindua eman gordetzeko paketea posizio horretan
                if(besoa.SavePackage(position)==0){
                        Serial.println("Dena ondo joan da egoeran: ST_SAVE\n");
                        return ST_DONE;
                }else{
                        Serial.println("Besoaren posizioarekin zeozer gaizki jun da");
                        return -1;
                }
        }else{
                Serial.println("Aurreko egoera ez da zuzena");
                return -1;
        }
break;

case ST_DONE:
//Prozesua ondo ailegatu bada egoera honetara, prozesua ondo joan dela esango du.
Honetan ez da ezer egiten.
        if (aurreko== ST_SAVE){
                Serial.println("Dena ondo joan da egoeran: ST_DONE\n");
                return 5;
        }else{
                Serial.println("Aurreko egoera ez da zuzena");
                return -1;
        }

break;
}
}
```

6.26. Kodea: InsertPackage funtzioaren ST_SAVE eta ST_DONE egoerak

ST_SAVE egoera azaltzera pasatuko gara orain:

- Lehenik eta behin, aurreko egoera egiaztatuko dugu.
- Ondoren `BesoRobotikoa` klasearen `SavePackage()` funtzioaren bitartez, beso robotikoak paketea plataformatik aurreko egoeran kalkulatu den biltegiratze-eremura mugituko du. Biltegiratze-eremuaren posizioa lehen esan dugun moduan, `position` aldagaian gordeko da.
- Paketea esleitu zaion biltegiratze-eremuan dagoela, hurrengo egoera den ST_DONE-ren identifikadorea bueltatuko da egoera hau bukatuz.

Azkenik, prozesu guztia bukatuta dagoelarik eta errorerik egon ez bada, ST_DONE egoeran sartuko gara. Hau bakarrik erabiliko da, prozesua bukatu dela egiaztatzeko.

Nabarmendu behar da, `InsertPackage()` funtzioan zehar agertu diren errore kodeak, balio negatiboak, zuzenean erlazionatuta daudela aurrerago 6.2.4 atalan definitu dugun wifi bitartezko komunikazioan eman daitezkeen erroreekin eta beraz, esanahi berdina izango dute.

6.2.3.2 ExtractPackage()

Funtzio honek 6.2.3.1 atalan ikusi dugun `InsertPackage()` funtzioarekin parekotasun handia du, egitura eta funtzionamendu aldetik. Kasu honetan, gordeta dagoen pakete bat ateratzeko erabiliko da. Nahiz eta funtzio kontrajarriak izan, hainbat egoerek antzeko prozesuei jarraitzen diete. Egoera hauen inplementazioa ikusi baino lehen, hauen funtzionamendua aztertzera pasatuko gara (B.2 irudia):

- EX_SAVE: Lehenik eta behin, atera nahi den paketea biltegian gordeta dagoen edo ez egiaztatuko da. Paketea gordeta dagoela eta non gordeta dagoen dakigula, beso robotikoa paketea gordeta dagoen biltegiratze-eremutik hartu eta plataforma mugikorraren gainean utziko du.
- EX_OPEN: Paketea plataformaren gainean dagoela, plataforma mugikorrari Bluetooth bitartez `extract` agindua bidaliko zaio. Jarraian, atera nahi den paketearen id-a bidaliko zaio. Behin, plataforma mugikorrak Arduino Megaren mezuei `OK` batekin erantzutean plataforma irekitzen hasiko da.
- EX_CLOSE: Plataforma irekita egon ondoren plataforma ixtean Arduino UNO-k mezu bat bidaliko dio Arduino Megari, paketea ateratzerakoan gertatukoa adieraziz. Posible da erabiltzaileak paketea hartu izana, erabiltzaileak paketea hartu ez izana (kasu honetan, Arduino Megak atera nahi zen paketea dagokion biltegiratze-eremuan gordetzera pasatuko da) edota guk bidali diogun id-a eta plataformaren gainean dagoen paketearen id-a berdinak ez izatea. Jasotzen den mezua edozein izanda mezu horri `OK` batekin erantzungo zaio, plataformarekin komunikazioa itxiz.
- EX_REGISTER: Paketea ondo atera bada, `Biltegia` klasetik pakete hori ezabatzera pasatuko gara, paketea okupatzen zuen biltegiratze-eremua beste pakete batentzako libre geratuz.
- EX_DONE: Prozesua ondo bukatu dela adierazteko balio digun egoera.

Egoera bakoitzaren funtzionalitatea aztertu ondoren, hauen inplementazioa ikustera pasatuko gara. Kasu honetan, funtzioak hartzen dituen parametroak eta bueltatzen dituen balioak 6.2.3.1 atalan `InsertPackage()` funtzioa aztertzerakoan ikusi ditugun berdinak direnez, hauek ez ditugu berriro azalduko. Zuzenean egoerak aztertzera pasatuko gara. Horretarako 6.27. kodean EX_SAVE eta EX_OPEN egoeren inplementazioa ageri da.

```
case EX_SAVE:
       if (aurreko== EX_SAVE){
       // lehenik eta behin id horrekin paketerik gordeta dagoen edo ez egiaztatu
              position = biltegia.savedPackage(id);
              if (position>=0){
              //Horrela bada pakete hori plataforma mugikorrean uztera pasatuko da
              //beso robotikoari agindua emanez. Posizioa paketea gordeta zegoen edo
ez egiaztatzean lortu dugun posizioa izango da.
                     if(besoa.ExtractPackage(position)==0){
                            Serial.println("Dena ondo joan da egoeran: EX_SAVE\n");
                            return EX_OPEN;
                     }else{
                            Serial.println("Beso robotikoari emandako posizioa ez da
existitzen");
                            return -1;
                     }
              }else{
                     Serial.println("Pakete hori ez dago sisteman gordeta");
                     return -3;
```

```
                }
                // TODO all the communication with the other board
        }else{
                Serial.println("Aurreko egoera ez da zuzena");
                return -1;
        }
break;

// Plataforma pakete bat ateratzeko prozesua hasteko aginduko zaio
case EX_OPEN:
        if (aurreko== EX_SAVE){
                // Extract agindua bidali ArduinoUNO-ri bluetooth bidez
                Serial1.write("extract");
                jaso="";
                Serial1.flush();
                 // Erantzuna itxaron
                Serial1read(jaso,1);
                Serial1.flush();
                // Erantzuna ondo joan bada
                if(jaso=="OK"){
        // Atera nahi den paketearen id bidali Arduino Uno-k bere egiaztapenak egin
dezan
                        komandoaezarri(id).toCharArray(agin,15);
                        Serial.println(agin);
                        Serial1.write(agin);
                        // Erantzuna jaso
                        Serial1read(jaso,1);
                        if (jaso=="OK"){
                                Serial.println("Dena ondo doa");
                                Serial.println("Dena ondo joan da egoeran:  ST_OPEN");
                                return EX_CLOSE;
                        }
                        // id ez bada ondo ailegatu
                        else if (jaso=="ER"){
                                Serial.println("Ez da ondo bidali id-a");
                                Serial1.write("OK");
                                return -2;

                        // zeozer arraro ailegatzen bada
                        }else{
                                Serial.println("Jakin ez den erantzuna lortu da");
                                return -2;
                        }
                        //zeozer arraro ailegatzen bada
                }else{
                        Serial.println("Jakin ez den erantzuna lortu da");
                        return -2;
                }
        }else{
                Serial.println("Aurreko egoera ez da zuzena");
                return -1;
        }

break;
```

6.27. Kodea: ExtractPackage funtzioaren EX_SAVE eta EX_OPEN egoerak

EX_SAVE egoera:

- Hasteko, aurreko egoeraren egiaztapena egingo dugu. Kasu honetan, hasierako egoera denez, aurretikan beste egoerarik ez dira bete. Ondoren, egiaztatuko dena hasierako egoera den ala ez izango da.
- Ondoren, atera nahi den paketearen id-a `id` parametroaren bidez lortuko dugu. Ondorioz, atera nahi den paketea biltegian gordeta dagoen edo ez egiaztatuko da, `savedPackages()` funtzioaren bitartez.
- Biltegiratuta badago, `savedPackages()` funtzioa gordeta dagoen biltegiratze-eremuaren posizioa bueltatuko digu. Orduan, beso robotikoari eskatuko zaio paketea biltegiratze-eremutik hartzea eta plataforma mugikorraren gainean jartzea. Horretarako, `BesoRobotikoa` klasearen `ExtractPackage()` funtzioa erabiliko da, parametro moduan

paketea gordeta dagoen posizioa pasatuz. Paketea plataformaren gainean utzi denean, egoera bukatutzat emango da.

- Hurrengo egoera den EX_OPEN egoeraren identifikadorea bueltatuko da.

Hurrengo egoera EX_OPEN izango da, eta honako prozesua jarraituko du:

- Hasteko, aurreko egoera den EX_SAVE ondo bete dela egiaztatuko da.
- Ondoren, plataforma mugikorrarekin komunikazioa hasi egingo da, Bluetooth bitartez, `extract` agindua bidaliz. Honekin, pakete bat kanpora ateratzeko prozesua hasiko da plataforma mugikorrean.
- Plataforma mugikorrak agindua jaso ondoren, OK erantzuna bidaliko du. Behin erantzun hau Arduino Megak jasotzen duenean, atera nahi den paketearen id-a bidaliko zaio plataforma mugikorrari; beharrezko den `id:identifikadorea` formaturarekin.
- Arduino UNO-k id-a ondo jasotzen badu, OK erantzuna bidaliko dio Arduino Megari eta honek egoera bukatutzat emango du.
- Hurrengo egoera den EX_CLOSE identifikadorea bueltatuko du.

Hasierako bi egoera hauek prozesatu eta gero, EX_CLOSE eta EX_REGISTER egoerak bete beharko dira. Hauen inplementazioa 6.28. kodean ageri da.

```
// Behin erabiltzailea paketea hartu duenean edo denbora baten ondorioz plataforma
itxi egin dela kotntrolatuko dugu
case EX_CLOSE:
        if (aurreko== EX_OPEN){
        // Arduino UNO-ren erantzuna itxaron
                Serial1read(jaso,3);
        // PLataforma itxi egin dela esango du
                Serial.println(jaso);
                if (jaso=="done"){
                        Serial1.write("OK");
                        Serial.println("Dena ondo joan da egoeran: ST_CLOSE\n");
                        return EX_REGISTER;
                }
                // Paketea ez dauka pasa den id-a
                else if(jaso=="ERR1"){
                        Serial.println("Paketea ez da ondo atera");
                        Serial1.write("OK");
                        besoa.SavePackage(position);
                        return -3;
                // erabiltzaileak ez du paketea hartu
                }else if(jaso=="ERR2"){
                        Serial.println("Erabiltzailea ez du hartu paketea");
                        Serial1.write("OK");
                        besoa.SavePackage(position);
                        return -4;
                }else{
                        Serial.println("Jakin ez den erantzuna lortu da");
                        return -2;
                }
        }else{
                Serial.println("Aurreko egoera ez da zuzena");
                return -1;
        }
break;
// Atera berri den paketea sistemaren biltegiratzea kontrolatzen duen biltegitik kendu
beharko da
case EX_REGISTER:
        if (aurreko== EX_CLOSE){
        // Atera den paketearen id-a biltegitik ezabatuko dugu
                position=biltegia.removePackage(id);
                if (position>=0){
                        Serial.println("Dena ondo joan da egoeran: ST_REGISTER\n");
                        return EX_DONE;
                }else{
                        Serial.println("Paketea ez da exisititzen biltegian");
                        return -3;
```

```
                }
        }else{
                Serial.println("Aurreko egoera ez da zuzena");
                return -1;
        }
break;
```

6.28. Kodea: ExtractPackage funtzioaren EX_CLOSE eta EX_REGISTER egoerak

EX_OPEN egoeraren bitartez plataforma mugikorra ireki da, ondoren plataforma mugikorra ixterakoan lortu den emaitza prozesatzeko EX_CLOSE egoera daukagu.

- Hasteko, beti bezala aurreko egoeraren balioa ondo dagoela egiaztatuko da.
- Jarraian, plataforma mugikorrak Bluetooth bitartez bidaliko duen mezuaren zain geratuko gara. Ailegatzen den komandoaren arabera hiru egoera ezberdinak definitu ditugu.
    1. `done` komandoa jasotzen bada, erabiltzaileak atera den paketea hartu duela adieraziko du eta, beraz, paketea ateratzeko prozesua ondo joan dela ondorioztatu dezakegu. Honekin, komunikazioa bukatutzat emango dugu, `OK` erantzun bat bidaliz Arduino UNO-ri eta funtzioak hurrengo egoerako den EX_REGISTER balioa bueltatuko du, egoera bukatutzat emanez.
    2. Guk bidalitako id-a eta plataformaren gainean utzitako paketearen id-ak berdinak ez izatea; orduan `ERR1` mezua jasoko dugu. Ondoren, paketea gordeta zegoen biltegiratze-eremuan gorde egingo da, `BesoRobotikoa` klasearen `SavePackage()` funtzioa erabiliz. Honek, errore bat bueltatuko du.
    3. Plataforma ireki eta gero, denbora bat pasa ondoren erabiltzaileak ez badu paketea hartzen, berriro ere sistemara sartuko da. Hau gertatzen bada `ERR2` mezua jasoko dugu. Paketea gordeta zegoen biltegiratze-eremuan gorde egingo da, `BesoRobotikoa` klasearen `SavePackage()` funtzioa erabiliz. Honek, funtzioak errore balio bat bueltatzea ekarriko du.

Erabiltzaileak paketea hartu duenean, paketea sistematik ezabatzera pasatuko gara. Horretarako EX_REGISTER egoera daukagu.

- Hasi baino lehen, aurreko egoeraren egiaztapena egingo da.
- Egiaztapenaren ondoren, id horrekin gordeta zegoen paketea sistematik ezabatzera pasatuko gara. Horretarako, `Biltegia` klaseak eskaintzen duen `removePackage()` funtzioa erabiliko dugu. Funtzio honek sistematik pakete hori ezabatuko du, eta paketea gordeta zegoen biltegiratze-eremua beste pakete batentzako libre geratuko da.
- Ondorioz, egoera hau bukatutzat emango genuke eta hurrengo egoera den EX_DONE egoeraren identifikadorea bueltatuko luke.

Azkenik EX_DONE egoera daukagu. Honek 6.2.3.1 atalean ikusi dugun ST_DONE egoeraren helburu berdina izango du, egoerak parekoak izanda. Ondorioz, egoera honen inguruan azalpen gehiagorik ez dela behar ondorioztatu dugu.

Honekin, gure biltegiratze-sistemak fisikoki paketeak ateratzeko eta gordetzeko erabiltzen duen prozesua modu orokorrean ikusi dugu. Beraz, azalpenak bukatzeko, funtzio hauek egiteko agintzen dion web aplikazioarekin mantentzen duen komunikazioaren protokoloa eta Arduino Megan nola inplementatu dugun aztertuko dugu, aurrerago 8. kapituluan web aplikazioaren komunikazioa aldea aztertuz.

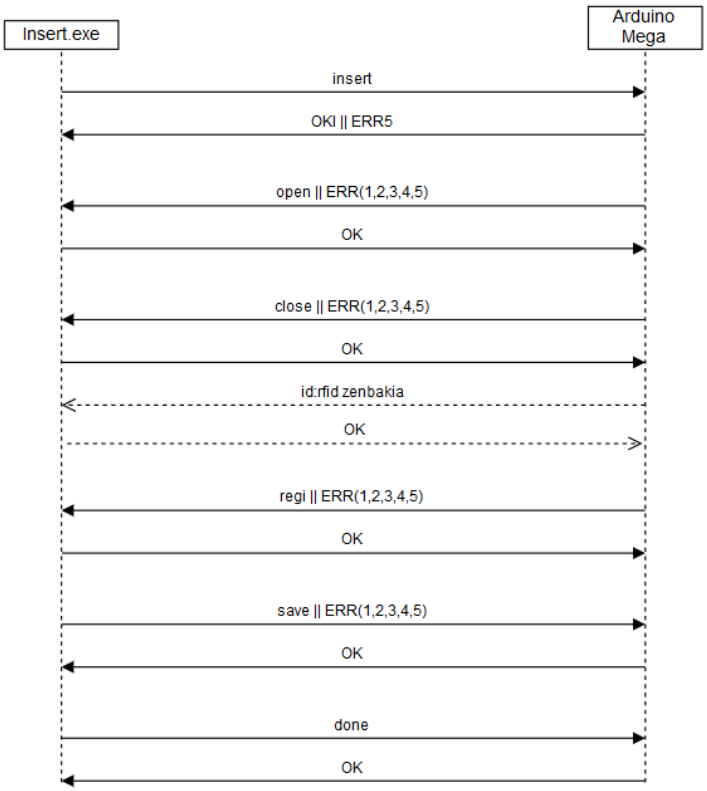
6.2.4 Wifi komunikazio protokoloa

Orain arte, Arduino Megak eta Arduino UNO-k betetzen dituzten funtzioak aztertu ditugu. Gainera, funtzio hauek nola inplementatu diren ikusi dugu, biltegiratze-sistemaren atal bakoitza modu zehatz batean azalduz.

Orain, sistema osoak web aplikazioarekin daukan komunikazioa aztertuko dugu. Komunikazio honetatik gure biltegiratze-sistemak aginduak jasoko ditu, eta jasotzen duen aginduaren arabera paketeak sartzeko eta ateratzeko prozesuak aurrera eramango ditu. Komunikazioa nola inplementatu dugun aztertu baino lehen wifi komunikaziorako definitu den komunikazio protokoloa aztertuko dugu.

6.2.4.1 Paketeak sartzeko protokoloa (Insert)

Beraz, pakete bat sartzerakoan erabili egingo dugun komunikazio protokoloa 6.11 irudian ikus dezakegu.



6.11. Irudia: Web aplikazioaren eta Arduino Megaren arteko komunikazioa protokoloa

Honako hauek izango dira gure komunikazioan egongo diren komandoak:

- `Insert`: Lehenik eta behin, web aplikazioak komunikazioa hasi egingo du, `insert` agindua Arduino Megari bidaliz. Honekin, Arduino Megari zer egin behar duen adieraziko dio. `insert` honi, Arduino Megak `OKI`-rekin erantzungo dio, dena ondo badoa. Erroreren bat badago `ERR5` errore mezuarekin erantzungo dio. Errore kodeak geroago aztertuko ditugu.

  Hasierako agindutik aurrera, komunikazioaren ekimena Arduino Megari pasatuko zaio, Arduino Mega komandoak bidaltzeaz arduratuko baita. Mezu hauek informazio gisa erabiliak izango dira, web aplikazioari biltegiratze-sisteman paketea sartzeko prozesuaren berri emateko.

- `Open`: Web aplikazioak plataforma mugikorra ireki dela ohartarazteko Arduino Megak `open` komandoa bidali egingo du. Plataforma irekitzean errore bat gertatzen bada `open` beharrean, errore kode bat bidali egingo dio. `Open` edo errorea jasotakoan web aplikazioak `OK` mezua bidali egingo dio. Erroreen kasuan, errore bat jasotzen den bakoitzean komunikazioa itxi egingo da eta web aplikazioak prozesua bukatutzat emango du, errore balioa bueltatuz.
- `Clos`: Plataforma mugikorra ixten denean, erabiltzaileak utzitako pakete berriarekin, `clos` komandoa bidali egingo dio Arduino Megak web aplikazioari. Prozesu horretan, errore bat egon bada, errore kode bat bidaliko da. Bi kasu hauei web aplikazioak `OK` mezu batekin erantzungo dio.
    - `clos` komandoa ondo jaso bada, jarraian Arduino Megak sartu berri den paketearen id-a bidali egingo dio "id:zenbakia" formatuarekin. Web aplikazioak id jaso ondoren `OK` erantzuna bidaliko dio Arduino Megari.
- `Regi`: Sartu berri den paketea, biltegiratze-sisteman erregistratu ondoren, Arduino Megak `regi` komandoa bidaliko dio web aplikazioari. Errore bat badago, errore kode bat bidaliko zaio. Web aplikazioak `OK` erantzungo dio.
- `Save`: Paketea sisteman erregistratu ondoren, sistemak paketea dagokion biltegiratze-eremuan gordetzera pasatuko da. Paketea ondo gorde bada, `save` komandoa bidaliko dio web aplikazioari, aldiz, errore bat egon bada errore kode bat bidaliko dio. Web aplikazioak `OK` erantzungo dio.
- `Done`: Azkenik, prozesua bukatutzat emateko `done` komandoa bidaliko dio Arduino Megak web aplikazioari. Honek `OK` erantzutean, komunikazioa itxi egingo da, web aplikazioan emaitza bistaratuz eta Arduino Megak eskaera berri baten zain geratuz.

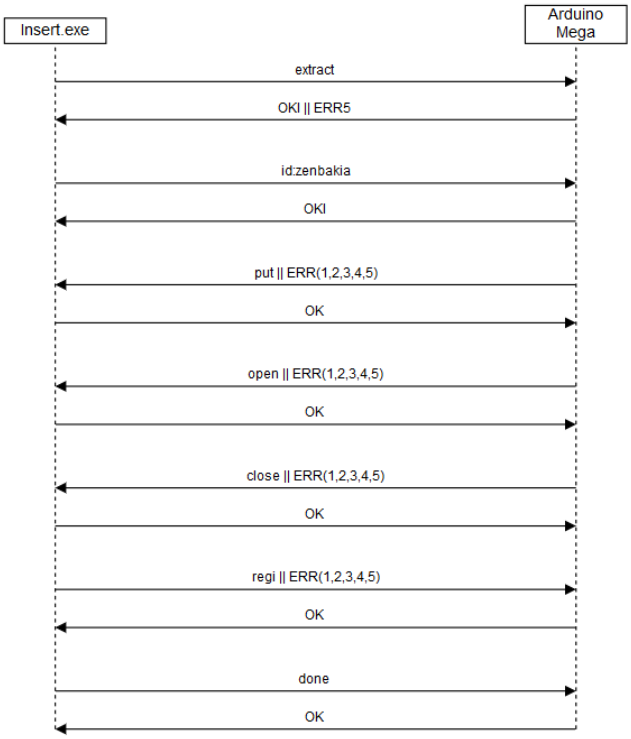
Behin komandoak aztertu ditugula, komunikazioan zehar jaso daitezkeen erroreak aztertzera pasatuko gara.

- `ERR1`: Arduino Megak paketea sartzeko prozesuaren momentu batean software errore bat egon dela adieraziko du.
- `ERR2`: Arduino Mega eta Arduino UNO (plataforma mugikorra kontrolatzen duena) arteko Bluetooth komunikazioan errore bat egon bada.
- `ERR3`: Sistemaren barruan sartutako paketeak ez dauka id balioduna. `Id`-ak formatu egokia ez daukalako edota jadanik sisteman id hori daukan beste pakete bat exisitizen delako. Sisteman pakete gehiago kabitzen ez direla adierazi dezake ere.
- `ERR4`: Erabiltzaileak ez badu pakete bat sartu.
- `ERR5`: Web aplikazioa eta Arduino arteko socket komunikazioan agindu ezezaguna ailegatzen bada.

Lehen 6.2.3 atalan `ServiceFuntions` liburutegiaren `InsertPackage()` eta `ExtractPackage()` funtzioak aztertzean, hauek errore kode batzuk bueltatzen zituztela ikusi dugu. Hauek ikusi berri ditugun errore kodeen esanahiekin zuzenki erlazionatuta daude. Hau da, `InsertPackage()` funtzioan -4 balioa bueltatzen denean, wifi komunikazioaren ERR4 kodearekin zuzenki erlazionatuta dago.

6.2.4.2 *Paketeak ateratzeko protokoloa (Extract)*

Orain pakete bat ateratzeko prozesua aztertzera pasatuko gara. Erabili egiten den komunikazio protokoloa 6.12 irudian ikusi dugunaren antzekoa izango da.



6.12. Irudia: Web aplikazioaren eta Arduino Megaren arteko komunikazio protokoloa

Kasu honetan id-a bidaltzen duena Arduino izan beharrean web aplikazioa izango da.

- `Extract` eta `id`: Hasteko web aplikazioak `extract` eta `id`-a bidaliko dizkio Arduino Megari. Arduinok mezu bakoitzari `OKI` batekin erantzungo dio. Hemendik aurrera, aurrekoan gertatzen den moduan komunikazioaren ekimena Arduino Megara pasatuko da. Bi komando hauekin biltegiratze-sistemak, sistematik zein pakete atera behar duen jakingo du.
- `Put`: Paketea sisteman dagoen edo ez bilatuko du Arduino Megak. Atera nahi den paketea sisteman badago, beso robotikoak gordeta dagoen biltegiratze-eremutik hartu eta plataforma mugikorraren gainean jarriko du. Hau egin ondoren `put` komandoa bidaliko dio web aplikazioari. Arduino Megak prozesuan zehar erroreak aurkitzen baditu, dagokion errore kodea bidaliko dio web aplikazioari. Web aplikazioak `OK` mezuarekin erantzungo dio.
- `Open`: Plataforma mugikorra ireki eta gero `open` komandoa bidali egingo du Arduinok. Errore bat badago, berriro ere komandoa ez ezik errore kode bat bidaliko dio. Honi berriro ere web aplikazioak `OK` batekin erantzungo dio.
- `Clos`: Plataforma itxi eta erabiltzaileak paketea hartu duela egiaztatzean `clos` komandoa bidaliko dio. Prozesuan errore bat badago, komandoa ez ezik errore kode bat bidaliko dio. Web aplikazioak `OK` erantzungo dio.
- `Regi`: Paketea biltegiratze-sistematik ezabatuko da eta `regi` mezua bidali egingo dio. Errore bat badago, berriro ere komandoa ez ezik errore kode bat bidaliko zaio. Honi berriro ere web aplikazioak `OK` erantzungo dio.

- `Done`: Atera nahi zen paketea ondo atera dela adierazteko komandoa. Honekin komunikazioa bukatutzat emango da.

Orain errore kodeak aztertzera pasatuko gara. Esan beharra dago, aurreko kasuaren antzekoak direla, baina ezberdintasun txikiak daudenez hauek aztertuko ditugu.

- ERR1: Arduino Megak pakete bat ateratzeko prozesuan zehar software errore bat gertatu dela adieraziko du.
- ERR2: Arduino Mega eta Arduino UNO (plataforma mugikorra kontrolatzen duena) arteko Bluetooth komunikazioan errore bat egon da.
- ERR3: Aplikazioak bidalitako id-a eta plataformaren gainean jarritako paketearen id-ak ezberdinak dira edota pakete honek daukan id-a sisteman ez bada existitzen.
- ERR4: Erabiltzaileak ez badu paketea hartu.
- ERR5: Web aplikazioa eta Arduino arteko socket komunikazioan agindu ezezaguna ailegatzen bada.

6.2.5 Wifi komunikazio inplementazioa (Arduino Megaren begizta nagusia)

Orain, 6.2.4 atalan aztertu dugun komunikazio protokoloak gure Arduino Megaren kodean nola inplementatu ditugun aztertuko dugu. Honekin, gure biltegiratze-sistemaren funtzionamendu osoa modu orokorrean ulertuko dugu.

6.2.5.1 Definitutako aldagaiak eta objektuak

Lehenik eta behin, geroago kodea ulergarriago suertatzeko, gure programak definituta dituen aldagai eta objektuak azaletik aztertuko ditugu.

```
#include “ServiceFunctions.h”
#include “Biltegia.h”
#include “BesoRobotikoa.h”
#include <SPI.h>
#include <WiFi.h>
#include <EepromUtil.h>
#include <EEPROM.h>
// Aldagai batzuen definizioak
#define MAX_BUF 1024
#define PORT 23
#define ARRAYSIZE 15
// Moduen definizioa
#define MOD_INST 0
#define MOD_EXT  1

String MODE [] = {”insert”,”extract”};
String KOMANDOAK [] = {”open”,”clos”,”regi”,”save”,”done”}; // insert
String KOMANDOAK1 [] = {”put”,”open”,”clos”,”regi”,”done”}; // extract
String err[] = {”ERR1”,”ERR2”,”ERR3”,”ERR4”,"ERR5"}; // Errore kodeak
char ok[] = “OKI”; // Komunikazioko erantzun zuzena

char ssid[] = “ea16d6”; //  your network SSID (name)
char pass[] = “····”;    // your network password (use for WPA, or use as key for WEP)

char aux[ARRAYSIZE],idr[ARRAYSIZE],error[ARRAYSIZE]; // Char array lagungarriak
String buf,eran,id; // Datuak gordetzeko erabiliko diren String-ak

/* Gure biltegia eta beso robotikoa kontrolatuko duten objektuak*/
Biltegia biltegia;
BesoRobotikoa besorob;
// Memoriaren paketeak gordetzen hasten den memoria helbidea, eta biltegiratze maximoa
const int HasMemory = 12;
const int MAX_BILTEGIA = 6;
// Komunikazioarako erabilgarriak, zein agindu jaso eta zein komando bidaltzeko
erabiliak
```

```
int egoera,command,er;
// Insert eta ExtractPackage() funtzioen egoerak kontrolatzeko aldagaiak
int aurrekoegoera,unekoegoera,hurrengoegoera,pos;
// Software zehar dauden loop ezberdinen kontrola eramateko erabilgarriak
boolean entry = true;
boolean attempting=false;
boolean finded=false;
boolean finish=false;
boolean prozesua=false;
boolean conn=false;
// Wifi konexioarako beharrezkoak diren objektuak
int status = WL_IDLE_STATUS;
WiFiServer server(23);
WiFiClient client;
```

6.29. Kodea: ArduinoMegaren aldagai eta objektu definizioak

6.29 kodean ikusi ditzakegu definitu diren aldagai eta objektu ezberdinak. Hauek bloke ezberdinetan banatu ditugu, beraien erabileraren arabera.

- Hasteko, softwarearen funtzionamendurako beharrezkoak diren liburutegiak adierazi ditugu. Ikusten dugun moduan, aurretikan 6.2 atalan ikusi ditugun liburutegi guztiak sartuko ditugu (`Biltegia` eta `BesoRobotikoa`). Hauetaz aparte, Arduinok eskuragarri dituen `<SPI>` eta `<Wifi>` liburutegiak behar dira, wifi-ra konektatzeko eta wifi socketaren kontrolarako beharrezkoak direlako. Gainera, Arduino Megaren memoria erabili ahal izateko `<EEPROM>` eta `<EepromUti>` dauzkagu.
- Gero `#define` batzuk txertatu ditugu, kodea ulergarriago izateko. Adibidez, gure biltegiratze-sistemak izango dituen moduen artean ezberdintzeko MOD_ definizioak ditugu. Honen bitartez, paketea ateratzeko edo sartzeko prozesuan gauden edo ez zehaztuko dugu.
- Honen ondorioz, prozesuan zehar erabili egingo diren `String`, `Char` eta `Array`-ak definitu ditugu. Esan beharra dago, karaktere-kateak gordetzeko eta maneiatzeko Arduinok eskaintzen duen `String` klasea erabiltzea erabaki dela. Hala ere, funtzio batzuek karaktere datu egitura eskatzen dute eta horregatik batzuk definitu behar izan ditugu. Beraz, `String` eta karaktere-kateen arteko eraldaketa asko izango ditugu kodean zehar.

  Alde batetik, gure sistemak web aplikaziotik jaso ditzakeen bi aginduak definituta ditugu `MODE[]` taulan. Beste lau taula ditugu. `KOMANDOAK[]`, `KOMANDOA1[]`, `err[]` eta `ok[]`. Hauek wifi komunikazio protokoloan definitu ditugun mezuak, erroreak eta erantzunak gordetzen dituzten taulak dira.

  Azkenik erabili egingo ditugun karaktere-kateak ditugu. `ssid` eta `pass` wifi konexioa ezartzeko beharrezkoak dira. Hauek konektatu behar garen sarearen izena eta pasahitza gordetzen baitute.

  Besteak, wifi komunikazioan erabili egingo diren aldagaiak dira. Nabarmendu `buf` karaktere-katean komunikazioan jasoko diren mezuak gordeko direla eta `id` karaktere-katean, berriz, sartu edo atera nahi den paketearen id-a.
- Sistemarentzeko ezinbestekoak diren `Biltegia` eta `BesoRobotikoa` objektuak. Hauek biltegiaren antolamendurako eta paketeak sistematik mugiarazteko beharrezkoak dira.
- Gero osoko aldagaiak ditugu. Hauek normalean, egoerak definitzeko edo taulen posizioak definitzeko erabiliko dira.
- Aldagai boolearrak, softwarean zehar ditugun begizta ezberdinen kontrola eramateko erabiliko dira.
- Azkenik, wifi komunikaziorako beharrezkoak diren `client` eta `server` objektuak definitu ditugu. Hauek socket komunikazioarako beharrezkoak dira. Komunikazioan, zerbitzaria

Arduino izango da eta bezeroa, berriz, web aplikazioa. Azkenik `status` aldagaia, wifi shield-a dagoen egoera gordetzeko erabiliko da.

Definitu ditugun aldagaiak alde batera utzita, Arduino Mega piztean egin beharreko konfigurazioa eta hasieraketa ezberdinak ikusiko ditugu. Hauek 6.30. kodean ditugu.

```
void setup() {
  //Hasieratu serie komunikazio guztiak.
  Serial.begin(9600);
  Serial1.begin(9600);
  besorob.Attach();
  besorob.StartPosition();
  PackageMemory(biltegia);
  // Wifi shield-a konektatuta dagoen edo ez egiaztatu
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    while(true);
  }
  connectWifi(conn);
 }

/* Arduino Megak pizten den bakoitzean, biltegian dagoen egoera kargatzeko balio digun
funtzioa. Honek EEPROM memorian gordeta dauden paketeen informazioa
 *  biltegia objektura pasatuko ditu, horrela Arduino itzaltzen bada gordeta dauden
paketeen informazioa ez da galduko. Honek, lehenik eta behin zenbat pakete gordeta
 *  dauden begiratuko du. Behin paketerik gordeta dagoen edo ez dakienean, orduan
pakete hauek zein biltegiratze espazioan aurkitzen diren bilatuko du. Paketeak
aurkitzean
 * diren biltegiratze eremuan biltegiko informazioa eguneratuko da, memorian dagoen
informazioarekin.
 * -- In: Paketeen administrazioa eremango duen biltegia objektua, hona EEPROM
memorian gordeta dagoen informazioa pasatuko da.
 */
void PackageMemory(Biltegia &biltegia) {
  int idatzita,gordeta,erabilita;
  int kont=0;
  char id[8];
  String rfid;
  EepromUtil::eeprom_read_int((HasMemory * 10),&gordeta);
  if (gordeta>0){
   for (int i=0;i<MAX_BILTEGIA;i++){
      EepromUtil::eeprom_read_int(i*2,&erabilita);
      if (erabilita==1){
        kont++;
        EepromUtil::eeprom_read_string((HasMemory+(8*i)),id,9);
        rfid=id;
        biltegia.SetValues(i,rfid);
        if (kont==gordeta)
          break;
      }
    }
  }
}
```

6.30. Kodea: ArduinoMega setup() eta PackageMemory()

Hasteko, komunikaziorako erabiliko diren bi serie-lerroak hasieratuko ditugu 9600 baud abiaduran. `Serial` Arduinok ordenagailuarekin izango duen komunikaziorako izango da. Honela Arduinoaren exekuzioa nola doan jakingo dugu, kodean zehar dauden inprimaketen bitartez. `Serial1` berriz, bluetooth bitartezko komunikazioa egiteko erabiliko da, 5.1. taulan ikusi dugun moduan TX1 eta RX1 pinak erabiliz.

Beso robotikoa kontrolatzen duen `besorob` objektuko `Attach()` eta `StartPosition()` funtzioak erabiliko dira beso robotikoa kontrolatzen hasteko eta hasierako posizioa hartu dezan.

Sortu dugun `Biltegia biltegia` objektuan EEPROM memorian sisteman aurretikan gordeta dauden paketeen informazioa idazteko `PackageMemory()` funtzioa daukagu.

- `PackageMemory()`: Hasteko 120. Helbidean, sisteman gordeta dauden paketeen kopurua irakurriko dugu, `gordeta` aldagaian gordez. Hemendik aurrera paketeak zein posiziotan gordeta dauden bilatuko dugu EEPROM memorian zehar. Paketeak aurkitzen ditugunean hauen balioa eta posizioa `biltegia` objektuan ezarriko dugu klase honen `SetValues()` funtzioa erabiliz. Behin gordeta dauden pakete guztiak aurkitu ditugula, funtzioa bukatuko da. Ondorioz, funtzioaren helburu nagusia, sisteman aurretikan gordeta dauden paketeen informazioa, prozesuan zehar paketeen administraziorako erabiliko den `biltegia` objektuan idaztea da. Horrela, gordeta dagoen informazioa ez da galduko, Arduinoa itzali eta piztean.

Azkenik, Arduino Megak wifi shield-a konektatuta daukala egiaztatuko du. Arduinok ez du aurrera jarraituko shield-a konektatu arte. Ondoren, `connectWifi()` funtzioa erabiliko da Arduino Megak guk zehaztutako wifi sarera konektatzeko. Honen funtzionamendua 6.2.5.2 atalan ikusiko dugu.

6.2.5.2 Wifi konexioa

Aldagai guztiak eta konfigurazioa aztertu ditugula, Arduino Megaren exekuzioarekin jarriko gara. Gure Arduino Megak egingo duen lehenengo gauza aukeratu dugun wifi sarera konektatzea izango da. Geroago sare berdinera konektatuta egongo den web aplikazioarekin komunikatzeko. Horretarako lau funtzio definitu ditugu, 6.31 Kodean ikusi daitezkenak.

