eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Informatika Ingeniaritzako Gradua

Software Ingeniaritza

Gradu Amaierako Lana

GidabotApp: GidaBot nabigazio sistemaren bidez erabiltzailea barruko inguruneetan bideratzeko aplikazio baten garapena eta integrazioa

Egilea

Iosu Sánchez Errazkin

2021

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Informatika Ingeniaritzako Gradua

Software Ingeniaritza

Gradu Amaierako Lana

GidabotApp: GidaBot nabigazio sistemaren bidez erabiltzailea barruko inguruneetan bideratzeko aplikazio baten garapena eta integrazioa

Egilea

Iosu Sánchez Errazkin

Zuzendariak

Ekaitz Jauregi eta Igor Rodriguez

Laburpena

EUS: Proiektu honen helburua, RSAIT ikerkuntza-taldeak garatutako GidaBot nabigazio-sisteman integratuko den Android aplikazio bat garatzea da. RSAITek garatutako nabigazio-sistema horren bitartez, bide-erakusle lanak egiten dituzten robot batzuek erabiltzaileak eraikin bateko helmuga zehatz batera eramaten dituzte.

Egungo egoeran, GidaBot sistemako edozein robotek gida-lan bat amaitzen duenean, beti azken helburuan gelditzen da. Ondorioz, beste erabiltzaile batek robota erabili nahi izanez gero, robotaren bila jo beharko du. Beraz, arazo honen soluzio posible bat da, gida lana bukatzean robotak puntu konkretu batzuetara mugitzea, erabiltzaileek robotak errazago aurkitu ditzaten. Bestalde, beste aukera bat, eta hain zuzen GrAL honetan aurkezten dena da, erabiltzaileak robotari bere kokalekua adieraztea, eta robota erabiltzailearengana hurbiltzea, azkenik helmugara gidatzeko.

Hori horrela, GrAL honetan diseinatu den — eta arazo horri aurre egingo dion — aplikazioari GidaBotApp izena eman zaio, eta memoria honetan, aplikazioa garatu ahal izateko burutu diren ataza guztiak azalduko dira; proiektuaren atazen planifikaziotik hasita, aplikazioaren inplementaziotik pasa eta proiektuaren ondorioetaraino.

CAS: El objetivo de este proyecto es desarrollar una aplicación Android que se integrará en el sistema de navegación GidaBot, desarrollado por el grupo de investigación RSAIT. A través de este sistema de navegación desarrollado por RSAIT, unos robots que hacen de guía trasladan a los usuarios a un destino concreto de un edificio.

En la situación actual, cuando cualquier robot del sistema GidaBot termina un servicio de guía, siempre se detiene en el objetivo final. En consecuencia, si otro usuario quiere utilizar el robot, deberá buscar el robot. Por lo tanto, una posible solución a este problema es que al finalizar el trabajo de guía los robots se desplacen a puntos concretos para que los

usuarios puedan encontrar los robots más fácilmente. Por otra parte, otra opción, y precisamente la que se presenta en este TFG, es que el usuario indique al robot su ubicación, y que el robot se acerque al usuario para conducirlo finalmente a su destino.

En este sentido, se ha denominado GidaBotApp a la aplicación diseñada en este TFG y que hará frente a este problema, y en esta memoria se explicarán todas las tareas realizadas para poder desarrollar la aplicación; desde la planificación de las tareas del proyecto, pasando por la implementación de la aplicación, hasta las conclusiones del trabajo realizado.

ENG: The aim of this project is to develop an Android application that will be integrated into the GidaBot navigation system, developed by the RSAIT research group. Through this navigation system developed by RSAIT, robots that act as guides take users to a specific destination in a building.

In the current state of affairs, when any robot in the GidaBot system completes a guidance service, it always stops at the end goal. Consequently, if another user wants to use the robot, they will have to look for the robot. Therefore, a possible solution to this problem is that, at the end of the guiding work, the robots move to specific points, so that users can find the robots more easily. On the other hand, another option, and precisely the one presented in this EDP, is for the user to indicate their location to the robot, and for the robot to approach the user to finally drive him to his destination.

In this sense, the application designed in this EDP and that will deal with this problem, has been called GidaBotApp, and this report will cover all the tasks carried out to be able to develop the application; from the planning of the project tasks, through the implementation of the application, to the conclusions of the work carried out.

Gaien aurkibidea

Irudien aurkibidea

Taulen aurkibidea

1. KAPITULUA

Sarrera

Ikasle edo pertsona berri bat Informatika Fakultatera iristen denean, gerta daiteke bere ikasgela non dagoen ez jakitea. Horri aurre egiteko, RSAIT taldeak GidaBot sistema garatu zuen, eta Tartalo, Kbot, Galtxagorri eta Marisorgin robotei Fakultateko gida izatearen ardura esleitu zien.

Robotak jakin badaki eskatzen diozun ikasgela edo bulegora gidatzen, baina hona hemen arazoa: ikasleak ez badaki non dagoen irakasle baten bulegoa fakultatean berria delako, jakingo al du robota non dagoen?

Arazo honi konponbide bat eman nahian, Gradu Amaierako Lan honetan Android Aplikazio bat garatu da, non erabiltzaileak robotari bere kokalekua adieraziko dion. Modu honetan, robota beregana hurbilduko da, eta erabiltzailea aplikazioan zehaztutako helmugara gidatuko du.

Oro har, aplikazio bat ondo garatzeak hainbat ataza hartzen ditu barne. Beraz, lehenik eta behin proiektu edo aplikazioaren irismena zehaztuko da, burutu beharko diren ataza guztiak planifikatzeko. Ondoren, bezero eskakizunen bilketa egingo da, aplikazioa zuzen diseinatu eta erabiltzailearen betekizunak betetzen dituela bermatzeko. Aplikazioaren inplementazioarekin hasi baino lehen, garapen-ingurunea berria izanik, formakuntza-prozesu bat jasoko da. Azkenik, aplikazioaren garapena edo inplementazioa burutuko da, eta proben bidez bermatu betekizun guztiak betetzen dituela.

Hori horrela, memoria honen helburua iragan diren fase guzti horiek dokumentatzea izango da.

2. KAPITULUA

Proiektuaren irismena

Plangintza, kudeaketa eta egindako lanaren jarraipen eta kontrola funtsezkoak dira proiektu baten arrakastan. Hori dela eta, atal honetan Gradu Amaierako Lan honen kudeaketa-plana aurkeztuko da.

Lehenik eta behin, proiektuaren irismena definituko da, proiektuaren helburuak zehaztu eta egin beharreko lana mugarrituz. Behin proiektuaren helburuak zehaztuta, burutu beharreko eginkizunen planifikazioa osatuko da. Horretarako, lehenik, zereginak lanpaketeetan banatuko dira, eta ondoren pakete horiek ataza txikiagoetan, hauei denbora-estimazio bat eman ahal izateko. Horrekin guztiarekin, proiektuaren bilakaeraren aurreikuspena aurkeztuko da Gantt diagrama baten bitartez.

Proiektuaren helburuak eta planifikazioa finkatu ostean, proiektuaren arriskuen kudeaketa egingo da. Arriskuen kudeaketaren helburua proiektuaren garapeneko arrisku nagusiak identifikatzea eta aztertzea izango da, ondoren arrisku horiek minimizatzeko estrategia bat finkatzeko.