```
/* Gure Arduino Megak guk definitu diogun sarera konektatzeko definitu dugun funtzioa.
Funtzio honek, jadanik sarera konektatuta gaudenean ez du ezer egingo
 *  ,berriz ez bagaude konektatuta konektatu nahi garen sarea eskuragarri dagoen edo
ez bilatuko du. eskuragarri badago sare horretara konektatuko da eta
 *  server moduan hasieratu egingo da arduino megak, geroago konexioak hartu ahal
izateko.
 * -- IN: boolean conn : Konektatuta gauden edo ez jakiteko erabiliko dugun booleanoa,
funtzioan sartzen garenean beti false moduan egon beharko da, client
 *                          berri baten zain geratzen garen bakoitzean dauakgun konexioa
egiaztatu egingo da.
 */
void connectWifi(boolean &conn){
  while (!conn){
        listNetworks();
        if (finded==true){
        // attempt to connect to Wifi network:
          while ( status != WL_CONNECTED) {
            if (!attempting){// wait until the arduino connects to the wifi
            Serial.print("Konexioa ezartzen sahitzen ari gara SSID: ");
            Serial.println(ssid);
            status = WiFi.begin(ssid,pass);
            attempting=true;
            }
          }
          attempting=false;
          // start the server:
          server.begin();
          Serial.println("Konektatuta zaude orain");
          // you're connected now, so print out the status:
          printWifiStatus();
          conn=true;
        }else{
         Serial.println("Konektatu nahi zaren sarea ez dago eskuragarri");
         timer(10000);
        }
  }
}

void software_Reset()
{
    if ( WiFi.status()!= WL_CONNECTED ){
      Serial.println("Ez dago konexiorik");
      timer(60000);
      asm volatile ("  jmp 0");
    }
```

```
}
/* Momentuan konektatuta gauden sareak Arduino Mega-ri esleitu dion IP helbide
pribatua. Honekin web aplikazioak zein ip helbidera
 *  egin behar dizkion eskaerak jakingo dugu, hau sare bat konektatu egiten garen
bakoitzean deituko zaio.
 *
 */
void printWifiStatus() {
  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);
}
/* Gure wifi shield-ak eskuragarri dauden sareen artean guk bilatzen dugun sarea
dagoen edo ez zehaztuko digun funtzioa
 *  Aldatze duen aldagaia, guk definitutako finded aldagai globala da.
 *
 */
void listNetworks() {
  // scan for nearby networks:
  //Serial.println("** Scan Networks **");
  int numSsid = WiFi.scanNetworks();
  if (numSsid == -1) {
    Serial.println("Ezin dira sarerik aurkitu");
    while (true);
  }
  // print the network number and name for each network found:
  for (int thisNet = 0; thisNet < numSsid; thisNet++) {
    if (strncmp(WiFi.SSID(thisNet),ssid,15) == 0){
      finded=true;
      return;
    }
  }
  finded=false;
}
```

6.31. Kodea: ArduinoMega-ren Wifi konexiorako funtzioak

- `printWifiStatus()`: Gure Arduino shield-a sare batera konektatuta dagoenean, sare horretan daukan IP-a inprimitzeko erabiliko den funtzioa.
- `listNetworks()`: Momentu horretan eskuragarri dauden sare guztietatik, Arduino bilatzen duena eskuragarri dagoen edo ez zehaztuko duen funtzioa.
- `connectWifi()`: Aurreko bi funtzioak erabili egiten dituen funtzio nagusia. Honen bitartez, guk nahi dugun sarera konektatuko da Arduino. Hasteko, jadanik sare batera konektatuta gauden edo ez egiaztatuko da. Jadanik konektatuta bagaude, funtziotik aterako gara, bestela konektatu nahi garen sarea bilatuko da `listNetworks()` funtzioaren bitartez. Sarea eskuragarri badago, konektatu egingo gara, socket zerbitzaria hasieratuz. Aldiz, eskuragarri dauden sareen artean gurea ez badago, hamar segundo itxaron eta gero, konektatu nahi garen sarea berriro bilatuko dugu. Arduinok horrela jarraituko du, definitutako sarea aurkitzen duen arte.
- `software_Reset()`: Behin sarera konektatu ondoren, Arduinok edozein prozesu hasi baino lehen sarera konektatuta dagoen edo ez egiaztatzeko balio digun funtzioa. Konexioa galdu egiten bada, funtzio honen bitartez software bidezko berrabiarazketa gertatuko da. Honela, Arduino berriro konfiguratu egingo da eta berriro sarera konektatzen saiatuko da. Hau beharrezkoa da, `client` eta `server` objektuak modu egoki batean hasieratu ahal izateko eta web aplikazioarekin konexioa ez galtzeko.

6.2.5.3 Arduino Mega: Begizta nagusia

Web aplikaziorekin egongo den komunikazioa azaltzen hasi baino lehen, hainbat xehetasun argitu behar ditugu.

Komunikazioaren zerbitzariak, hau da, Arduino Megak, zerbait bidaltzean, bidalketa ondo egin den edo ez egiaztatuko du. Bidalketak `write()` funtzioaren bitartez egingo dira. Honek bidali diren byte kopurua itzuliko du. Edozein momentutan bidalketa horietan erroreren bat gertatzen bada, prozesua bertan behera geratuko da, hasieratu beharreko aldagaiak hasieratuz eta komunikazioa itxiz.

Bestetik, web aplikaziotik erantzun bat espero bada, Arduino ez da etengabe erantzun horren zain geratuko, edozein gauzagatik web aplikazioak funtzionatzeari utzi diezaiokeelako, komunikazioa bertan behera geratuz. Ondoren, Arduino Mega blokeatuta ez geratzeko, tenporizadore bat jartzea erabaki da, zentzuzko denbora batez zain egoteko. Beraz, denbora hau pasatzen bada ezer jaso gabe, aldagaiak hasieratu, komunikazioa itxi eta eskaera berri baten zain geratuko da sistemak.

Web aplikazioren eta Arduino Megaren arteko wifi bidezko komunikaziorako funtzio lagungarri batzuk definitu ditugu (6.32. kodean).

```
/*
* ‘string’ parametroa ‘string_zerr’ bektoreko karaktere kateetako batetik hasten den
egiaztatzen du. ‘string_zerr’ bektoreko azkeneko elementua NULL izan behar da.
* ‘string’ parametro karaktere katearen hasierarekin bat egiten duen ‘string_zerr’
bektoreko lehenego karaktere katearen indizea itzuliko du. Ez bada bat-egiterik egon
balio negatibo bat itzuliko du.
*/
int search_command(String buf, String *komandoak)
{
  for(int i=0; i < ARRAYSIZE; i++){
    if(buf == komandoak[i])
      return i;
  }
  return -1;
}
/* Komunikazioaren edozein momentuan erroreren bat egon bada, bidali egin behar den
errore kodea bidaliko da. KOmunikazioaren arazoen errore kodea ERR5
*/
void unexpected ()
{
  err[4].toCharArray(error,ARRAYSIZE);
  server.write(err);
}
/* Wifi socketaren bitartesz datozen mezuak irakurtzeko erabili egingo dugun funtzioa
 *  denboragailuarekin.
-- IN : String buf -> Mezua gorde egingo den bufferra*/
int Socketread(String &buf){
  buf="";
  boolean recv =false;
  char character;
  long cur = millis();
  long pre = millis();
  long interval = 0;
  while (!recv && interval<20000){
    if (client.available()>1){
      while (client.available()){
        character = client.read();
        buf = buf + String(character);
      }
      recv=true;
    }
    cur=millis();
    interval=cur-pre;
  }
  recv=false;
  client.flush();
  if (interval<20000){
    return 0;
  }else{
   return -1;
  }
}
/* InsertPackage() eta ExtractPackage() funtzioak buletatu ditzakeen errore kodeak,
daogkion socketaren
```

```
 *  errore mezura bilaktu egingo duen funtzioak. Funtzioen eta errore mezuen arteko
erlazioa zuzena da,
 *  ikusi dezakegun moduan.
 *  --IN: int errorea: Biltegiaren funtzionamendurako definitutako funtzioak bueltatu
ditzakeen errore kodeak
 *  --OUT Strign errorea: Errore kode horri dagokoin errore mezua bueltatuko du.
 */
String searcherror(int errorea){
  switch (errorea){
    case -1:
      return “ERR1”;
    break;

    case -2:
      return “ERR2”;
    break;

    case -3:
      return “ERR3”;
    break;

    case -4:
      return “ERR4”;
    break;
  }
}

/* timer bat, itxoin dezan sartzen diren milisegunduak
--In : long interval -> Itxoin behar diren milisegunduak
*/
void timer(long interval){
  unsigned long currentMillis,previousMillis;
  currentMillis = millis();
  previousMillis = millis();
  while (currentMillis-previousMillis<interval){
      currentMillis=millis();
  }
}
```

6.32. Kodea: ArduinoMega-ren funtzio lagungarriak

- `Search_command()`*:* Jasotzen diren mezuak protokoloan definituta dauden aginduen artean dauden edo ez egiaztatzeko funtzioa. Agindua definituta badago, lehen ikusi ditugun tauletan (`MODE`, `KOMANDOAK`, `KOMANDOAK1`) daukaten posizioa bueltatuko du. Honekin, sistema zein egoeratan dagoen jakingo du, pakete bat sartu edo ateratzeko egoeran gauden edo ez. Funtzio honek bi parametro hartzen ditu, komunikazioaren bitartez jaso den mezua eta komandoak eta aginduak definituta dauden tauletako bat.

- `Unexpected()`*:* Web aplikaziotik agindu ezezaguna ailegatzen bada `ERR5` errorea bidaltzeaz arduratuko den funtzioa.

- `Socketread()`*:* Lehen, komunikazioetatik jasoko diren mezuak gordetzeko ikusi ditugun hainbat funtzioen antzekoa. Wifi socketaren bitartez, datozen mezuak irakurtzeko balio digun funtzioa. Hau itxaroten geratuko da mezuren bat ailegatu arte edota definitu dugun denbora maximoa agortu arte (20 segundo). Denbora tartea web aplikazioari erantzuteko emango zaion denbora maximoa izango da. 20 segundo horietan erantzunik jaso ez bada errore bat gertatu dela adieraziko du. Definitutako denbora web aplikazioari bidalitako mezuak erantzuteko behar duen denbora kontuan izanda hartu da, honi sarean gertatu daitezkeen faktoreak gehituta. Mezua karakterez-karaktere sartzen da, mezuak gordetzeko definitu dugun `buf` aldagaian.

- `Searcherror()`*:* Sistemaren prozesuak aurrera eramango dituen `ServiceFunctions` liburutegiko `InsertPackage()` *eta* `ExtractPackage()` bueltatu ditzazketen errore kodeak, web aplikazioari bidaliko zaion errore mezuetara bilakatuko dituen funtzioa da.

- `Timer()`*:* Denbora jakin bat itxaroteko erabiliko den funtzioa. Parametro moduan itxaron behar den denbora milisegundutan pasatuko zaio.

Beraz, behin funtzio lagungarri hauek aztertu ditugula, sistemaren funtzionamendua aztertzera pasatuko gara.

Hasteko, Arduino Megak exekutatzen dituen lehenengo hiru prozesuak aztertuko ditugu. Hauek, wifi sarerako konexioa dagoela egiaztatu, komunikazio berri bat hasteko bezero berri baten konexioari itxaron eta bezero berriarekin konexioa ezarri ondoren, bidaliko den lehenego mezuaren tratamendua izango dira. Inplementazioa 6.33 kodean ageri da.

```
void loop() {
   // Loop nagusia, kode hau beti errepikatzen egongo da, eskaeraz eskaera.
  // Lehen esan dugun moduan eskaera berri baten zain geratzen garen bakoitzean gure
arduinok
  // daukan konexioa egiaztatu egingo da.
1  software_Reset();
   if (conn==true){
   // wait for a new client
2  client = server.available();
   // when the client sends the first byte, say hello:
3  if (client) {
         client.flush();
         Serial.println("Eskaera berri bat lortu da");
         client.println("Hello, client!");
         entry = true;
         while(entry){
4              // client (web aplikazioak) bidaliko duen aginduaren zain geratu
               er=Socketread(buf);
               if (er<0){
                 client.stop();
                 Serial.println("Errore bat gertatu da komunikazioan, zerbat espero
zen baina ez da ailegatu");
                 entry=false;
                 break;
               }
               client.flush();
               Serial.println("Lortu den agindua "+buf);
           // lortutako agindua, espero diren aginduen artean dagoen edo ez egiaztatu
               if((command=search_command(buf,MODE)) < 0){
                  unexpected();
                  client.stop();
                  entry=false;
                  break;
               }
               // agindua ondo badago OKI mezua bidali erabiltzaileari
5              if(server.write("OKI")<0){
                 Serial.println("Error when communicating with the client");
                 client.stop();
                 entry=false;
                 break;
               }
               timer(2000);
               finish=false;
               prozesua=false;
          // jaso den komandoaren arabera pakete bat atera edo sartzeko prozesuak egin
               switch(command){
```

6.33. Kodea: ArduinoMega wifi konexioa eta client berri baten konexioa

1. Arduino Megak wifi sarera daukagun konexioa egiaztatzen hasiko da.
2. Behin sarera konektatuta gaudela egiaztatu ondoren, bezero berri baten zain geratuko gara, hau da, web aplikazioren eskaera berri baten zain geratuko gara.
3. Eskaera bat ailegatzen denean, konexioa ezarriko da, eta web aplikaziotik bidaliko den aginduaren zain geratuko da sistema.
4. Agindua jasotzerakoan, `search_command(buf,MODE)` bitartez, jasotako mezua definituta ditugun bi aginduen (`insert` eta `extract`) artean dagoela egiaztatuko da. Honekin ere jaso dugun aginduaren arabera sistema prozesatu beharrezko modua zehaztuko da, pakete bat ateraz edo sartuz. Definituta dauden bi aginduak ez diren beste mezu bat

jasotzen badugu, ERR5 errore mezua bidaliko zaio web aplikazioari. Ondorioz, agindua behin jasota eta interpretatu dugula, agindu jakin bat bada web aplikazioari OKI mezua bidaliko zaio.

5. Behin OKI mezua bidaltzen dela, jasotako aginduari dagokion prozesuarekin hasiko gara. 6.7 eta 6.8 irudietan ikusten dugun moduan, agindua jaso ondoren, komunikazioaren ekimena Arduino Megari pasatuko zaio, honek prozesua aurrera joan ahala web aplikaziori prozesuaren berri eman ahal izateko.

Jaso daitezkeen aginduak bi dira: `insert` eta `extract`. Hauek, pakete bat biltegian sartzeko edota gordeta dagoen pakete bat ateratzeko aginduak dira, hurrenez hurren. Hori dela eta, Arduino Megaren begizta nagusia jaso den aginduaren arabera bitan banatuko da. Bi prozesuak banaka aztertuko ditugu.

6.2.5.3.1 Pakete bat sartzeko kasua (Insert modua)

Beraz, web aplikazioak `insert` agindua bidaltzean, biltegiratze-sistemak pakete bat sisteman sartu beharko du. Horretarako 6.2.3 atalean ikusi dugun `InsertPackage()` funtzioa erabiliko dugu, honek pakete bat sartzeko egin beharreko prozesu guztiak kontrolatzen baititu. Lehen aztertu dugun moduan `InsertPackage()` egoeraz egoera betetzen joango da bete beharreko prozesua. Egoera bakoitza bukatzerakoan egoera horren berri bidaliko zaio web aplikazioari 6.9 irudian ikusi dugun moduan. Behin web aplikazioari egoeraren berri eman zaiola, hurrengo egoera betetzera pasatuko gara.

`InsertPackage()` funtzioari egiten zaion deia 6.34 kodean ageri da.

```
case MOD_INST: //Insert Package
    while (!finish){
      hurrengoegoera=InsertPackage(unekoegoera,aurrekoegoera,id,pos,besorob,biltegia);
```

6.34. Kodea: MOD_INST kasuaren InsertPackage() funtzioaren deia

Ikusi dugun moduan, `search_command()` bueltatutako balioak eta `insert` moduak definituta daukan `MOD_INST` egoerak, identifikadore berdina daukate. `Insert` aginduak `MODE[]` taulan 0 posizioa daukalako eta `MOD_INST` aldagaiak 0 balioa.

Behin modua definitu dugula, begizta baten barruan sartuko gara. Begizta honen baldintza `finish` aldagai boolearra `false` izatea da. Ondorioz, prozesu guztia bukatzen ez den arte edota errore bat gertatzen ez den arte, `finish` balioa ez da `true` balio izatera pasako.

Ondoren, `InsertPackage()` funtzioari egiten zaion deia dago. Kontuan izan behar da funtzioari dei egiten zaion bakoitzean `unekoegoera` aldagaian zehaztuta dagoen egoera prozesatuko dela. Funtzioak egoera bukatzean bueltatuko duen balioa, hurrengo egoeraren identifikadorea izango da eta hau `hurrengoegoera` aldagaian gordeko da.

Egoera bat betetzen denean `hurrengoegoera` aldagai honen balioaren arabera, bi kasu ezberdindu ditzazkegu. Alde batetik, `hurrengoegoera` balioa negatiboa daukanean. Honek, prozesuan errore bat egon dela esan nahiko du. Beraz, web aplikazioari egon den errorearen mezua bidali beharko zaio. Horretarako 6.35 kodean ikusten dugun kodea exekutatuko da.

```
}else{
   timer(2000);
1  searcherror(hurrengoegoera).toCharArray(error,ARRAYSIZE);
   Serial.print("Aplikazioari bidali zaion errore kodea: ");
   Serial.println(error);
1  if(server.write(error)<0){
      client.stop();
      Serial.println("Errore bat gertatu da komunikazioan, zerbat espero zen baina ez
da ailegatu");
      entry=false;
```

```
      finish=true;
      unekoegoera=0;
      aurrekoegoera=0;
      continue;
      }
2  er=Socketread(eran);
   if (er<0){
      client.stop();
      Serial.println("Errore bat gertatu da komunikazioan, zerbat espero zen baina ez
da ailegatu");
      entry=false;
      finish=true;
      unekoegoera=0;
      aurrekoegoera=0;
      continue;
    }
 // client-ak ondo jaso badu errorea komunikazioa eta prozesamendua modu egokian
bukatu
   if (eran.equals("OK")){
       client.stop();
       entry=false;
       finish=true;
       aurrekoegoera=0;
       unekoegoera=0;
       Serial.println("Errore batekin bukatu da, baina komunikazioa ondo joan da\n");
 // bestela berdin ere bukatu prozesua, baina ez dakigu beste aldeak zer nolako
egoeran geratu den
3  }else{
       client.stop();
       entry=false;
       finish=true;
       aurrekoegoera=0;
       unekoegoera=0;
       Serial.println("Errore batekin bukatu da, eta komunikazio gaizki jun da\n");
   }
 }
}
```

6.35. Kodea: MOD_INST egoeran prozesuan errore bat egon bada

1. `InsertPackage()` funtziotik lortu den errore kodea 6.32. kodean ikusi dugun `searcherror()` funtzioaren bitartez, bidali behar den errore mezu batengatik ordezkatuko da. Behin errore kodea daukagula socketaren bitartez errorea web aplikazioari bidaliko zaio.
2. Web aplikazioak errore mezua jasotzerakoan, erantzun mezu bat bidaliko dio bueltan Arduino Megari. Arduino Megak estimatutako denboran erantzunik jaso ez badu errore moduan bukatuko da. Arduino Megak `OK` bat jaso badu, mezua ondo ailegatu dela adieraziko du, prozesua bukatutzat emanez. Horretarako beharrezko aldagaiak hasieratu beharrezko mezua pantailaratu eta komunikazioa itxiko da; eskaera berri baten zain geratzera pasatuz.
3. `OK` mezua ez bada jaso, berdina gertatuko da, baina pantailaratuko den mezua, errore mezua izango da. Honekin komunikazioa itxi egingo da eta berriro ere bezero berri baten zain geratuko da.

Aldiz, dena ondo joan bada eta `hurrengoegoera`-n hurrengo egoeraren identifikadorea bueltatu bada, uneko egoera bete dela esan beharko zaio web aplikazioari eta jarraian, hurrengo egoera exekutatzeko prestakuntzak egin.

Hau dena, 6.36 kodean ikusi dezakegu.

```
if(hurrengoegoera>0){
1       KOMANDOAK[unekoegoera].toCharArray(aux,ARRAYSIZE);
        Serial.print("Web aplikazioari bidaliko zaion mezua: ");
        Serial.println(aux);
        if(server.write(aux)<0){
            Serial.println("Errore bat egon da komunikazioan");
```

```
            client.stop();
            entry=false;
            finish=true;
            continue;
         }
         //Bidalitakoari client-ak erantzun duenaren arabera ondo edo errorea
         er=Socketread(eran);
         if (er<0){
             client.stop();
             Serial.println("Errore bat gertatu da komunikazioan, zerbat espero zen
baina ez da ailegatu");
             entry=false;
             finish=true;
             unekoegoera=0;
             aurrekoegoera=0;
             continue;
         }
         Serial.println(eran);
2        if (eran.equals("OK")){
a          if (unekoegoera==ST_DONE){
               finish=true;
               client.stop();
               entry=false;
               unekoegoera=0;
               aurrekoegoera=0;
               Serial.println("Paketea ondo sartu eta gorde da");
b           }else if(unekoegoera==ST_CLOSE){
               eran="";
               id.toCharArray(idr,ARRAYSIZE);
               if(server.write(idr)<0){
                  Serial.println("Errore bat egon da komunikazioan");
                   client.stop();
                   unekoegoera=0;
                   aurrekoegoera=0;
                   entry=false;
                   finish=true;
                   continue;
                  }
                 er=Socketread(eran);
                 if (er<0){
                     client.stop();
                     Serial.println("Errore bat gertatu da komunikazioan, zerbat
espero zen baina ez da ailegatu");
                     entry=false;
                     finish=true;
                     unekoegoera=0;
                     aurrekoegoera=0;
                     continue;
                 }
                  client.flush();
                  if (eran.equals("OK")){
                       aurrekoegoera=unekoegoera;
                       unekoegoera=hurrengoegoera;
                        Serial.println("Egoera eta komunikazioak ondo bukatu dira");
c                   }else{
                       aurrekoegoera=unekoegoera;
                       unekoegoera=hurrengoegoera;
                       Serial.println("Egoera eta komunikazioak ondo bukatu dira");
                   }
               }else{
                  Serial.println("Errore bat egon da komunikazioan");
                  client.stop();
                  entry=false;
                  finish=true;
                  prozesua=true;
                  aurrekoegoera=0;
                  unekoegoera=0;
               }
```

6.36. Kodea: MOD_INST egoerarn prozesuaren berri komunikatu eta hurrengo egoerarako prestatu

1. Ondorioz, dena ondo joan bada, bete den egoerarari dagokion komandoa bidaliko zaio web aplikaziori, honek prozesuaren berri izateko. Bidaliko diren komandoak eta bete

diren egoeren artean erlazio zuzena dago, 6.9 irudiko protokoloan ikusi daiteken moduan.

2. Jarraian, web aplikazioren erantzunaren zain geratuko gara. Mezua jasotzean `OK` ez den zeozer jasotzen badugu errore bat egon dela suposatuko da, prozesua bertan behera geratuz. Aldiz, `OK` jasotzen bada, hurrengo egoera prestatzera pasatuko gara. Horretarako hiru kasu ezberdin definitu ditugu.

    a. ST_DONE: Bete berri den egoera, hau da, `unekoegoera` aldagaian dagoen balioa ST_DONE egoeraren identifikadorea bada, pakete bat sartzeko prozesu osoa errorerik gabe bukatu dela adieraziko du. Ondorioz, prozesua bukatutzat emango da, konexioa itxiz eta aldagai guztiak hasieratuz. Hemendik, programa bezero berri batek bidaliko duen beste eskaera baten zain geratuko da.

    b. ST_CLOSE: ST_CLOSE egoera bukatzen dugunean eta web aplikazioari honen berri emandakoan, 6.9 irudian ageri den protokoloan ikusi daitekeen moduan, web aplikazioa sartu berri den paketearen id-aren zain geratuko da. Ondorioz, ST_CLOSE egoera bete eta gero, id-a bidaliko diogu web aplikazioari. Bidali eta gero web aplikazioaren erantzunaren zain geratuko gara. `OK` erantzuna jaso ondoren, `hurrengoegoera` aldagaian zegoen egoera uneko egoerara izatera pasatuko da. `InsertPackage()` funtzioari egingo zaion hurrengo deian prozesatuko den egoera izatera pasaz.

    c. Beste egoera guztiak: Beste edozein egoera bukatzean eta web aplikaziori honen berri eman ondoren, `hurrengoegoera unekoegoera` izatera pasatuko da. Horrela, `InsertPackage()` funtzioari egingo zaion hurrengo deian hurrengo egoera prozesatu egingo da.

Ondorioz, ikusi dezakegu modu zikliko batean egoera guztiak prozesatuz joango direla Arduino Megak `InsertPackage()` funtzioari deitzen dion bakoitzean, egoera sekuentzia aproposa exekutatuz. Horrela prozesua kontrolatu ahal izango dugu, eta erroreren bat gertatuz gero web aplikazioak errorea ezagutuko du.

Beraz, prozesu honekin 6.9 irudian ageri den komunikazio protokoloa betetzen dela egiaztatzen dugu.

6.2.5.3.2 Pakete bat ateratzeko kasua (Extract modua)

`extract` modu honetan sartuko gara, web aplikazioak `extract` agindua bidaltzen digunean. Behin modu honetan sartzen garenean `insert` kasuan ez bezala, prozesuarekin hasi baino lehen Arduino Megak atera nahi den paketearen id-a lortu beharko du web aplikaziotik, atera nahi den paketea identifikatu ahal izateko. Web aplikazioak id-a bidali ondoren, lehen ikusi dugun antzeko prozesua aurrera eramango da, egoeren bitartez funtzionatzen duen `ExtractPackage()` funtzioa erabiliz.

Honetan ere, egoera bakoitza bukatzerakoan, honen berri eman beharko zaio web aplikazioari 6.10 irudiko protokoloan definituta agertzen den moduan. Behin egoera bukatuta eta web aplikazioari mezua bidali zaiola, egoera berri bat prozesatzera pasatuko gara.

Hasteko `extract` moduan id-a nola jaso eta prozesua nola hasten den aztertuko dugu.

```
 case MOD_EXT: //Paketea kentzeko modua
   while (!finish){
      er=Socketread(eran);
      if (er<0){
         client.stop();
         Serial.println("Errore bat gertatu da komunikazioan, zerbat espero zen baina
ez da ailegatu");
```

```
          entry=false;
          finish=true;
          unekoegoera=0;
          aurrekoegoera=0;
          continue;
       }
       client.flush();
       Serial.println(buf);
       id=buf;
       if(server.write("OKI")<0){
        Serial.println("Errore bat egon da komunikazioan");
        client.stop();
        entry=false;
        finish=true;
        continue;
        }
       timer(5000);
       while(!prozesua){
      hurrengoegoera=ExtractPackage(unekoegoera,aurrekoegoera,id,pos,besorob,biltegia);
```

6.37. Kodea: MOD_EXT Id jasotzea eta prozesuaren hasiera

6.37. kodean ikusi dezakegun moduan, `MOD_INST` moduan ez bezala, `MOD_EXT` honetan paketearen identifikadorea web aplikaziotik jaso behar dugu, sistemak atera nahi den paketearen id lortzeko. Web aplikaziotik mezua jasotzeko `Socketread()` funtzioa erabiliko dugu. Behin id-a jaso dugula `OKI` erantzuna bidaliko diogu zerbitzariari eta komunikazioaren ekimena Arduino Megara pasatuko da, paketea ateratzeko prozesua aurrera joan ahala honen berri web aplikazioari bidaliz.

Beraz, paketea ateratzeko prozesua `ExtractPackage()` funtzioaren bitartez hasiko da. Hau, 6.2.3 atalan ikusi dugun moduan egoeraz egoera prozesatuko da. Lehenengo egoeratik hasiko da, eta egoera bakoitza bukatzerakoan hurrengo egoeraren identifikadorearen kodea dena bueltatuko du funtzioak, web aplikazioari honen berri emanez egoera bakoitzari dagokion komandoa bidaliz. Ziklo hau etengabe errepikatuko da azkenengo egoerara ailegatu arte.

Ziklo honetan ematen diren kasu ezberdinak, 6.38. kodean ikusi daitezke.

```
while(!prozesua){
  hurrengoegoera=ExtractPackage(unekoegoera,aurrekoegoera,id,pos,besorob,biltegia);
  if(hurrengoegoera>0){
     KOMANDOAK1[unekoegoera].toCharArray(aux,ARRAYSIZE);
     Serial.println(aux);
     if(server.write(aux)<0){
       Serial.println("Errore bat egon da komunikazioan");
       client.stop();
       entry=false;
       finish=true;
       continue;
     }
     timer(3000);
     er=Socketread(eran);
    if (er<0){
        client.stop();
        Serial.println("Errore bat gertatu da komunikazioan, zerbat espero zen baina
ez da ailegatu");
        entry=false;
        finish=true;
        unekoegoera=0;
        aurrekoegoera=0;
        continue;
     }
     Serial.println(eran);
     client.flush();
     if (eran.equals("OK")){
       if (unekoegoera==EX_DONE){
         finish=true;
         client.stop();
         entry=false;
         prozesua=true;
         aurrekoegoera=0;
```

```
          unekoegoera=0;
          Serial.println("Atera nahi zen paketea ondo atera da");
        }else{
          aurrekoegoera=unekoegoera;
          unekoegoera=hurrengoegoera;
          Serial.print("Egoera eta komunikazioa ondo bukatu da");
        }
      }else{
        Serial.println("Errore bat egon da komunikazioan");
        client.stop();
        entry=false;
        finish=true;
        prozesua=true;
        aurrekoegoera=0;
        unekoegoera=0;
       }
```

6.38. Kodea: MOD_EXT prozesatu berri den egoeraren komunikazioa eta hurrengo egoerarako prestakuntza

Kasu hau, `insert` kasua baino errazagoa da. Honetan `ExtractPackage()` funtzioak egoera bat prozesatzen duenean erroreren bat gertatzen bada, funtzioak errore kode bat bueltatuko du eta 6.35 Kodean ikusi dugun prozesu berdina jarraituko da. Honetan errore kode horri dagokion errore mezua bidaliko zaio web aplikazioari eta prozesatzen ari zen eskaera bertan behera utziko da, beste berri baten zain egotera pasaz.

Aldiz, prozesua ondo bete bada, hurrengo egoeraren identifikadorea bueltatuko da. Beraz, egoera bukatu egin dela jakinaraziko zaio web aplikazioari.

1. Horretarako, egoera bakoitzari dagokion komandoa bidali egingo zaio web aplikazioari. Egoeraren identidikadorea eta bidali behar den komandoa `KOMANDOAK1[]` taulan daukan posizioa berdinak izango dira.

Behin komandoa bidali diogula web aplikazioari, honen erantzunaren zain geratuko gara. Bi aukera daude: `OK` batekin erantzutea edota definituta ez dagoen mezu bat ailegatzea.

2. Definituta ez dagoen mezu bat ailegatzen bada, errore moduan tratatuko dugu, komunikazioa itxiz eta aldagai guztiei hasierako balioak emanez. Horrela eskaera berri baten zain egotera pasatuko gara.
3. Berriz, `OK` mezua jasotzen badugu, prozesuan bi aukera definitu dira.
    a. EX_DONE: Azkenengo egoera ondo prozesatu bada, prozesua ondo bukatu dela adieraziko du. Beraz, komunikazioa itxi egingo da eta aldagaiak hasieratu egingo dira, etorkizunean egingo diren eskaerentzako. Hasiera batean azaldu dugun moduan, `finish` aldagaia izango da zikloa kontrolatu egingo duen aldagaia eta beraz, honi `true` balioa emanda ziklotik aterako gara eta Arduino Megak eskaera berri baten zain egotera pasatuko da
    b. Beste egoera guztiak: `unekoegoera aurrekoegoera` izatera pasatuko da eta `hurrengoegoera unekoegoera` izatera pasatuko da. Horrela `ExtrackPacakge()` funtzioari egingo zaion hurrengo deian hurrengo egoera prozesatuko da. Modu orokor batean, egoeraz egoera pasatuko gara azkenengora ailegatu arte.

7. Web aplikazioaren diseinua

7.1 Web aplikazioa

Web aplikazioak, lehen azpimarratu dugun moduan, bi funtzio nagusi izango ditu. Batetik, sistemaren “hardware” zatiarekin komunikazioa mantentzea eta beraz, komunikazioan bidali eta jasotze horren kudeaketa lanak egitea. Bestetik, sistemak erabiltzaile ezberdinekin elkarrekintza bideratzearen arduraduna izango da. Horretarako, aplikazioaren diseinua eta irisgarritasuna egokiak izan behar dira.

Hasteko, aplikazioak daukan egitura ikusiko dugu, dituen erabiltzaile mota, datu-base eta karpeta egiturak aztertuz. Geroago, aplikazioak daukan diseinua eta garapena aztertzera pasatuko gara (orrialdeen itxura, nola funtzionatzen duten, etab), eta azkenik, 4. kapituluan definitu ditugun funtzio guztien sekuentzia-diagramak ikusiko ditugu.

7.1.1 Aplikazioaren egitura

Egiturarekin hasi baino lehen, argi utzi behar ditugu web aplikazioak izango dituen bi helburu nagusiak. Alde batetik, erregistratuta dauden erabiltzaileak biltegiratze-sisteman paketeak gorde ahal izatea eta, bestetik, sisteman jadanik gordeta dauden paketeak biltegiratze-sistematik atera ahal izatea. Bi funtzio hauek asetzeko helbururarekin egituratu da web aplikazioa.

7.1.1.1 Erabiltzaile motak

Web aplikazioak dituen orrialde eta funtzio guztiak aztertu baino lehen, gure aplikazioak dituen datu-base eta erabiltzaile motak aztertuko ditugu. Gure aplikazioak erabiltzaile ezberdinak izango ditu. Erabiltzaileak bi multzotan banatu ditzakegu:

- Aplikazioan sartzerakoan erabiltzaile guztiak erabiltzaile anonimoak izango dira. Erabiltzaile anonimo batek dituen baimenekin, sisteman erregistratu edota jadanik erregistratuta badago sisteman sartu daiteke, email-a eta pasahitza erabilita.
- Behin sisteman gaudela, dugun kontuaren baimenen arabera, hiru erabiltzaile mota definitu ditugu: erabiltzaile arrunta, banatzailea edota administratzailea.

7.1.1.2 Datu-baseak

Dauden erabiltzaile motak aztertuta, erabiltzaile hauek, beste hainbat baliorekin batera nola gordetzen diren aztertuko dugu. Aplikazio honek erabili behar dituen datuak modu egoki batean gordetzeko MySQL datu-base bat sortzea erabaki da. XML orrietan gordetzeko aukera egonda ere, datu-baseak datuak kudeatzerakoan eskaintzen duen egonkortasun eta erraztasuna direla eta, gure eskakizunetara hobekien moldatzen den teknologia da.

Beraz, aplikazio honen datuen kudeaketarako erabili dugun datu-basean definitu ditugun taula ezberdinak aztertuko ditugu jarraian.

7.1.1.2.1 Erabiltzaileak

Izenaz ondorioztatu dezakegunez, taula hau erabiltzaile erregistratuak gordetzeko erabiliko dugu. Taula honek hainbat zutabe izango ditu, erabiltzailearen informazio ezberdina gordeko

dutenak. 7.1 irudian ikusi daiteke taulak izango duen zutabe bakoitza eta datu bakoitzak izan beharko duen egitura.

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado
1	**Email** 🔑	char(30)	latin1_spanish_ci		No	*Ninguna*
2	**Izena**	text	latin1_swedish_ci		No	*Ninguna*
3	**Abizena**	text	latin1_spanish_ci		No	*Ninguna*
4	**Abizena2**	text	latin1_spanish_ci		Sí	*NULL*
5	**Pasahitza**	text	latin1_spanish_ci		No	*Ninguna*
6	**Mugikorra**	text	latin1_spanish_ci		No	*Ninguna*
7	**Galdera**	text	latin1_spanish_ci		No	*Ninguna*
8	**Erantzuna**	text	latin1_spanish_ci		No	*Ninguna*
9	**Saialdi**	int(3)			No	*Ninguna*
10	**Baimen**	text	latin1_spanish_ci		No	*Ninguna*

7.1. Irudia: Erabiltzaile taularen formatua

Taulan sartuko den erabiltzaile bakoitzak 10 parametro izango ditu. Irudian ikusi daiteken moduan, email-a izango da taularen gako nagusia[10] eta ondorioz, sisteman email bakoitza bakarra eta errepikaezina izango da.

Honetaz aparte, erabiltzailei buruz testu soilean gordeko den informazioa hurrengoa izango da:

- `Izena` : Erabiltzailearen izena.
- `Abizena` : Erabiltzailearen lehenego abizena.
- `Mugikorra`: Erabiltzailereakin kontaktuan jartzeko mugikor zenbakia.
- `Galdera`: Pasahitza ahaztu bada, pasahitza berreskuratzeko erantzun beharreko galdera.
- `Baimena`: Zer motako erabiltzailea den zehazten duen balioa. Kontuan izan behar da parametro honen balioa sistemak zuzenean emango diola erabiltzaileari eta honek ezingo duela erabaki zer erabiltzaile mota den. Hiru balio posible izango ditu: ikasle(erabiltzaile arrunta), banatzaile eta admin.

Parametro hauekin batera, testu soilean gordeko den beste bat aurkitu dezakegu, bigarren abizena gordetzeko erabiliko den `abizen2`. Hala ere, parametro hau hautazkoa da eta, beraz, erabiltzaileak bete gabe utzi dezake.

Bukatzeko beste bi parametro ditugu: pasahitza eta pasahitza berreskuratzeko segurtasun galderaren erantzuna. Hauek zifratuta gordeko dira sisteman SHA1 teknologia erabiliz, datuen integritatea, neurri baxuan bada ere, ziurtatzeko.

Azkenik, testua ez den parametro bakarra aurkituko dugu, `saialdi` parametroa. Hau, erabiltzaile batek kontu batean sartzeko duen aukera kopurua izango da. Parametroak gehienez hiru saiakera baimenduko ditu. Ondorioz, erabiltzaile bat hiru aldiz saiatzen bada kontu batean sartzen, eta pasahitza ez badu ondo jartzen, kontu hori zuzenean blokeatuko da eta administratzailearekin kontaktuan jarri beharko da, kontua desblokeatzeko.

[10] *Gako nagusia* taula batean elementu bakoitza identifikatzen duen eremuari edo eremu-konbinazioari deitzen zaio. Gako nagusia bakarra eta errepikaezina da.

Adibide moduan 7.2. irudian hiru erabiltzaile moten sarrerak ikusi daitezke.

banatzaileak@azkar.com	Azkar		cb45c671cbc500627ea424eea5f91996221b5935	945338866	Banatzailea kodea	bfe54caa6d483cc3887dce9d1b8eb91406f1ea7a	0	Banatzaile
jone@gmail.com	Jone	Hernandez Perez	cb45c671cbc500627ea424eea5f91996221b5935	698123456	Lehengo Ikasleixea	b4ea93b07284ef6e504f8205cacf1cb31a4bfa2a	0	ikasle
bermejo@admin.com	Eneko	Bermejo	*NULL* 21c61bf5057f09d4814143ee04d88ec31ca78795	692708133	Zein da zure administradore kodea?	7110eda4d09e062aa5e4a390b0a572ac0d2c0220	0	Admin

7.2. Irudia: Erabiltzaile taularen sarrerak

7.1.1.2.2 Paketeak

Taula honetan gure biltegiratze-sisteman momentu oro gordeta dauden paketeen zerrenda izango dugu. Erabiltzaileen taularekin gertatzen den moduan, pakete bakoitzak hainbat parametro izango ditu. Taulak izango dituen parametroak 7.3 irudian ikus daitezke.

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado
1	**id** 🔑	char(8)	latin1_spanish_ci		No	*Ninguna*
2	**email**	char(30)	latin1_spanish_ci		No	*Ninguna*
3	**banatzailea**	text	latin1_spanish_ci		No	*Ninguna*
4	**deskribzioa**	text	latin1_spanish_ci		No	*Ninguna*

7.3. Irudia: Pakete taularen egitura

7.3 irudian ikus daitekeen bezala, sarrera bakoitzak 4 parametro/zutabe izango ditu. Erabiltzaileen taulan email-arekin gertatzen den moduan, `id` parametroa gako nagusia izango da eta, ondorioz, taulan id hori daukan pakete bakar bat egon daiteke. Id parametro hau 8 luzeerako `char` motako datu batean gordeko da. Aurretik 6.1.3 atalean ikusi dugun moduan id-ak 4 byte hamaseitarrez irudikatzen baitu.

Hiru parametro gehiago izango ditugu:

- `Email-a`: Hartzailearen email-a izango da, hau da, paketea zuzenduta dagoen erabiltzailearen email-a.
- `Banatzailea`: Paketea utzi duen pertsonaren izena (email-a). Banatzaileen kasuan, enpresaren izena agertuko da.
- `Deskripzioa`: Paketeak barruan daukanaren deskripzio txiki bat, erabiltzaileak paketeak identifikatu ditzan.

Kasu honetan `email` parametroak erabiltzaileen taulan daukan datu egitura berdina izango du. Bestetik, `banatzaile` eta `deskripzio` parametroak testu soil moduan gordeko dira. 7.4 irudian ikusi dezakegu sisteman gordeta egon daitezkeen bi paketeen adibideak.

id	email	banatzailea	deskribzioa
6e5c7cb2	a@a.com	Azkar(banatzaileak@azkar.com)	GAP
eee47db2	a@a.com	Eneko(a@a.com)	Sistema Eragile Apunteak

7.4. Irudia: Pakete taulako sarreren adibidea.

7.1.1.2.3 Egiaztapenkodeak

Taula honen helburua banatzaileak erregistratzeko behar dituzten kode bereziak gordetzea izango da. 4. kapituluan azaldu dugun moduan enpresa banatzaileak kode bat lortu beharko du administratzailearengandik, sisteman erabiltzaile berri bat erregistratu aurretik. Honekin, iruzurrak ekidin daitezke, kode hau enpresak eta sistema adminstratzaileak bakarrik jakingo dutelako.

Kodeak erabilera bakarra izango du, hau da, bakarrik kontu bat erregistratzeko balioko du. Gainera, erabiltzaile arruntekin gertatzen den ez bezala, pasahitza berreskuratzeko galdera kasu honetan egiaztapen kodea izango da. Kode hauen banaketa ez da definitu, bakarrik errealitatearen adibide moduan erabili egin direlako.

Beraz, guzti hau kontuan hartuz, taulak 7.5 irudian ikusi daitekeen egitura izango du.

Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado
kodea	char(40)	latin1_spanish_ci		No	*Ninguna*
banatzailea	text	latin1_spanish_ci		No	*Ninguna*
Mugikorra	text	latin1_spanish_ci		No	*Ninguna*
Email	char(30)	latin1_spanish_ci		No	*Ninguna*
Erabil	tinyint(1)			No	*Ninguna*

7.5. Irudia: Egiaztapen kodeen taularen egitura

7.5 irudian ikusten den moduan, egiaztapen kodea bera da gako nagusia. Gainera 40 karaktere luzeko datu formatuan gordeko da. Horrez gain, segurtasuna dela eta, kodea ez da zuzenean gordeko, SHA1 teknologiaren bitartez enkriptatua sartuko da taulan.

Kodeaz gain beste 4 parametro izango ditugu.

- `Banatzailea`: Enpresa banatzailearen izena.
- `Mugikorra`: Beraiekin kontaktuan jartzeko mobila.
- `Email-a`: Erabiltzailea sortzeko erabiliko den email-a.
- `Erabil`: Parametro boolearra da hau, 0 edo 1 balioak izango ditu, sistemak kodea erabili den edo ez jakin dezan. Horregatik aukeratu den datu-egitura `tinyint` izango da, datu-egitura honek bakarrik byte bat erabiltzen du eta.

Parametro guztiak aztertu ondoren, taula honen sarrera batzuk ikustera pasatuko gara. Adibide gisa 7.6 irudian ikusi ditzakegu bi enpresa ezberdinen kodeak.

kodea	banatzailea	Mugikorra	Email	Erabil
bfe54caa6d483cc3887dce9d1b8eb91408f1ea7a	Azkar	945338866	banatzaileak@azkar.com	1
f7c3bc1d808e04732adf679965ccc34ca7ae3441	Seur	945132894	banatzaileak@seur.com	1

7.6. Irudia: Egiaztapen kodeko taulen sarreren bi adibide

7.1.1.2.4 Konexioak

Taula hau ez da ezinbestekoa, baina sistema monitorizatzeko momentuan erabilgarri suertatu dakioke administratzaileari. Kasu honetan, taularen ideia, erabiltzaileak sisteman sartu edo irteten denean denbora eta datuak gordetzea izango litzateke. Honela, sisteman noiz sartu eta zenbat denboran egon den jakin ahal izango dugu. Horretarako, 7.7 irudiko egitura duen taula erabili da.

Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado
Email	char(30)	latin1_spanish_ci		No	*Ninguna*
Mota	text	latin1_spanish_ci		No	*Ninguna*
Ordua	datetime			No	*Ninguna*
LogInOut	text	latin1_spanish_ci		No	*Ninguna*

7.7. Irudia: konexio taulako egitura

Taula 4 parametroz osatuta dago:

- `Email-a`: Sisteman sartu edo irteten den erabiltzailearen email-a.
- `Mota`: Erabiltzaile mota.

- `Ordua`: Bi akzioak noiz egin diren. Kasu honetan ordua da gako nagusia. Sistema informatikoek, web aplikazioek bereziki, momentu berean bi erabiltzaile sartzeko ahalmena daukate. Hala ere, gure sisteman web aplikazioa bakarrik biltegiratze-sistemarekin komunikatzen den ordenagailuan atzigarri egongo da eta beraz, momentu bakoitzean erabiltzaile bakarra egon ahal izago da sisteman sartuta. Gainera gure biltegiratze-sistemak eskaerak bakarka prozesatu behar ditu eta ondorioz, aldi berean sesio bat baino gehiago irekita eukitzeak arazoak ekarriko lituzke.
- `LogInOut`: Sisteman sartu edo irteten den.

Konexio taularen itxura 7.8. irudian ikus daiteke.

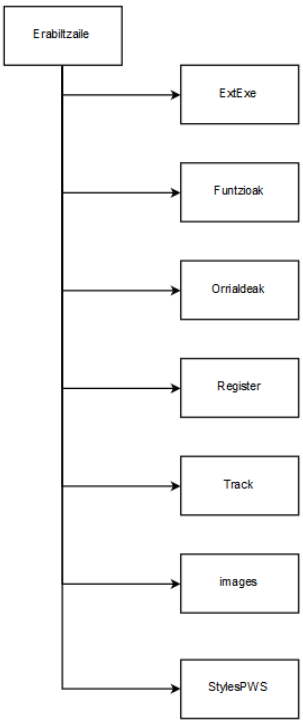
banatzaileak@seur.com	Banatzaile	2017-03-22 10:19:08	LogIn
banatzaileak@seur.com	Banatzaile	2017-03-22 10:24:24	LogOut
jbermejo@ikasle.ehu.eus	ikasle	2017-03-22 11:38:00	LogIn
jbermejo@ikasle.ehu.eus	ikasle	2017-03-22 12:42:25	LogOut
jbermejo@ikasle.ehu.eus	ikasle	2017-03-22 13:07:11	LogIn
jbermejo@ikasle.ehu.eus	ikasle	2017-03-22 13:16:39	LogOut

7.8. Irudia: Konexio taulako adibidezko sarrerak.

7.1.1.3 Fitxategi-sistema eta aplikazioaren diseinua

Aplikazioaren datu-base eta erabiltzaileak ikusita gure aplikazioak izango duen fitxategi-sistemaren egitura aztertzera pasako gara. Hau gure aplikazioan dauden elementu ezberdinak banatzeko erabili dugu, modu orokor batean aplikazioa ulergarriagoa suertatu dadin.

7.9 irudian ikusi dezakegu gure web aplikazioarentzako diseinatutako fitxategi-sistema.



7.9. Irudia: Aplikazioaren fitxategi sistema

Hauek izango dira gure aplikazioak izango dituen elementu ezberdin nagusiak. Orain bakoitza bakarka aztertzera pasatuko gara, bakoitzaren barruan aurkitzen diren orrialde edota funtzioak zerrendatuz eta zer nolako helburua duten azalduz.

7.1.1.3.1 ExtExe karpetaren edukia

Karpeta honen barnean bi motako osagaiak daude. Arduinoarekin komunikazioa egiteko erabiliko diren C++ aplikazioak eta C++ aplikazio hauek deitzeaz arduratuko diren PHP funtzioak. Alde batetik, pakete berri bat sartzeko behar dugun `Insert.php` eta `Insert.exe` izango ditugu eta bestetik, `ExtactPackage.php` eta `extract.exe`. Bikote moduan aztertuko ditugu, bakoitzaren funtzioak eta diseinua aztertuz.

- `Insert.php` eta `insert.exe`: Kasu honetan erabiltzaileak sistemari pakete bat sartu behar dela agintzen dionean, aplikazioak `Insert.php` funtzioari deituko dio. Hau `insert.exe` exekutatzeaz arduratuko da. `Insert.exe` wifi socket-a sortzeaz eta Arduino Megarekin komunikazioaz (6.2.4 atalan ikusitako protokoloaren web aplikazioaren aldea inplementatuko duena) arduratuko den aplikazioa izango da. `Insert.exe` honek sartu den paketearen id-a lortuko du komunikaziotik. `Insert.exe` exekuzioa bukatzerakoan testu-fitxategi batean utziko du pakete horren id-a, `Insert.php`-k hortik hartu dezan.

  `Insert.exe` bukatzerakoan, `Insert.php`-k `insert.exe`-ren exekuzioak bueltatutako balioaren arabera, prozesua nola joan den jakingo du eta erabiltzaileari komunikatuko dio. Gainera prozesuan zehar `insert.exe`-k jarraitutako pausuak `Track` karpetako testu-fitxategi batean gordeko dira. Fitxategi hauen diseinua eta funtzionamendua `Trak` karpeta aztertzean ikusiko dugu.

- `ExtractPackage.php` eta `extract.exe`: Kasu hau aurrekoaren antzekoa da. Kasu honetan erabiltzaileak pakete jakin bat atera nahi duenean web aplikazioak `ExtractPackage.php`-ri deituko dio atera nahi den paketearen id-a pasatuz. Hortaz, `ExtractPackage.php` funtzioak `extract.exe` aplikazioari deituko dio paketearen id-a argumentu bezala pasaz. Ondorioz, `extract.exe` sistemaren Arduino Megarekin komunikatu egingo da paketea ateratzeko. Paketea behin atera dela eta komunikazioa bukatu delarik, `extract.exe`-k bueltatutako balioaren arabera `ExtractPackage.php`-ek jakingo du prozesua nola joan den. `extract.exe` honek jarraitutako pausuak `Track` karpetako testu-fitxategia batean gordeko ditu ere.

7.1.1.3.2 Track karpeta

Karpeta honen barruan web aplikazioaren erabiltzaile bakoitzeko testu-fitxategi bat aurkituko dugu. Testu-fitxategiaren izena erabiltzailearen email-a izango da, eta erabiltzaile bakoitzak aplikazioari eskatutako ekintzen monitorizazio bat egiteko erabiliko da.

Beraz, lehen esan dugun moduan, fitxategi hauetan `insert.exe` eta `extract.exe` exekuzioen trazak gordeko dira, erabiltzaile bakoitzak zer nolako eskaerak egin dituen jakiteko eta pakete bakoitzak sistemaren barruan izandako mugimenduak ikusteko.

Hasteko traza guztiek exekuzioa gertatu den ordua izango dute, geroago traza guztien bilaketa errazago izan dadin. Hortik aurrera, aplikazioak erabiltzaileak eskatutako ekintza aurrera eramateko exekuzioaren trazak egongo dira. Adibidea, 7.10 irudian daukagu.