Azkenik, Gradu Amaierako Lanaren garapenerako erabilitako informazio-sistema deskribatuko da, komunikazio-sistemarekin batera.

2.1 Produktuaren irismena

Gradu Amaierako Lan honen helburu nagusia GidaBot sisteman integratuko den Android aplikazio bat garatzea da.

Proiektuaren bizi-zikloan zehar garatuko den aplikazioari esker, Donostiako Informatika Fakultatean dagoen erabiltzaile batek bere kokapenetik gertuen dagoen robota beregana hurbiltzeko eskaera burutu ahalko dio GidaBot sistemari. Gainera, aplikazioaren bidez erabiltzaileak robotari esan ahalko dio zein tokira joan nahi duen, eta robotak erabiltzailea helmugaraino gidatuko du. Hari beretik, proiektuaren irismenetik kanpo geratzen da robotek GPS bidez aurkitzea erabiltzaileak, GPS seinaleak ez baitu eraikinen barruko kokapenak burutzeko balio.[69]

Hortaz, ondorengoak izango dira aplikazioaren helburu zehatzak:

H1 Android Studio garapen-ingurunean Android aplikazio bat garatzea Java lengoaia erabiliaz.

H2 Aplikazioa ROS *framework*-a eta GidaBot sistemaren arteko komunikazioa burutzeko gai izatea.

H3 Edozein erabiltzailek erabiltzeko moduko interfaze grafikoa diseinatu eta garatzea.

H4 Fakultateko kokaleku nagusiak gordeko dituen datu-basea sortu eta aplikazioan integratzea.

H5 Aplikazioa Informatika Fakultateko solairu guztien mapa erakusteko gai izatea, erabiltzaileak mapako bulego eta kokaleku nagusiak ikusteko.

H6 Aplikazioaren bitartez erabiltzailea uneko solairuko edo beste solairu bateko helburu batera iristeko gai izatea.

2.1.1 Irismen minimoa

Gradu Amaierako Lan honek izaera ikertzailea duenez, oso zaila da proiektuaren hasieratze-fasetik definitzea proiektuaren irismen osoa zein izango den, nekez kalkula daitekeelako ataza bakoitzari dedikatuko zaion denborara zehatza. Hori dela eta, proiektuaren amaieratze-fasean proiektuaren arrakasta ebaluatu ahal izateko, proiektuaren irismen minimoa definitu beharko da, proiektuaren arrakasta balioztatuko duena.

Hortaz, honakoak izango dira aplikazioaren helburu minimoak:

1. Android Studio garapen-ingurunean Android aplikazioa garatzea Java lengoaiaz, aplikazioa GidaBot sistemarekin komunikatzeko gai izatea, eta aplikazioaren interfazea edozein erabiltzailek erabiltzeko modukoa izatea (H1, H2 eta H3).

2. Aplikazioaren bitartez erabiltzailea solairu bakar bateko helburu batera iristeko gai izatea (H6, baina solairu bakar batean soilik).

Hori horrela, proiektuaren amaiera-fasean aplikazioak helburu minimo horiek bete baditu, proiektuak arrakasta izan duela esan ahalko dugu. Gainera, behin helburu minimo horiek beteta, eta nahiko arin egin badira, azken urrats modura fakultateko solairu guztiak gehituko lirateke aplikazioan (H6).

2.2 Bizi-zikloa

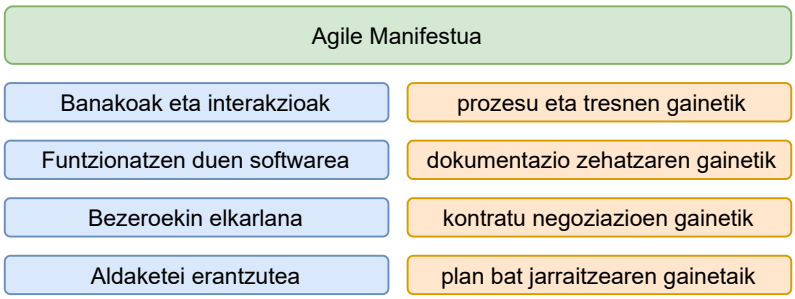
Aipatu bezala, proiektuaren irismen osoa ez da hasieratik definituta egongo, eta beraz, proiektua modu iteratiboan garatuko da, Software Garapen Arinaren edo *Agile* printzipioak jarraituz.

2.2.1 *Agile*

Software Garapen Praktika Arinak (*Agile Software Principles*), talde-proiektuen kudeaketa zein softwarearen garapenaren ikuspegi iteratibo bat dira, zeintzuk bezeroei emangarri txiki baina kudeagarriak entregatzeko jardunbide egokiak biltzen dituzten. Beraz, lantalde batek egindako lan guztia entregagarri handi bakar batean entregatu ordez, talde arin batek lana gehikuntza txiki baina kudeagarriagoetan entregatzen du. Horretarako, eskakizunak, plangintzak eta emaitzak etengabe balioztatzen dira, taldeek aldaketei azkar erantzuteko mekanismo naturalak izan ditzaten. Garapen-teknika bereziak definitu baino, feedback ziklo estuak eta etengabeko hobekuntzarako konpromisoa erakusten duen metodologia-talde bat da *Agile*.[5]

2001. urtean argitaratu zen Agile Manifestuan[31] aipatzen den legez, 2.1 irudiko eskubiko zutabeko balioek garrantzia duten arren, ezkerreko zutabeko balioak gehiago balioetsi beharko lirateke software-garapeneko proiektuetan.

Printzipio arinen jatorrizko manifestuan ez zen zehazten bi asteko iterazioak egin behar zirenik, edota zein zen lantalde baten tamaina egokiena. Horren ordez, *Agile*k pertsonak lehenesten dituen balioak ezartzen ditu. Hori dela eta, gaur egun oso erabiliak diren metodologia asko oinarritu dira printzipio arinetan: *Scrum*[70], *Kanban*[6] edota *XP* (*Extreme Programming*)[3], kasu.