```
Process date and time is: Thu May 18 10:31:30 2017
Arduino Megari bidalitako agindua: extract
Jaso den erantzuna: OKI
Atera nahi den paketearen id-a bidali da: 6e5c7cb2
Jaso den erantzuna: OKI
While-era sartuko naiz
Jaso den komandoa: put
Paketea utzi da plataforman
Jaso den komandoa: open
Paktetea ateratzera doa
Jaso den komandoa: clos
Paketea ez da bueltatu
Jaso den komandoa: regi
Paketea sistematik atera da
Jaso den komandoa: done
Dena ondo joan da
```

7.10. Irudia: Track adibidea

7.1.1.3.3 Orrialdeak karpeta

Gure web aplikazioaren karpetarik garrantsitzuena. Hemen izango ditugu erabiltzaileak ikusiko dituen interfazeak eta, beraz, web aplikazioak erabiltzaileekin egingo duen elkarrekintza, hauen bitartez egingo da. Horregatik, web aplikazioaren muina dela esan dezakegu, hau da, erabiltzaileen eskaerak administratu eta erantzungo dituzten orrialdeak. Hainbat orrialde ezberdin daude, hauek banaka analizatuko ditugu.

- `Hasiera.php`: Web aplikaziora sartzerakoan ikusiko dugun lehengo orrialdea. Orrialdea hau atzitzean erabiltzaile anonimoen baimena izango dugu, oraindikan ez baitugu sesiorik ireki. Honetan bi aukera izango ditu, `Login` (jadanik sisteman erregistratuta dagoen norbaitek sisteman sartzeko erabiliko duena) eta `SignUp`, sisteman erregistratzeko erabiliko dena.

  Behin web aplikazioan sesioa hasieratu ondoren, orrialde hau dinamikoa denez, sisteman sartu den erabiltzaile motaren arabera, aukera ezberdinak agertuko dira.

  - Erabiltzaile arruntek: Pakete bat sartzeko, pakete bat ateratzeko eta sesioa bukatzeko aukerak izango dituzte.
  - Banatzaileak: Paketeak sartzeko eta sesioa bukatzeko aukerak izango dituzte.
  - Administratzaileak: Paketeak ateratzeko, erabiltzaileak ezabatzeko eta sesioa bukatzeko aukerak izango dituzte.

- `signIn.php`: Esan bezala, `Hasiera.php`-tik sesiorik ireki ez badugu, sesio bat hasteko aukera izango dugu. Orrialde honetan, erabiltzailea sisteman erregistratuta badago sesio berri bat hasteko aukera izango du. Erabiltzaileak bere kontuaren email eta pasahitza sartuz hasiko du sesio berri bat. Sisteman sartu ondoren `Hasiera.php` orrira bueltatuko da, baina lehen esan dugun moduan, orrialdean erabiltzaile motaren arabera beste aukera batzuk izango ditu eskuragarri.

  `signIn.php` honetan, erabiltzaileak pasahitza ahaztuz gero pasahitza berri bat lortzeko aukera izango du, segurtasun galdera erabilita.

- `berreskuratuPass.php` eta `pasaBerria.php`: Erabiltzaile batek pasahitza ahazten badu berri bat ezartzeko aukera izango du, eta horretarako bi orrialde hauek daude. Hasiera batean erabiltzaileak pasahitza berreskuratu nahi duela aukeratzean, `berreskuratuPass.php` orrialdera birzuzendua izango da. Honetan 3 pausu jarraitu beharko ditu:

  1. Erabiltzaileak email-a idatzi beharko du. Sistemak email horrekin konturik dagoen edo ez egiaztatuko du.

2. Erabiltzailea existitzen bada, kontu horrek duen segurtasun galdera aurkeztuko zaio erabiltzaileari, galdera erantzuteko eremu batekin.

3. Erabiltzaileak galderari ondo erantzuten badio, sistemak pasahitza aldatzeaz ziur dagoen edo ez galdetuko dio. Baiezkoa bada, `pasaBerria.php`-ra pasatuko gara, ezezkoa bada, berriz, hasiera orrialdera bueltatuko gara.

Behin `pasaBerria.php`-n, pasahitz berria sartzeko eremua agertuko zaio erabiltzaileari. Hemen pasahitz berria sartu, sistemak beharrezko egitura duela egiaztatu, eta datu- basean pasahitz zaharragatik ordezkatuko da.

- `AukeraSignUp.html`: `signIn.php`-rekin gertatzen den moduan, `Hasiera.php` orritik sisteman kontu berri bat erregistratzeko aukera izango dugu. Aurretik azaldu den moduan erregistratzeko momentuan bi erabiltzaile mota bezala erregistratu gaitezke, erabiltzaile arrunt bezala edo banatzaile moduan. Ondoren, aukera orrialde honetan bi erregistratzeko moduen artean aukeratu ahal izango dugu. Bata edo bestea aukeratuta, erregistratzeko formulariora birbidaliko gaituzte, `SignUp.html` edo `Egiaztapenkodea.php`, hurrenez hurren.

  Hala ere, kapitulu honen hasieran esan dugun moduan, banatzaile moduan erregistratzeko (`Egiaztapenkodea.php`) kode berezi bat beharko da. Ondorioz, `Egiaztapenkodea.php`-n dagoen formularioan sartu baino lehen, `AukeraSignUp.html` honetan banatzaileen kode bereziaren egiaztapena egingo da.

- `SignUp.html` eta `EgiaztapenKodea.php`: `AukeraSignUp.html` orritik bi orrialde hauetara helduko gara. Biak batera azaltzea erabaki da, biak funtzio berdina eta aukera berdinak aurkezten dituztelako, ezberdintasun minimo batekin. Biak erregistratzeko formulario bat aurkeztuko digute, non erabiltzaileak erabiltzaile berri bat sortzeko beharrezko informazioa sartzeko eremu ezberdinak izango dituen. Informazio eremuak 7.2.1.3 atalan, erabiltzaile datu-baseko taula analizatzerakoan ikusitakoak izango dira.

  Orrialde hauek erabiltzaileak sartzen duten informazioaren kontrol minimoaz arduratuko dira. Beharrezko informazio guztia betetzen dela eta sartzen den informazioak egitura egokia duela kontrolatuko dute. Behin erabiltzaileak informazio guztia sartzen eta bidaltzen duenean, geroago `Funtzioak` karpeta aztertzean (7.1.1.3.4 atala) ikusiko dugun moduan, `Erregistratu.php` funtzioa informazioa datu-basean sartzeaz arduratuko da.

  Lehen esan bezala, erregistatzeko bi aukeren artean ezberdintasun bat dago. Erabiltzaile arrunt batek `SignUp.html` orrialdean informazio eremu guztiak bete behar dituen bitartean, banatzaileek 7. kapituluaren hasieran datu-baseen eremuan komentatu dugun moduan, eremu batzuk dagoeneko beteta izango dituzte, adibidez, email, mugikor eta pasahitza berreskuratzeko eremuak. Funtzionamendu honen bitartez sisteman erregistratzen diren banatzaileak kontrolatu ahal izango dira.

  Bukatzeko, erabiltzaileen datu-basea aztertzean (7.1.1.2.1 atala) ikusi dugu nola `Baimen` eremua (erabiltzaile mota) automatikoki sistemak definituko du, erregistroa egin den orrialdearen arabera. `SignUp`-etik datozen eskaerak erabiltzaile arruntak izango dira, `Egiaztapenkodea.php`- tik datozenak, berriz, banatzaileak.

Behin erabiltzaile anonimo batek egin ditzakeen erabilpen kasuak ikusita, erabiltzaile erregistratuak egin ditzaketen funtzioak posible egingo dituzten orrialdeak aztertzera pasatuko gara.

- `InsertPackage.php`: Kasu honetan banatzaile eta erabiltzaile arruntak sartu ahal izango dira. Orrialdeak sarreren kontrola eramango du, baimena ez duen inor sartu ez dadin.

Erabiltzaileak erregistratzean gertatzen den moduan, formulario bat izango dugu. Formulario honek datu-basearen paketeen taulak dituen eremuak izango ditu (ikus 7.2.1.4 atala). Formulario honetan, bai erabiltzaile arrunten kasuan, bai banatzaileen kasuan `banatzaile` eremua beraien email-arekin zuzenean beteko da eta aldaezina izango da. Honekin, norbaitek beste baten izenean pakete bat sartu ahal izatea ekiditzen dugu.

Bestetik, jasotzailea aukeratzerakoan, momentu horretan sisteman erregistratuta dauden erabiltzaile arrunten izenak agertuko zaizkigu. Bukatzeko paketeak barruan daukanaren deskribapen txiki bat sartzeko eremua izango dugu. Behin eremu guztiak bete ditugunean, lehen azaldu dugun `ExtExe` karpetako `Insert.php` orrialdera bidaliak izango dira. Hau, lehen azaldu bezela, sistema fisikoarekin komunikatuko den `Insert.exe` aplikazioari deitzeaz, honen exekuzioa kontrolatzeaz eta komunikazio horretatik lortutako pakete berriaren id lortzeaz arduratuko da. Ondoren, datu-basean sartu berri den paketearen informazioa sartzeaz arduratuko da, erabiltzaileak osatutako formularioaren informazioa eta biltegiratze-sistemak bidali duen sartu berri den paketearen id-a baliatuz. Behin dena eginda, edo prozesuan zehar erroreren bat gertatzen bada, `InsertPackage.php` orrialdeak `Insert.php`-tik erantzun bat jasoko du eta honek erabiltzaileari adierazteaz arduratuko da, `Hasiera.php`-ra bueltatu baino lehen.

- `ListPackages.php`: Paketeak ateratzeaz arduratuko da. Orrialde honetara bakarrik erabiltzaile arruntak sartu ahal izango dira.

  Orrialde honek erabiltzailearen izenean gordeta dauden pakete guztiak aurkeztuko dizkio. Izan ere, pakete bakoitza bere ezaugarriekin aurkeztuko da, erabiltzaileak atera nahi duena aukeratu ahal izateko. Pakete bakoitzak bere botoia izango du, erabiltzaileak botoi honen bitartez paketea aukeratu ahal izateko.

  Pakete bat aukeratzean, pakete bat sartzerakoan gertatzen den moduan, `ExtExe` karpetan dagoen `ExtractPackage.php` funtzioari bidaliko zaio atera nahi den paketearen informazioa. Funtzio horrek jaso duen informazioarekin, biltegiarekin (Arduino Megarekin) komunikatuko den `extract.exe` aplikazioari deituko dio eta paketea ateratzeko eskatuko zaio biltegiratze-sistemari. Prozesua bukatzean `ExtractPackage.php` funtzioak atera den paketea datu-basetik ezabatuko du eta lortutako erantzuna, `ListPackages.php`-ri bidaliko dio. Honek jaso eta erabiltzaileari aurkeztuko dio, `Hasiera.php`-ra bueltatu baino lehen.

- `ListPackagesAdmin.php`: `ListPackages.php`-ren funtzionamendu berdina izango du, baina honetan administratzaileak bakarrik sartu ahal izango dira, eta erabiltzaile baten paketeak ikusi beharrean, sisteman gordeta dauden pakete guztiak ikusi eta atera ahal izango dituzte.

- `ListUsersAdmin.php`: Funtzio hau ere, administratzailearen esku bakarrik geratuko da. Honetan sisteman erregistratuta dauden erabiltzaile guztiak agertuko dira. Ondorioz, administratzaileak aukeratutako erabiltzaile bat ezabatu ahal izango du sistematik, beti ere erabiltzaile horrek sisteman paketerik ez badu. Horrela bada, erabiltzaile horren izenean dauden pakete guztiak atera beharko dira, gero erabiltzaile hori ezabatu ahal izateko.

7.1.1.3.4 Funtzioak

Bukatzeko `Funtzioak` karpeta aztertuko dugu. Honetan gure web aplikazioak egin behar dituen funtzio batzuk aurrera eramaten duten PHP funtzio batzuk daude. Hauek erabiltzaileak ez ditu eskuragarri edukiko, bakarrik web orrialdeak atzitu ahal izango ditu. Hauek aurreko atalan ikusi

ditugun orrialdeak baliatu egingo dituzte, bereziki datu-basearekin egin beharreko komunikazio aurrera eramateko.

- `Erregistratu.php`: Orrialde atalean aipatu dugun moduan, erabiltzaileak `SignUp.html` eta `EgiaztapenKodea.php` orrialdeen formularioak bete dituenean, `Erregistratu.php`-ri deitzen zaio, formularioaren informazioa bidaliz. Informazioa datu-basean sartzeaz arduratuko da.

- `Logout.php`: Behin erabiltzaile bat sisteman sartu dela, dagoen orrialdean dagoela, goiko eskuineko izkinan `Logut` egiteko botoi bat izango du. Honen bitartez `logout.php` funtzio honi deituko zaio, sesioa bukatuz eta `konexio` datu-basean sarrera eginez. Jarraian, `Hasiera.php` orrira birzuzenduta izango da erabiltzailea, erabiltzaile anonimoa izatera pasaz.

- `Segurtasunerantzuna.php` eta `seguratsungaldea.php`: Pasahitza berreskuratzeko prozesuan `berreskuratuPass.php` deskribatu dugun orrialdean, erabiltzaile bakoitzari dagokion galdera datu basean bilatzeko `segurtasungaldera.php` funtzioari deituko dio. Galdera zein den lortu ondoren, `berreskuratuPass.php` galdera hau bistaratuko du orrialdean. Erabiltzaileak galdera horri erantzuten dionean, `segurtasunerantzuna.php` funtzioari deituko zaio, erabiltzaileak idatzitakoa eta datu-basean gordeta dagoen erantzuna berdina diren ala ez egiaztatzeko.

  Funtzio hauei AJAX teknologia erabilita deituko zaie, dinamismo eta errendimendua hobetu ahal izateko.

- `DeleteUser.php`: Funtzio honek `ListUsersAdmin.php`-n aukeratzen den erabiltzailearen informazioa jasoko du AJAX teknika erabiltzen duen `jquery`-ko `ajax` objetuaren bitartez (3.2.2.5 atala). Honek jasotzen duen erabiltzailea ezabatuko du datu-basetik eta prozesu horren emaitzarekin erantzungo dio `ListUsersAdmin.php`-ri.

`StylesPWS` eta `Images` karpetak, web aplikazioaren itxurarako erabili diren CSS[11] eta irudiak dituzte. Proiektuaren funtzionamendurako eragina ez daukatenez, ez dira azalduko.

7.2 Sekuentzia-diagramak

Behin web aplikazioaren egitura eta hau osatzen duten osagaiak aztertuta, web aplikazioak dituen funtzio ezberdinak aurrera eramateko osagai hauen artean dagoen informazio-fluxua aztertuko dugu. Horretarako erabilera kasu bakoitzerako sekuentzia-diagrama bat egin da. Sekuentzia-diagramak aplikazioaren nora-norakoak ulertzeko oso erabilgarriak suertatu daitezke, erabilpen kasu bakoitzak aplikazioan duten eragina eta aplikazioan zehar ematen diren datu-fluxu egitura guztiak azaltzen direlako. Azalpenekin hasi baino lehen esan behar da gure kasuan web-arakatzailean ikusten ditugunaz hitz egiten dugunean, erabiltzaileak ikusiko duen web aplikazioaz hitz egiten ari gara.

7.2.1 Aplikazioan saioa hasi

Erabiltzaile bat web aplikazioan sartzen denean saioa hasi ez duen erabiltzaile moduan aurkituko da `Hasiera.php` orrian. `Hasiera.php` orri honetan `SignIn` botoia sakatzerakoan, zerbitzariari `SignIn.php` orrialdea eskatuko zaio eta honek orrialdera bidaliko gaitu. Momentu horretan erabiltzaileak web-arakatzailean `SignIn.php` orrialdea ikusiko du. Lehen aztertu dugun moduan

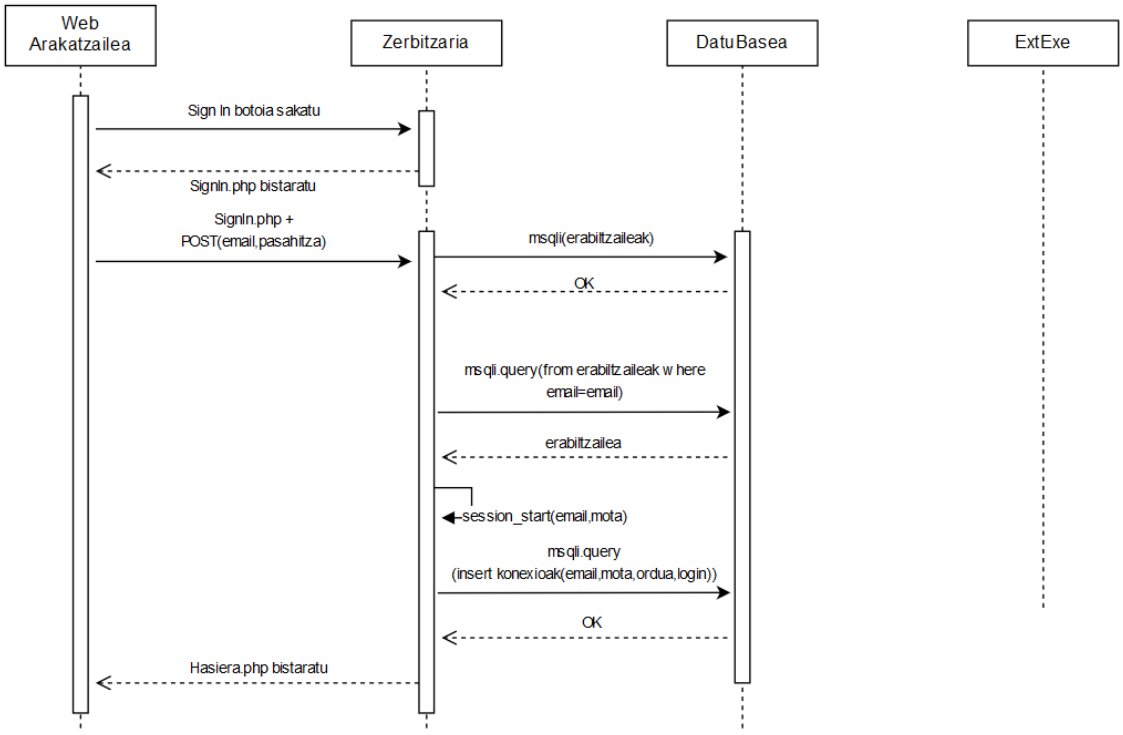
[11] Kaskadako estilo-orriak edo CSS (ingelesez, Cascading Style Sheets) HTML edo XML (beraz, XHTML ere bai) lengoaian egituratutako dokumentu baten aurkezpena definitzeko balio duen lengoaia da. W3C da beronen definizioa mantentzen duen erakundea.

`SignIn.php` orrialdeak sisteman saioa hasteko beharrezkoak diren email eta pasahitza eremuak dituen formulario bat izango du. Eremu hauek informazio egokiarekin bete eta `Bidali` botoia sakatuko dugu, sisteman saioa hasteko.

`Bidali` botoia sakatzerakoan `SignIn.php` orrialdera deituko dio berriro ere, baina kasu honetan POST metodoaren bitartez erabiltzaileak betetako eremuko informazioa (email-a, pasahitza) bidaliko zaio. Orduan, zerbitzariak, datu-basearekin konexioa egingo du eta erabiltzaileak pasatako email-a duen erabiltzailea eskatuko dio datu-baseri.

Zerbitzariak lortu duen informazioarekin erabiltzailea existitzen dela eta sartutako pasahitza zuzena dela egiaztatuko du. Orduan, zerbitzariak `session_start()` funtzioarekin saio berri bat sortuko du, email-a eta erabiltzaile-mota parametro bezala dituela. `Session` objektu honen bitartez gure web aplikazioak jakingo du sesio bat ireki egin dela eta zer nolako ezaugarriak dituen sesio berri horrek.

Saioa hasita, sarrera berri bat sartuko da `konexioak` taulan, behar den informazioarekin (email, mota, ordua, login). Hau egin ondoren, zerbitzariak `Hasiera.php` orrialdera bidaliko dio web-arakatzaileari, sisteman sartu berri den erabiltzaile horren baimenak ahalbidetzen dituen aukerak bistaratuz. Lehen esan dugun moduan, `Hasiera.php`-k eduki ezberdinak izango ditu erabiltzaile motaren arabera. Guzti hau modu grafikoan ikusi daiteke 7.11 irudian.



7.11. Irudia: Saioa hasi sekuentzia-diagrama

7.2.2 Erregistratu sisteman

Aplikazioaren diseinua aztertzean azaldu dugun moduan, erregistratzeko bi modu daude. Batetik erabiltzaile arrunt moduan erregistratzea eta, bestetik, banatzaile moduan erregistratzea. Biak sekuentzia diagrama ia berdina izan arren, banaka azaltzea erabaki da, duten ezberdintasuna esanguratsua baita.

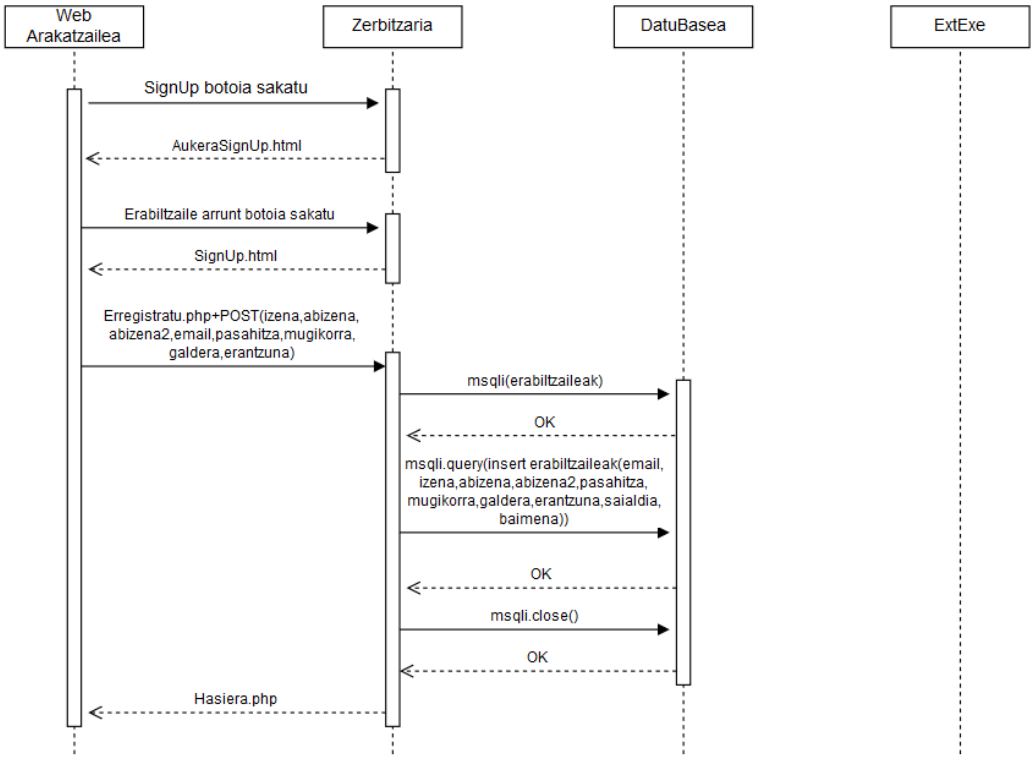
Hasteko erabiltzaile arrunt batek erregistratzean jarraitzen den sekuentzia diagrama aztertuko dugu, 7.12 irudiaren laguntzaz. Erabiltzailea, web-arakatzailean, `Hasiera.php` orrian sisteman

sartu gabe aurkitzen da. Honetan `SignUp` botoia sakatzen badu, `AukeraSignUp.html` orrialdea eskatuko dio zerbitzariari.

`AukeraSignUp.html` honetan bi botoi nagusi izango ditugu, erabiltzaile arrunt edo banatzaile moduan erregistratzearen artean aukeratzeko balio digutenak. Erabiltzaileak erabiltzaile arrunt botoia sakatzen badu, zerbitzariari `SignUp.hmtl` orria eskatuko zaio eta zerbitzariak hori bidaliko digu. Behin erabiltzaileak `SignUp.html` orria ikusi ditzakenean, bertan ageri den formularioa bete beharko du. Esan dugun moduan, orrialde berak informazio guztia sartu dela eta informazioak egitura ona daukan egiaztatzeko mekanismoak ditu. Bertan ageri den formularioak 7.2.1.3 atalean ikusi dugun informazioa eskatuko dio erabiltzaileari.

Informazioa sartu ondoren, `Bidali` botoia sakatzean, informazioak formatu egokia daukala bermatuko du eta jarraian `Erregistratu.php` funtzioari deituko zaio, formularioaren parametroak POST metodoaren bitartez pasatuz.

Zerbitzarian `Erregistratu.php` funtzioak POST bitartez jaso dituen parametroak erabilita datu-baseko `erabiltzaile` taulan sarrera berri bat sartuko du. Datu-basearekin komunikazio osoa bukatzean, zerbitzariak `Hasiera.php` orrira birbidaliko digu, web-arakatzailean erregistroa ondo joan dela jakinaraziz.

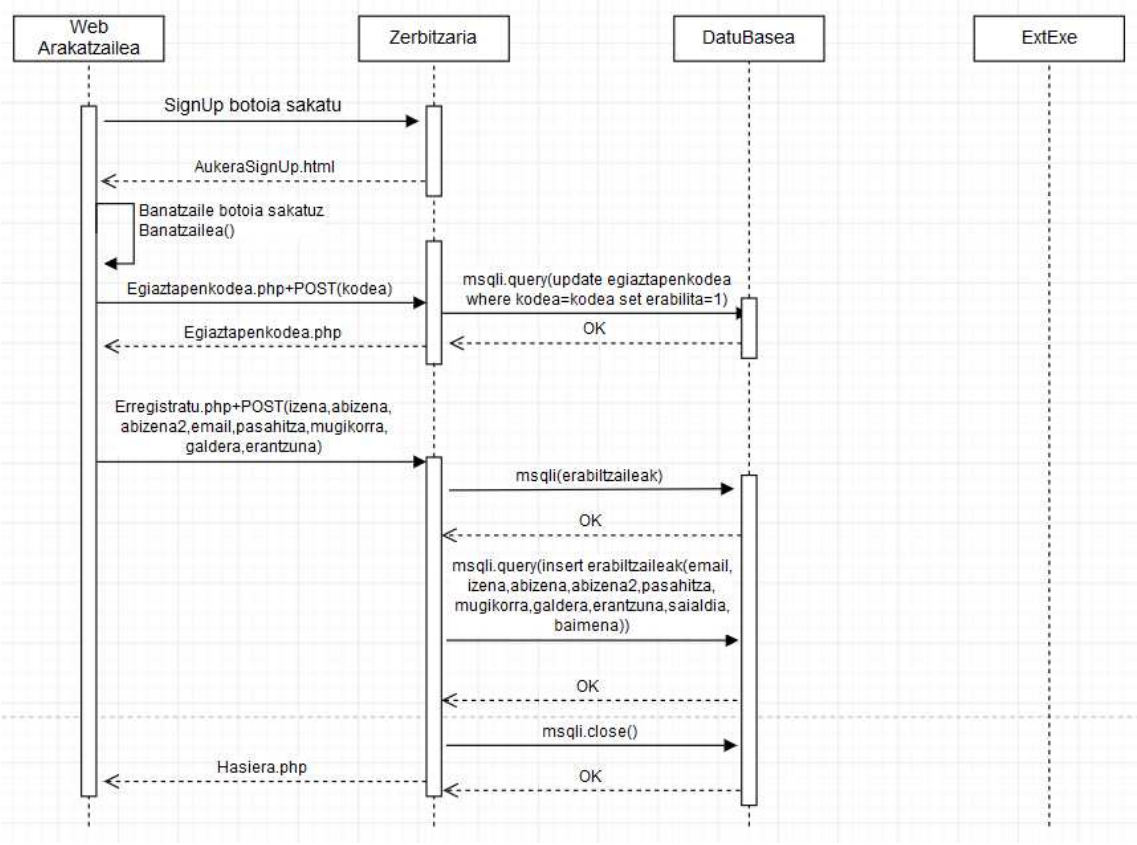


7.12. Irudia: Erabiltzaile arrunt erregistroa

Bestetik 7.13. irudian banatzaile batek erregistratzean emango den fluxu-diagrama ikusiko dugu. Erabiltzaile arruntaren kasuan bezala, `Hasiera.php` orriko `SignUp` botoia sakatuz hasiko da fluxua. `SignUp` botoia sakatzerakoan `AukeraSignUp.html` orrira eramango gaitu aplikazioak. Honetan `Erabiltzaile` botoia sakatu beharrean, `Banatzaile` botoia sakatuko dugu. Botoia sakatzerakoan orrialde bertan Javascript bitartez definituta dagoen `Banatzaile()` funtzioari deituko zaio eta, ondorioz, orrialde behekaldean eremu berri bat bistaratuko da. Honetan erabiltzaileak banatzaile moduan erregistratzeko beharrezkoa den egiaztapen kodea sartu beharko du.

Kodea idatzi eta `Bidali` botoia sakatzean, zerbitzariari `EgiaztapenKodea.php` orria eskatuko zaio POST parametro bezala, erabiltzaileak bete berri duen kodea pasatuz. Behin kodea egiaztatu eta egiaztapen kodea datu-basean "erabilia" bezala ezarri delarik, web arakatzailera bidaliko da `EgiaztapenKodea.php`. Orrialde hau, erabiltzaile arrunt-ek bete beharrezko formularioaren antzekoa da, baina kasu honetan eremu batzuk aurretik beteta agertuko zaizkio erabiltzaileari.

Behin formularioa betetzen denean, `Erregistratu.php` funtzioari deituko zaio formularioaren eremuak POST bitartez bidaliz. Hemendik aurrera erabiltzaile arrunten prozesu berdina jarraituko da.



7.13. Irudia: Banatzaileak erregistratu

7.2.3 Pasahitz berri bat definitu

Hurrengo diagramak azaltzen hasi baino lehen, hemendik aurrera datu-basera egingo diren eskaerak bakarrik irudikatuko direla jakinarazten dugu, hau da datu-basearekin egin beharreko konexioak ez dira adieraziko. Datu-basearekin konexioa ezarri eta ixteko deiak 7.11 eta 7.12 irudietan ikusten den moduan egiten dira beti, ondorioz, eskemak sinplifikatzearren, dei hauek hemendik aurrera ez dira adieraziko.

Jarraian deskribatuko dugun prozesuaren sekuentzia-diagrama 7.14 irudian ikus daiteke.

Erabiltzaile batek pasahitz berria ezarri nahi duenean edo zeukana ahaztu duenean, hurrengo sekuentzia jarraitu beharko du berri bat lortzeko. Erabiltzaileak `Hasiera.php` orrian dagoela, `Login` botoia sakatu ondoren, zerbitzariak `SignIn.php` orria birbidaliko du.

Honetan, email eta pasahitza sartzeko eremuen azpian `Pasahitza ahaztu duzu?` esteka aurkituko du erabiltzaileak. Hau sakatzerakoan, zerbitzariari `berreskuratuPass.php` orria eskatuko zaio.

Behin erabiltzaileak orri hori ikusgai duela, bere web-arakatzailean, pasahitz berria ezartzeko prozesua hasiko da. Hasieran, erabiltzaileak eremu bakarra ikusiko du. Eremu honetan aldatu nahi den kontuaren email-a idatzi beharko du. Hau bete ondoren eta `Bilatu` botoia sakatuz, orrialdea AJAX teknologiaren bitartez komunikatuko da zerbitzariarekin.

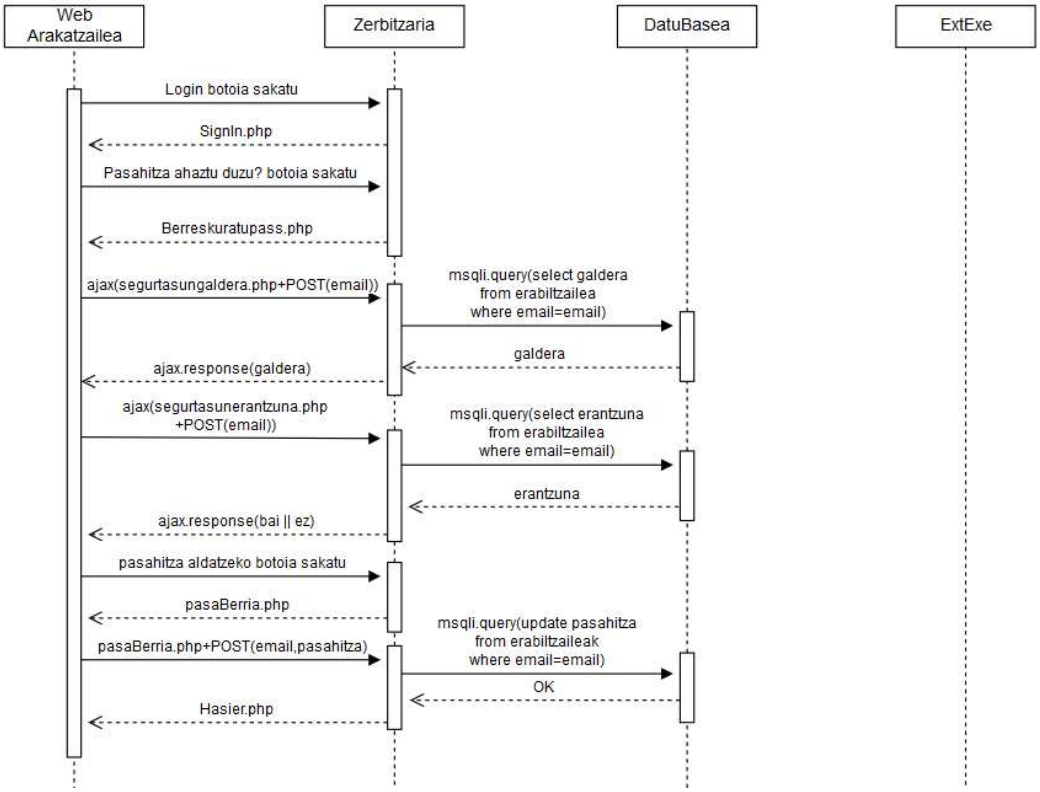
Beraz, AJAX teknika erabiltzen duen `XMLHttpRequest` objektuaren bitartez, erabiltzaileak idatzitako email-aren kontuan definitutako segurtasun galdera eskatuko dio. Horretarako `segurtasungaldera.php` funtzioari deituko zaio, POST bitartez email-a pasaz. Funtzioak, orduan, galdera lortzeko datu-basera deia egingo du eta galdera daukanean zuzenean orrialdeari pasatuko dio.

`berreskuratuPass.php` orrialdeak erantzuna jasotzean bi eremu berri ikusgai jarriko ditu. Bata lortu berri duen segurtasun galderarekin, eta bestea, erabiltzaileak segurtasun gadera horretarako erantzunarekin bete beharko duen eremua.

Erabiltzaileak erantzun eremua betetzean eta `Balidatu` botoia sakatzerakoan, berriro ere `XMLHttpRequest` objektu baten bitartez zerbitzariarekin komunikatuko da. Kasu honetan `segurtasunerantzuna.php` funtzioari deituko zaio, parametro bezala erantzuna pasatuz. Orduan `segurtasunerantzuna.php` funtzioak datu-baseari kontu horri dagokion segurtasun galderako erantzuna eskatuko dio eta datu-basetik lortutako erantzuna eta erabiltzailetik lortutako erantzuna konparatuko ditu. Konparazio horren emaitza bidaliko dio bueltan `berreskuratuPass.php` orriari. Ezezkoa bada `Hasiera.php` orrialdera birzuzendua izango da erabiltzailea. Aldiz, baiezko bada, pasahitza aldatzeaz ziur dagoen edo ez galdetuko zaio erabiltzaileari.

Beraz, erabiltzaileak pasahitza aldatu nahi badu, zerbitzariari `pasaBerri.php` orrialdea eskatuko dio. Honetan, erabiltzaileak pasahitza berria sartuko du eta balidazio baten ondoren, `pasaBerri.php` orrialdea eskatuko zaio zerbitzariari berriro ere, POST bitartez email eta pasahitza berria bidaliz. Orduan zerbitzariak bi parametro hauek hartuta, email horretako kontuan datu-basean daukan pasahitz zaharra, pasahitz berri batengatik aldatuko du. Ondoren, prozesua bukatutzat emango da eta `Hasiera.php` orrira birzuzendua izango da erabiltzailea.

AJAX teknologia ahalbidetzen du eskaera guzti horiek atzeko plano batean egitea, erabiltzailearen esperientzia hobetuz.



7.14. Irudia: Pasahitz berria ezartzeko sekuentzia-diagrama

7.2.4 Pakete bat sartu

Nahiz eta erabiltzaile mota ezberdinek pakete bat sartzeko aukera izan, kasu guztietan aplikazioak jarraituko duen eskema berdina da. Erabiltzaileak `Hasiera.php` orrialdean pakete bat sartzeko aukera ikusgai izateko, sisteman sartu egin beharko da bere kredentzialak erabiliz. Sisteman sartu ondoren, `Hasiera.php` orrian `InsertPackage` esteka ikusgai izango du. Prozesu osoaren sekuentzia-diagrama 7.15 irudian ikusi dezakegu.

Esteka hau sakatuz gero, zerbitzariak `InsertPackages.php` orrira bidaliko digu. `InsertPackages.php` orria aztertu dugunean ikusi dugu nola formulario egitura daukan, 3 eremuz osatuta, banatzailea, hartzailea eta deskripzio eremuak, alegia. Banatzaile eremua jadanik beteta etorriko zaigu, gure email eta erabiltzaile izenekin. Horretarako zerbitzariak orria bidali baino lehen datu-basera dei bat egingo du, erabiltzailearen izena lortzeko.

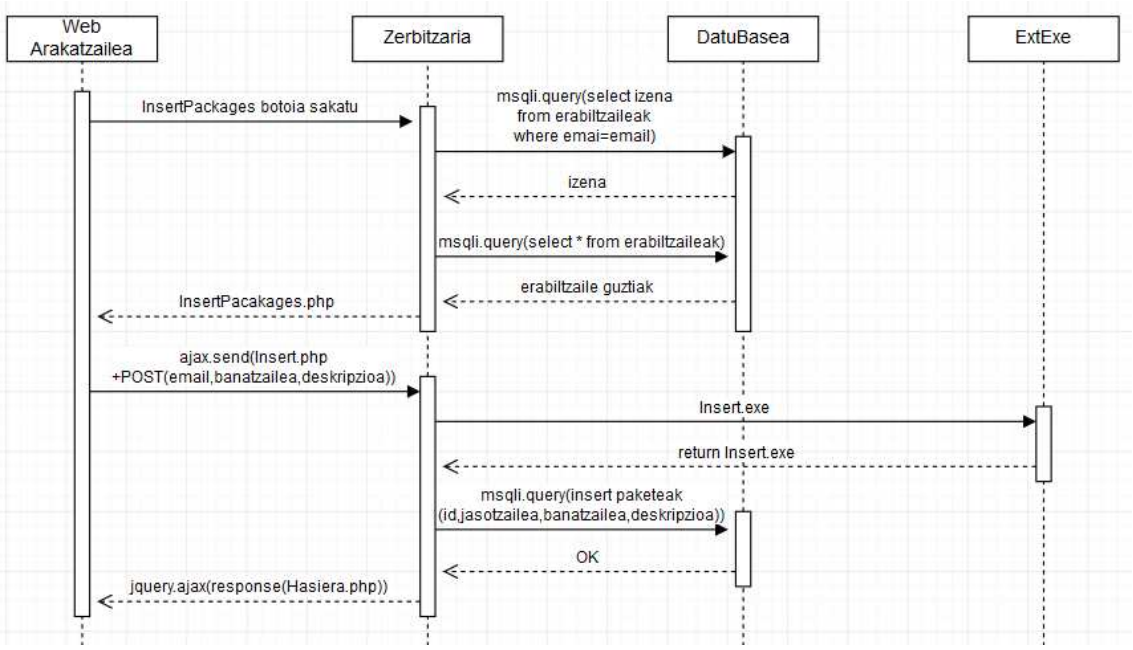
Bestalde, jasotzaile eremua zerrenda hedagarri bat izango da, sisteman dauden erabiltzaile arrunt guztien email-ak osatuta. Zerrenda hau betetzeko, datu-baserako bigarren deia egiten da, erabiltzaile arrunt guztiak lortzeko.

Behin informazio guztia bildu duela, zerbitzariak `InsertPackages.php` orria osatu eta web-arakatzaileari bidaliko dio. Erabiltzaileak formularioa bete duenean eta `Insert` botoia sakatzen duenean `jquery.ajax` objektuaren bitartez, `Insert.php` funtzioari deituko dio POST metodoaren bitartez lehen ikusi ditugun 3 eremuetan dauden balioak pasatuz.

Zerbitzarian `Insert.php` honek `ExtExe` karpetan dagoen `Insert.exe` programari deituko dio. Hau, lehen zehaztu dugun moduan, Arduino Megarekin komunikatuko da, sisteman pakete berri bat sartzeko agindua bidaliz. Hauen arteko komunikazioa 6.2.4 atalan zehaztu dugu. Behin biltegiratze-sisteman prozesua eta komunikazioa bukatu direnean `Insert.exe` bukatuko da. `Insert.exe` bueltatuko duen balioa `Insert.php` prozesatuko du (honela sisteman pakete bat

sartzeko prozesua nola joan den jakingo dugu) eta `rfid.txt` fitxategian `Insert.exe`-k idatzitako sartu berri den paketearen `id` balioa irakurriko du.

Prozesua ondo joan bada, `Insert.php` pakete berriaren datuak sartuko ditu datu-basean. Sartuko dituen balioak, `InsertPackages.php` orrialdean erabiltzaileak betetako formularioaren hiru eremuak gehi biltegiratze-sistematik lortu den paketearen id-a izango dira. Ondoren, `Hasiera.php` orrialdera birzuzendua izango da erabiltzailea, orrialdean dena ondo joan dela mezu baten bitartez adieraziz.



7.15. Irudia: Pakete bat sartzeko sekuentzia-diagrama

7.2.5 Pakete bat atera

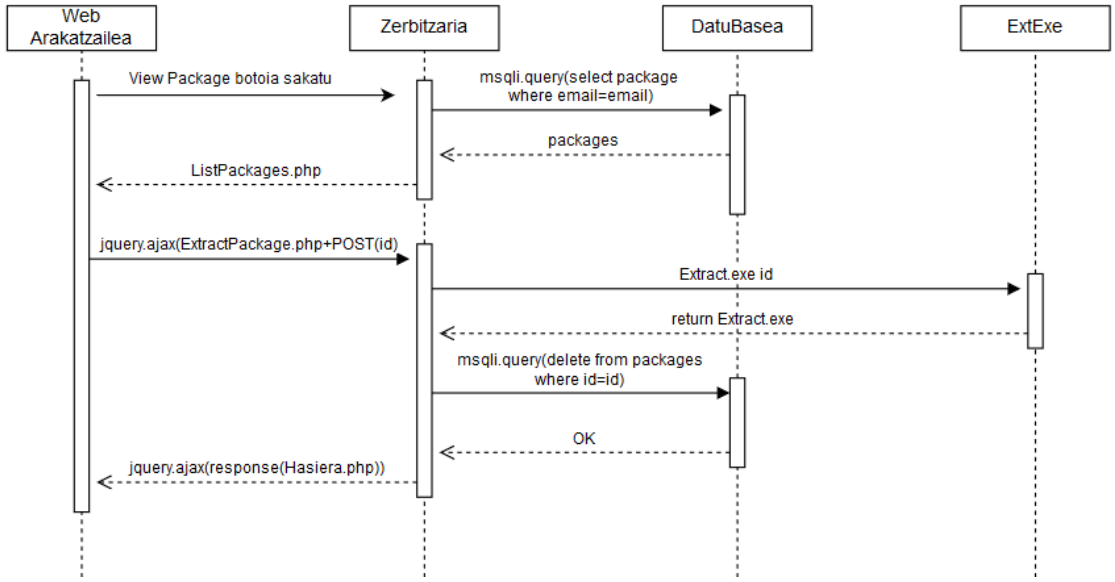
Pakete bat ateratzeko funtzionalitatea erabiltzaile mota ezberdinak eskuragarri daukate, hain zuzen ere, erabiltzaile arruntek eta administratzaileek. Hala ere, aurreko kasuan egin dugun moduan nahiz eta prozesua erabiltzaile motaren arabera pixka bat aldatu, bakarrik kasu baten sekuentzia-diagrama ikusiko dugu (7.16 irudia).

Erabiltzaileak web aplikazioan sartuta daudelarik, `Hasiera.php` orrian `ViewPackages` botoia sakatuz, zerbitzariari erabiltzaile motaren arabera `ListPackages.php` edo `ListPackagesAdmin.php` orriak eskatuko zaizkio. `ListPackages.php`-n erabiltzaile arruntak bere izenean gordeta dituen paketeak zerrenda moduan bistaratuko dira, `ListPackagesAdmin.php` berriz, administratzaileak sisteman dauden pakete guztiak ikusiko ditu. Zerrenda hau osatu ahal izateko, zerbitzariak paketeen informazioa eskatuko dio datu-baseari. Paketeen informazioa izanda, zerrenda osatuko da eta `ListPackages.php` orria web-arakatzaileari bidaliko zaio.

Behin erabiltzaileak `ListPackages.php` orria ikusgai duela, atera nahi duen paketea aukeratu beharko du. Ikusgai dagoen zerrendaren errenkada bakoitzean pakete baten informazioa aurkituko da. Honekin batera, pakete hori ateratzeko `Atera` botoi bat dago ere. Edozein paketeren botoia sakatzean, orrialdeak `jquery.ajax` objektu bat osatuko du. Honela, komunikazioa sortuko da zerbitzariarekin. Komunikazio honen bitartez zerbitzariaren `ExtractPackage.php` funtzioari dei egingo zaio POST bitartez, paketearen id (RFID identifikadorea) bidaliz.

`ExtracPackage.php` funtzioak `ExtExe`-n aurkitzen den `extract.exe` programari deituko dio, parametro moduan atera nahi den paketearen id-a pasaz. Pakete bat sartzeko kasuan bezala,

`extract.exe` hau izango da biltegiratze-sistemarekin komunikatuko dena, Arduino Megari paketea ateratzeko agindua bidaliz. Fisikoki paketea sistematik atera denean, `ExtractPackage.php` `extract.exe`-k bueltatutako balioa aztertuko du. Dena ondo joan bada, `ExtractPackage.php` atera berri den paketea datu-basetik kentzeko eskaera egingo du. Datu-baseak paketea ezabatzean prozesu guztia ondo joan dela adieraziko du eta web-arakatzaileari lehen `jquery.ajax` objetuaren eskaerari erantzun moduan, dena ondo joan dela adierazteko mezua bidaliko dio. Ondoren, `Hasiera.php` orria birbidaliz.



7.16. Irudia: Pakete bat ateratzeko sekuentzia-diagrama

7.2.6 Erabiltzaile bat sistematik ezabatu

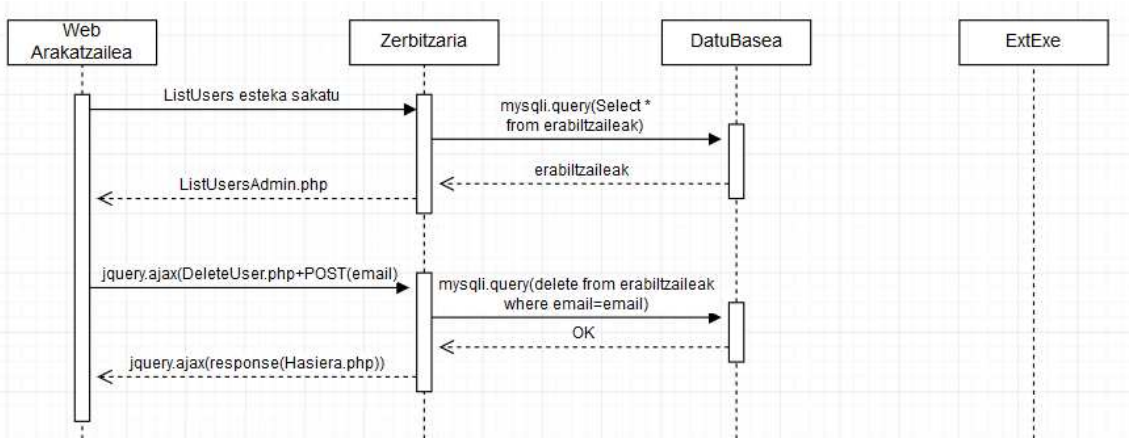
Funtzio hau, bakarrik administratzaile baimena daukaten erabiltzaileek izango dute eskuragarri. Funtzio honen sekuentzia-diagrama 7.17. irudian daukagu.

Administratzaileak `ListUsers` esteka sakatzerakoan web-arakatzaileak zerbitzariari `ListUsersAdmin.php` orria eskatuko dio. Zerbitzariak, orri hau osatzeko erabiltzaile guztiak behar ditu eta horretarako datu-baseari erabiltzaile guztien informazioa eskatuko dio.

Behin erabiltzaile guztiak dituela, erabiltzaile guztiekin zerrenda bat osatuko du `ListUsersAdmin.php` orrian eta web-arakatzaileari bidaliko dio.

Zerrendako errenkada bakoitzak erabiltzaile baten informazioa bistaratuko du. Honetaz gain, errekanda bakoitzaren azkengo elementua erabiltzaile hori ezabatzeko erabiliko den `Kendu` botoi aurkituko dugu. Erabiltzaileak zerrenda ikusita, ezabatu nahi duen erabiltzailearen `Kendu` botoia sakatzen badu, `jquery.ajax` objektua osatuko da, zerbitzariko `DeleteUsers.php` funtzioari POST bitartez aukeratu den erabiltzailearen email-a pasaz.

`DeleteUsers.php` honek jaso duen email-aren izenean paketerik gordeta dauden edo ez egiaztatuko du. Paketeren bat gordeta badauka, erabiltzailea ezabatu aurretik pakete hori atera beharko da sistematik. Aldiz, paketerik gordeta ez badu, erabiltzailea sistematik ezabatzeko eskaera egingo dio datu baseari. Erabiltzailea datu-basetik ezabatzean, `ajax.jquery` objketuari erantzun zuzenarekin erantzungo zaio. Jarraian, `Hasiera.php` orrira birbidaliz.



7.17. Irudia: Erabiltzaile bat ezabatzeko sekuentzia-diagrama

8. Web aplikazioaren inplementazioa

Atal honetan web aplikazioaren inplementazioa aztertuko dugu. Kontuan hartu behar dugu inplementazioa aurrera eramateko 7. kapituluan ikusi dugun diseinua gida moduan erabili dugula eta, beraz, 7. kapituluan azaldu eta ikusi ditugun hainbat atal berrikusiko ditugu.

Inplementazioa modu orokor eta antolatu batean ikusteko, erabiltzaile motaren arabera banatuko dugu azalpena. Hasiko gara oraindikan sisteman sartuta ez dauden erabiltzaileekin, erabiltzaile arruntekin jarraituz, banatzaileetara eta azkenik administratzaileak aztertuz.

Web aplikazioaren garapen guztia azaltzea luzeegia geratuko litzatekeenez; aplikazioaren puntu kritikoenak aztertuko ditugu, funtzionamendu orokorra ulertzeko. Hala ere, kodean ulertzeko erabilgarriak suertatu daitezkeen iruzkinak aurkitu daitezke.

8.1 Erregistratu gabeko erabiltzailea

Beti ere edonor web aplikazioan sartzerakoan `Hasiera.php` orrira sartuko da. 7. kapituluan ikusi dugun moduan. `Hasiera.php` orrialde honek aukera ezberdinak ikusgai utziko ditu, erabiltzaile motaren arabera. Beraz, orrialdeak egitura berdina mantenduko du, nahiz eta erabiltzaile ezberdinen arabera agertzen diren aukerak ezberdinak izan.

Erabiltzaile batek sesioa irekita daukan edo ez eta zer motako erabiltzailea den jakiteko PHP lengoiak eskuragarri duen `session` aldagaia erabiliko dugu.

Web aplikazioan sartzerakoan erabiltzaile batek 8.1. irudiko ikuspegia izango du. Menu sinple bat daukagu, erabiltzaileak dituen aukera ezberdinekin:

- `Login`: Jadanik sisteman erregistratuta dagoenak, bere kontuarekin sesio bat hasteko erabiliko duena.
- `SignUp`: Oraindik sisteman erregistratuta ez dagoen erabiltzaile batek sisteman kontu berri bat sortzeko erabiliko duena.

8.1. Irudia: Hasiera.php, sesio ireki gabeko erabiltzailea

Bistaraketari dagokion inplementazioa 8.1 kodean ikusi daiteke.

```
session_start();
if(isset($_SESSION['baimena'])&&$_SESSION['baimena']=="Admin"){
    echo'<span class="right"><a href="ListUsersAdmin.php">List Users</a></span>';
    echo'   ';
    echo'<span class="right"><a href="ListPackagesAdmin.php">List Packages</a></span>';
    echo'   ';
    echo'<span class="right" ><a href="../Funtzioak/logout.php">Logout</a></span>';
```

```
}else if(isset($_SESSION['baimena'])&&$_SESSION['baimena']=="ikasle"){
    echo'<span class="right"><a href="InsertPackages.php">Insert Packages</a></span>';
    echo'   ';
    echo'<span class="right"><a href="ListPackages.php">View Packages</a></span>';
    echo'   ';
    echo'<span class="right" ><a href="../Funtzioak/logout.php">Logout</a></span>';

}else if(isset($_SESSION['baimena'])&&$_SESSION['baimena']=="Banatzaile"){
    echo'<span class="right"><a href="InsertPackages.php">Insert Packages</a></span>';
    echo'   ';
    echo'<span class="right" ><a href="../Funtzioak/logout.php">Logout</a></span>';

}else{
    echo'<span class="right"><a href="signIn.php">Login</a></span>';
    echo'   ';
    echo'<span class="right"><a href="AukeraSignUp.html">SignUp</a></span>';
}
```

8.1. Kodea: Hasiera.php

Ikus daitekeen moduan lehenik eta behin `session_start ()` funtzioari deitzen zaio, `session` aldagaia erabili ahal izateko. Hemendik aurrera, lau kasu posible daude. Lehenik eta behin, `session` aldagaia ezarrita dagoen edo ez aztertuko da eta, horrela bada, sesioa zer nolako baimenak dituen begiratuko da. Kasuaren arabera `Hasiera.php` orrian eskuragarri egongo diren aukerak ezberdinak izango dira.

Sesioa hasi ez duen erabiltzaile bat garenez azkenengo kasuan gaude. Azkenengo kasu honetan irudian agertzen diren `Login` eta `SignUp` estekak aurkitzen ditugu.

8.1.1 Sisteman sartu

Erabiltzaileak 8.1 irudian ikusten dugun `Login` esteka klikatzen badu, `SignIn.php` orrira birbidalia izango da. Honetan, erabiltzaileak sesio bat hasteko aukera izango du. Horretarako kontuaren email-a eta pasahitza sartu beharko ditu. Informazio sarrera hau 8.2 irudian ikusi dezakegun formularioaren bitartez egingo da.

Honetaz aparte, `Atzera` botoi bat daukagu, berriro ere `Hasiera.php` orrira bueltatzeko. Bestetik, `Bidali` botoia daukagu, behin email eta pasahitza eremuak bete ditugularik sesioa hasteko erabiliko dena. Bukatzeko, `Pasahitza ahaztu duzu?` esteka daukagu, gure kontuan pasahitza berri bat ezartzeko balio diguna.

8.2. Irudia: SignIn.php

`Bidali` botoia sakatzerakoan `SignIn.php` orrira bidaliko dira formularioaren balioak, POST metodoaren bitartez. Kontuan izan behar dugu orri honek PHP-n inplementatutako atal bat daukala. Atal honek POST parametroaz email-a eta pasahitza jasotzen dituenean zerbitzarian

hainbat ekintza aurrera eramango ditu. Definitutako ekintza hauek zein diren ikusteko 8.2. kodean dugu inplementazioaren sasi-kodea.

```
1       $sql=("SELECT * FROM erabiltzaileak WHERE Email='".$_POST['email']."';");

        $emaitza =  query($sql);

        // existitzen bada orduan pashitza konprobatzera pasa
2       if ($emaitza == 1){
            //aurkituta
            $user = $emaitza ->fetch_array(MYSQLI_ASSOC);
            // erabiltzaileari 3 aukera eman pasahitza ondo sartzeko
            if($user['Saialdi']<3){
                if($user['Pasahitza']==sha1($pasahitza)){
                    // pasahitza ondo badago, datu basean saialdiak 0-ra jarri
                      $sqla=("UPDATE erabiltzaileak SET Saialdi=0
WHERE Email='". $_POST['email'] ."';");

                    // sesioa hasieratu eta behar diren balioak eman, baimenan etab
3                   session_start();
                    $_SESSION['erabiltzaile']=$email;

                    if($user['Baimen']=="ikasle"){
                        $_SESSION['baimena']="ikasle";

                    }elseif ($user['Baimen']=="Admin"){
                        $_SESSION['baimena']="Admin";

                    }elseif ($user['Baimen']=="Banatzaile"){
                        $_SESSION['baimena']="Banatzaile";
                    }
                    // Konexio taularako beharrezkoak diren datuak egituratu.
4                   $data = new DateTime(null, new DateTimeZone('Europe/Madrid'));

                    $dataformat= $data->format('Y-m-d H:i:s');
                    // Log In-a noiz egin den taulan sartzeko, ordua, baimena eta zein
akzioa.
                    $mota=$_SESSION['baimena'];
                    $akzioa="LogIn";
                    // konexio taulan sartu sarrera
                    $sqlc="INSERT INTO konexioak
    (Email,Mota,Ordua,LogInOut)
VALUES('$_POST[email]','$mota','$dataformat','$akzioa')";


                    // berriro bueltatu hasierako orrialdera, baimenak ezarrita.
5                    header('Location: ./Hasiera.php');

                }else{

                //pasahitza ez badago ondo jarrita, saialdi bat gehitu eta db gorde.
                    $saialdi=$user['Saialdi']+1;
                    $sqlb=("UPDATE erabiltzaileak SET Saialdi='". $saialdi ."'
     WHERE Email='". $_POST['email'] ."';");

            }else{
                // 3 saialdi egin ditu kontua blokeatuta
            }
        }else{
            // email-a ez da existitzen
        }
```

8.2. Kodea: SignIn.php, php atalaren sasi-kodea

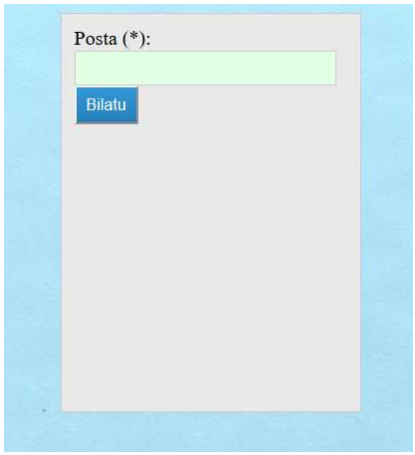
Hasteko, sasi-kodean ikusi dezakegu nola POST bitartez zeozer jaso duen edo ez aztertzen dela. POST bitartez datuak jasotzen baditu, orduan sesio berri bat sortzera pasatuko da:

1. Datu-basearekin, `mysqli` objektu baten bitartez konexioa ezarriko da. Konexioa ezarrita, datu-baseari erabiltzaile taulatik erabiltzaileak idatzitako email-arekin existitzen den kontua eskatuko dio.

2. Behin erabiltzailearen datuak datu-basetik lortu dituela, gordeta dagoen pasahitza eta erabiltzaileak emandako pasahitza konparatuko dira. Berdinak badira aurrera, ez badira berdinak saiakera balioa inkrementatu egingo da eta erabiltzaileari datuak berriro sartzeko eskatuko zaio. Gogoratu, sisteman sartzeko gehienez erabiltzaile bakoitzak hiru saiakera izango dituela. Hiru saiakera hauek erabilita, kontua blokeatu egingo da.
3. Egiaztapena egin ondoren, `session` aldagaia sortzera eta osatzera pasatuko gara, `session` aldagairen `erabiltzailea` eta `baimen` eremuak kontuaren informazioarekin betez.
4. Hau egin eta gero, sesio berri bati dagokion traza edukitzeko datu-baseko `konexioak` taulan sarrera berri bat sartuko da, beharrezkoak diren datuekin.
5. Azkenik, `Hasiera.php` orrira birbidaliak izan aurretik, erabiltzaile horrek sisteman sartzeko dituen saiakera kopurua 0-ra ezarriko da.

8.1.2 Pasahitz berria ezartzeko

8.2 irudian ikusi dugun `Pasahitza ahaztu zaizu?` esteka sakatzen badugu, `berreskuratuPass.php` orrira birbidaliak izango gara. Honetan erabiltzaileak 8.3 irudiko ikuspegia izango du. Erabiltzaileak email bat sartzeko eremua izango du, erabiltzaileak pasahitza aldatu nahi duen kontuaren email-a sartu ahal izateko.



8.3. Irudia: berreskuratuPass.php

Erabiltzaileak email eremua bete eta `Bilatu` botoia sakatzean, `berreskuratuPass.php`-n javascritp-en definituta dagoen `bilatu()` funtzioari deituko zaio. Honek `XMLHttpRequest` objektuaren bitartez, AJAX teknika erabiliz, `segurtasungaldera.php` funtzioari deituko dio, erabiltzaileak eremuan idatzitakoa POST parametro bezala bidaliz. 8.3. kodean `bilatu()` funtzioa daukagu. Honetan ikus dezakegun moduan, datuak bidali baino lehen hauek egiaztatu egiten dira, datu egokiak bakarrik bidaltzeko.

Egiaztapenak egin ondoren, `XMLHttpRequest` objektua osatuko dugu. Hemen ere, jasotzen den erantzunaren arabera egin beharrekoak definitzen ditugu. Behin dena definituta daukagula, bidali egingo zaio eskaera `segurtasungaldera.php` funtzioari.