2.1 Irudia: Agile manifestua

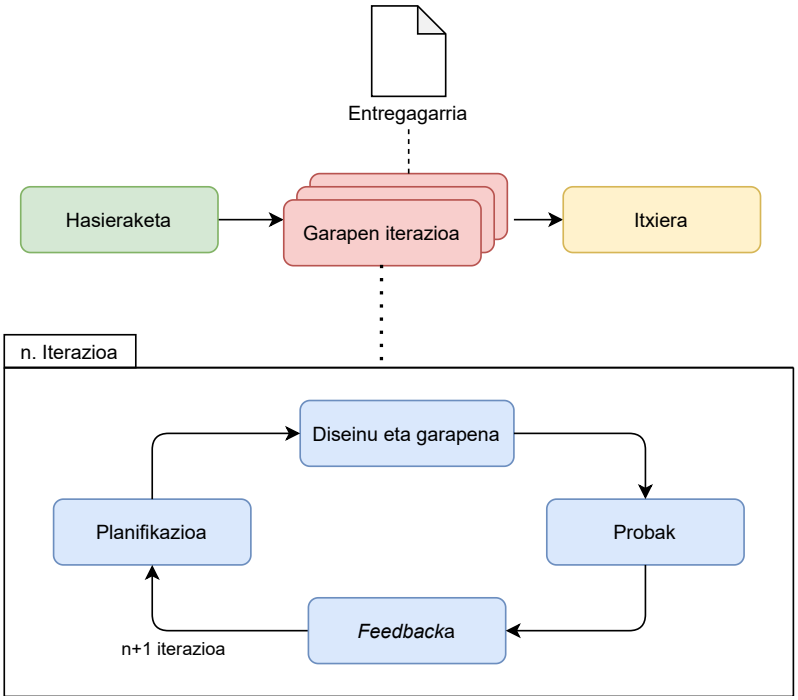
2.2.2 Oinarri arinak proiektuan

Azaldu bezala, oinarri arinen funtsetako bat proiektu osoan burutu beharreko lana entregagarri txikitan banatzea da. Beraz, burutu beharreko aplikazioan printzipio hauek aplikatuz, helburuak modu iteratiboan garatuko dira, iterazio bakoitzerako entregatze-data bat finkatuz. Hortaz, iterazio bakoitzaren amaieran bezeroarekin bilera bat burutuko da, non iterazioan zehar garatutako funtzionalitateak erakutsiko diren. Halaber, iterazio-amaierako bilera horietan bezeroak aplikazioa hobetzeko iradokizunak egingo ditu, eta hurrengo iterazioan funtzionalitate berriak gehitu baino lehen, aplikazioa hobetu egingo da iradokizun edo eskakizun berri horien arabera. Honekin guztiarekin, 2.2 irudian proiektuaren bizi-zikloko prozesuen fluxua azaltzen da.

Horiek horrela, aplikazioaren lehen iterazioaren helburua irismen minimoa betetzea izango da, eta behin aplikazioak bezeroaren behar minimo horiek bete dituela, funtzionalitate gehigarriak inplementatuko dira, esaterako, datu-basearen inplementazioa, Fakultateko maparen bisualizazioa edota hainbat solairuko bidaiak.

2.3 Plangintza

Burutu beharreko lan guztia ondo planifikatzeak berebiziko garrantzia izango du proiektua behar bezala garatzeko. Atal honetan Lanaren Deskonposaketa Egitura (LDE) diagramaren bitartez lana paketeetan zatituko da. Ondoren, pakete horiek ataza txikiagoetan banatu, bakoitzari denbora estimazio bat emanez, eta Gantt diagrama egingo da proiektuaren bilakaera aurreikusteko. Azkenik, proiektuaren emangarriak eta garapen mugarriak azalduko dira.



2.2 Irudia: Proiektuaren bizi-zikloa. Prozesuen fluxua

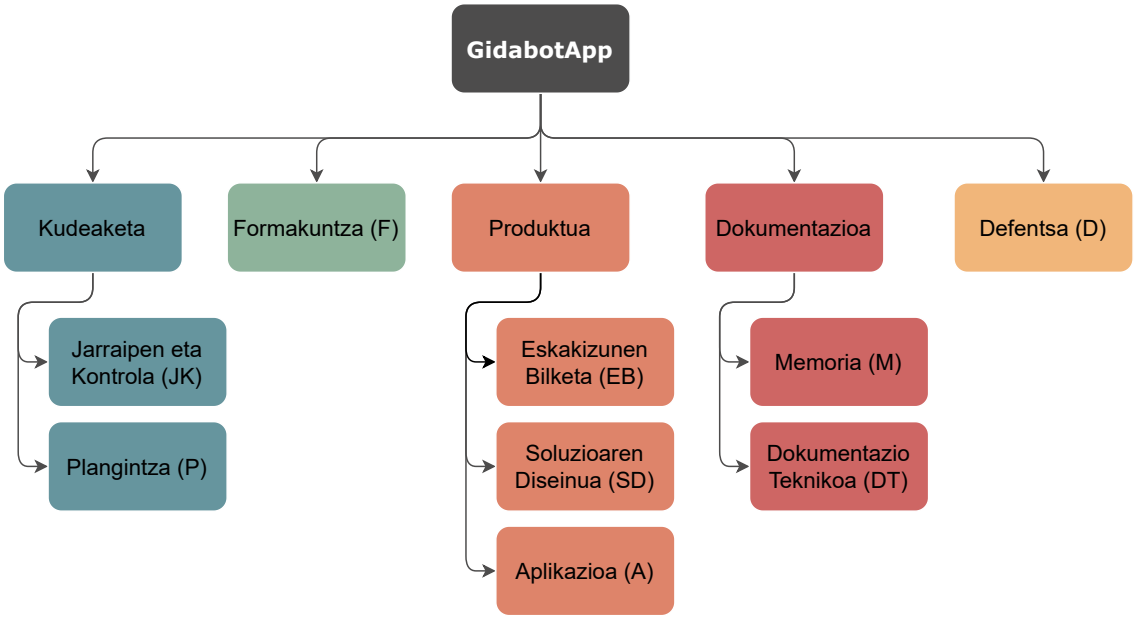
2.3.1 Lanaren Deskonposaketa Egitura (LDE)

Lanaren Deskonposaketa-diagraman proiektuan burutu beharreko lan guztia pakete desberdinetan banatu da, hainbat multzotan sailkatuta: Kudeaketa, Formakuntza, Produktua, Dokumentazioa eta Defentsa, hain zuzen.

Lan-paketeen deskribapena

LDE diagraman azaldu den modura, hauek dira proiektuko eginkizunen lan-paketeak:

- **Jarraipen eta Kontrola (JK)** paketean, proiektuaren garapen egokia bermatuko duten atazak bilduko dira, hots, plangintzarekiko desbiderapena edota zuzendariekin egindako bilerak.
- **Plangintza (P)** lan-paketeak, hasierako plangintza egiteko atazak biltzen ditu, baita plangintza eguneratuta mantentzeko beharko direnak ere.
- **Formakuntza (F)** lan-paketeak, proiektuaren garapenerako beharrezkoak diren ezagutzak eskuratzeko atazak biltzen ditu.



2.3 Irudia: LDE diagrama

- **Eskakizunen Bilketa (EB)** lan-paketean, soluzioa diseinatu baino lehen burutu beharreko atazak izango ditugu, hau da, bezeroaren eskakizunak betetzen direla bermatzeko atazak.

- **Soluzioaren Diseinua (SD)** lan-paketeak, aplikazioa inplementatu aurretik egikaritu beharreko lanak bilduko ditu, esaterako, aplikazioaren arkitekturaren definizioa, edota aplikazioaren klase-diagrama.

- **Aplikazioa (A)** paketeak, Android aplikazioa garatzeko burutu beharrekoak barne hartuko ditu; ROS eta Android gailuen komunikazioa eta Aplikazioaren interfaze grafikoaren inplementazioa, besteak beste.

- **Memoria (M)** lan-paketeak, memoria hau garatzeko beharrezko atazak barneratzen ditu.

- **Dokumentazio Teknikoa (DT)** paketeak, garatu(ko) den aplikazioaren funtzionamendua azalduko duten atazak biltzea du helburu. Besteen artean, *README* motako fitxategiak edota softwarearen kodea ulergarriagoa izateko baliagarriak izan daitezkeen dokumentu edota iruzkinak.

- **Defentsa(D)** lan-paketeak, garatutako proiektua epaimahai baten aurrean aurkezteko burutu beharreko eginkizunak jasotzen ditu.

2.3.2 Atazen deskonposaketa

Egin beharreko lana paketeetan banatu ondoren, pakete horiek ataza txikiagoetan banatu dira, eta ataza bakoitzari denbora-estimazio bat esleitu zaio, ataza bakoitza noiz burutuko den planifikatu ahal izateko (ikus 2.1 taula).

2.3.3 Gantt diagrama

Ataza bakoitza noiz burutuko den planifikatzeko, Gantt diagrama bat osatu da. Bertan azaltzen da aste bakoitzean zein ataza dagoen planifikatuta, eta ataza bakoitzak zein menpekotasun dituen. Diagramaren zutabe bakoitza proiektuaren garapeneko aste bat izango da, kontuan izanik proiektuaren hasiera-data 2021ko otsailaren 1 dela. Diagrama honi esker, atazen programazio guztiaren ikuspegi orokor bat izango dugu, eta aste bakoitzerako zein ataza amaitu beharko den (ikus 2.4 irudia).

2.3.4 Garapen mugarriak eta emangarriak

Entregagarriei dagokienez, bi emangarri mota egongo dira proiektuarekin lotuta: batetik, kudeaketarekin lotutakoak, eta proiektuaren garapenarekin daudenak bestetik.

Kudeaketarekin lotutako emangarriak

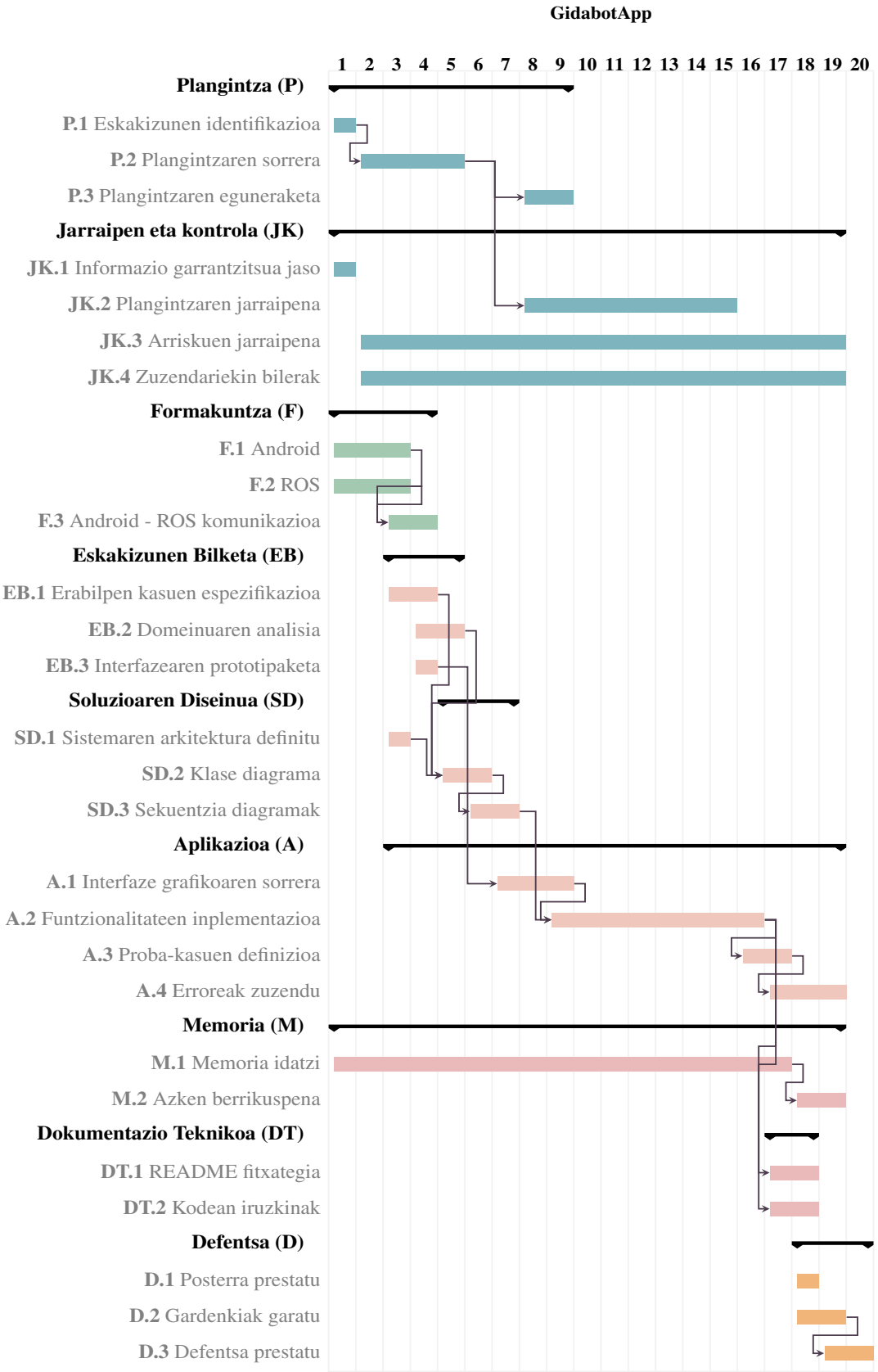
Proiektuaren kudeaketarekin lotuta emangarri bakarra dago: hots, proiektua hasi baino lehen Gradu Amaierako Lan honen zuzendariek egile honi aurkeztu zioten dokumentua, non lanaren eskakizun eta helburuak zehazten ziren. Dokumentu hau bi parteek (hots, zuzendariek eta egileak berak) berrikusi eta adostu zuten, behin proiektuaren oinarriak finkatuta proiektuari ekin ahal izateko.

Proiektuaren garapenarekin lotutako emangarriak

Maila honetan lau emangarri izango ditugu. Lehena, proiektuaren inplementazioarekin lotutako kodearen azken bertsioa izango da, bezeroari, hau da, RSAIT taldeari entregatuko zaiona. Beste hiru emangarriak Gradu Amaierako Lanean ebaluatuko direnak izango dira: garapenaren inguruko zehaztasunak gordeko dituen memoria-dokumentua, defentsarako erabiliko den aurkezpena, eta proiektuaren posterra azkenik.