```
function bilatu(){
    document.getElementById("erantzuna").innerHTML="";
    document.getElementById("sar").style.display = 'none';
    document.getElementById("eran").value="";
    var email=document.getElementById("posta").value;
    var re = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-
9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
1   if (email==""){
        alert("Email eramua bete behar duzu");
    }
    else if(re.test(email)==false){
```

```
        alert("Email ez dauka formatu ona");
    }else{
2       xmlhttp= new XMLHttpRequest();
        var posta=document.getElementById("posta").value;
        xmlhttp.onreadystatechange = function()
        {
3           if (xmlhttp.readyState==4 && xmlhttp.status==200){
                if(xmlhttp.responseText=='Ez'){
                    alert("Ez dago erabiltzailerik email hori daukana");
                }else{
                document.getElementById("sar").style.display = 'block';
                document.getElementById("gald").style.display="block";
                document.getElementById("gald").value=xmlhttp.responseText;
                document.getElementById("posta").disabled=true;
                document.getElementById("gald").disabled=true;
                }
            }
        }
4       xmlhttp.open("POST","../Funtzioak/segurtasungaldera.php");
        xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
        xmlhttp.send("email="+posta);
    }

    }
```

8.3. Kodea: berreskuratuPass.php-ko bilatu() funtzioa

Ondoren, kodean ikusi daiteken hainbat ezaugarri azalduko ditugu:

1. Erabiltzaileak email eremua bete duela eta formatu egokia daukala egiaztatu egingo da.
2. `XMLHttpRequest` objektua hasieratu, eta honen konfigurazioa egitera pasatuko gara.
3. Bidaliko den eskeraren erantzuna jaso egitean egin beharrekoa definituko dugu. Kasu honetan bi aukera daude. `Ez` balioa jaso, email horrekin konturik ez dagolea adieraziko duena edota, kontu horri dagokion segurtasun galdera jasotzea. Orduan, hau pantailaratzeko beharrezko funtzioak deitu egingo dira.
4. Egingo den eskaera egituratzea. Lehenik eta behin, zein funtzioari deituko zaion definituko dugu `open()` funtzioarekin. Ondoren `send()` funtzioa erabiliko dugu, `open()` funtzioan definitutako funtzioari deia eginez. Parametro moduan POST bitartez bidaliko zaion informazioa definitu dugu.

Hortaz, `segurtasungaldera.php` funtzioak POST bitartez erabiltzaileak idatzitako email-a jasoko du. Email horrekin datu-basean kontu horri dagokion segurtasun galdera bilatuko da. Email horrekin konturik existitzen ez bada, `Ez` mezua erantzungo dio `bilatu()` funtzioari. Aldiz, erantzun bezala, kontua aurkitzen bada, kontu horri dagokion segurtasun galdera erantzun bezala bidaliko dio.

`bilatu()` erantzuna jaso ondoren `berreskuratuPass.php` leihoak 8.4 irudiko itxura hartuko du.

8.4. Irudia: berreskuratuPass bilatu() funtzioari deitu ondoren.

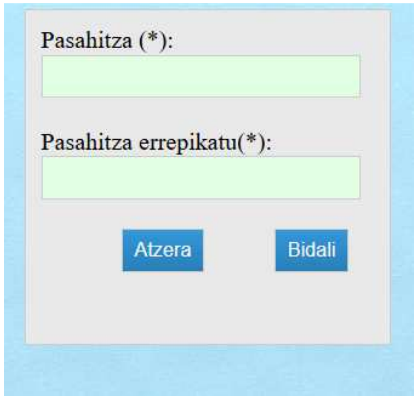
Orain ikus dezakegun moduan beste bi eremu berri agertu dira, 8.3. kodean `bilatu()` funtzioak erantzun zuzena jasotzean bistaratu dituenak, alegia.

Beraz, behin erabiltzaileak erantzuna sartzean eta `Balidatu` botoia sakatzen duenean `bilatu()` funtzioaren antzekoa den `balidatu()` funtzioari deituko dio. Honek `bilatu()`-k (8.3. kodea) egiten duen moduan `XMLHttpRequest` objektu bat sortuko du. Kasu honetan, `segurtasungaldera.php` funtzioari deitu beharrean, `segurtasunerantzuna.php`-ri deituko dio. Honi POST bitartez erabiltzaileak idatzi duen erantzuna eta email-a bidaliko dizkio.

`Seguratsunerantzuna.php` honek jasotako datuekin datu-baseari dei egingo dio kontu horri dagokion segurtasun galderaren erantzuna lortzeko. Erantzuna lortzen duenean, datu-basetik lortu berri den erantzuna eta erabiltzaileak bidalitako erantzuna konparatu egingo ditu, konparazio horren emaitza `balidatu()` funtzioari erantzun bezala bidaliz. Baiezkoa bada 8.5 irudia ikusiko dugu.

8.5. Irudia: berreskuratuPass.php erantzuna balidatu ondoren

Irudian ikus dezakegun moduan, erabiltzaileari pasahitza aldatzea nahi duen edo ez galdetuko zaio. Erabiltzaileak `Ez` botoia sakatzen badu `Hasiera.php` orrira bueltatuko da, bestela `pasaBerri.php` orrira pasatuko gara (8.6 irudia).



8.6. Irudia: pasaBerri.php

Honetan erabiltzaileak pasahitz berria sartu beharko du eta `Bidali` botoia sakatu. Hau sakatzerakoan, `pasaBerri.php` berak datu-basean pasahitz berria sartzeaz arduratuko da 8.4. kodean daukagun bezala.

```
1  $mysqli = new mysqli("localhost", "root", "", "erabiltzaileak");
     /* comprobar la conexión */
     if ($mysqli->connect_errno) {
                printf("Konexioan errore bat egon da: %s\n", $mysqli->connect_error);
                 exit();
     }
        // erabiltzaile taulan email horrekin dauden erabiltzaileak ater
2    $pasahitzaberria =("UPDATE erabiltzaileak SET Pasahitza='$pasazifra' WHERE
Email='$email'");
3    $emaitza_query=  $mysqli->query($pasahitzaberria);
     if (!$emaitza_query){
         die('Error: ' . mysqli_error($mysqli));
     }
     $mysqli->close();
```

8.4. Kodea: pasaBerri.php datu base konexioa eta update

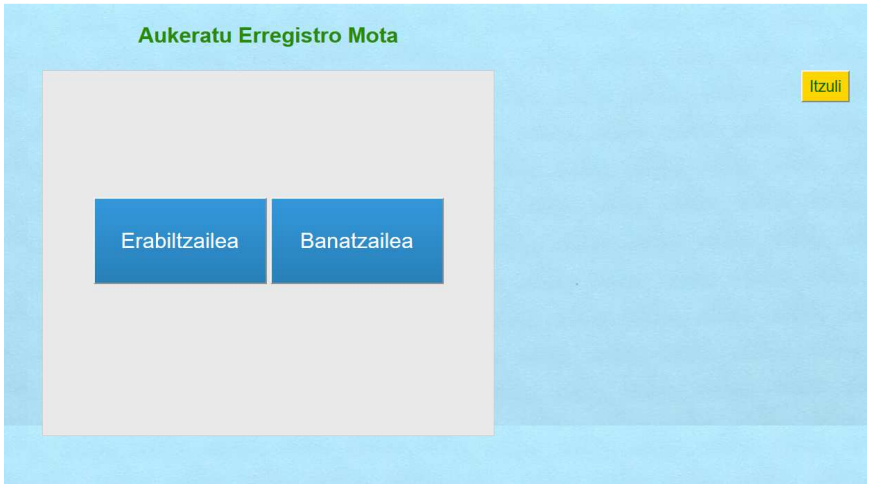
Kode honekin, aplikazioan zehar datu-basera egingo diren eskaeren adibide orokorra aztertu dezakegu.

1. Lehenik eta behin, datu-basearekin konexioa ezartzen dugu. Ikus dezakegun moduan, `erabiltzaileak` datu-basera konektatzen gara.
2. Datu-baseari egingo zaion eskaera egituratzen dugu. Kasu honetan `erabiltzaileak` taulan kontu baten pasahitza balio berri batengatik eguneratuko dugu.
3. `Query()` funtzioa erabiliko da datu-baseari eskaera egiteko. Funtzio honek eskaeraren emaitza bueltatuko du. Honetan errorerik ez badago, datu-basearekin daukagun konexioa itxi egingo dugu.

Honekin erabiltzaileak pasahitza aldatzea lortuko du.

8.1.3 Erregistratzea

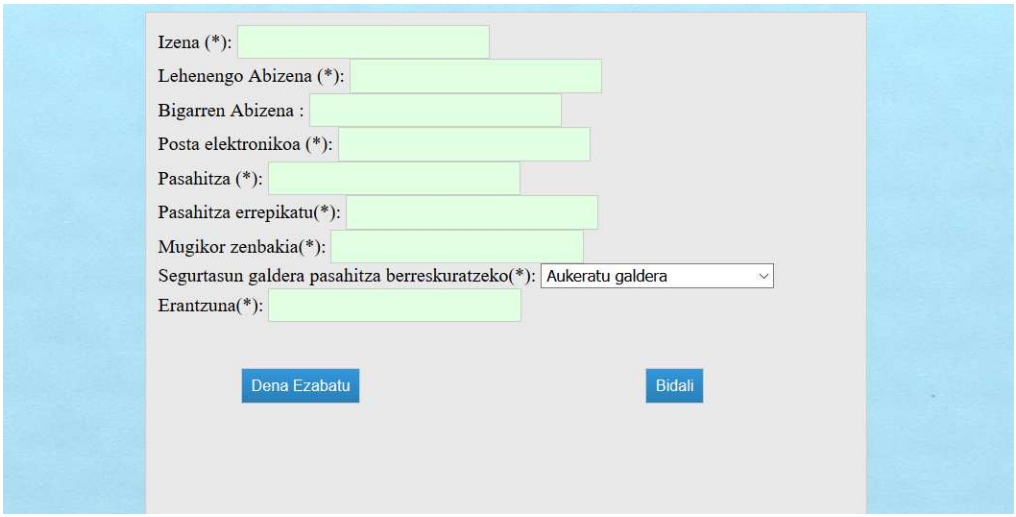
Sisteman erregistratzeko dauzkagun aukerak aztertuko ditugu orain. 8.1. irudiko `Hasiera.php` orrialdean `SignUp` esteka sakatzean `AukeraSignUp.html` (8.7.irudia) orrira birbidaliak izango gara.



8.7. Irudia: AukeraSignUp.html orrialdea

8.7 irudian ikus dezakegun moduan erabiltzaileak bi aukera izango ditu, hauen artean aukeratzeko bi botoi handi izanda. 7. kapituluan ikusi dugun moduan, erregistratzeko bi aukera daude. Erabiltzaile arrunt moduan erregistratzea (`Erabiltzailea` botoia) edo banatzaile moduan erregistratzea (`Banatzailea` botoia).

`Erabiltzailea` botoia sakatzen badugu, zuzenean `SignUp.html` orrialdera birbidaliak izango gara. Honetan formulario bat izango dugu; 8.8 irudian ikus dezakegu daukan itxura.



8.8. Irudia: SignUp.html orrialdea

Informazioa bete ondoren, erregistratzeko `Bidali` botoia sakatu beharko dugu. Horrela datu-basean sarrera sortuko duen `Erregistratu.php` funtzioari deituko zaio. Hala ere, `Erregistratu.php` funtzioari deitu aurretik `SignUp.html`-n Javascript bidez definitu dugun funtzio baten bitartez, erabiltzaileak sartutako datuak balioztuko dira.

Egiaztapen honek, lehenik eta behin, bete behar diren eremu guztiak bete diren edo ez balioztuko du, ondoren sartutako datu batzuen formatu eta egitura egiaztatuz. Egiaztapen funtzio honi `balidatu()` deitu egin eta 8.6 kodean ikus dezakegu daukan egitura.

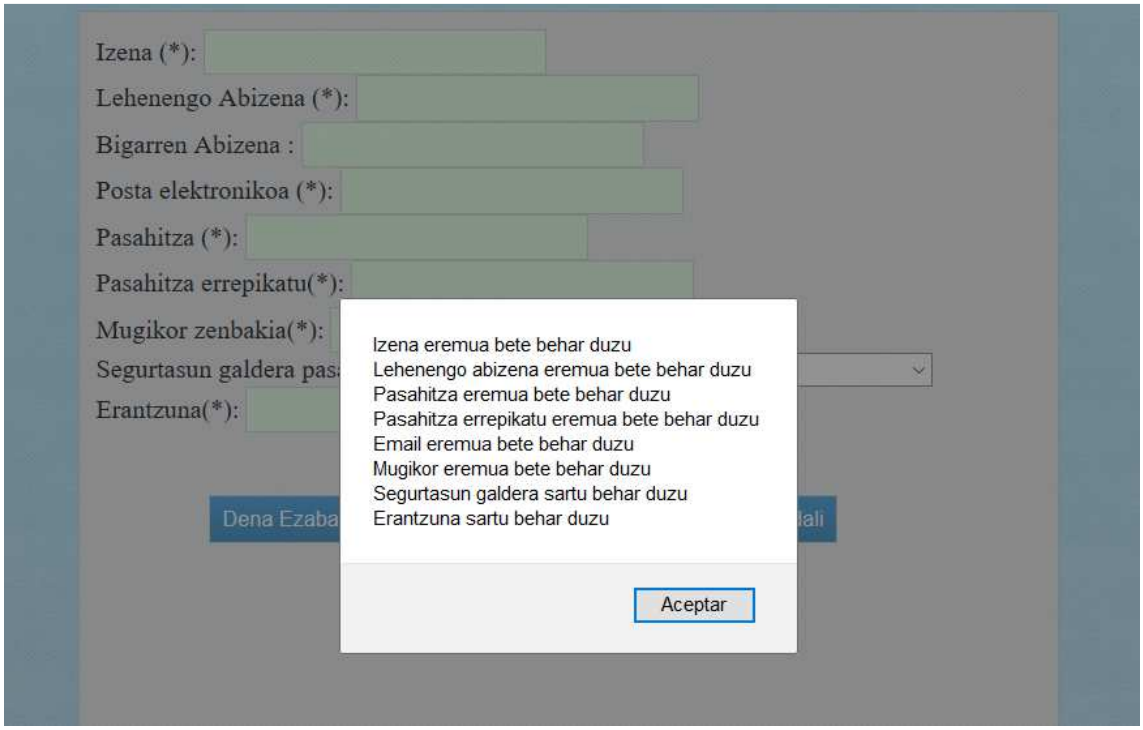
```
function balidatu(){
    var errorea="";
    var re = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-
9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
    var balidatzekomugikorra = new RegExp("^[6|9]{1}-? ?[0-9]{8}$");
    var izen = document.getElementById("izena").value;
    var abizena1 = document.getElementById("abizena1").value;
    var pasahitza = document.getElementById("pasahitza").value;
    var email = document.getElementById("email").value;
    var mugikorra = document.getElementById("mugikorra").value;
    var pasahitzaerrep=document.getElementById("pasahitzaerr").value;
    var galdera=document.getElementById("galdera").value;
    var erantzuna=document.getElementById("erantzuna").value;
    if(izen==""){
        errorea+="Izena eremua bete behar duzu \n";
    }
    if(abizena1==""){
        errorea+="Lehenengo abizena eremua bete behar duzu \n";
    }
    if(pasahitza==""){
        errorea+="Pasahitza eremua bete behar duzu \n";
    }
    if(pasahitzaerrep==""){
        errorea+="Pasahitza errepikatu eremua bete behar duzu \n";
    }
    if(pasahitza!=pasahitzaerrep){
        errorea+="Pasahitza desberdinak jarri dira\n";
    }
    if(email==""){
        errorea+="Email eremua bete behar duzu\n";
    }else{
        if(re.test(email)==false){
        errorea+="Email-a ez dauka formatu egokia \n";
        }
    }
    if(mugikorra==""){
        errorea+="Mugikor eremua bete behar duzu\n";
    }else{
        if(balidatzekomugikorra.test(mugikorra)==false){
            errorea+="Telefono eremua ondo bete behar duzu, hau da karaketererik gabe
eta 9 zenbaki \n";

        }
    }
    if(galdera==""){
        errorea+="Segurtasun galdera sartu behar duzu \n";
    }
    if(erantzuna==""){
        errorea+="Erantzuna sartu behar duzu \n";
    }
    if(errorea!=""){
        alert(errorea);
        return false;
    }
    else{
        alert("Dena ondo bete duzu, bidalketa egin da.");
        return true;
    }
}
```

8.5. Kodea: SignUp.html balidatu() funtzioa

Ikusi daitekeen moduan DOM bitartez erabiltzaileak eremu bakoitzean idatzitako balioak aldagai batean gorde egingo ditu. Jarraian, eremu bakoitza bete den edo ez egiaztatu egingo da eta bete diren hainbat eremuak beharrezko formatu daukaten egiaztatuko da ere. Balioztapen hauetan erroren bat aurkitzen bada, `errorea` aldagaian gehitzen joango dira errore bakoitzari dagokion mezua. `errorea` aldagaia utzik ailegatzen bada bukaerara eremu guztiak modu egokian bete direla adieraziko du.

Aldiz, `errorea` aldagaia errorez beteta badago, hauek pantailaratu egingo dira, eta erabiltzaileak eremuak berriro bete beharko ditu, modu zuzenean.

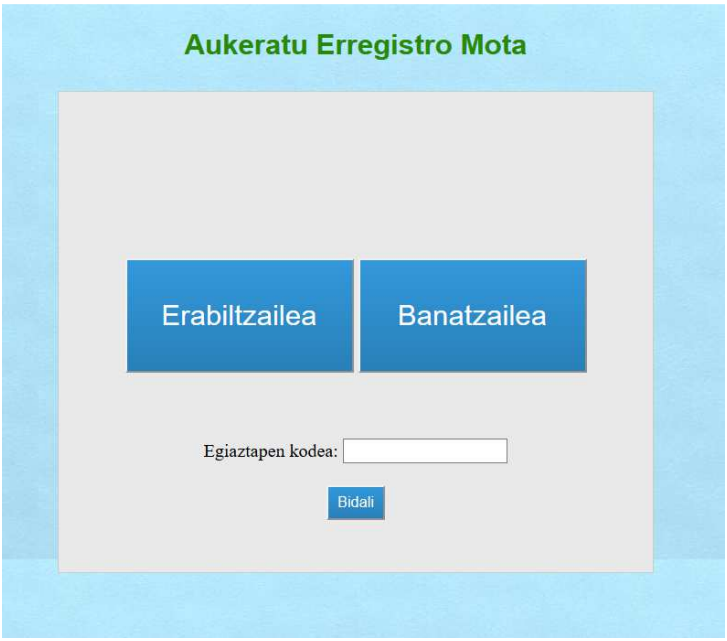
`Balidatu()` (8.6 Kodea) funtzioak eremuren batean informazioa falta bada edota izan behar ez duen egitura duela antzematean 8.9 Irudian ikus daitekeen mezua aterako du.



8.9. Irudia: SignUp.html balidapen mezua

`Balitadu()` funtzioan eremu guztiak ondo bete direla egiaztatzen bada, `Erregistratu.php` funtzioari POST bidez formularioko eremuen informazioa bidaliko zaio. `Erregistratu.php` aztertu baino lehen erregistratzeko dugun beste metodoa aztertuko dugu, biek erabiltzen baitute `Erregistratu.php` funtzioa.

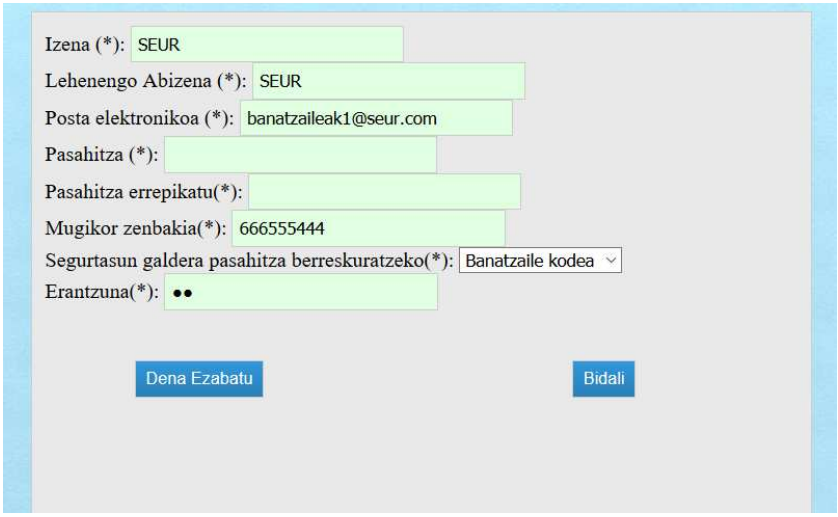
Beraz, berriro ere `AukeraSignUp.html` (8.6 irudia) orrira bueltatuko gara. Honetan `Banatzailea` aukera sakatzerakoan 8.10 irudian ikusten dugun moduan, eremu berri bat agertuko da.



8.10. Irudia: AukeraSignUp.html Banatzaile botoia sakatzerakoan

Eremu berri hau 7. kapituluan aztertu dugun moduan, banatzaileak erregistratzeko behar duten egiaztapen kodea sartzeko izango da. Kodea sartu eta `Bidali` botoia sakatzerakoan, POST bitartez erabiltzaileak idatzitako kodea `EgiaztapenKodea.php` orriari bidali dio. Hau kodearen egiaztapenaz arduratuko da, emandako kodea datu-basean daudenekin alderatuz.

Beraz, kodea baliozkoa bada `EgiaztapenKode.php` orria kargatuko da web-arakatzailean, 8.11 irudian ikusi daitekeen moduan.



8.11. Irudia: EgiaztpapenKodea.php kodea egiaztatuta

Hortaz, lehen erabiltzaile arruntaren erregistroan (8.8 irudia) ikusi dugun formulario berdina izango dugu. Hala ere, kasu honetan hasieratik pasahitza izan ezik, beste eremu guztiak beteta agertuko dira, beraien balioa aldaezina izanik. Balio hauek, kodea egiaztatzerakoan datu-basetik lortu den informaziotik lortuko dira. Gogoratu, egiaztapen kodea taulan (7.2.1.5 atalan) gordetzean enpresaren informazioa gorde egiten dela ere, hain zuzen ere, erregistroan erabiltzen den informazioa da.

Beraz, pasahitz bat definitu ondoren, `Bidali` botoia sakatzerakoan `Erregistratu.php` funtzioari deituko zaio POST bitartez formularioaren eremu guztien informazioa pasaz.

Ondorioz, berriro ere `Erregistratu.php` funtzio honetan aurkitzen gara; orain bai honen inplementazio aztertuko dugu (8.6. kodea).

```
if (isset($_SERVER['HTTP_REFERER'])){
1   $aux=$_SERVER['HTTP_REFERER'];
    $web=explode("/",$aux);
    if( $web[6]=="SignUp.html" || $web[6]=="EgiaztapenKodea.php"){
2       if (filter_var($_POST['email'], FILTER_VALIDATE_EMAIL) &&
preg_match("/^[6|9]{1}-? ?[0-9]{8}$/", $_POST['mugikorra'])) {
3           $mysqli = new mysqli("localhost", "root", "", "erabiltzaileak");
            /* comprobar la conexión */
            if ($mysqli->connect_errno) {
                printf("Konexioan errore bat egon da: %s\n", $mysqli->connect_error);
                exit();
            }
            $passZifra=sha1($_POST['pasahitza']);
            $eranZifra=sha1($_POST['erantzuna']);
4           if (isset($_POST['modu'])){
                $sql="INSERT INTO erabiltzaileak(Email,Izena, Abizena,
Abizena2,Pasahitza,Mugikorra,Galdera,Erantzuna,Saialdi,Baimen) VALUES
('$_POST[email]','$_POST[izena]','$_POST[abizena]','$_POST[abizena2]','$passZifra','$_
POST[mugikorra]','$_POST[galdera]','$eranZifra',0,'$_POST[modu]')";
            }else{
                $sql="INSERT INTO erabiltzaileak(Email,Izena, Abizena,
Abizena2,Pasahitza,Mugikorra,Galdera,Erantzuna,Saialdi,Baimen) VALUES
('$_POST[email]','$_POST[izena]','$_POST[abizena]','$_POST[abizena2]','$passZifra','$_
POST[mugikorra]','$_POST[galdera]','$eranZifra',0,'ikasle')";
```

```
            }
5           if (!$mysqli->query($sql))
            {
                die('Errorea:'.mysqli_error($mysqli));
            }
            $mysqli->close();
6           echo'<script>alert("Erregistroa ondo egin da.")</script>';
            echo'<script>window.location = "../Orrialdeak/Hasiera.php"</script>';
        }else{
            echo'<script>alert("Datu desegokiak; email edo mugikorra ez daukate
formatu ona")</script>';
            echo'<script>window.location = "../Orrialdeak/Hasiera.php"</script>';
        }
    }else{
        echo'<script>alert("Ez daukazu baimena orrialde honetan sartzeko")</script>';
        echo'<script>window.location = "../Orrialdeak/Hasiera.php"</script>';
    }
}else{
    echo'<script>alert("Ez daukazu baimena orrialde honetan sartzeko")</script>';
    echo'<script>window.location = "../Orrialdeak/Hasiera.php"</script>';
}
```

8.6. Kodea: Erregistratu.php

1. Lehenik eta behin, funtzioak dei egin dion orrialdearen kontrola egingo du, ekiditeko edonork funtzio honi deitzea.
2. Hau egin eta gero, POST bitartez ailegatu zaion informazio kritikoaren egiaztapena egingo du. Nahiz eta aurreko pausuan, web-arakatzailean `balidatu()`funtzioak (8.6 kodea) datuak egiaztatu, baliteke erabiltzaileak Javascript desgaituta izatea eta, beraz, komenigarria da zerbitzari aldean ere jasotzen den informazioaren egiaztapena egitea. Kasu honetan bakarrik email eta mugikorren egiaztapena egitea erabaki da, kritikoenak direlako. Egokiena eremu guztien egiaztapena egite izango litzateke, etorkizu batean egitea aurreikusi dena.
3. Datu-basearekin konexioa ezarriko da eta datu-basean zifratuta gordetzen diren pasahitza eta segurtasun galderako erantzuna zifratuko dira. SHA1 funtzioa erabili da, nahiz eta oso segurua ez izan, enkriptazio azkar eta erraz bat egiteko.
4. Konexioa ezarrita, datu-basean datuak sartzera pasatuko gara. Bi erabiltzaile mota ezberdinak erregistratzeko prozesuetatik ailegatu gaitezke `Erregistratu.php` honetara, eta beraz, datuak sartzerakoan ezberdindu egin beharko ditugu. Ezberdintasun hau `modu` balioaren arabera egingo da. Banatzaileen kasuan `EgiaztapenKodea.php`-tik `modu`-ak izan behar duen balioa POST bitartez bidaliko da ere, aldiz, `SignUp.html`-tik balio hori ez da bidaliko. Beraz, ezaugarri hori oinarri bezala hartuz, bi kasuak ezberdindu dira. Bi kasuetan erabiltzaileak taulan sarrera berri bat sortuko da, formularioaren bitartez lortu den informazio guztiarekin.
5. Behin datu-basean datuak sartu direlarik, datu-basearekiko konexioa itxi egingo da eta funtzioa bukatuko da.
6. Ondorioz, `Hasiera.php` orrira birbidaliko digu.

8.2 Erabiltzaile erregistratuak

Erabiltzaile erregistratuak beste hainbat aukera dauzkate, 7.3 atalan ikusi ditugunak. Funtzio bakoitza azaltzen hasi baino lehen, erabiltzaile mota bakoitzak eskuragarri dituzten funtzioak definituko ditugu. Hasteko, erabiltzaile mota bakoitza sisteman sartu ondoren, ikusiko duten `Hasiera.php` orria aztertuko dugu. Ordenean, erabiltzaile arrunta (8.12 irudia), banatzailea (8.13 irudia) eta administratzailea (8.14 irudia). `Hasiera.php` orriaren dinamismoa 8.1 kodean azaldu dugu.

8.12. Irudia: Hasiera.php erabiltzaile arrunt baten ikuspuntutik

8.13. Irudia: Hasiera.php banatzaile baten ikuspuntutik

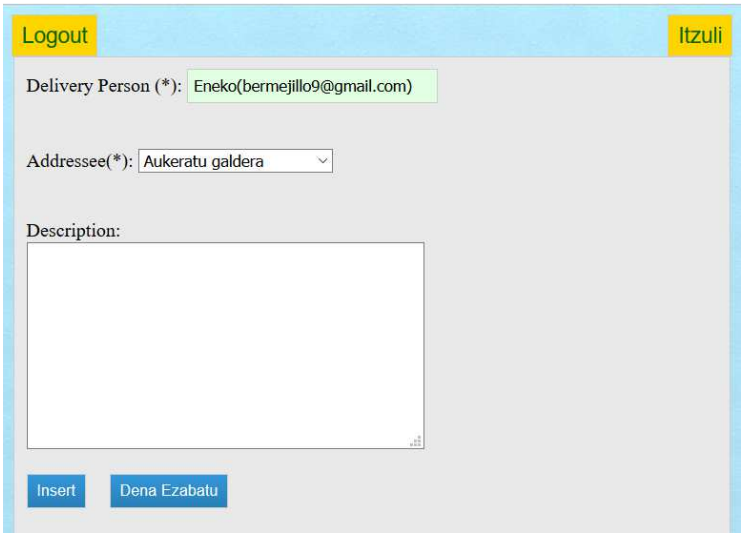
8.14. Irudia: Hasiera.php administratzaile baten ikuspuntutik

8.2.1 Pakete bat biltegiratu

8.12 eta 8.13 irudietan ikus dezakegun moduan, bai erabiltzaile arruntek baita banatzaileek ere `Insert Packages` aukera daukate beraien menuetan. Beraz, bi erabiltzaile mota hauek pakete bat biltegiratzeko aukera izango dute.

Bien artean existitzen den ezberdintasun bakarra, erabiltzaile arrunt batek biltegiratzen duen paketea, bai berarentzako edo sistemako edozein beste erabiltzaile arrunt batentzako gorde dezakeela da. Aldiz, banatzaileek bakarrik sistemako erabiltzaile arruntentzako biltegiratu ditzakete paketeak (enpresa banatzaileen funtzionamendua imitatuz).

Erabiltzaileak `Insert Package` esteka sakatzen dutenean `InsertPackages.php` orrira pasatuko dira. (8.15 Irudia)



8.15. Irudia: InsertPackages.php

8.15 irudian ikusten dugun moduan, sisteman sartu eta gero erabiltzaileek momentu oro `Logout` egiteko botoia izango dute eskuragarri. Honen bitartez, ireki duten sesioa itxi eta `Hasiera.php` orrira bueltatuko dira, erabiltzaile anonimo baten baimenekin.

Bestalde, pakete bat sisteman sartzeko beherrezko informazioa osatzeko formulario bat dago. Antzeman daitekeen moduan, lehenengo eremua sistemak beteko du, irekita dagoen sesiotik informazioa lortuz. Beti ere, eremua osatzerakoan "Izena (email)" formatuari jarraituko dio. Lehenengo eremu honek, paketea sisteman sartu duen erabiltzailearen informazioa izango du.

Bigarren eremua paketea nori bideratuta dagoen ezartzeko balio digu. Kasu honetan, zerrenda hedagarri bat izango du erabiltzaileak, dituen aukeretatik bat aukeratzeko. Honetan, lehen azaldu den moduan, erabiltzaile arrunt guztiak agertuko dira. Paketea sartuko duen erabiltzailea erabiltzaile arrunta bada bere kontua ere agertuko da.

Hirugarren eremuan paketeak barnean daukanaren inguruan deskripzio txiki bat egiteko aukera izango du erabiltzaileak.

Behin datu guztiak sartu eta gero, erabiltzaileak `Insert` botoia sakatzerakoan `jQuery.ajax` obtektu bat egituratu eta zerbitzariari informazioa bidali egingo zaio (8.7 Kodea). `jQuery.ajax` objektu honek lehen ikusi dugun `XMLHttpRequest()`-en antzeko lana egiten du.

```
function InsertPk(){
1   var errorea="";
    var bana=document.getElementById("banatzailea").value;
    var dest=document.getElementById("destinatario").value;
    if(bana==""){
        errorea+="The Delivery Person field is empty \n";
    }
    if(dest==""){
        errorea+="the Addressee field is empty\n";
    }

    if(errorea!=""){
        alert(errorea);
    }else{
2       var desk = document.getElementById ("Description").value;
        var str1="email=";
        var str2="banatzailea=";
        var str3="deskribzioa=";
  var bidaltzekodatuak=str1.concat(dest)+'&'+str2.concat(bana)+'&'+str3.concat(desk);
3    $.ajax({
            url : "../ExtExe/Insert.php",
            type: "POST",
            data : bidaltzekodatuak,
```

```
4            beforeSend:function()
             {
               $('#erakutsi').html('<div><img src="../images/loading.gif"/></div>');
             },
5            success: function(datuak)
            {
               $('#erakutsi').fadeIn().html(datuak);
             },
6            error: function ()
             {
$('#content').fadeIn().html('<p class="error"> <strong> Zerbitzariak ez  duela
erantzuten dirudi </p>');
             }
             });
         }
}
```

8.7. Kodea: InsertPackages.php jQuery.ajax objektua

8.7. kodean ikus daitekeen moduan, `jquery.ajax` objektua Javascript-en idatzitako `InsertPk()` funtzioaren barnean aurkitzen da, eta erabiltzaileak `Insert` botoia sakatzerakoan exekutatuko da. Funtzio honen egitura eta eginkizunak aztertuko ditugu orain:

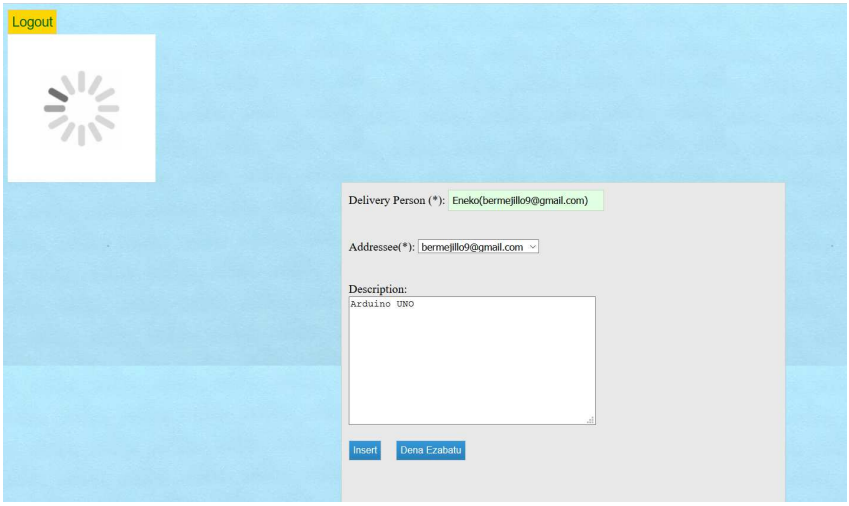
1. Hasteko, erabiltzaileak bete behar dituen eremuak beteta daudela egiaztatuko da.
2. Hau eginda, gure AJAX objektua egituratzen hasiko gara. Hasteko, POST bitartez bidaliko diren datuak egituratuko dira; email-a, banatzailea eta deskripzioa.
3. Ondoren, AJAX objektuak nori egin behar dion eskaera ezarriko dugu. Kasu honetan, `ExtExe` karpetan dagoen `Insert.php` funtzioari deituko dio. Honi, lehen esan moduan, POST bitartez bidaliko zaizkio aurretik `bidaltzekodatuak` aldagai barruan definitu ditugunak.
4. Objektu honek, funtzio ezberdinak izango ditu. Hasteko, eskaera bidali aurretik eta erantzun bat jasotzen dugun arte, `jquery.ajax` objektuak egin beharrekoa `beforesend` funtzioan definituko dugu. Honetan, erantzun bat jaso arte, InsertPackages.php-n definituta dugun `erakutsi` eremuan `Loading.gif` (8.16. irudia) gif-a ikusgai utziko da. Honekin erabiltzaileari bere eskaera prozesatzen ari dela adieraziko diogu.

8.16. Irudia: Loading.gif

5. Hortaz, erantzuna jasotzerakoan (`success` funtzioa) `Loading.gif` kenduko da eta mezu bat agertuko zaio erabiltzaileari, komunikazioaren erantzuna bistaratuz. Erantzuna zuzena izan daiteke edo errore bat agertu daiteke.
6. Azkenik, erroren bat egongo balitz komunikazioan (`error` funtzioa) `InsertPackage.php`-n definituta dagoen `content` elementuan errore mezu bat agertuko da.

Objektu honek komunikazio bat hasiko du zerbitzariarekin. Hain zuzen ere, `Insert.php` funtzioarekin, hau da, biltegiratze-sistema kontrolatzen duen Arduino Megarekin komunikazioa kudeatuko duen funtzioarekin.

Beraz, erabiltzaileak `Insert` botoia klikatzean eta komunikazioa egiten den bitartean, 8.17 irudikoa ikusiko du web-arakatzailean.



8.17. Irudia: Insertpackage.php pakete bat sartzen den bitartean

Komunikazio horretan, deitu egiten den `Insert.php` funtzioak betetzen dituen prozesuak hobeto ulertzeko kodea aztertuko dugu (8.8 kodea).