Ataza	Deskribapena	Orduak
Plangintza (P)		**17**
P.1	Proiektuaren eskakizunen identifikazioa, hasierako erabakiak hartzea, informazioaren analisia eta zalantzen ebazpena.	4
P.2	Hasierako plangintzaren sorrera.	10
P.3	Plangintzaren eguneraketa, beharrezkoa izango balitz.	3
Jarraipen eta Kontrola (JK)		**11**
JK.1	Proiektuaren garapenari buruzko informazio garrantzitsua jaso.	3
JK.2	Plangintzarekin, jarraipeneko informazioaren konparaketa burutu. Desbideratze esanguratsuak eta arrisku berriak identifikatu.	-
JK.3	Arriskuen kudeaketan aipatutako neurriak betetzen direla bermatu.	-
JK.4	Proiektuaren garapen zuzena bermatzeko zuzendariekin jarraipen bilerak burutu.	8
Formakuntza (F)		**75**
F.1	Android aplikazio bat garatzeko oinarrizko prestakuntza jaso.	35
F.2	ROS sistemarekin lan egiteko oinarrizko prestakuntza jaso.	20
F.3	Android eta ROS sistemen arteko komunikaziorako informazioa bildu.	20
Eskakizunen Bilketa (EB)		**12**
EB.1	Erabilpen kasuen espezifikazioa.	5
EB.2	Domeinuaren analisia.	4
EB.3	Interfazearen prototipaketa.	3
Soluzioaren Diseinua (SD)		**18**
SD.1	Sistemaren arkitektura definitu.	3
SD.2	Klase diagrama osatu.	5
SD.3	Sekuentzia diagramak osatu.	10
Aplikazioa (A)		**100**
A.1	Aplikazioaren interfaze grafikoa diseinatu eta inplementatu.	15
A.2	Funtzionalitateak inplementatu modu iteratiboan.	70
A.3	Aplikazioaren funtzionamendu egokirako proba-kasuak definitu eta egikaritu.	5
A.4	Proba-kasuetan aurkitutako akatsak zuzendu.	10
Memoria (M)		**75**
M.1	Memoriaren idazketa eta berrikuspen periodikoa.	70
M.2	Azken berrikuspena eta maketazioa.	5
Dokumentazio Teknikoa (DT)		**15**
DT.1	README fitxategia sortu, sistemaren konfigurazio eta funtzionamenduaren azalpenekin.	10
DT.2	Kodearen dokumentazioa burutzea, ulergarriagoa egingo duten iruzkinak gehituz.	5
Defentsa (D)		**18**
D.1	Proiektuaren posterraren prestaketa.	3
D.2	Defentsarako gardenkiak garatu.	10
D.3	Proiektuaren defentsa prestatu.	5
GUZTIRA		**341**

2.1 Taula: Atazen deskonposaketa eta estimazioa



2.4 Irudia: Gantt diagrama

Proiektuaren emate-datak argi izatea eta betetzeak berebiziko garrantzia izango du arrakastan; horregatik, 2.2 taulan, proiektuaren mugarriak adierazi dira, eta emangarriak dituztenen kasuan data horretako emangarriaren izena zehaztu da.

Mugarria	**Data**	**Emangarria(k)**
Proiektuaren hasiera	otsailak 1	-
Lanaren matrikulazioa	ekainak 11	-
Defentsa eskaera	ekainak 11	-
Lana ADDI plataformara igo	ekainak 20	Memoria Posterra Kodea
Lanaren defentsa	ekainak 28 - uztailak 9	Aurkezpena
Behin behineko kalifikazioak	uztailak 9	-
Behin betiko kalifikazioak	uztailak 21	-
Proiektuaren amaiera	uztailak 21	-

2.2 Taula: Proiektuaren mugarri eta emangarriak

2.4 Arriskuen kudeaketa-plana

Proiektuaren arriskuen kudeaketaren bidez proiektuarekin erlazionatutako mehatxuei buruzko ziurgabetasuna maneiatuko da. Horretarako, arriskuen identifikazioa, analisia eta ebaluazioa burutuko dira, identifikatutako arrisku horientzat tratamendu-estrategiak proposatzeko.

2.4.1 Arriskuen identifikazioa

A1 **Erabilitako softwarearen bertsio-aldaketek sortutako arazoak.** Teknologien kapituluan aipatzen den edozein softwareren bertsio aldaketak proiektuaren garapenean eragin zuzena izango du. Esaterako, sistema eragilearen eguneraketa batek ROS sistemarengan eragin dezake, aplikazioaren eta robotaren arteko komunikazioan funtsezkoa den atalen bat kaltetuaz, modu horretan, sortutako sistemaren funtzionalitateak baliogabetuz.

A2 **Aplikazioaren garapenaren desbiderapena.** Aurretik aipatu den legez, egile honek garatutako lehen Android aplikazioa izango da proiektu hau. Horrez gain, apli-

kazioak ROS sistemarekin elkarlanean aritu beharko du, egilearentzat ere berria dena. Tresna berriekin lan egiten denean denbora-estimazioak egitea oso konplexua izaten da, eta horrenbestez, ziurrenik proiektuko garapenean zehar plangintzarekiko desbiderapenak egongo dira. Hainbat faktorek eragin dezakete estimatutako orduen desbideratzea; hots, Android-garapenean zehar sor daitezkeen erroreek, edota Android eta ROS arteko sistemen komunikaziorako arkitekturaren diseinuan sor daitezkeen arazoek.

A3 **Helburuak ez lortzea.** Ataza bakoitzari dedikatu beharreko denborarekiko desbiderapena oso esanguratsua izango balitz, baliteke Produktuaren irismenean zehaztutako helburu guztiak ez betetzea.

A4 **Datuen galera.** Proiektu baten garapenean zehar, ohikoa izaten da ezustean fitxategiren bat ezabatze edo galtzea. ROS konfigurazio-fitxategi baten ustekabeko aldaketa batek edota proiektuaren memoriaren ustekabeko ezabatzeak, proiektuan eragin handia izango luke. Hortaz, funtsezkoa izango da halako ezbeharrak aurreikustea oinarrizko fitxategien segurtasun-kopiak eginez.

A5 **Zuzendari edota ikaslearen erabilgarritasun-egoeraren aldaketa.** Proiektuaren garapenean zehar, gerta daiteke zuzendarien edo egile honen erabilgarritasun-egoera aldatzea. Inplikatuen erabilgarritasunaren aldaera hori hainbat faktorek sor dezakete, esaterako, irakasgairen batek eskatu lezakeen lan-gainkarga batek, edota arazo pertsonal batek.

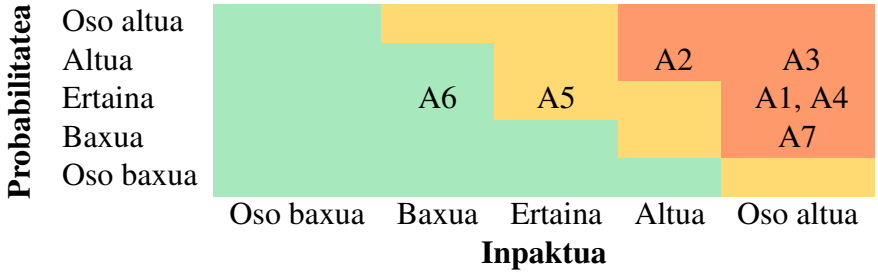
A6 **COVID-19ak eragindako itxialdia.** Nahiz eta gaur egun COVID-19aren pandemiaren egoerak hobera egin duen, oraindik ere gerta liteke egile hau edo zuzendarietako bat denbora batez itxialdian egotera behartuta egotea.

A7 **Ordenagailu Eramangarria hondatzea**. Eramangarria hondatzeak denborak proiektuko estimazioen desbiderapen handia ekar lezake. Izan ere, Gidabot sistema martxan jartzeko burutu beharreko instalazio-pausu guztiak berriz ere egikaritu beharko lirateke.