```
 session_start();
 // Orrialdea irekitzeko eta erabiltzeko beharrezkoak baimenak egiaztatu
 if(isset($_SESSION['baimena'])){
  if ($_SESSION['baimena']=="ikasle" || $_SESSION['baimena']=="Banatzaile"||
$_SESSION['baimena']=="Admin"){
   // InsertPackage.php ajax bidez bidalitako informazioa gorde
1    $email=$_POST['email'];
     $banatzailea=$_POST['banatzailea'];
     $deskribzioa=$_POST['deskribzioa'];
     // Komunikazioa mantenduko duen aplikazioari deitu
2    exec('insert.exe',$out,$return);
   // Aplikazioak idatzitako print guztiak gorde erabiltzaile horren track fitxategian
3     if (file_exists('../Track/'.$email.'.txt')) {
        $actual = file_get_contents('../Track/'.$email.'.txt');
       }else{
          $actual="";
       }
       $total="";
       foreach ($out as &$valor) {
         $valor =$valor."\r\n" ;
         $total.=$valor;
       }
       unset($valor); // rompe la referencia con el último elemento
       $actual .= $total;
       $put=$actual."\r\n";
       file_put_contents('../Track/'.$email.'.txt', $put);
       // Aplikazioak izandako return balioa aztertu
4      if ($return==0){
  // Dena ondo joan bada, aplikazioak paketearen rfid zenbakia gorde duen fitxategitik
 // informazioa lortu.
        $myfile = fopen("rfid.txt", "r") or die("Unable to open file!");
        $rfid=fgets($myfile);
        fclose($myfile);
        unlink("rfid.txt");
        if ($rfid==""){
          echo'<script>alert("Paketea identifikatzen erroren bat egon da")</script>';
          echo'<script>window.location = "../Orrialdeak/Hasiera.php"</script>';
        }else{
        / informazioa guztiarekin datu basean sartu paketea
          $mysqli = new mysqli("localhost", "root", "", "erabiltzaileak");
          if ($mysqli->connect_errno) {
            printf("Konexioan errore bat egon da: %s\n", $mysqli->connect_error);
            exit();
        }
        // egin beharrekoa sql deia egituratu
        $package="INSERT INTO paketeak (id,email, banatzailea, deskribzioa) VALUES
                            ('$rfid','$email','$banatzailea','$deskribzioa')";
        // query-a bidali datu basera
```

```
        $emaitza_packages = $mysqli->query($package);
        if (!$emaitza_packages)
        {
          die('Errorea:'.mysqli_error($mysqli));
        }
        $mysqli->close();
        echo'<script>alert("Paketea ondo sartu da sisteman")</script>';
       }
// Insert.exe bueltatu duen errore kode ezberdinen arabera zer egin behar den definitu
5       }else if ($return==1){
          echo'<script>alert("Errore bat egon da zure eskaera prozesatzerakoan, ezin
da ezer egin")</script>';
        }else if ($return==2){
          echo'<script>alert("Plataformarekin komunikatzerakoan arazoak egon
dira")</script>';
        }else if ($return==3){
          echo'<script>alert("Sartu den paketea ez dauka formatu ona edo ez dago
lekurik gordetzeko")</script>';
        }else if ($return==4){
          echo'<script>alert("Ez duzu paketea sartu, berriro ere sahiatu
sartzen")</script>';
        }else{
          echo'<script>alert("Web aplikazioa eta biltegiaren arteko komunikazioan
errore bat egon da")</script>';
        }
     }
 }
```

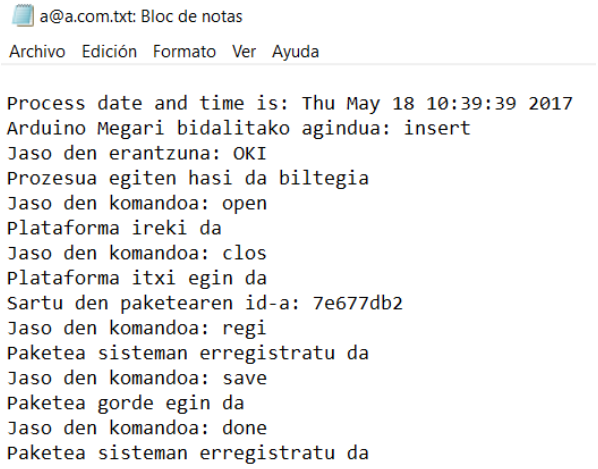
8.8. Kodea: Insert.php funtzioa

Beraz, kodea pausoz pauso aztertzera pasatuko gara.

1. Hasteko, lehen ikusi dugun `InsetPackages.php`-ren `InsertPK()`-k (8.7. kodea) `jquery.ajax` bitartez bidali dizkion hiru parametroak aldagaietan gordeko ditu.

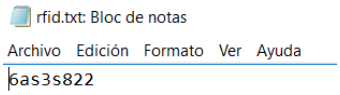
2. `ExtExe` honetan aurkitzen den `Insert.exe` programari dei egingo zaio. `Insert.exe` modu sakon batean 8.3 atalean aztertuko da, bere parekoa den `extract.exe`-rekin. Atal horretan, Arduino Megarekin daukaten komunikazioa modu zehatz batean ikusiko dugu. Ondorioz, `insert.exe` izango da biltegiratze-sistemari, hau da, Arduino Megari pakete bat sartzeko agindua bidaliko diona eta honekin komunikazioa mantenduko duena. Paketea biltegian sartu eta gorde ondoren eta komunikazioa bukatu delarik, `insert.exe` aplikazioa amaituko da. Bukatzean `Insert.php` `insert.exe` programak bueltatutako balioa aztertuko du. Bueltatuko balioen arabera, `Insert.php`-k prozesua ondo joan den edo ez jakingo du, erabiltzaileari adieraziz, `InsertPK()` funtzioari erantzuna bidaliz.

3. Honekin batera, prozesuan zehar inprimitu diren mezuak gorde egingo dira, erabiltzaile bakoitzak bere izenean `Track` karpetan daukan testu-fitxategian (8.18 irudia).



8.18. Irudia: Track baten adibidea

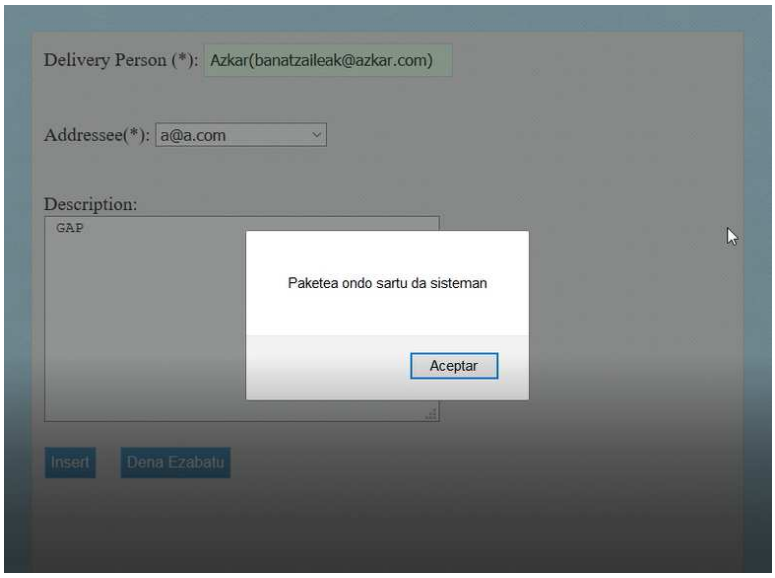
4. Lehen esan dugun moduan, aplikazioak bueltatutako balioak aztertuko ditu `Insert.php` honek. Beraz, dena ondo joan bada, 0 bueltatuko da. Paketearen informazio datu-basean sartu aurretik, sartu berri den paketearen id-a behar dugu. Horretarako, `Insert.exe` aplikazioak, `rfid.txt` fitxategia sortuko du `ExtExe` karpetan. Fitxategi honen barruan sartu berri den paketearen id-a izango dugu (8.19 irudia). Beraz, `Insert.php`-k fitxategia ireki eta id-a irakurriko du. Fitxategia irakurri ondoren, fitxategia ezabatuko da. Paketearen informazioa guztia dugula, datu-basean sartuko dugu paketea. Datu-basean sartuko den informazioa, `Insert.exe`-tik lortutako id-a eta erabiltzaileak `InsertPackages.php`-tik POST bidez bidali dizkigun 3 parametroak izango dira. Datu-basean informazioa sartu ondoren, erabiltzaileari informazio leiho bat bidaliko zaio, dena ondo joan dela adieraziz .



8.19. Irudia: rfid.txt adibidea

5. Aldiz, `insert.exe` programaren exekuzioan erroren bat gertatu bada, programak 0 ez den beste balio positibo bat bueltatuko du. Bueltatzen duen balioaren arabera, errore mota ezberdinak adieraziko ditu. Errore bakoitzak mezu ezberdina bidaliko dio bueltan `InsertPackages.php`-ko `jquery.ajax` objektuari. Mezu hauen bitartez erabiltzaileari egon den errorea adieraziko zaio.

Honenbeste, behin paketea sisteman sartuta dagoela, `Hasiera.php` orrira birbidaliak izango gara, prozesua amaituz (8.20. irudia).



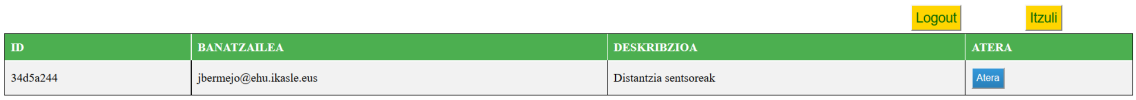
8.20. Irudia: Paketea ondo sartzean agertuko zaigun mezua

8.2.2 Pakete bat atera

Kasu honetan 8.12 eta 8.14 irudietan ikus daitekeen moduan, paketeak ateratzeko baimena, erabiltzaile arruntek eta administratzaileek izango dute. Berriro ere, pakete bat sartzerakoan gertatzen den moduan, nahiz eta ezberdintasun txiki batzuk izan, prozesuak orokorrean

berdinak dira bi erabiltzaile motarako, horregatik erabiltzaile arrunt baten kasu partikular batekin geratuko gara.

Erabiltzaileak `View packages` edo `List Packages` aukerak sakatzen baditu, 8.21 irudian ikusten dugun antzeko taula bat ikustera pasatuko da. Erabiltzaile arruntaren kasuan, `ListPackages.php` orriak erabiltzaile horrek biltegiratze-sisteman bere izenean dituen paketeak bistaratuko ditu. Administratzailearen kasuan, berriz, sisteman dauden pakete guztiak bistaratuko dizkio `ListPackageAdmin.php`-k.



8.21. Irudia: ListPackages.php

Paketeak bistaratzeko irudian ikusten dugunaren antzeko taula bat izango dugu, non errekanda bakoitzak sisteman gordeta dagoen pakete bat irudikatuko duen. Errenkada bakoitzaren azkenengo eremua botoi bat izango da, `Atera` botoia. Botoi hau sakatzeak, pakete hori sistematik ateratzeko prozesua martxan jarriko du.

Pakete bat gordetzean gertatzen den moduan, pakete bat ateratzea aukeratzean `ListPackages.php` edo `ListPackageAdmin.php` orrialdetan definitu dugun `AteraPaketea()` funtzioaren bitartez komunikatuko gara zerbitzarian dagoen `ExtractPackage.php` funtzioarekin. Horretarako, berriro ere `jQuery.ajax` objektua erabiliko dugu (8.9. kodea).

```
function AteraPaketea(kontagailua){
        var errorea="";
        if(kontagailua==""){
                errorea+="Zeozer gaizki jun da, id ez da ondo hartu \n";
        }
        if(errorea!=""){
                alert(errorea);
        }else{
            var rfid = document.getElementById ("paketea"+kontagailua).innerText;
            var str2="id=";
            var bidaltzekodatuak=str2.concat(rfid);
            $.ajax({
                url : "../ExtExe/ExtractPackage.php",
                type: "POST",
                data : bidaltzekodatuak,
                beforeSend:function()
                                {
                 $('#erakutsi').html('<div><img src="../images/loading.gif"/></div>');
                                },
                success: function(datuak)
                                {
                 $('#erakutsi').fadeIn().html(datuak);
                 window.location = "./Hasiera.php";
                                },
                error: function ()
                {
                 $('#content').fadeIn().html('<p class="error"> <strong> Zerbitzariak
ez duela erantzuten dirudi </p>');
                }
            });
        }
    }
```

8.9. Kodea: ListPackages.php Atera Paketea funtzioa

Lehen aztertu dugun 8.7 kodeko `InsertPK()` funtzioaren antzekoa denez, azalpen sakonagorik ez da behar. Funtzionamendua berdina izango da, aldatzen den bakarra zerbitzarian dagoen funtzioa eta honi bidaliko zaizkion parametroak izanik. Funtzio honek, lehen aztertu dugun moduan, `ExtractPackage.php` funtzioari eskatu dio POST bitartez bidaltzen dion id-a daukan paketea sistematik ateratzeko.

Ondorioz, `Atera` botoia sakatuz komunikazioa hasiko da, `InsertPackages.php`-rekin gertatzen den moduan, `Loading.gif` ikusiko du erabiltzaileak, prozesua aurrera eramaten den bitartean (8.22 irudia).

8.22. Irudia: ListPacakages.php Atera botoia sakatu ondoren

Kasu honetan, `ExtractPackage.php`-k honako egitura izango du (8.10. kodea). Egia da `Insert.php`-ren antzeko egitura duela, baina ezberdintasun batzuk ditu.

```
    session_start();
    if(isset($_SESSION['baimena'])){
     if ($_SESSION['baimena']=="ikasle" || $_SESSION['baimena']=="Banatzaile"||
$_SESSION['baimena']=="Admin"){
            $id=$_POST['id'];
            $email=$_SESSION['erabiltzaile'];
1           exec('extract.exe '.$id.'',$out,$return);
            if (file_exists('../Track/'.$email.'.txt')) {
                $actual = file_get_contents('../Track/'.$email.'.txt');
            }else{
                $actual="";
            }
            $total="";
            foreach ($out as &$valor) {
                $valor =$valor."\r\n" ;
                $total.=$valor;
            }
            unset($valor);
            $actual .= $total;
            $put=$actual."\r\n";
            file_put_contents('../Track/'.$email.'.txt', $put);

            if ($return==0){
                $mysqli = new mysqli("localhost", "root", "", "erabiltzaileak");
                if ($mysqli->connect_errno) {
                 printf("Konexioan errore bat egon da: %s\n", $mysqli->connect_error);
                 exit();
                }
                $package =("DELETE FROM paketeak WHERE id='".$id."';");
                $emaitza_packages = $mysqli->query($package);
                if (!$emaitza_packages)
                {
                    die('Errorea:'.mysqli_error($mysqli));
                }
                $mysqli->close();
                echo'<script>alert("Paketea '.$id.' atera da ")</script>';
            }else if ($return==1){
                echo'<script>alert("Errore bat egon da zure eskaera prozesatzerakoan,
ezin da ezer egin")</script>';
            }else if ($return==2){
                echo'<script>alert("Plataformarekin komunikatzerakoan arazoak egon
dira")</script>';
            }else if ($return==3){
                echo'<script>alert("Paketea ez da aurkitu edo errore bat egon da
paketea ateratzean")</script>';
            }else if ($return==4){
                echo'<script>alert("Ez duzu paketea hartu, berriro ere gorde egin
da")</script>';
            }else{
                echo'<script>alert("Web aplikazioa eta biltegiaren arteko
komunikazioan errore bat egon da")</script>';
```

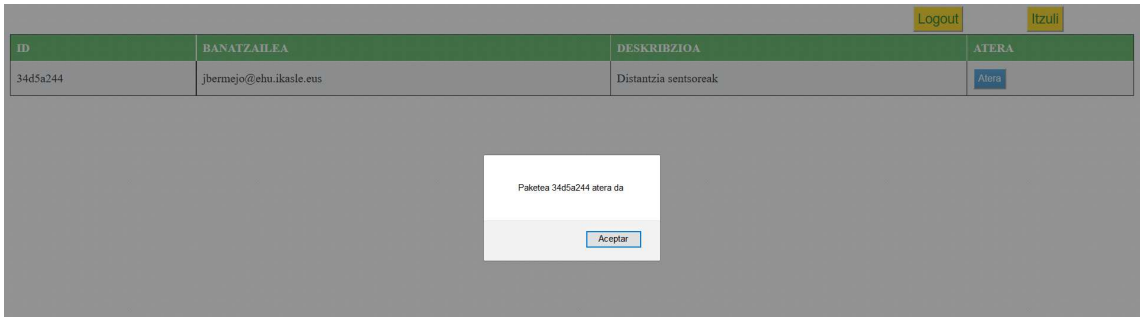
```
            }
        }else{
            echo'<script>alert("Ez daukazu baimena orrialde honetan
sartzeko")</script>';
            echo'<script>window.location = "../Orrialdeak/Hasiera.php"</script>';
        }
    }else{
        echo'<script>alert("Ez daukazu baimena orrialde honetan sartzeko")</script>';
        echo'<script>window.location = "../Orrialdeak/Hasiera.php"</script>';
    }
```

8.10. Kodea: ExtractPackage.php

`Insert.php`-rekin (8.8 kodea) ikus dezakegun ezberdintasun bakarra, paketearen id-arekin egiten den prozesua da. `Insert.php` funtzioaren kasuan paketearen id-a biltegiratze-sistematik lortu behar da eta horretarako `insert.exe` programak sortzen duen `rfid.txt` fitxategi irakurtzen da. Aldiz, pakete bat ateratzerakoan web aplikazioak da biltegiratze-sistemari atera nahi den paketearen id-a bidali behar diona. Horretarako ikusi dezakegun moduan `exec()` funtzioak `extract.exe` programari deitzerakoan parametro moduan atera nahi den paketearen id-pasatuko zaio (kodean 1 erreferentzia).

Kodearen beste guztia, `Insert.php` (8.8 kodea) ikusi dugunaren antzekoa da. Adibidez, kasu horretan pakete berria sartuko da datu-basean, honetan, berriz, id hori daukan paketea datu-basetik ezabatuko da.

Dena ondo badoa, hurrengo mezua jasoko du erabiltzaileak (8.23. irudia) eta berriro ere `Hasiera.php` orrira bueltatua izango da, prozesua amaituz. Erroreak gertatzen badira, `Insert.php`-n ikusi dugun prozesu berdinari jarraituko zaio.



8.23. Irudia: ListPackages.php erantzuna jasotzean

8.2.3 Erabiltzaileak ezabatu

Funtzio hau administratzaileek bakarrik izango dute eskuragarri. 8.14 irudiko `List Users` esteka klikatzerakoan `ListUsersAdmin.php` orriak, pakete bat ateratzerakoan gertatzen den moduan, sisteman erregistratuta dauden erabiltzaileen taula bat bistaratuko du (8.24 irudia).

Logout Itzuli

Email	Name	Family Name	Baimena	KENDU
asd@asd.com	Eneko	Bermejo	ikasle	Kendu
banatzaileak@azkar.com	Azkar	Azkar	Banatzaile	Kendu
banatzaileak@seur.com	Seur	Seur	Banatzaile	Kendu
bermejillo9@gmail.com	Eneko	Enejko	ikasle	Kendu
jone@gmail.com	Jone	Hernandez	ikasle	Kendu

8.24. Irudia: ListUsersAdmin.php

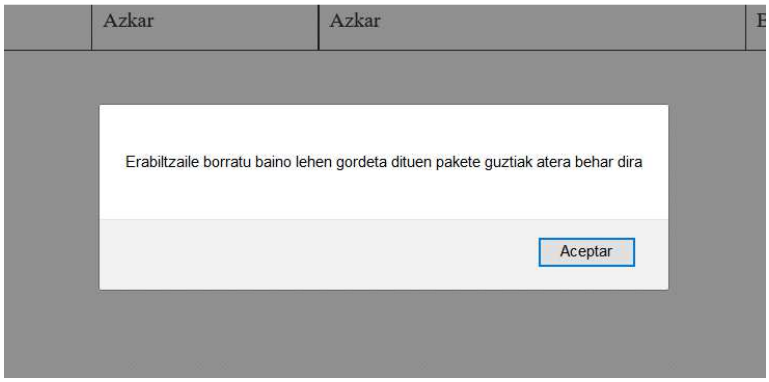
Pakete bat ateratzeko kasuan gertatzen den moduan `Kendu` botoiek Javascript-en idatzitzako funtzio bati deituko diote. Funtzioa, kasu honetan, `KenduErabiltzailea()` izango da (8.11. kodea).

```
function KenduErabiltzailea(kontagailua){
        var errorea="";
        if(kontagailua==0){
                errorea+="Zeozer gaizki jun da, id ez da ondo hartu \n";
        }
        if(errorea!=""){
                alert(errorea);
        }else{
            var email = document.getElementById(kontagailua).innerText;
            var str2="email=";
            var bidaltzekodatuak=str2.concat(email);
            $.ajax({
                url : "../Funtzioak/DeleteUser.php",
                type: "POST",
                data : bidaltzekodatuak,
                beforeSend:function()
                {
                 $('#erakutsi').html('<div><img src="../images/loading.gif"/></div>');
                },
                success: function(datuak)
                        {
                   $('#erakutsi').fadeIn().html(datuak);
                },
                error: function ()
                {
                   $('#content').fadeIn().html('<p class="error"> <strong>
Zerbitzariak ez duela erantzuten dirudi </p>');
                }
               });
          }
    }
```

8.11. Kodea: ListUsersAdmin.php orriko KenduErabiltzaile() funtzioa

Berriro ere, AJAX teknologia erabiltzen duen `jquery.ajax` objektua erabiliko da, kasu honetan erabiltzailearen datuak `DeleteUsers.php` funtzioari bidaliz, honek erabiltzaile hau datu- basetik ezabatzeko.

`DeleteUsers.php` funtzioak jasotzen duen erabiltzailea datu-basetik ezabatzeko erabiltzen da. Hala ere, erabiltzaile bat ezabatu baino lehen, datu-basean bere izenean paketerik dagoen edo ez egiaztatuko da. Datu-basean, bere izenean paketerik egotekotan, erabiltzailea ezingo da ezabatu, pakete guztiak sistematik kentzen diren arte (8.25 irudia).



8.25. Irudia: ListUsersAdmin.php ezabatzen sahiatu den erabiltzailea paketea badauzka

Paketerik ez balu zuzenean erabiltzailea ezabatuko da sistematik eta berriro ere `Hasiera.php` orrira birbidalia izango da administratzailea.

8.3 Web aplikazioaren eta Arduinoaren arteko komunikazioa

8.2 kapituluan ikusi dugun moduan, `insert.exe` eta `extract.exe` programak erabiliko dira gure web aplikazioa biltegiratze-sistemarekin (Arduino Mega) komunikazioa egiteko. Wifi socket bitartezko komunikazio honek izango duen protokoloa 6.2.4 atalan ikusi dugu. Jarraian, web aplikazioaren aldean protokoloa nola definitu dugun ikusiko dugu. Hain zuzen ere, 6.7 eta 6.8 irudietan definituta dauden komunikazio protokoloak jarraituko ditugu.

8.3.1 Insert.exe

Lehenik eta behin, komunikaziorako beharrezkoa den socket-a egituratu behar dugu, 8.12 kodean ikusi dezakegun moduan.

```
        strcpy(zerbitzaria, SERVER);
        // Socketa sortu.
1       if((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
                perror("Errorea socketa sortzbugsean");
                exit(1);
        }
        // Zerbitzariko socketaren helbidea sortu.
2       zerb_helb.sin_family = AF_INET;
        zerb_helb.sin_port = htons(portua);
        if((hp = gethostbyname(zerbitzaria)) == NULL)
        {
                herror("Errorea zerbitzariaren izena ebaztean");
                exit(1);
        }
        memcpy(&zerb_helb.sin_addr, hp->h_addr, hp->h_length);
3       if(connect(sock, (struct sockaddr *) &zerb_helb, sizeof(zerb_helb)) < 0)
        {
                perror("Errorea zerbitzariarekin konektatzean");
                exit(1);
```

8.12. Kodea: Insert.exe socket egituraketa

1. Hasteko socket-a sortzen dugu.
2. Geroago, zerbitzariaren helbidea eta portuak definitu eta hauek beharrezkoa den `hostent` [12] egituran sartzen ditugu.
3. Konfigurazio osoa prest daukagula, zerbitzariarekin konektatzera pasatuko gara.

Behin socket-a prest dagoela, komunikaziora pasatuko gara. Komunikazioa, 6. atalean zehaztutako egitura jarraituko du. Baina komunikazioarekin hasi baino lehen, komunikazioa aurrera eramateko definitu diren funtzio lagungarriak aztertuko ditugu. Hauek azaltzea ezinbestekoa da , geroago ikusiko dugun kodea ulergarria gertatzeko (8.13 kodea).

```
/* Zerbitzaritik jasotako erantzuna aztertzen du.
* Jasotakoa OKI, 0 balioa itzuliko du eta bestela errorearen kode zenbakia.
*/
int parse(char *status)
{
        if(!strncmp(status,"OKI",3))
                return 0;
        else if(!strncmp(status,"ERR",3))
                return(atoi(status+3));
        else
        {
```

[12] *Hostent* egitura zerbitzari baten informazioa gordetzeko erabiltzen da. Egitura honetan zerbitzariaren izena, IPv4 helbidea, portua eta informazio gehiago gordetzeko erabiltzen da. Egitura hau socket-ek erabiltzen dituzte.

```
                fprintf(stderr,"Ustekabeko erantzuna.\n");
                exit(1);
        }
}
* Funtzio honek gure bufferra garbituko du write egin eta gero horrela read egindako

* bufferra zabor gabe egongo da.
*/
int readline(int stream, char *buf, int tam){
        memset(buf,0,strlen(buf));
         tv.tv_sec = 90;  /* 90 Secs Timeout */
         tv.tv_usec = 0;
         FD_ZERO(&rfds);
        FD_SET(stream,&rfds);
        if(select(stream+1,&rfds,NULL,NULL,&tv)<=0){
             perror("Itxoiteko denbora pasatu da");
             return 1;
         }
        if(recv(stream,buf,tam,0)<=0){
                fprintf(stderr,"Ustekabeko erantzuna.\n");
                 return 1;
        }
        return 0;
}
/* buf array clears*/
int clearbuf(char *buf){
    memset (buf,0,strlen(buf));
    return 0;
}
/* Jasotako mezua definituta dauden komandoekin konparatu egingo da, eta
 aurkitzen bada, komandoaren posizioa bueltatu egingo da.*/
int searchcommand(char* buf,const std::vector <std::string>& komandoa){
    for (int i=0;i < komandoa.size();i++){
         if (strcmp(komandoa.at(i).c_str(),buf)==0){
             return i;
         }
    }
    return -1;
```

8.13. Kodea: Insert.cpp funtzio lagungarriak

Lau funtzio lagungarri ditugu:

1. `Parse()`: Hau Arduinotik jasotzen den bai `OKI` eta bai errore mezuak interpretazeko balioko digu. Erroreen kasuan errore kodea ateratzeko gai izango da, errore prozesapena erraztuz.
2. `Readline()`: Lehen socketetik datozen komandoak eta mezuak irakurtzeko funtzioa da. Kasu honetan ez da infinituki mezuen zain geratuko, baizik eta, segurtasun neurri bat bezala, 90 segundu zain geratuko da. 90 segundu hauek pasatzen badira ezer jaso gabe, aplikazioaren exekuzioa bukatu egingo da eta errore balio bat bueltatuko du. 90 segunduak aukeratzearen arrazoia biltegiratze-sistemak egin behar dituen prozesu guztien artean, denbora gehien ematen duen prozesuak gutxi gorabehera 80 segundu irauten duelako da. Beraz, 90 segunduak jarrita biltegiratze-sistemak denbora soberan dauka bere prozesuak aurrera eramateko.
3. `Clearbuf()`: Bufferra 0-z betetzeko balio digun funtzioa. Honekin bufferra garbitu dezakegu.
4. `Searchcommand()`: Jasotako mezuak zein komandoak diren zehazteko balio digun funtzioa. Honek `bufferraren` informazioa aurretik definituta ditugun komandoekin konparatuko ditu, berdinak diren edo ez aztertuz. Horrela Arduino Megatik ailegatzen diren komandoak identifikatuko ditugu.

Beraz, guzti hau ikusita komunikazio protokoloa nola inplementatzen den ikusiko dugu jarraian (8.14. kodea).

```
1       if(sprintf(buf,"insert")<= 0){
            perror("Errorea bufferrera pasatzean komandoa:");
            printf("Errorea bufferrera pasatzean komandoa\n");
            exit(1);
        }
        printf("Arduino Megari bidalitako agindua: %s\n",buf);
        if(write(sock,buf,strlen(buf))<0){
            printf("Errorea bidaltzean\n");
            perror("Errorea bidaltzean:");
            exit(1);
        }
        if (clearbuf(buf)!=0){
            exit(1);
        }

        // Erantzunaren zain geratu
2       if(readline(sock, buf, MAX_BUF)!=0){

            printf("Errorea bufferra irakurtzean, denbora muga pasa da\n");
            exit(1);
        }
3       status = parse(buf);   //Erantzuna konparatu.
        printf("Jaso den erantzuna: %s\n",buf);
    // erantzunaren arabera errore edo aurrera jarraitu
        if(status != 0)
        {
         fprintf(stderr,"Errorea: ");
         fprintf(stderr,"%s",ER_MEZUAK[status].c_str());
         printf("Errore bat egon da mezua jasotzean: %s\n",ER_MEZUAK[status].c_str());
            exit(1);
        }
        if (clearbuf(buf)!=0){
            exit(1);
        }
        // prozesura sartuko gara
        printf("Biltegia prozesua egiten hasi da\n");
```

8.14. Kodea: Insert.cpp agindua bidali

1. Hasteko, esan dugun moduan `insert` agindua bidaliko zaio (`write()` funtzioaren bitartez) Arduino Megari eta honen erantzunaren zain geratuko gara (`readline()`).
2. `OKI` jasotzen badugu, aurrera jarraituko dugu. Aldiz, erroreren bat jasotzen badugu, errorea bistaratu eta exekuziotik errore kode batekin aterako gara, komunikazioa bukatuz.
3. Ondoren, `OKI` jaso badugu Arduino Megatik komandoak jasotzera pasatuko gara (8.15 kodea).

```
        while(!finish)
        // betiere komando berri baten zain geratu
        {
            if(readline(sock, buf, MAX_BUF)!=0){
                printf("Errorea bufferra irakurtzean\n");
                close(sock);
                exit(1);
            }
            printf("Jaso den komandoa: %s\n",buf);
            // komandoa zein den prozesatu
1           aukera=searchcommand(buf,komandoak);
            // komandoaren arabera kasu bat edo beste
            switch(aukera)
            {
2           case AUK_OPEN:
                sprintf(buf,"OK");
                if(write(sock,buf,strlen(buf))<0){
                    printf("Errorea bidaltzean\n");
                    perror("Errorea bidaltzean:");
                    close(sock);
                    exit(1);
                    break;
```

```
                }
                printf("Plataforma ireki da\n");
                break;

3           case AUK_CLOSE: // Close box.
                sprintf(buf,"OK");
                if(write(sock,buf,strlen(buf))<0){
                    printf("Errorea bidaltzean\n");
                    perror("Errorea bidaltzean:");
                    close(sock);
                    exit(1);
                    break;
                }
                printf("Plataforma itxi egin da\n");
                if(readline(sock, id, MAX_BUF)!=0){
                    printf("Errorea bufferra irakurtzean\n");
                    close(sock);
                    exit(1);
                }
                printf("Sartu den paketearen id-a: ");
                printf(id);
                printf("\n");
                sprintf(buf,"OK");
                if(write(sock,buf,strlen(buf))<0){
                    printf("Errorea bidaltzean\n");
                    perror("Errorea bidaltzean:");
                    close(sock);
                    exit(1);
                    break;
                }
                break;

            case AUK_REGISTER:
                sprintf(buf,"OK");
                if(write(sock,buf,strlen(buf))<0){
                    printf("Errorea bidaltzean\n");
                    perror("Errorea bidaltzean:");
                    close(sock);
                    exit(1);
                    break;
                }
                printf("Paketea sisteman erregistratu da\n");
                break;

            case AUK_SAVE:
                sprintf(buf,"OK");
                if(write(sock,buf,strlen(buf))<0){
                    printf("Errorea bidaltzean\n");
                    perror("Errorea bidaltzean:");
                    close(sock);
                    exit(1);
                    break;
                }
                printf("Paketea gorde egin da\n");
                break;

4           case AUK_DONE:
                sprintf(buf,"OK");
                if(write(sock,buf,strlen(buf))<0){
                    printf("Errorea bidaltzean\n");
                    perror("Errorea bidaltzean:");
                    close(sock);
                    exit(1);
                    break;
                }
                printf("Paketea sisteman erregistratu da\n");
                close(sock);
                finish=true;
                break;
5           default:
                errorea=parse(buf);
                sprintf(buf,"OK");
                if(write(sock,buf,strlen(buf))<0){
                  printf("Errorea bidaltzean\n");
                  close(sock);
                  exit(1);
```

```
                  break;
                }
                fprintf(stderr,"Errorea: ");
                fprintf(stderr,"%s\n",ER_MEZUAK[errorea].c_str());
                printf("Errore bat egon da prozesuan:
%s\n",ER_MEZUAK[errorea].c_str());
                exit (errorea);
                break;
        }
```

8.15. Kodea: Insert.cpp Arduinotik jasotako mezuak interpretatuz

Arduino Megari agindua bidali ondoren, honek prozesuaren berri ematearen zain geratuko da `insert.exe`. Ondorioz, begizta baten barruan sartuko gara, non etengabe Arduino Megaren mezu baten zain geratuko gara, beti ere lehen esan dugun 90 segundu horiek pasatu arte.

1. Hortaz, behin mezu bat jasotzean, `searchcommand()` funtzioarekin komunikazio protokoloan definitu ditugun komandoekin konparatuko dugu. Definituta ditugun komandoak (`open`,`clos`,`regi`,`save`,`done`) Arduino Megak jarraitu behar duen prozesuarekin bat datoz.

2. Jasotako komandoa aurretikan definituta geneuzkan komandoen artean badago komando horri dagokion mezua eta erantzuna prestatuko da. Kasu guztietan `OK` mezua bidaliko zaio Arduino Megari eta `insert.exe`-ren trazak sortzeko `printf` bidez jaso den komandoaren berri emango da, prozesuaren informazioa inprimatuz. Trazak lehen aztertu ditugun `Track` karpetan aurkitzen diren erabiltzaile bakoitzeko testu- fitxategietan gordeko dira.

3. `clos` komandoaren kasuan beste komandoekin egiten den prozesuaz gain, sartu berri den paketearen id-aren zain geratuko da. Hau jaso ondoren, `OK` bidaliko dio berriro ere Arduino Megari eta komunikazioarekin jarraituko da.

4. `done` komandoa ailegatzerakoan, komunikazioa itxi egingo da eta begiztatik aterako gara, Arduino Megak prozesu guztia modu egokian bukatu duela adieraziz.

5. Aldiz, Arduino Megaren prozesuaren edozein momentuan errore bat gertatzen bada, errore mezu bat jasoko du `insert.exe`-k. Errore kodeak definitu ditugun komandoen artean ez daudenez, `default:` kasuan aurkituko gara. Honetan jaso dugun errorearen kodea aterako dugu, lehen ikusi dugun `parse()` funtzioa erabiliz, eta errore horri dagokion mezua inprimatu. Ondoren, komunikazio itxi egingo da eta `insert.exe` programak errore kode horren balioa bueltatuko du. Adibidez, Arduino Megatik `ERR4` mezua ailegatzen bada, `parse()` bitartez 4 balioa aterako dugu. Horrekin `ER_MEZUAK[]` taulan 4. posizioan definitu dugun mezua inprimatuko da eta insert.exe programak 4 osoko balioa bueltatuko du. Horrela web aplikazioak programak zein egoeratan bukatu den jakingo du.

```
std::ofstream myfile;
myfile.open ("rfid.txt",std::ofstream::out);
myfile << id;
myfile << "\n";
myfile.close();
return 0;
```

8.16. Kodea: Insert.cpp id-a rfid.txt fitxategian utzi

`done` komando jaso ondoren eta komunikazioa bukatu dela, jasotako paketearen id-a `rifd.txt` fitxategian idaztziko dugu (8.16 Kodea). Horrela, lehen esan dugun moduan web aplikazioak sartu berri den paketearen id-a irakurri dezake.

Honekin bukatutzat emango genuke `Insert.exe` honen exekuzioa eta inplementazioa.

8.3.2 Extract.exe

Kasu honetan, `insert.exe` aztertzean ikusi dugun socket-aren egitura (8.12 Kodea) eta funtzio lagungarriak (8.13 kodea) berdinak direnez, zuzenean komunikazio nola inplementatu dugun ikustera pasatuko gara.

Hasteko `extract.exe`-k `extract` agindua eta `id`-a bidaliko dizkio Arduino Megari, honek erabiltzaileak atera nahi duen paketea ateratzeko prozesuarekin hasi dadin (8.17 kodean).

```
1       if(sprintf(buf,"extract")<= 0){
            perror("Errorea bufferrera pasatzen komandoa");
            exit(1);
        }
        if(write(sock,buf,strlen(buf))<0){
            printf("Errorea bidaltzean");
            perror("Errorea bidaltzean");
            exit(1);
               }
        if (clearbuf(buf)!=0){
            exit(1);
        }
        if(readline(sock, buf, MAX_BUF)!=0){

            printf("Errorea bufferra irakurtzean");
            exit(1);
        }
       status = parse(buf);   //Erantzuna konparatu.
       printf(buf);
       printf("\n");
      if(status != 0)
      {
             fprintf(stderr,"Errorea: ");
             fprintf(stderr,"%s",ER_MEZUAK[status]);
             exit(1);
       }

        if (clearbuf(buf)!=0){
            exit(1);
        }
        // id-a pasa argumentu bezala
2       if(sprintf(buf,id)<= 0){
            perror("Errorea bufferrera pasatzen komandoa");
            exit(1);
        }
        if(write(sock,buf,strlen(buf))<0){
            printf("Errorea bidaltzean");
            perror("Errorea bidaltzean");
            exit(1);
               }
        if (clearbuf(buf)!=0){
            exit(1);
        }
        if(readline(sock, buf, MAX_BUF)!=0){

            printf("Errorea bufferra irakurtzean");
            exit(1);
        }
       status = parse(buf);   //Erantzuna konparatu.
       printf(buf);
       printf("\n");
      if(status != 0)
      {
             fprintf(stderr,"Errorea: ");
             fprintf(stderr,"%s",ER_MEZUAK[status]);
             exit(1);
       }

        if (clearbuf(buf)!=0){
            exit(1);
        }
```

8.17. Kodea: Extract.cpp aginduak eta id bidali

8.17. kodean ikusi dezakegun moduan bitan errepikatzen da prozesu guztia, hau da, hasteko `extract` agindua bidaltzen zaio (1) Arduino Megari eta erantzunaren zain geratzen da aplikazioa. Erantzuna `OKI` bada, aurrera jarraituko du. Aldiz, errore bat jasotzen badugu, komunikazioa bertan behera utziko da eta prozesua amaitu egingo da.

Beraz, `extract` agindua ondo jaso bada, prozesu berdina errepikatuko da, atera nahi den paketearen `id:zenbaki` bidaliz (2). Bidalketa hau ondo badoa, `insert.exe`-n ikusi dugun begiztara sartuko gara (8.18 kodea) eta Arduino Mega prozesua betetzen joatearen zain geratuko da `extract.exe`.