2.4.2 Arriskuen analisi kualitatiboa

Arriskuen kudeaketa-planaren helburua da arrisku negatiboen probabilitatea edo/eta inpaktua murriztea, proiektuaren arrakasta-aukerak optimizatzearren. Hori dela eta, Probabilitate-

Inpaktu matrizean arrisku bakoitza ebaluatu da, ondoren arrisku esanguratsuenei tratamendu bat proposatzeko.



2.3 Taula: Arriskuen Probabilitate-Inpaktu matrizea

2.4.3 Arriskuen tratamendua

Aztertu berri ditugun arriskuei erantzun bat emateko, PMBOK-en aipatzen diren mehatxuei erantzuteko estrategiak jarraituko dira: **ekidin, eskualdatu, arindu** eta **onartu**. [36, *11.5.2 Planificar la Respuesta a los Riesgos: Herramientas y Técnicas*]

Hots, matrizeko alde gorrian ez dauden arriskuak **onartu** egingo dira; beraz, **A5** (Erabilgarritasun-egoeraren aldaketa) eta **A6** (COVID-19ak eragindako itxialdia) mehatxuak onartu egingo dira. Izan ere, egile honen eta zuzendarien arteko komunikazioaren gehiengoa telematikoa izaten ari denez, gutako batek positibo emango balu edo akaso positibo batekin kontaktu zuzena izan, proiektuan lan egiteko baldintzak ez lirateke aldatuko, non eta birusak osasun arazo larririk eragiten ez dituen.

Matrizearen alde gorrian dauden arriskuak, ordea, banaka aztertuko ditugu:

A1 arriskuaren, hau da, Bertsio-aldaketek sortutako arazoen inpaktua arintzeko, sistemaren konfigurazioei segurtasun-kopia periodikoak egingo zaizkie, arriskuaren inpaktua **arindu** edo murriztuz. Aurrerago Konfigurazioen kudeaketaren atalean azalduko denez, Timeshift erabiliko da eramangarriko konfigurazio-fitxategien segurtasun-kopiak egiteko. Babeskopia horiei esker, edozein eguneraketak garapenean arazorik ekarriko balu, atzera egiteko aukera egongo litzateke.

Garapenaren denbora-desbiderapenaren (**A2** arriskua) probabilitatea **murrizteko**, planifikazioan aplikazioaren inplementazioari dagozkion ataza bakoitzari estimatutako denbora baino zertxobait gehiago esleitu zaio. Modu horretan, atazaren batean ustekabeko arazoren bat izanez gero, proiektuan izango duen eragina minimizatuko da, eta denbora egongo da desbiderapen horiei aurre egiteko.

Helburuak ez lortzearen arriskua (**A3** arriskua) **murrizteko**, inplementazioaren atalean, lehenik eta behin helburu minimoak beteko dira, eta behin oinarrizko funtzionalitateak finkatuta, funtzionalitate osagarriak inplementatuko dira.

Proiektuaren garapenerako baliatutako informazio guztia leku seguru eta ezagun batean gordetzea oso garrantzitsua izango da. Memoriaren garapenerako erabilitako irudi, taula eta diagrama guztien bertsio editagarriak, esaterako, kontrolpean izatea beharrezkoa izango da. Izan ere, diagrama bat aldatu behar izango bagenu bertsio editagarriak izan gabe, diagrama guzti horiek berregin beharko lirateke, horrek dakarren denbora-desbiderapenarekin. Zer esanik ez, Android Studioko proiektua galdu, edota proiektuko memoriaren fitxategi nagusia ezabatuko balitz.

Hortaz, **A4** arriskuaren inpaktua **minimizatuko** da, funtsezko fitxategien segurtasun-kopia periodikoak eginez laino-zerbitzuetan (ikus 2.6 atala informazio-sistemaren deskribapenerako). Besteak beste, Github erabiliko da kodeari dagozkien fitxategiak gordetzeko, Mega.nz eramangarriko fitxategientzat, eta Overleaf memoriarekin lotutakoentzat.

Azkenik, **A7** arriskua **onartu** egingo da. Arriskua arintzeko aukera bat izan liteke mahaigaineko ordenagailuan babeskopia modura GidaBot sistemaren instalazio osoa egitea, baina horrek, hondamendia gertatu aurretik ordu horiek bigarren instalazio horretan inbertitzea ekarriko luke. Hau da, segurtasun instalazioa eramangarria hondatu baino lehen egiteak edo eramangarria hondatu ondoren egiteak denbora berdina eskatuko luke, eta beraz, eramangarria hondatu arte instalazioa burutzen ez badugu denbora hori aurreztuko dugu.

2.5 Kalitatearen kudeaketa-plana

Atal honetan, garapen prozesuan eta honen bidez lortutako produktuan egon daitezkeen akatsak edo desbideratzeak saihestea helburu duten ekintzak definituko dira. Horretarako, lehenik produktuaren kalitatea neurtzeko irizpideak aurkeztuko dira, eta irizpide horien arabera produktuaren kalitate desberdinak zeintzuk izan daitezkeen azalduko dira.

2.5.1 Kalitatearen planifikazioa

Kalitatearen planifikazio on batek gure proiektuaren kalitatea bermatuko du, ezarritako irizpideen arabera. Izan ere, garatuko den produktuaren kalitatearen kontrolerako ez badugu neurririk ezartzen, kalitate-ezaren kostu eta inpaktua oso handia izan daiteke.

Gradu Amaierako Lan honen produktua Android aplikazioa bera izango denez, proiektuaren kalitatea neurtzeko, aplikazioaren kalitatea neurtuko dugu. Hori dela eta, produktuaren kalitatea balioztatzeko erabiliko diren irizpideak proiektuaren inplementazioa hasi aurretik zehaztea komeni da.

Aplikazioaren kalitatea neurtzeko irizpideak ezartzerakoan, aukera zuhurrena industriako estandarretara jotzea da. Beraz, kasu honetan Android aplikazio bat garatzea dugunez helburu, industriako estandarrak zehazten diren orrialdera joko dugu: Android Developers, hain zuzen ere. Bertan, bai aplikazio baten kalitatea neurtzeko irizpideak, baita irizpide horiek egiaztatzeko proba kasuak ere zehazten dira. [48]

Hala ere, proiektuaren helburua ez denez, inondik ere, aplikazio guztiz profesional bat garatzea, proiektuaren irismenetik kanpo geratzen da industriako estandar guztiak jarraitzen dituen aplikazio bat sortzea. Horregatik, kalitate-irizpide horietatik garrantzitsuenak hautatzeaz gain, hainbat irizpide propio zehaztu dira, garatutako aplikazioaren funtzionalitate zehatzen kalitatea ebaluatzeko (ikus 2.4 taula). Gainera, irizpide estandarizatu horiek soilik aplikazioaren kalitatea nahikoa denean kontuan hartuko dira, hau da, ez dira beharrezkoak izango aplikazioak kalitate minimoa izateko.

2.5.2 Kalitatearen kontrola

Jarraipen eta kontrola kapituluan proiektuaren kalitatea ebaluatzeko egin diren probak aurkeztuko dira. Proba horiei esker, ebaluatuko da ezarritako irizpideetatik zeintzuk betetzen diren, eta zeintzuk ez. Hortaz, behin aplikazioa probatuta, honako irizpideen arabera aplikazioaren kalitatea zein den ondorioztatu ahalko da:

- Aplikazioak **kalitate minimoa** duela esango dugu, KO-1 irizpidea betetzen badu, hori baita proiektuaren oinarrizko helburua.
- Aplikazioak KO-1 eta MP-1..3 irizpideak betetzen baditu, **kalitate oneko** produktutzat hartuko dugu.
- Aplikazioak 2.4 taulako irizpide guztiak betez gero, **kalitate bikaina** duela esango dugu, ezarri diren irizpide guztiak beteko balitu.
- Aldiz, aplikazioak ez badu behintzat KO-1 irizpidea betetzen, **kalitate urriko** produktutzat joko dugu, ez baitu aplikazioaren oinarrizko funtzionalitatea beteko.

Arloa	ID	Deskribapena	Probak
Nabigazioa	VX-N1	Aplikazioak Atzera botoiaren nabigazio estandarra erabiltzen du, eta pantailan ez ditu botoi bereziak erabiltzen Atzera botoiarentzat.	CR-3
	VX-N2	Elkarrizketa-koadro guztiak atzera botoiarekin itxi daitezke	CR-3
	VX-N3	Hasiera botoia edozein unetan sakatuta gailuaren pantaila nagusira joaten da.	CR-1
Jakinarazpenak	VX-S2	Aplikazioak jakinarazpenak honetarako soilik erabiltzen ditu: Erabiltzailearekin lotutako testuinguru aldaketa bat (esaterako, jasotako mezu bat), eta uneko gertaerarekin lotutako informazio edo kontrolak erakusteko	CR-9
Baimenak	FN-P2	Aplikazioaren oinarrizko funtzionalitateekin lotuta egon ezean, aplikazioak ez du datu konfidentzialik atzitzeko baimenik eskatzen (hots, kontratuak edota sistemaren erregistroa) edota erabiltzaileari dirua kostatu diezaioketen zerbitzuetara.	SC-4
SDK	PS-T1	Aplikazioa Android sistemaren azken bertsio publikoan exekuta daiteke, bere funtzionalitate nagusia galdu gabe	CR-0
Komunikazioa	KO-1	Aplikazioak eskaintzen dituen aukeren bidez robota erabiltzailearengana hurbiltzeko gai da.	MN-1
Mapa	MP-1	Aplikazioa Informatika Fakultateko mapa erakusteko gai da, bertako kokapen garrantzitsuenekin batera.	MN-1
	MP-2	Aplikazioa Fakultateko hainbat solairu erakusteko gai da, eta solairu-aldaketa egin daiteke, solairu bakoitzean dagokion robotaren kokalekua ikusten delarik.	MN-2
	MP-3	Erabiltzaileak bidaia hasi egiten duenean, mapan robotaren kokalekua eguneratzen da, eta kokaleku hori errealitatekoarekin bat dator.	MN-1

2.4 Taula: Aplikazioaren kalitate-irizpideak

(*Goialdean Android estandarrek definitutako irizpideak, eta behealdean proiektu honen ebaluaziorako definitutakoak.)

2.5.3 Ekintzak

Horiek horrela, proiektuaren garapenean zehar, probak periodikoak egingo dira, produktuaren kalitatea bermatzeko. Proba horiek bi taldetan sailkatuta egongo dira, batetik aplikazioaren gaineko barne-probak, eta kanpo-probak bestetik.

Barne probak

Barne probei dagokionez, kodearen helburua izango da bezeroaren eskakizunak betetzea, baina baita ere errorerik edo *bug*-ik ez izatea. Horretarako, egokiena izango litzateke test automatizatuak erabiltzea, lortu nahi ditugun emaitza zehatzak definituz, jakiteko ea garatzen dugun softwareak emaitza horiek jasotzen dituen. Hala ere, aplikazioaren inplementazioa asko luzatzen bada, gerta daiteke denborarik ez izatea probak automatizatzeko, eta egileak berak eskuz probatu behar izatea aplikazioa.

Kanpo probak

Printzipio arinen atalean azaldu den modura, iterazio bakoitzaren amaieran, eta noski, garatzaileak barne-probak exekutatu ostean, bezeroari aplikazioaren erakusketa bat egingo zaio, telematikoki edota presentzialki (pandemiaren egoerak ahalbidetzen bagaitu).

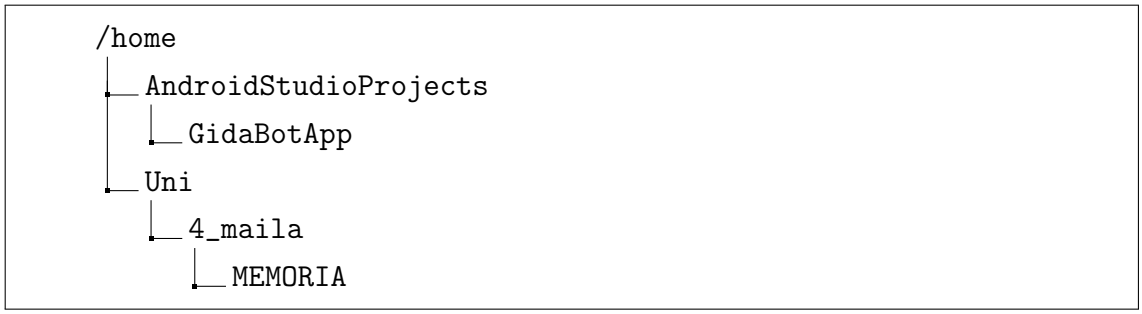
2.6 Informazio-sistema

2.6.1 Hardwarea

Informazio-sistemari dagokionez, aipatzekoa da proiektuaren garapenerako bi gailu erabiliko direla; ordenagailu eramangarria bata, eta Android sistema eragilea daraman telefono mugikorra bestea. Eramangarria aplikazioaren eta memoriaren garapenerako erabiliko da, eta telefono mugikorra, aldiz, garatutako produktuaren funtzionalitateak probatzeko. Telefono mugikorra probetarako soilik erabiliko denez, eramangarriko biltegiratzea aztertuko da soilik.

2.6.2 Fitxategien biltegiratze-egitura

Oro har, Eramangarriko biltegiratze-egituran, alde batetik memoriarekin lotutako fitxategiak izango ditugu, eta bestetik aplikazioaren kodearekin lotutakoak. Aplikazioaren proiektuak *home/AndroidStudioProjects* karpetan gordeko dira, eta memoriarekin lotutakoak, aldiz, *home/Uni/4_maila/GrAL/MEMORIA* karpetan. Hala ere, memoria Overleafen garatuko denez, soilik irudi, taula eta antzekoekin lotutako fitxategi editagarriak gordeko dira karpeta horretan.