```
while(1)
     {
       if(readline(sock, buf, MAX_BUF)!=0){
         printf("Errorea bufferra irakurtzean\n");
         close(sock);
         exit(1);
       }
       printf("Jaso den komandoa: %s\n",buf);
       aukera=searchcommand(buf,komandoak);
       switch(aukera)
       {
          case AUK_PUT:
               sprintf(buf,"OK");
               if(write(sock,buf,strlen(buf))<0){
                  printf("Errorea bidaltzean\n");
                  perror("Errorea bidaltzean:");
                  close(sock);
                  exit(1);
                  break;
               }
               printf("Paketea utzi da plataforman\n");
               break;

         case AUK_OPEN:
               sprintf(buf,"OK");
               if(write(sock,buf,strlen(buf))<0){
                  printf("Errorea bidaltzean");
                  perror("Errorea bidaltzean:");
                  close(sock);
                  exit(1);
                   break;
               }
               printf("Paktetea ateratzera doa\n");
              break;
          case AUK_CLOSE:
               sprintf(buf,"OK");
               if(write(sock,buf,strlen(buf))<0){
                  printf("Errorea bidaltzean");
                  perror("Errorea bidaltzean:");
                  close(sock);
                  exit(1);
                  break;
               }
               printf("Paketea ez da bueltatu\n");
               break;

          case AUK_REGISTER:
               sprintf(buf,"OK");
               if(write(sock,buf,strlen(buf))<0){
                  printf("Errorea bidaltzean");
                  perror("Errorea bidaltzean:");
                  close(sock);
                  exit(1);
                  break;
               }
               printf("Paketea sistematik atera da\n");
               break;

           case AUK_DONE:
               sprintf(buf,"OK");
               if(write(sock,buf,strlen(buf))<0){
                  printf("Errorea bidaltzean\n");
```

```
                    perror("Errorea bidaltzean:");
                    close(sock);
                    exit(1);
                    break;
                }
                printf("Dena ondo joan da\n");
                close(sock);
                exit(0);
               break;
          default:
                errorea=parse(buf);
                fprintf(stderr,"Errorea: ");
                fprintf(stderr,"%s",ER_MEZUAK[errorea].c_str());
                printf("Errore bat egon da prozesuan:
%s\n",ER_MEZUAK[errorea].c_str());
                sprintf(buf,"OK");
                if(write(sock,buf,strlen(buf))<0){
                   printf("Errorea bidaltzean\n");
                   perror("Errorea bidaltzean:");
                   close(sock);
                   exit (errorea);
                   break;
                }
                close(sock);
                exit (errorea);
                break;
               }
        }
close(sock); // Socketa itxi.
```

8.18. Kodea: Extract.cpp Arduino Megaren mezuen zain

`insert.exe`-n gertatzen den moduan, Arduino Mega komandoak bidaltzen joango da, paketea ateratzeko prozesuaren ataza ezberdinak bete ahala. Ondorioz, komando hauen zain geratuko da web aplikazioa, hau da, `extract.exe`.

Komandoa jaso ondoren, komandoa edo errore kodea identifikatuko du. Definituta dauzkagun komandoak komunikazio protokoloan ikusi ditugunak dira (`put`,`open`,`clos`,`regi`,`done`). Komando bakoitzak bere egoera izango du, hala ere, guztietan barruan egingo den prozesua berdina da. Komandoa ondo jaso dela Arduino Megari adierazteko `OK` erantzuna bidaliko zaio, ondoren, prozesuaren berri emateko informazioa inprimatuz. Inprimatzen den informazioa, track fitxategian gordeko den testua izango da.

Ordea, aplikazioak jasotako mezua komandoen artean aurkitzen ez bada, errore bat jaso dela suposatuko da. Ondorioz, jaso den errore mezutik errore kodea aterako da `parse()` funtzioa erabiliz. Gainera, errore horri dagokion mezua inprimatuko da, `Track` karpetako dagokion fitxategian errorea agertzeko eta, bestetik, `extract.exe` programak errore kodearen balioa bueltatuko du, web aplikazioak errorea gertatu dela jakin dezan.

Honekin bukatutzat ematen dugu gure web aplikazioaren inplementazioaren azalpena.

9. Sistemaren ebaluazioa eta hobekuntzak

Kapitulu honetan, produktuaren garapenean zehar egin diren probak azalduko dira, baita proba horien ondorioz detektatu diren arazoak eta hauei emandako konponbideak ere.

Produktuaren atal guztiak, independienteki frogatzeko garatutako kodea, gorde da, produktua garatzen joan ahala berriro ere erabili ahal izateko. Behin sistema osoa garatuta izan dugunean, azkenengo proba orokorrak egin ditugu. Proba hauetan sistemak ezuzteko egoeren aurrean dauzkan erantzunak aztertu dira.

Egindako probak hiru multzo nagusietan banatu ditzazkegu: Web aplikazioaren funtzionamendu zuzena bermatzeko egindako proba isolatuak, bitegiratze-sistemaren funtzionamendua zuzena egiaztatzeko probak eta, azkenik, sistema osoari egindako proba unitarioak.

Sistemak behar bezala funtzionatzen duela frogatzeko, nahiko proba sinpleak egin ditugu: plataforma mugikorra ireki eta ixtea, plataforma mugikorrak paketeen identifikazioa, biltegiaren paketeen erregistroa, beso robotikoa biltegiratze-posizio bakoitzetik paketea hartu eta utzi, web aplikazioak egiten dituen errore kontrolak, informazioa datu-basean ondo sartzen den edo ez aztertu. Lortu diren emaitzak nahiko onak izan dira orokorrean. Hala ere, iragarri den moduan, akats batzuk aurkitu dira. Aurkitutako arazo batzuk azaltzera pasatuko gara orain, arazo hauei eman zaien konponbidea emanez.

9.1 Web aplikazioa

Web aplikazioa inplementatzean, aurretikan graduaren beste ikasgai baterako garatutako web aplikazioa hartu dugu oinarri bezala eta beraz, errore asko ez ditugu identifikatu, hauek aurretik konponduta genuezkan eta. Hala ere, `insert.exe` eta `extract.exe` programak deitzerako momentuan arazo batzuk izan genituen.

PHP-tik kanpoko programa bat exekutatzeko bi aukera aztertu ziren, `exec()` eta `shell_exec()`. Hauek erabili dira web aplikaziotik biltegiratze-sistemarekin komunikatuko diren programei deitzeko. Hasiera batean `shell_exec()` funtzioa erabiltzea pentsatu zen, honen erabilpena erraza zelako. Funtzio hau komando interpretean egin daitezkeen ekintzak simulatzen baititu. Hala ere, komunikazioaren garapenean aurrerago konturatu ginen, errore kontrolerako eta prozesuaren monitorizaziorako (`Track` karpetako fitxategiak sortzeko) funtzio honek eskaintzen zizkigun ezaugarriak guk behar genituenak baino gutxiago zirela. Ondorioz, `exec()` funtzioa erabiltzea erabaki genuen. Honek hainbat parametro hartzen ditu:

1. `Command`: Exekutatuko den komandoa edo programa.
2. `Output`: Programaren exekuzioan zehar irteera kanaletik atera den informazioa gordetzen duen aldagaia. Gure kasuan trazak egituratzeko balio izan digu, honetan programaren exekuzioan zehar egindako inprimaketa guztiak gordetzen baitira. Aldai honetan kate moduan gode egingo dira irteera kanaletik ateratako informazioa.
3. `Return_var`: Programak exekuzioa bukatzerakoan bueltatutako balioa.

Honekin gure web aplikazioak eta komunikazioa mantentzen duten `insert.exe` eta `extract.exe`-ren arteko elkarrekintza hobetu eta erraztu genuen.

9.2 Arduino UNO

Arduino UNO-k dauzkan funtzionalitateak egiaztatzeko momentuan, hainbat errore identifikatu ditugu.

9.2.1 NFR2401 irrati-uhin transmisorea (Bi arduinoen arteko komunikazioa)

Produktuaren lehenengo diseinua egin genuenean, gure biltegiratze-sistema osatzen duten bi plaken (Arduino Mega eta Arduino UNO) arteko komunikazioa irrati-uhin teknologia bitartez egitea erabaki genuen, horretarako NRF2401 moduluak erabiliz.

Modulu hauek erabiltzeko, produktuaren garapenean hainbatetan ikusi dugun SPI busa erabiltzea beharrezkoa da. Bus hau, une bakoitzean bakarrik modulu batek erabili dezake. Honek arazo batzuk sortu zituen RFID-RC522 modulua erabiltzeko momentuan, modulu honek ere SPI busa erabiltzen baitu. Hauek erabiltzen genituen bakoitzean busa erabiltzeko eskaera egin behar baitzen. Gainera, bi modulu hauek erabiltzeko guk inplementatu ez ditugun liburutegiak behar dira eta biak elkarrekin erabiltzeak SPI busaren erabilpen guztia aldatzea ekarriko luke.

Arduino Megan wifi-ra konektatzeko erabili den wifi shield-arekin arazo berdina aurkitu genuen.

Posible izango litzake modulu guztiak SPI busaren bitartez funtzionatzen jartzea, baina honek modulu hauen liburutegietan aldaketa sakonak eskatuko lituzke eta horretarako behar den denbora ez dugu eskuragarri, proiektu hau garatzeko denbora-muga baitugu.

Ondorioz, bi Arduinoen arteko komunikazioa egiteko beste teknologia edo moduluak aztertu genituen. Hainbat aztertu ondoren Bluetooth teknologia erabiltzen duten HC-05 moduluak erabiltzea erabaki zen. Bi arrazoiengatik aukeratu ziren:

- Kablerik gabeko komunikazio teknologia erabili nahi zelako eta Bluetooth teknologia oso zabalduta dagoelako gaur egun.
- HC-05 moduluak ez du SPI busa behar eta, gainera, honen konfigurazioa erreza ez bada ere, erabilpena oso erraza da.

9.2.2 Plataforma mugikorra (distantzia-sentsoreak)

Arduino UNO-rekin jarraituz, plataforma mugikorraren mugimenduak inplementatzerako momentuan, proben ondorioz ikusi zen distantzia-sentsore batek ematen zituen irakurketak ez direla oso zehatzak. Kontuan izanda sentsoreek ziurgabetasun maila bat daukatela, sentsoreak guk espero genituen irakurketak ez ditu egiten. Honen arrazoi posible bat, hainbat uhin eta kableak egonda, interferentziak egotea da. Berriro ere, hau konpontzeko hardwarearen muinean sartzea ekarriko luke eta honek desbideraketa handiak ekarriko lituzke proiektuan. Beraz, errorea software bitartez konpontzea erabaki da.

Plataforma ireki eta ixterako momentuan, hauen irakurketei esker, Arduino UNO-k plataforma mugitzen duen serbomotorra noiz gelditu behar den jakingo du. Beraz, irakurketa hauetan erroreak egotea, plataforma erdi irekita edo erdi itxita uztea ekarri dezake, sistema osoaren funtzionamendua oztopatuz.

Ondorioz, irakurketek plataforma itxita edo irekita dagoela adierazten dutenean, irakurketa hauek errepikatuko dira, aurreko irakurketa fidagarria dela egiaztatzeko. Honekin, hardwarearen integritatea eta plataformaren funtzionamendu zuzena bermatzen dugu. Etorkizunerako helburu bat, espero ez ditugun irakurketen arrazoia argitzea izango da.

9.2.3 Plataforma mugikorra (RFID-RC522 modulua)

Arduino UNO-rekin bukatzeko, paketeak identifikatzeko momentuan RFID irakurlearekin izandako arazoa aztetuko dugu.

RFID irakurleak pakete bat identifikatzen duenean ez genuen lortzen paketearen id-a berriro irakurtzea. Hau, paketea aldatu ez dela edo paketea plataformatik kendu den edo ez jakiteko beharrezkoa dugu. Horretarako liburutegiaren dokumentaziora jo genuen eta ikusi genuen RFID irakurleak bakarrik `ready` egoeran dauden paketeak identifikatzen dituela eta, beraz, behin etiketa identifikatuta etiketa egoeraz aldatzen da. Ondorioz, ezin daiteke berriro etiketa hori identifikatu.

Arazoa konpontzeko, etiketak identifikatzeko erabiltzen genuen `PICC_IsNewCardPresent()` funtzioa aztertzea erabaki genuen. Hemen konturatu ginen paketeak detektatzeko erabiltzen den funtzioa, `ready` egoeran dauden etiketak detektatzeaz gain, `ready` egoeran ez dauden etiketak `ready` egoeran jartzen dituela.

Ondorioz, ikusi genuen behin etiketaren id-a irakurri ondoren, etiketa berriro ere identifikatu ahal izateko birritan deitu behar geniola `PICC_IsNewCardPresent()` funtzioari. Etiketa identifikatuta dugula, berriro `PICC_IsNewCardPresent()`funtzioari deituz etiketa identifikagarria bilakatzen da berriro, etorkizun batean plataforman dagoen paketea berriro irakurgai utziz. Adibidez 6.9 kodean pakete bat sartzeko momentuan, plataformaren gainean paketea uzten denean honen identifikazioa ikusi dezakegu.

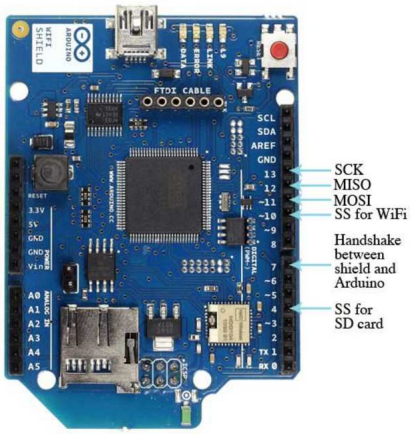
9.3 Arduino Mega

Behin plataforma mugikorrarekin izandako arazoak ikusita, Arduino Megaren funtzionalitateak inplementatzeko momentuan izan ditugun arazoak aztertzera pasatuko gara. Gogoratzeko, Arduino Megak beso robotikoaren eta biltegiratuta dauden paketeen administrazioa kontrolatzen ditu. Gainera, web aplikazioarekin komunikazioa izateko wifi sarera konektatuta egon behar da eta horretarako wifi shield-aren kontrola garatu da ere. Azkenengo funtzionalitate honetatik hasiko gara, wifi konexioarekin, alegia.

9.3.1 Wifi konexioa

Wifi konexiorako, Arduinok eskaintzen duen wifi shield-a erabili dugu. Honen kontrolarako, Arduinok lana asko errazten duen `<Wifi>` liburutegia erabili da. Beraz, hau inplementatzeko momentuan arazo handirik ez ditugu izan. Hala ere, bi errore aurkitu ditugu.

- Wifi shield-aren funtzionamendua independenteki probatu ondoren eta funtzionamendu zuzena zeukala egiaztatu ondoren, Arduino Megaren beste elementuekin batera probatzera pasatu ginen. Hau egiterako momentuan, ikusi genuen Arduino Megak ez zuela antzematen wifi shield-a konektatuta zeukala, nahiz eta wifi shield-a ondo konektatua egon. Beraz, arazo hau aztertu genuen. Wifi shield-arekin independienteki egindako proben artean eta orain egiten ari ginenen proben artean, aldatzen zen bakarra, beso robotikoaren pin-ak Arduino Megara konektatu genituela zen. Ondorioz, arrazoia hor egon behar zela ondorioztatu genuen. Beraz, proiektua hasi baino lehen egindako dokumentazioaren azterketara erreparatuz zazpigarren pin digitala ezin zela erabili ikusi genuen, hau erabiltzen baitzen wifi shield eta Arduino plakaren arteko komunikaziorako. Beraz, horretan ezin genuen ezer konektatu. Pin egitura aldatu genuen, orain daukagun egiturara (7.1.2 atalan). Wifi shield-ak kontrolarako erabiltzen dituen pin-ak 9.1 Irudian ikusi daitezke.



9.1. Irudia: Wifi-Shield pin erabilera.

- Bestetik, proba ezberdinak egiterako momentuan, ikusi genuen gure kodean wifi konexioa ezarri ondoren, ez genuela berriro ere konexioa ondo zegoen edo ez egiaztatzen. Probak egiterako momentuan, wifi sarea eskuragarri egoteari uzten bazion gure Arduino blokeatuta geratu egiten zen. Ondorioz, wifi-ra konektatzeko funtzio bat inplementatzea erabaki genuen, non sarera konektatuta gauden edo ez egiaztatzen duen eta, konektatuta ez bagaude, wifi sarera konektatzen saiatuko den. Beraz, hemendik aurrera eskaera berri baten zain geratzen den bakoitzean, Arduinok sarera konexioa daukan egiaztatuko du. Honekin wifi konexioa galduz gero, automatikoki berriro konektatzeko gai izango da sistema.

9.3.2 Beso robotikoa

Wifi konexioarekin izandako arazoak aztertuta, beso robotikoarekin izandako arazo nagusienak aztertuko ditugu. Hainbat errore agertu zaizkigu, behin beso robotikoa inplementatuta geneukanean. Agertu zaizkigun errore guztiak erlazionatuta dauden arren, hauek banaka ikusiko ditugu.

- Beso robotikoa inplementatuta genuekanean, pakete eta biltegiratze-eremu ezberdinekin probak egiten hasi ginen. Proba hauetan hainbat biltegiratze-eremuetan paketea utzi eta gero, beso robotikoa hasierako egoerara bueltatzen zenean, beste biltegiratze-eremuetan utzitako paketeak ukitu eta mugiarazten zituela konturatu ginen. Honek, paketea posizio desegoki batean uztea eragiten zuen. Honetaz aparte, antzeman genuen goiko biltegiratze-eremuek paketeak bazituzten eta beheko pakete bat uztera pasatzen baginen arazoak sortzen zirela. Izan ere, beso robotikoa hasierako posizio honetara bueltatzean, goiko biltegiratze-eremuan aurkitzen zen paketea mugiarazten baitzuen. Ondorioz, hau ekiditzeko, paketeak utzi edo hartuz gero, beso robotikoa hasierako posizioara zuzenean bueltatu baizik, segurtasun mugimendu batzuk egiten ditu, paketeekin talkarik egin ez dezan.
- Hasiera batean, beso robotikoaren mugimenduak inplementatu genituenean, ez genion beso robotikoari agindutako posizioak hartzeko inongo denborarik ematen. Ondorioz, batzuetan beso robotikoak hurrengo agindua ailegatu baino lehen agindutako posizioetara ailegatzeko denborarik ez zeukanez, portaera arraroa erakusten zuen. Orduan, mugimenduen artean denbora espazio bat uztea erabaki genuen (horretarako 6. kapituluan ikusitako `timer()` funtzioa inplementatu genuen). Honela, beso robotikoari denbora ematen zitzaion posizioak ondo hartzeko eta bidalitako aginduak ondo betetzeko. Hala ere, beso robotikoak mugimendu sorta ezberdin asko dituenez, mugimenduen artean itxaron behar den denbora oso ezberdina da. Beraz, itxaron

beharreko balio hau estandarizatzeko asmoz eta errendimendu handiago lortzearren `StopTime` balioa definitu genuen `BesoRobotikoa` klasean. `StopTime` aldagai honen balioa denbora gehien kostatzen zaion mugimenduarena izatea erabaki da. Hala ere, etorkizun batean itxaron denbora hauek dinamikoki esleitzea pentsatu da, errendimendua hobetu nahian.

- Azkenik, beso robotikoa elikatzerako momentuan arazo batzuk izan genituen. Hasiera batean Arduino Megaren konexioak finkatzerakoan, beso robotikoa plakak eskuragarri daukan 5V Vout pinarekin elikatzea pentsatu genuen. Hala ere, probak egiterako momentuan, pin honek ematen duen korrontea ez dela nahiko antzeman genuen. Beraz, beso robotikoa elikatzeko kanpoko elikadura behar zela ondorioztatu genuen. Kanpoko elikadura iturria aukeratzeko momentuan bi gauza izan behar dira kontuan, eskainiko duen tentsioa eta korrontea. Beti ere jakinda erabilitako serbomotorrak 5V eta 6V tartean funtzionatzen dutela, eskuragarri geneukan 5V eta 2A elikagai-iturri batekin egin genuen proba, eta honekin lortu genuen gure beso robotikoa modu egokian elikatzea.

9.3.3 Paketeen administrazioa

Dokumentuan zehar aztertu dugun moduan, Arduino Megaren beste funtzionalitateen artean, biltegian gordeta dauden paketeen administrazioa daukagu. Hau lehen azaldu den moduan `Bitegia` klasearen bitartez inplementatu da. Hala ere, honen probak egiterakoan, errore batekin aurkitu ginen.

- Arduino edozein momentuan itzaltzen bada edozein arrazoirengatik, momentu horretan biltegian gordeta dauden paketeen informazioa galduko litzateke. Ondorioz, Arduino berriro ere pizterakoan, `Biltegia` klasea berriro hasieratzean, biltegia osoa libre bezala agertuko zen. Hau integritate errore handia da, sistema osoak gaizki funtzionatzea ekarriko lukelako. Arazo honi erantzun bat emateko, Arduino plaka guztiek eskuragarri daukaten EEPROM memoria erabiltzea pentsatu genuen. Horrela, gordeta dauden paketeen informazioa momentu oro eguneratuta mantenduko dugu ROM memoria honetan eta paketeen informazioa ez da galduko.

  EEPROM memoria honetan bytez byte idatzi behar da, baina guk `Biltegia` klaseko bai `inUse[]` eta `packages[]` taulak gorde nahi ditugu. Osokoak eta karaktere-kateak EEPROM-en idazteko eta EEPROM-etik irakurtzeko liburutegi bat aurkitu genuen Arduino foroan (https://playground.arduino.cc/Code/EepromUtil), `<EEPROMUtil>`. Honen bitartez, posible dugu gure paketeen informazioa memorian gorde modu erraz batean, byte kontrolaz arduratu gabe. Beraz, orain Arduino Mega hasieratzen den bakoitzean, EEPROM-en gordeta dauden balioak hasieratu berri den `Biltegia` klaseko obejktura pasatuko dira. Horrela, Arduinoan dauden paketeen informazioa ez da galduko eta sistemak ondo funtzionatuko du.

9.3.4 Komunikazioko erroreak

Azkenik, komunikazioan zehar gertatu diren erroreak aztertzera pasatuko gara. Gogoratu bi komunikazio mota ikusi ditugula proiektu honetan, wifi socket bidezko komunikazioa eta Bluetooth bitartekoa. Wifi komunikazioan gertatu direnak ikustera pasatuko gara.

- Arduino Megari esleitzen zaion IP helbidearekin arazo batzuk izan genituen. Izan ere, proiektuan definitu den moduan, zerbitzari lanak Arduino Megak egingo ditu, bezero lanak, berriz, web aplikazioak. Beraz, web aplikazioan socket-a egituratzerakoan, zerbitzariaren IP helbidea finkatuta daukagu aldagai batean. Honek, arazo batzuk sortarazten dizkigu, sarez aldatzen dugunean Arduino Megari esleitzen zaion IP helbidea ezberdina izan daitekelako. Web aplikazioak socket-a sortzerakoan definituta daukan

helbidea eta Arduino Megak daukan IP helbideak berdinak ez badira, errore bat gertatuko da. Arazo honetarako, bi soluzio posible ikusten dira, web aplikazioan funtzionalitate berri bat sartzea, non administratzaileak eskuz sartu ahal izango luke IP helbidea. Horrela komunikazioa aurrera eramaten duten programei (Insert.exe eta Extract.exe) IP helbide hau argumentu bezala pasatuko zitzaien (gogoratu orain programa hauetan Arduino Megaren IP helbidea kodean eskuz txertatuta dagoela). Honekin IP helbidea aldatzea ez zen izango arazoa eta gure sistemaren errendimendua eta erabilgarritasuna asko hobetuko luke. Hala ere, soluzio hau inplementatzea dugun denbora eta definitutako proiektutik kanpo geratzen den soluzioa da. Beste aukera bat, sistema konektatzen den sareak Arduino Megari IP estatiko bat esleitzea izango litzateke. Azkenengo soluzio hau egitea zentzuzkoena dela pentsatu da, baliabideen aldetik eta logistikaren aldetik. Gainera, sistemak leku finko batean egoteko diseinatu da, beti sare berdinera konektatuta egoteko. Hala ere, denbora faltaren ondorioz, erantzun honen inplementazioa etorkizunerako utzi da.

Honekin, proiektua garatu dugun heinean, aurkitu ditugun errore nagusienak aztertu ditugu eta hauei eman zaien konponbideak azaldu ere.

10. Jarraipen eta kontrola

Kapitulu honetan proiektuaren garapenean eta egindako plangintzaren arteko ezberdintasunak azaltzen dira, proiektuaren garapenean izandako desbideraketak, kalitatearen kontrola eta baita identifikatutako arriskuek izan duten eragina ere.

10.1 Planifikazioarekiko desbideraketak

Proiektuko garapenean, hasierako plangintzarekin alderatuz, desbideraketa batzuk gertatu dira. 10.1 irudian daukagu estimatutako eta errealitatean erabilitako denboraren arteko ezberdintasunak. Desbideraketa garrantsitzuak azalduko ditugu.

10.1.1 Garapen prozesua

Produktua eta proiektuak garatzeko prozesuan izandako desbideraketa totalari erreparatzen badiogu, desbiderapen nahiko txikia sufritu dugu. Ezin da gauza bera esan garapeneko ataza bakoitzari erreparatuz gero. Adibidez, erabiltzaile eta administratzaile web apikazioak egiteak uste baino denbora gutxiago eraman digu. Aldiz, bai plataforma mugikorraren eta baita biltegiaren kontrolaren garapenak planifikatutakoa baino denbora gehiago kostatu zaigu.

A.3 atazako biltegiratze-sistemaren garapenak (A3.1, A3.2, A3.3) modu orokor batean begiratuta, ikusi dezakegu desbideraketa handia suposatu duela, aurretikan planifikatuta geneukan ordueekiko. Izan ere, sistemaren atal garrantsitzuena izanda, lortu nahi genuen kalitate maila, errendimendu maila eta funtzionalitateak behar bezala garatzeko, denbora gehiago inbertitzera behartuta ikusi gara.

Hala ere, lehen esan dugun moduan, proiektuaren beste atal batzuetan espero genuena baino denbora gutxiago inbertitu dugu. Adibidez, web aplikazioa garatzean, aurretikan inplementatutako beste aplikazio bat oinarri bezala hartu genuenez, honek denbora asko aurreztu digu. Bestetik, garapena hasi baino lehen teknologia berrien informazioa eta ikasketei eskainitako denboran ere desbideraketa batzuk gertatu dira. Adibidez, Arduinoren funtzionamendua eta garatzen hasi baino lehenago lortu beharreko ezagutzak lortzeak, guk estimatutako baino denbora gutxiago kostatu zaigu. Informazio bilketa biltegi-sistemaren garapena baino lehenago egin dugunez, biltegia garatzerako momentuan bagenekien eskuragarri geneukan denbora gehiago zela eta horregatik, azkenean proiektuan gertatu den desbideraketa ez da hain handia izan. Laburbilduz, pairatu dugun desbideraketa A1, A2, A3, A4 atalak elkartzen baditugu, 2 ordukoa izan da, arbuiagarria, beraz.

10.1.2 Dokumentazioa

Gure proiektuak sufritu duen desbideraketa handiena dokumentazioa egiterakoan izan da. Nahiz eta honetarako estimatutako denbora nahiko luzea izan, azkenean estimatutako baino denbora gehiago inbertitu behar izan dugu. Desbiderapen honen arrazoi nagusia, memoria idazterako momentuan produktibitate falta izan da. Hau esperientzia faltatik etorri da, ez baikaude ohituta proiektu hain handi bat azaltzera.

Memoria idazten joan ahala ohartu gara planifikatutako baino denbora gehiago beharko genuela. Hala ere, memoriak kalitate maila egokia izateko beharrezko den denbora erabiltzea erabaki dugu. Azken finean, dukumentu hau proiektuan zehar egindako lan guztiaren laburpena baita.

Lan paketea	Estimazioa(ordutan)	Erreala	Desbideraketa	
A1.Kudeaketa	31,00	29,00	-2,00	-6,45%
A1.1 Plangintza	20,00	20,00	0,00	0,00%
A1.2 Jarraipen eta kontrola	9,00	7,00	-2,00	-22,22%
A1.3 Tutoreekin bilerak	2,00	2,00	0,00	0,00%
A2. Analisia eta Prestakuntza	42,00	30,00	-12,00	-28,57%
A2.1 Informazioa bilketa	8,00	8,00	0,00	0,00%
A2.2 Arduino eta moduluak	15,00	10,00	-5,00	-33,33%
A2.3 Erabitzaile eta administradore aplikazioak	10,00	4,00	-6,00	-60,00%
A2.4 Garapenaren ingurunea	9,00	8,00	-1,00	-11,11%
A3. Garapena	150,00	166,50	16,50	11,00%
A3.1 Biltegia eraiki	10,00	11,50	1,50	15,00%
A3.2 Platafoma mugikorra	45,00	60,00	15,00	33,33%
A3.3 Beso Robotikoa kontrola	55,00	65,00	10,00	18,18%
A3.4 Erabiltzaile eta administradore aplikazioa garatu	30,00	20,00	-10,00	-33,33%
A3.5 Elementu guztien arteko komunikazioa	10,00	10,00	0,00	0,00%
A4. Egiaztapena	10,00	10,00	0,00	0,00%
A4.1 Produktuaren probak	3,00	3,00	0,00	0,00%
A4.2 Emaitzak aztertu	1,00	1,00	0,00	0,00%
A4.3 Errore identifikatu eta zuzendu	6,00	6,00	0,00	0,00%
A5. Dokumentazioa	115,00	130,00	15,00	13,04%
A5.1 Memoria idatzi	75,00	90,00	15,00	20,00%
A5.2 Eranskinak	15,00	15,00	0,00	0,00%
A5.3 Erabilpen kasua	15,00	15,00	0,00	0,00%
A5.4 Aurkezpen Prestatu	10,00	10,00	0,00	0,00%
TOTALA	**348,00**	**365,50**	**17,50**	**5,03%**

10.1. Irudia: Hasierako plangintzarekiko izandako desbiderapenak

10.2 Kalitatea

Gure proiektua garatuta daukagula, hasiera batean definitutako kalitate mailarekin konparatuko dugu, lortutako kalitate-maila zehaztu nahian. Horretarako plangintzan definitutako kalitate plana oinarri bezala hartuko dugu (1.6 atala).

10.2.1 Adierazle kuantitatiboak

- Plangintzaren jarraipena: Nahiz eta batzuetan finkatutako plangintza ez dugun jarraitu, orokorrean plangintza bete dugula esan dezakegu. Plangintza egokia diseinatzea eta honi ahalik eta gehin jarraitzeak garrantzi handi dauka proiektuaren arrakastan. Plangintzari jarraitzeak, epeak errespetatzea eta proiektuaren oinarrizko maila betetzeak ekartzen du. Egia da desbiderapen batzuk gertatu direla, baina hauek proiektuaren arrakastan ez dute eragin handirik izan.
- Errekurtso minimoak erabili: Proiektuaren hasieran, helburu bezala, aurrekontua ahalik eta txikiena izatea jarri genuen. Honetarako, beste hainbat proiektuetan erabili ditugun elementuak eta moduluak erabiltzea erabaki genuen. Honela, aurrekontua minimizatu dugu, beharrezko elementuak beste lekuetatik lortuz. Helburu hau lortu dugula esan dezakegu, Arduino Mega eta RFDI-RC522 moduluez aparte, ez baitugu beste elementurik erosi behar izan. Biltegia erakitzerakoan, etxean geneuzkan materialen birziklapena egin da.

10.2.2 Adierazle kualitatiboak

- Biltegi fisikoa: Biltegi finko eta erabilgarri bat eraikitzea lortu dugu. Honetan, elementu guztientzako lekua aurkitu dugu. Gainera, plataforma mugikorraren mugimendu mekanikoa, arazo handienak eman arren, modu egokian konpondu dugu.

- Plataforma Mugikorra (Arduino UNO): Bete behar dituen bi funtzioak ondo betetzen ditu. Paketeak biltegitik atera eta sartzen ditu, paketeen identifikazioa eginez. Aurkitu dugun arazo bakarra, NRF2401 irrati-uhin transmisoreekin izan da. Hala ere, HC-05 Bluetooth moduluengatik aldatuz, Arduino Megarekin komunikazioa arazorik gabe funtzionatzea lortu dugu.
- Beso Robotikoa (Arduino Mega): Bete beharreko funtzio guztiak zuzen betetzen ditu. Beso robotikoak modu kontrolatu batean paketeak biltegian zehar mugiarazten ditu. Gainera, paketeen administrazioa eramateaz arduratzen da, paketeen arteko erroreak ekidituz. Bestetik, plataforma mugikorrarekiko komunikazioa finko eta fidagarria ezartzea lortu da, HC-05 bluetooth moduluen bitartez. Azkenik, web aplikaziorekin wifi bitartezko komunikazioa sortzea eta mantentzea lortu da, honetan gertatu daitezkeen erroreak kontrolatuz.
- Web aplikazioa: Biltegiratze-sistemari aginduak bidaltzeko erabiliko den interfazea da. Honek, erabiltzaileari modu erraz eta irisgarri batean gure biltegiarekin elkarrekintza egiteko aukera ematen dio. Sitemaren funtzonamenduaren arduraduna izan arren, lortu da ere erabiltzailentzako erabil erraza izatea.
- Sistemaren fidagarritasuna: Sistema osoaren fidagarritasuna ezinbestekoa da, hardware elementuen integritatea bermatu ahal izateko. Ondorioz, proba askoren ondorioz, detektatu diren erroreak behin konponduta, egindako azkenengo probetan ez da fidagarritasun arazorik aurkitu eta beraz, gure sistema fidagarria dela esan dezakegu.
- Kodearen ulergarritasuna: Produktu informatikoa garatzen denean ezinbestekoa da garatu den kodea ulergarria izatea, etorkizun batean hau berrerabili behar duenak, arazorik ez izateko. Ondorioz, gure kodea oharrez eta iruzkinez beteta dago, funtzio eta metodo bakoitzak egiten duena ulergarria izateko. Horretaz aparte, funtzio eta aldagientzako izenak aukeratzean, saiatu gara hauek egiten dutenaren adierazgarri izatea, ulermena erraztuz.

  Kodearen argitasuna alde batera utzita, prozesu osoan zehar mezu inprimaketa asko ageri dira, horiek denak arazketari begira informazio baliagarri ematen dutenak. Honela, programa erabiltzerakoan honek egiten duena ikusi daiteke.

10.3 Arriskuak

1.10 atalan proiektuan gerta zitezkeen arriskuak zehaztu genituen. Hauek gure garapenean izan duten eragina aztertzera pasatuko gara orain.

- Hardwarearekin zerikusia daukaten guztiak: Hardwarearekin ez dugu arazorik izan, guztiak funtzionatu behar duen moduan funtzionatzen baitu. Egia da distantzia-sentsore batekin arazoak izan ditugula, honek batzuetan egiten zituen irakurketak ez zirelako egokiak. Arazo hau software bitartez konpontzea lortu dugu. Bestetik, wifi konexioarekin ez dugu arazo asko pairatu.
- Datuen galera: Zorionez, ez dugu datu galerarik pairatu. Hala eta guztiz ere, definitu genituen jarraibideak jarraitu ditugu eta periodikoki segurtasun kopiak egin ditugu, bai sarean eta baita gailu fisiko batean ere. Horrela, segurtasun maila handia lortu dugu.
- Denbora falta: Egia da, lehen azadu den moduan, desbideraketak direla eta, estimatuta zeuden orduak baino gehiago inbertitu ditugula. Honek, denboran eragina izan du, hala eta guztiz ere, proiektua bukatu dugun eta aurkeztu behar den data mugaren artean, denbora tarte bat utzi genuen planifikazioan, ezusteko baten aurrean ekintza eremu bat eduki ahal izateko.

11. Ondorioak

Kapitulu honetan proiektuarekin lortutako emaitzak eta atera diren ondorioak azalduko ditugu, bukatzeko, proiektua garatzen jarraituko bagenu, hobetzeko zein aukera edukiko genituzkeen aztertuz.

11.1 Lortutako emaitzak

Biltegiratze-sistema automatizatu bat garatzea lortu dugu, hasiera batean jarritako helburu guztiak betez. Alde batetik, web aplikazio bat garatu dugu, erabiltzaileak gure sistemarekin daukan elkarekintza erraztuz eta ulergarriago eginez. Bestetik, web aplikazio honetatik ailegatzen diren aginduak asetzen dituen biltegiratze-sistema garatzea lortu dugu, paketeak gordez eta ateraz. Beraz, esan dezakegu gure sistemak helburu bezala paketeak gorde eta ateratzea duela, eta hau bera egiten duen sistema bat garatzea lortu dugula.

Gure sistemaren funtzionamenduaren adibide bat izateko bideo bat prestatzea erabaki da. Honetan, gure sistemaren funtzionamendua aztertzen da, sistemak dituen elementu desberdinak aztertuz. Bideoa bi zatitan banatu egiten da.

- Lehenengo atalan gure biltegiratze-sistemaren funtzionamendua aztertuko dugu, erabilpen kasu ezberdinak aurkeztuz. Erabiltzaile arrunt bat gordeta daukan pakete bat ateratzea, etab.
- Bigarren atalean gure biltegiratze-sistemak egiten duen errore kontrola aztertuko dugu, aurreikusi egin diren errore batzuen aurrean gure sistemak daukan jarrera azalduz.

Bideoa hurrengo estekan aurkitzen da:

https://youtu.be/8eZ2diY4Mqk

Lortu den emaitzarekin benetan gustura gaude. Gure sistemak ez dauka, momentu honetan, exisititzen diren biltegiratze-sistema profesionalen ezaugarri, ez funtzionalitate berdinak. Garatu den produktua, sistema erreal baten prototipo txikia dela esan dezakegu. Hala ere, egin behar dituen funtzioak, modu egoki batean betetzen ditu eta, gainera, erabiltzaile ez-aditu batek web aplikazioaren bitartez erabili dezake.

11.2 Ikasitakoa

Proiektu honen helburua, gradu osoan zehar lortutako gaitasunak frogatzea da. Gaitasun hauek, ez dira bakarrik programen diseinuan eta kodeketan lortu direnak, baizik eta orokorrean proiektu baten aurrean egin beharreko guztien inguruko gaitasunak ere. Aurrera eraman den proiektua guk pentsatutako eta diseinatutako proiektua izanda, bagenekien arazo asko ekarri zitzakeen proiektua zela. Hala ere, gure burua proiektuari aurre egiteko moduan ikusten genuen eta lortu dugun emaitzarekin oso gustora gaude. Kontuan izanda proiektua, beti bezala, hobetu daitekeela.

Proiektuak iraun duen denboran zehar, hainbat gauza ikasi ditugu. Batzuk, zuzenean gure informatika gaitasunekin zerikusia daukatenak dira eta beste batzuk, gure etorkizunerako, proiektu bat planifikatzerakoan egin beharrekoen inguruan direnak. Bestalde, presioaren aurrean eta denbora luzerako proiektu bat egiterakoan, pertsonalki izan dugun portaerak ezagutzeko balio izan digu. Segidan azalduko ditugu proiektu honetan zehar ikasi ditugun eta interesgarriak diren hainbat lekzio.

- Nola ez, aurretikan erabili ez ditugun hainbat teknologiak erabili ondoren, hauetan hainbat gaitasun lortu ditugu. Adibidez, Arduino munduaren inguruan, egin den garapenaren ondoren, Arduino plaka ezberdinak erabiltzen ikasi dugu. Honetaz aparte, plaka hauekin erabili diren shield, modulu eta sentsore/eragingailuen inguruan asko ikasi dugu ere. Arduino mundua alde batera utzita, aurretikan, web garapenean geneuzkan ezagutzak praktikan jartzeko eta ezagutze hau zabaltzeko balio izan digu ere.
- Proiektu luze bat aurrera eramatean ezinbestekoa da, hasi baino lehen planifikazio egoki bat egitea. Horrela, gure denbora eta lana ondo antolatzen badugu, denbora falta eta itomen sentzazioak ekiditu egin ahal izango ditugu, motibazioa ez galduz eta proiektuarekiko nekea gutxituz. Proiektua hasi ondoren, oso garrantsitzua da ere egunero helburu txikiak ezartzea, hauek betetzen joateko, motibazioa mantenduz. Honekin batera, egunero egiten den lana eta dedikatutako orduak apuntatzeak asko laguntzen du, geroago, dokumentazioa egiterako momentuan informazioa izateko.
- Esan dugun moduan, proiektua garatzen den bitartean dokumentazioa idazten joatea oso erabilgarria da, geroago, hainbat gauza galdu edo ahaztu daitezkelako. Zer dokumentatu behar den hautatzerako momentuan, memoriarako garrantsitzuak diren gauzak bakarrik dokumentatu behar dira.
- Kodea garatzerako momentuan, oso garrantzitsua da aurretikan dokumentatzea eta egin nahi den funtzionalitateak egiteko tresna egokiak bilatzea. Honek garapenean denbora asko aurreztea ekarriko digu, jadanik definituta dauden liburutegiak aurkitu ditzazkegulako. Hau gainera, Arduino bezalako hain zabalduta dagoen komunitateaz hitz egiten dugunean, sarean dagoen informazioa mordoa da.
- Hainbat elementuz osatuta dagoen proiektu bat garatzeko momentuan, oso garrantsitzua da elementu bakoitzak proiektuaren barruan dauzkan helburuak eta funtzioak zehaztea. Honekin batera, beraien arteko komunikazioa aurretikan ondo definitzea ezinbestekoa da, geroago egon daitezkeen erroreak ekidin ahal izateko. Gainera, diseinu fisikoa (biltegia) eta elementu elektroniko/informatikoak elkartzerakoan, hainbat gauza izan behar dira kontuan. Espazioa, kableak, mugikortasuna etab.
- Bestetik, kodea ondo dokumentatzea eta iruzkintzea ezinbestekoa da, geroago kodea irakurtzean, kodea ulertu ahal izateko. Honekin batera, kode garbi eta modularra izatea oso garrantsitzua da. Horrela errore bat gertatzen bada, aldatu beharreko kodea zatitxo bat edo funtzio bat izango da, eta ez kode osoa.
- Babes-kopiak edukitzeran baliagarritasuna eta garrantzia egiaztatu da. Ez bakarrik segurtasun maila handia izateko, baizik eta informazioa momentu oro eskuragarri izateko. Honek ere, garapena hainbat ordenagailuetan egitea ahalbidetzen duelako.

11.3 Etorkizunerako lana

Gure proiektua hemen bukatzen den arren, oraindikan ere badira hobetzeko alderdiak. Hauek, bi taldetan banatu daitezke. Alde batetik, orain arte egindako lana hobetzeko egin daitezkeen hobekuntza sinpleak, denbora faltagatik inplementatu ez direnak eta, bestetik, gure irismenetik kanpo geratzen ziren hainbat futzionalitate berri.

- Biltegia eta web aplikazioaren komunikazioa bat-batean galtzen denean, biltegiaren funtzioak erantzun baten zain egon daitezkeen denbora pasatzean, errorea dagoela identifikatzen du. Etorkizunean errore hori identfikatzerakoan, biltegia zein egoeratan dagoen identifikatu eta biltegia egoera egoki batean uzteko egin beharreko prozesuak inplementatu beharko lirateke. Orain arte, adibidez, paketea plataformatik sartu berri

denean komunikazioa bat-batean ixten bada, Arduino Megak komunikazioan errore bat dagoela identifikatuko du, eta zuzenean eskaera berri baten zain egotera pasatuko da. Arazoa sartu berri den paketearekin dago, gaur egungo sisteman pakete horrek plataformaren gainean jarraituko du, eta administratzaileak eskuz kendu beharko du. Beraz, ondo legoke, adibidez, paketea berriro ateratzea edo behin-behineko biltegiratze-eremu batean uztea, hurrengo eskaeren funtzionamendua ez oztopatzeko.

- Biltegiaren kontrol manuala: Denbora dela eta, ezin izan dugu horrelakorik garatu. Honen bitartez administratzaileak urruneko kontrolaren bitartez biltegiratze-sistema erabiliko luke joystic baten bitartez.
- Web aplikazioaren itxura: Denbora dela eta, egin den web aplikazioren interfazea oso basikoa da. Ondo legoke, azkenengo tekonologiak (HTML5 eta CSS3) erabiliz interfaze dinamiko eta polit bat egitea. Honekin batera ondo legoke orrialdea mugikorrentzako ere egokitzea.
- Distantzia-sentsorearen funtzionamendua: Lehen esan den moduan, ondo legoke, distantzia-sentsorearen irakurketa desegokiaren zergatia argitzea.
- Arduino Megaren IP helbideraren arazoa: Bi modu aztertu dira. Biltegiratze-sistema erabiltzen ari garen sarean Arduino Megari IP helbide estatiko bat esleitzea. Hau inplementatzeko denbora asko ez da behar. Hala ere, sistema errealitatean leku eta sare zehatz batean finkatzen ez den arte, konponbide hau inplementatzeak ez da egokia. Bestetik, web aplikazioan funtzionalitate berri bat gehitu ahal izango litzateke, non administratzaileak eskuz sartu ahal izango luke Arduino Megaren IP helbidea. Honekin errendimendua eta erabilgarritasuna asko handituko litzateke, baina hau inplementatzea aurreko soluzioa baino denbora kostu gehiago suposatuko luke.

Bestetik, etorkizunerako gehitu daitezkeen funtzionalitate batzuk aztertuko ditugu orain.

- Pakete ezberdinak sartzeko ahalmena: Garatu den proiektuan, maneiatzen diren pakete guztiek, egitura berdina daukate, gure beso robotikoak daukan ahalmen txikia dela eta. Beraz, etorkizun batean, beso robotiko handiago bat funtzionalitate handiagoarekin erabiltzeak pakete ezberdinekin lan egitea ahalbidetuko luke.
- Beso robotikoak plataforma mugikorretik paketeak hartzeko modua: Gure proiektuan, beso robotikoak guk utzitako paketeak ondo hartu ditzan, paketeak posizio eta modu jakin batean utzi behar dira plataforman. Beraz, ondo legoke etorkizunerako beso robotikoari kamara bat edo sentsore batzuk gehitzea, berak paketearen posizioa eta norabidea kalkulatuta paketeak modu automatikoan hartu ditzan. Horrelako funtzionalitatea inplementatzeko Arduino plakak ez luke ahalmen nahikorik izango. Ondorioz, beste sistema txertatu potente batengatik aldatu beharko genituzke (Rasperry Pi). Funtzionalitate hau ROS frameworkaren bitartez inplementatzea posible izango litzateke.
- Paketeak RFID etiketa ezberdinak izatea: Produktu honetan, dokumentuan zehar esan dugun moduan, exisititzen diren RFID etiketa ezberdinak erabiltzeko ahalmena gehitu nahi da. Gainera, kontuan izanda, etiketa hauek 1 edo 4 Kb-eko EEPROM memoriak dituztela, ondo legoke hauetan nolabaiteko informazio gehigarria gordetzea, paketeen kontrola zehatzago izan dadin.
- Email edo SMS bidezko abisua: Ondo legoke erabiltzaileak gure sisteman ekintzaren bat betetzen duen bakoitzean email-era konfirmazioa bidaltzea, segurtasun neurri bezala. Honekin ere, sisteman erregistratzerakoan, email-a zuzena dela egiaztatu ahal izango litzateke. Bestetik, sisteman erroreren bat detektatzen denean, administratzaileei mezuren bat bidaltzea pentsatu da.

- Sisteman gertatzen diren komunikazio eta datuen enkriptazio zuzen bat garatzea: Alde batetik, web aplikazioan gordetzen den informazio konfidentziala azkenengo enkriptazio tekniken bidez inplementatzea, segurtasun maila handitu ahal izateko. Momentu honetan SHA1 teknologia erabili egiten da, baina enkriptazio mota hau gaur egun ez da segurua. Bestetik, adibidez, Arduino Mega eta web aplikazioren arteko socket komunikazioa, komunikazioa enkriptatua bilakatzea egokia izango litzateke.
- Bukatzeko, diseinatu eta eraiki den biltegia, gure proiekturako prototipo moduan egin da. Beraz, honen egituran, hainbat hobekuntza egitea posible izango litzateke. Adibidez, biltegiratze-eremu gehiago definitzea edota barruan gertatzen dena jakiteko, kamara bat instalatzea.

ERANSKINAK

A. Erabilpen gida

A.1 Biltegia prestatu

Biltegiaren prestakuntza modu erraz batean egingo dugu. Kontuan izan behar dugu, biltegiaren oinarrian, beso robotikoa eta plataforma mugikorra finko egongo direla. Bestetik, Arduinoak paretara itsatsita egongo dira, velcro batzuen bidez. Beraz, hauek kendu edo jarri ahal izango dira edozein momentuan.

Bestetik, biltegia funtzionatzeko beharrezko konexio guztiak eginda daude. Edozergatik kableak deskonektatuko balira, 5.1 atalean egin beharreko konexio guztiak definituta daude, eta bertan dauden eskemak jarraituz, erabiltzaileak sistemak funtzionatzea lortu beharko luke.

Beraz, behin kable guztiak prest daudela, biltegia erabiltzera pasatuko gara. Biltegiratze-sistema osoaren prestakuntzaren barruan wifi konexioa da korapilatsuena. Dokumentu osoan zehar esan dugun moduan, sistema beti wifi konexio berdinera, konfigurazio berdinarekin konektatuta egoteko garatu da. Ondorioz, wifi sarea aldatzen bada hainbat aldaketa egin beharko dira softwarean.

Alde batetik, birprogramatu egin beharko da Arduino Mega, sare berriari konektatu ahal izateko. Hau egiteko, Arduino Mega erabiltzen duen ArduinoMega.ino programaren `ssid` eta `pass` aldagaiak aldatu beharko dira. Behin hau eginda, Arduino birprogramatu eta Arduino pizterakoan eta sare berriari konektatzerakoan sareak Arduinori emandako IP-a gorde egin beharko dugu. IP helbidea Arduinok inprimatzen dituen mezuetatik atera ahal izango dugu.

Bestetik, web aplikazioak Arduinorekin komunikaziorako erabiltzen dituen bai `Insert.exe` eta `Extract.exe` programak birkonpilatu beharko ditugu, beraien `insert.h` eta `extract.h` fitxategietan, sare berriak Arduinoari emandako IP-a definituz. Behin hauek birkonpilatu ditugula eta `.exe` berriak ditugula, hauek ordezkatu beharko ditugu, gure web aplikazioaren `ExtExe` karpetan dauden `.exe` fitxategiengatik. Horrela gure proiektuak sare berrian funtzionatuko egingo du. IP helbidearen esleipena ez da modu oso eraginkorrean egiten, baina hala ere etorkizunean beste modu bat inplementatzea pentsatu da.

Behin sare berri batera konektatzeko prozesua ikusita eta beharrezko kableak konektatuta daudela, sistema piztera pasatuko gara.

Biltegia pizteko hurrengo eskema jarraitu beharko dugu.

1. Arduino UNO: Arduino UNO elikatzeko bi modu daude. Bat USB kablearen bitartez, ordenagailura konektatzea. Modu honen bitartez, Arduino IDE-k eskuragarri daukan *Monitor Serie* bitartez, Arduino UNO-ren prozesuak jarraitu ditzazkegu. Bestetik, kanpoko elikadura iturri batera konektatzen badugu gogoratu 7-12V artekoa izan behar dela.
2. Arduino Mega: Arduino UNO-rekin gertatzen den moduan, plaka elikatzeko bi aukera daude: USB bitartez elikatzea edo kanpoko elikadura iturri bat erabiltzea. Behin elikadurara konektatuta dagoela, ezarri zaion sarera konektatzean wifi shield-eko `LINK` led-a berdez jarriko da. Orduan, Arduinok aginduak jasotzeko prest egongo da. Bestetik, bi Arduinoak piztuta daudelarik, ikusiko dugu nola HC-05 moduluak parekatu egin direnean, 2 segunduro bi moduluen led-ak birritan kliskatuko egingo dutela, bi moduluak parekatuta daudela adieraziz. Nabarmendu behar da Bluetooth moduluaren led-aren kolorea gorria dela.

3. Azkenik, beso robotikoaren kanpoko elikagai iturria entxufe batera konektatu beharko dugu. Hau eginda, ikusiko dugu nola beso robotikoak hasierako posizio hartzen duela.

Honekin gure biltegiratze-sistema guztiz funtzionala egongo da eta web aplikaziotik aginduak jasotzeko prest egongo da.

A.2 Web aplikazioa prestatu

Web aplikazioa erabiltzeko, guk erabili dugun herraminta XAMPP instalazio paketea izan da. Hau, modu erraz batean instalatu egiten da Windows sistema-eragilean. Esan beharra dago, probak bakarrik egin direla Windows konputagailu batean, eta beraz, beste sistema-eragileetan funtzionatzen duen edo ez, ez da ziurtatzen.

Beraz, behin XAMPP instalatuta daukagula gure web aplikazioko karpeta gure zerbitzariaren `htdocs` karpetan uzten badugu, aplikazioa atzigarria egon beharko litzateke, erabiltzeko prest. `Htdocs` karpeta hau XAMPP instalatu dugun helbidean atzigarri egongo da. Web aplikazioak MariaDB datu-basea administraziorako eta Apache zerbitzari batekin exekutatzea posible izango litzateke.

Ohar moduan esatea, bi atalak, bai biltegia eta baita aplikazioa exekutatzen ari den ordenagailua ere, sare berdinean egon behar direla.

A.3 Biltegiratze sistema erabiltzen

Behin biltegia piztuta eta erabiltzeko prest dagoelarik eta XAMPP martxan daukagularik gure ordenagailuan, biltegia erabiltzeko prest egongo gara.

Hasteko gure hasierako orrialdera joan beharko gara, horretarako edozein web-arakatzailean, hurrengo estekara sartuz, hasierako orrialdea izango dugu.

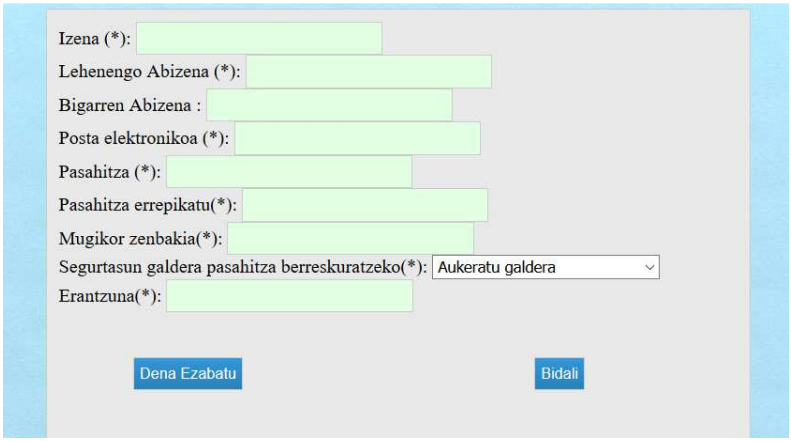
http://localhost/Erabiltzaile/Aplikazioa/Orrialdeak/Hasiera.php

Oso garrantzitsua da erabiltzen dugun web-arakatzaileak JavaScript blokeatuta ez izatea, hau gure aplikazioaren funtzionamendurako ezinbestekoa baita. Behin hona sartuta A.1 irudiko bista izango dugu. Honetan bi aukera ditugula ikusiko dugu, bat sisteman sartzeko eta bestea sisteman erregistratzeko.

A.1. Irudia: Hasierako orrialdea

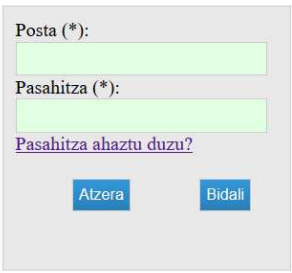
Sisteman erregistratu nahi bagara, bi aukera izango ditugu, A.2 irudian ikusi daiteken moduan, erabiltzaile moduan erregistratzea edota banatzaile enpresa moduan erregistratzea. Banatzaile moduan erregistratzeko administratzaileak emandako kode jakin bat behar dugu, iruzurrak ekiditzeko asmoz. Erabiltzaile moduan edo banatzailaile moduan erregistratzen bagara, A.3 irudian daukagun formulario batera ailegatuko gara, gure erabiltzaile datuak bete ahal izateko.

A.2. Irudia: Erregistroa egiteko aukerak



A.3. Irudia: Erregistroa egiteko formularioa

Behin erregistroa eginda, sisteman sartu ahal izango gara gure kredentzialak erabilita. Sisteman sartzeko A.1 irudian ikusi dugun `Login` botoia sakatu beharko dugu, agertzen zaigun orrialdean, gure erabiltzaile kredentzialak sartuz. Agertuko zaigun orrialdea A.4 irudian daukagu.

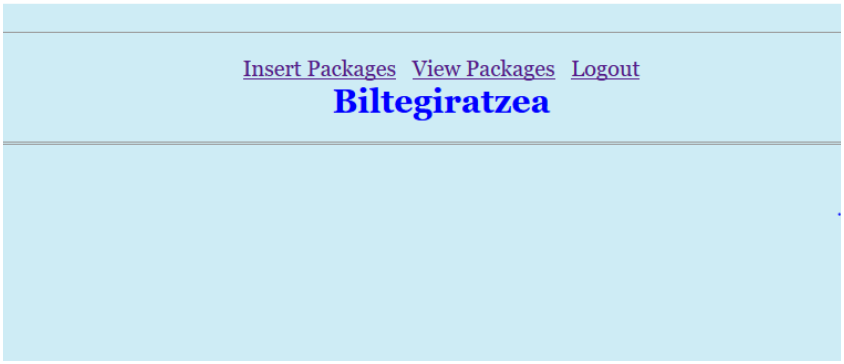


A.4. Irudia: Login egiteko orrialdea

Behin aplikazioan identifikatzen garenean, erabiltzaile motaren arabera, funtzio ezberdinak izango ditugu eskuragarri.

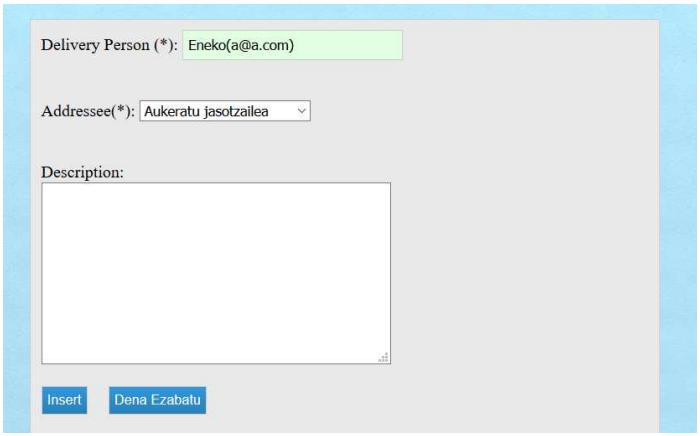
A.3.1 Erabiltzaile arrunta

Erabiltzaile arrunt moduan identifikatzen bagara, eskuragarri izango ditugun funtzioak bi izango dira, A.5 irudian ikusi daitekeen moduan.



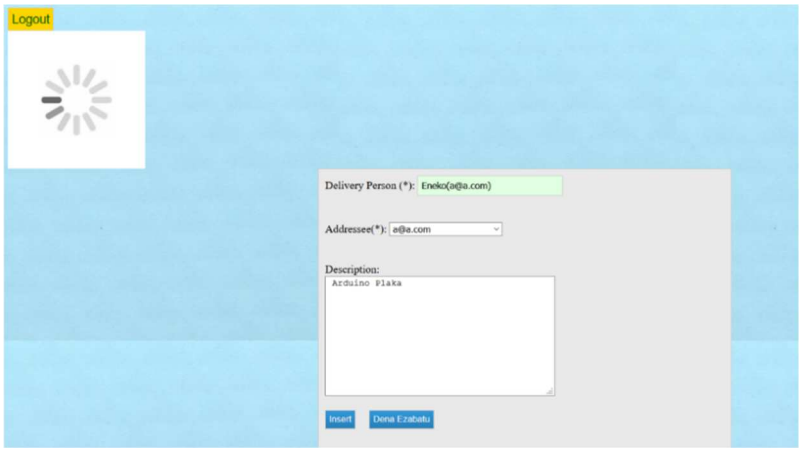
A.5. Irudia: Erabiltzaile arruntaren menua

Paketeak sartzeko eta ateratzeko aukerak izango ditugu. Alde batetik, pakete bat sartu nahi badugu, A.6 irudian daukagun formulario bete beharko dugu, paketeak fisikoki sartu baino lehen.



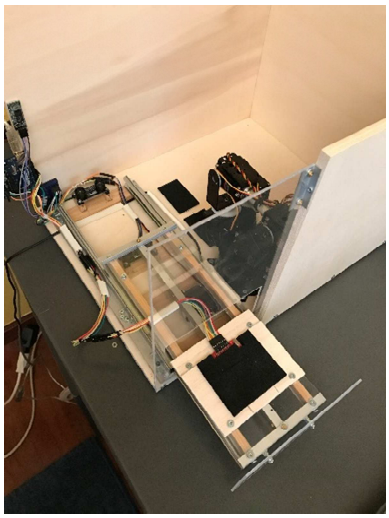
A.6. Irudia: Paketea sartzeko aukera

Behin paketea nori bideratuta dagoen eta deskripzioan paketearen barruan dagoena idatzi ondoren, `Insert` botoia sakatu behar dugu. Jarraian pakete berri bat sisteman sartzeko prozesua hasi egingo da, web aplikazioak biltegiratze-sistemari agindua bidaliz, honek pakete berria sartzeko prozesua hasiz. Prozesuak irauten duen bitartean erabiltzaileak A.7 irudia izango du pantailan.



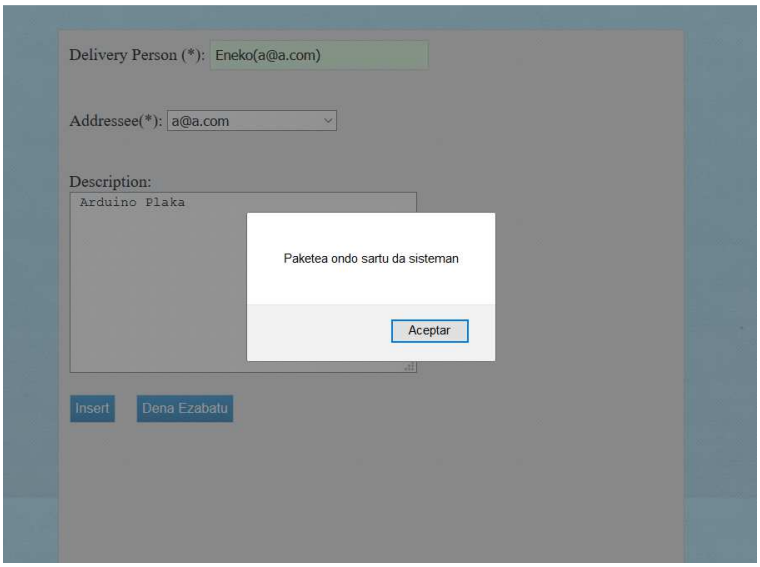
A.7. Irudia: Paketea sartzeko prozesua betetzen den bitartean

Prozesuan zehar biltegiaren plataforma mugikorra ireki egingo da, eta erabiltzaileak sartu nahi duen paketea barrura sartu beharko du, plataforman paketeentzako definituta dagoen zonaldean paketea utziz (A.8 irudia). Paketea plataforman utzi ondoren, barrura sartuko da eta biltegiak paketea gordeko du.



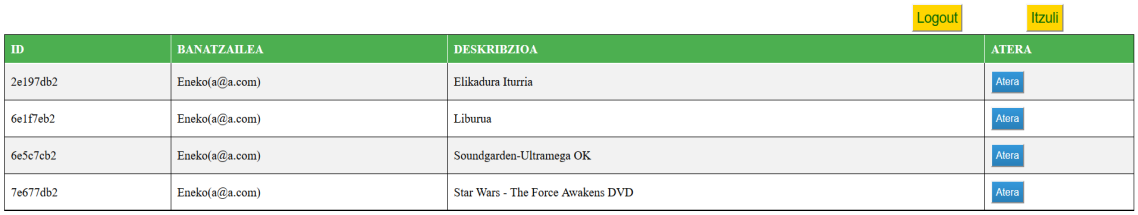
A.8. Irudia: Plataforma mugikorra prest

Behin prozesua amaituta edo errore batekin amaitu bada, erabiltzaileari mezu bat agertuko zaio, prozesuaren berri emateko. Prozesua modu egokian bukatzen bada, erabiltzaileak A.9 irudian daukagun mezua ikusiko du.



A.9. Irudia: Pakete berria sisteman sartu denean

Bestetik, beste aukera gure izenean gordeta dagoen pakete bat ateratzea izango da. Horretarako `ExtractPackage` esteka aukeratu egin beharko dugu eta jarraian gure izenean gordeta ditugun paketeak agertuko zaizkigu pantailan (A.10 irudia).

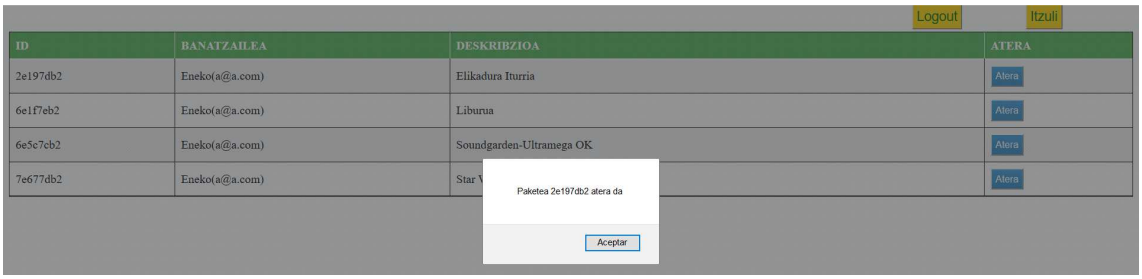


A.10. Irudia: Erabiltzaileak gordeta dituen paketeak

Gordeta ditugun paketetako, bat ateratzea aukeratzen dugunean, biltegiarekin komunikazio hasiko da eta prozesua abiatuko da. Prozesua exekutatzen den bitartean erabiltzaileak web aplikazioan A.11 irudia ikusiko du. Bitartean ezin da beste botoirik sakatu.

A.11. Irudia: Paketea ateratzen den bitartean

Behin prozesua aurrera doala eta errorerik ez badago, plataforma mugikorra ireki egingo da erabiltzaileak eskatutako paketea ateraz. Erabiltzaileak paketea hartzerakoan, plataforma itxi egingo da eta prozesua amaitu egingo da, A.12 irudian ikusi daitekeen mezuarekin.



A.12. Irudia: Paketea ondo atera bada

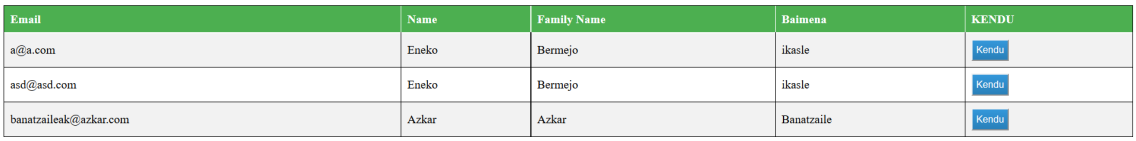
A.3.2 Banatzaile erabiltzaileak

Erabiltzaile mota hauek bakarrik sisteman paketeak sartzeko aukera izango dute. Horretarako, A.3.1 atalean ikusi dugun prozesua jarraitu beharko da.

A.3.3 Administratzaileak

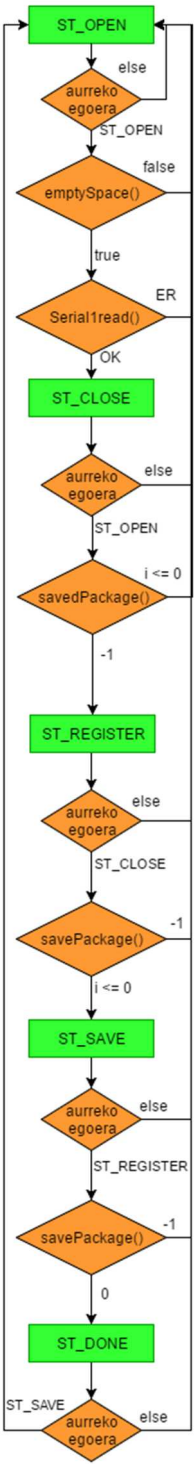
Administratzaileak bi funtzio posible izango ditu. Bat sisteman gordeta dauden paketeak ateratzea, eta bestetik, sistemako erabiltzaileen administrazioa eramatea. Lehenengo funtzionalitatea, A.3.1 atalan ikusi dugun prozesu berdina jarraituko du, kasu honetan, erabiltzaile bati dagokion paketeen artean aukeratu beharrean, sisteman gordeta dauden pakete guztien artean aukeratu ahal izanez.

Bestetik, sistematik erabiltzaileak ezabatzeko aukera izango du administratzaileak. Horretarako paketeen kasuan gertatzen den moduan, administratzaileak erabiltzaile zerrenda bat izango du, nahi duen erabiltzailea ezabatu dezakela. Hala ere, ezabatu nahi den erabiltzaileak ezingo du sisteman paketerik gordeta izan. Bestela paketerik gordeta daukan erabiltzaile bat ezabatzen saiatzen bagara errore bat agertuko zaigu. Erabiltzaile zerrendaren adibide bat A.13 Irudian daukagu.

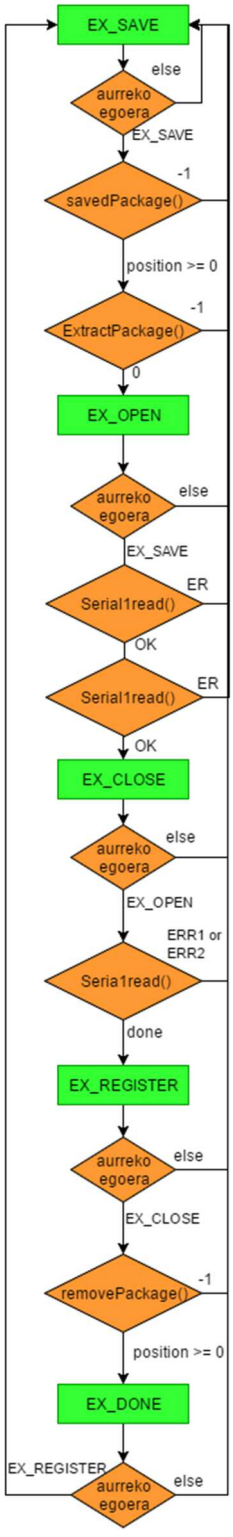


A.13. Irudia: Erabiltzaile ezabatzeko zerrenda

B. InsertPackage eta ExtractPackage automatak



B.0.1. Irudia: InsertPackage() automata



B.0.2. Irudia: ExtractPackage() automata

C. Bilera aktak

C.1 Konstituzio bilera

Data: 2017/02/06, 12:00 Lekua: Informatika Fakultatea

Hurbilduak:

- Eneko Bermejo
- Elena Lazkano

Helburuak:

- Proiektuaren helburuak eta nondik norakoak finkatu.

Bileran hitz egindakoa eta hartutako erabakiak:

- Biltegiratze-sistema garatuko da.
- Egindako irismenaren zuzenketak.
- *Sistema Txertatuen* ikasgaian erabilitako ALD5 beso robotikoa erabiliko da.

Egin beharrekoak:

- Arduino plaken inguruko informazioa bilatu.
- Biltegia eta web aplikazioa eraikitzeko diseinua egin.
- Proiektuaren plangintza burutu.
- Behar diren materialak zehaztu.

Elenak:

- Eskatutako materialak bilatu.

C.2 Bigarren bilera

Data: 2017/02/23, 12:00 Lekua: Informatika Fakultatea

Hurbilduak:

- Eneko Bermejo
- Elena Lazkano

Helburuak:

- Proiektuaren diseinua ikusi eta egindako dokumentuaren zuzenketa.
- Proiektua garatzen hasiko gara.

Bileran hitz egindakoa eta hartutako erabakiak:

- Proiektuaren garapenarekin hasiko da.
- Diseinuaren zuzenketak.

Egin beharrekoak:

- Proiektuaren garapenarekin hasi.
- Memoria idazten hasiko da.

C.3 Hirugarren bilera

Data: 2017/05/23, 09:15 Lekua: Informatika Fakultatea

Hurbilduak:

- Eneko Bermejo
- Elena Lazkano
- Txelo Ruiz

Helburuak:

- Egindako dokumentuaren zuzenketa.

Bileran hitz egindakoa eta hartutako erabakiak:

- Biltegiaren funtzionamenduaren demoa garatu.
- Dokumentuan zuzenketa sakonak egin.

Egin beharrekoak:

- Dokumentua zuzentzen hasiko gara.

Bibliografia

(Arduino, 2017) Arduino (2017). Arduino Reference: https://www.arduino.cc/en/Reference/HomePage. Atzitua [2017/05].

(PROMETEC, 2017) PROMETEC (2017). El módulo Bluetooth HC-05: http://www.prometec.net/bt-hc05. Atzitua [2017/05].

(PROMETEC, 2017) PROMETEC (2017). RFID: Identificación por RF: http://www.prometec.net/arduino-rfid/. Atzitua [2017/06].

(Currey, 2014) Currey, Martyn (2014). Connecting 2 Arduinos by Bluetooth using a HC-05 and a HC-06: Pair, Bind and Link. http://www.martyncurrey.com/connecting-2-arduinos-by-bluetooth-using-a-hc-05-and-a-hc-06-pair-bind-and-link/. Atzitua [2017/05].

(DFRobot, 2017) DFRobot (2017). URM35 V4.0 Ultrasonic Sensor. https://www.dfrobot.com/wiki/index.php/URM37_V4.0_Ultrasonic_Sensor_(SKU:SEN0001). Atzitua [2017/05].

(Arduino, 2017) Arduino (2017). EEPROM liburtegiaren kodea eta azalpena. http://playground.arduino.cc/Code/EepromUtil. Atzitua [2017/06]

(Balboa, 2012) Balboa, Miguel (2012). MFRC522: Arduino library for MFRC522 and other RFID-RC522 based modules. https://github.com/miguelbalboa/rfid. Atzitua [2017/04].

(Group, 2017) The PHP Group (2017). PHP: exec-Execute an external program. http://php.net/manual/en/function.exec.php. Atzitua [2017/05].

(RobotShop inc., 2017) Lynxmotion (2017). The ALD5 robotic arm. http://www.lynxmotion.com/c-130-al5d.aspx. Atzitua [2017/06].

(IntelligentDelivery,S.A., 2017) hapiick (2017). Online erosketak jasotzeko kutxatilak. http://www.hapiick.com/eu/interneteko-erosketak-jaso/. Atzitua [2017/06].

(ULMA Handling Systems, 2017) ULMA (2017). Bernardo Ecenarro (BESA) abre sus puertas a la automatizacion de la mano de ULMA Handling Systems. http://www.ulmahandling.com/es/casos-de-exito/automatizacion-logistica-besa. Atzitua [2017/07].

(Tam, 2014) Tam, Donna (2014). Conoce a Kiva: el hacendoso empleado robotico de Amazon. https://www.cnet.com/es/noticias/conoce-al-robot-kiva-el-hacendoso-empleado-de-amazon/. Atzitua [2017/06].

(Oracle Corporation , 2014) NetBeans (2014). Configuring NetBeans IDE 8.0 for C/C++/Fortran. https://netbeans.org/community/releases/80/cpp-setup-instructions.html. Atzitua[2017/05].

(EHU, 2017) EHU, I. F. (2017). Gradu amaierako proiektuari buruzko arautegia. http://www.ehu.eus/documents/340468/2334257/GAP_araudia.pdf. Atzitua [2017/02].