2.5 Irudia: Memoria eta kodearen fitxategien egitura

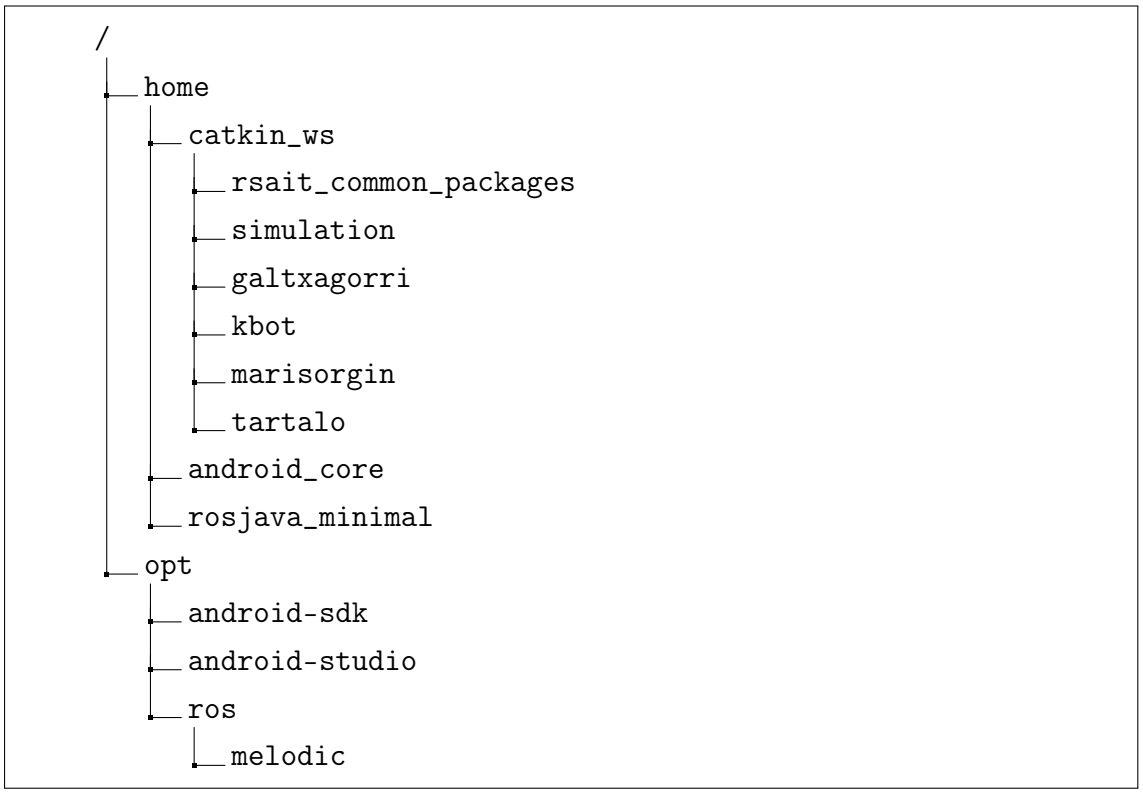
Memoriaren eta kodearen segurtasun-kopiei dagokienez, memoriako fitxategien segurtasun-kopia periodikoak egiten dira Mega.nz bidez hodeian.

Aplikazioaren kodearen babeskopientzat, aldiz, Githuben biltegi bat sortu da, non iturrikodearen bertsio desberdinen kopiak gordeko diren.

2.6.3 Konfigurazio-fitxategien kudeaketa

ROSen instalazioarekin erlazionatutako karpetak */home* eta */opt* karpetetan kokatuko dira. Bigarren honetan, Android Studioren instalazio-gunea ere egongo da (ikus 2.6 irudia konfigurazio-fitxategien egitura ezagutzeko).

Aipatzekoa da, Timeshift softwarearen bidez, konfigurazio-fitxategi guzti hauen segurtasun-kopia periodikoak egiten direla egunero, edozein momentutan Android Studio edo ROSek ustekabeko erroreren bat emango balu, fitxategien bertsio zaharrago bat berreskuratu ahal izateko.



2.6 Irudia: Konfigurazio-fitxategien egitura

2.7 Komunikazioen kudeaketa

Komunikazio eraginkor batek proiektu batean parte hartzen duten eragile desberdinen arteko zubia sortzen du, antolakuntza ingurune desberdinak, esperientzia maila desberdinak, eta proiektuaren exekuzio edo emaitzetan ikuspegi eta interes anitzak lotuz.

Hori dela eta, Gradu Amaierako Lan honen kasuan, proiektuko barne-komunikazioentzat, hau da, egilearen eta zuzendarien arteko komunikazio laburrak gauzatzeko, unibertsitateko posta elektronikoa erabiliko da.

Komunikazio luzeagoetarako edota zuzendarien laguntza behar den kasuetarako bilera telematikoak egingo dira, kasu horretarako *Blackboard Collaborate* plataformak eskaintzen dituen bilera-gelak erabiliz, aldez aurretik zuzendariekin ordu bat finkatuz.

2.8 Interesatuak

Proiektuaren interesatu nagusia memoria eta aplikazioaren egilea bera da, berari dagokiolako tribunalaren aurrean proiektua defendatzea, eta gainera, kalifikazio finala ere berari dagokiolako.

Bigarrenik, bezero rola betetzen duen RSAIT[1] ikerkuntza taldea proiektuko interesatua izango da, eta bere kide diren Ekaitz Jauregui eta Igor Rodriguez aldi berean proiektuko zuzendaria eta zuzendarikidea izango dira, hurrenez hurren.

Azkenik, proiektuaren defentsaren ebaluaketa burutuko duen epaimahaia bigarren mailako interesatua izango da, euren ardura fakultateak definitutako gidalerroak jarraituz proiektua kalifikatzea izango delarik.

[1] RSAIT ikerkuntza taldea: http://www.sc.ehu.es/ccwrobot/

3. KAPITULUA

Aurrekariak

Kapitulu honetan proiektuaren aurrekariak aurkeztuko dira. Lehenik eta behin, RSAIT taldeak garatutako GidaBot nabigazio-sistemaren funtzionamendua gainetik azalduko da, eta ondoren, proiektu honek planteatzen duen arazoari irtenbide bat eman diezaioketen proiektuak aurkeztuko dira.

3.1 GidaBot

GidaBot nabigazio-sistema *ROS framework*-ean oinarritutako robot-talde heterogeneo bat antolatu eta kudeatzeko diseinatutako aplikazioa da. GidaBot sistemako roboten helburua da gidari gisa aritzea solairu anitzeko eraikinetan. Nahiz eta sistemako erabiltzaileek egiten dituzten ibilbideak solairu anitzekoak izan daitezkeen, robot bakoitzak solairu bakarrean ematen du zerbitzua, ezin baititu ez eskailerak igo, ezta igogailua erabili ere. Modu horretan, zehazten den ibilbidearen arabera bi robot edo gehiagoren arteko kolaborazioa beharko da. Hortaz, RSAIT taldeak diseinatutako sistemak roboten arteko komunikazio-estrategia bat erabiltzen du gida-eginkizunetan helburuak eta bideak partekatzeko.[66]

Gainera, GidaBoten interfaze grafikoari esker, sistema inoiz erabili ez duten edota eraikinean berriak diren bisitariek helmuga-kokaleku bat hautatzeko aukera izango dute, eta robotek hautatutako kokalekuraino gidatuko dituzte. Proiektu honetan erabili den GidaBot bertsioa Donostiako Informatika Fakultatean lan egiteko dago egokituta, baina *Gazebo*n oinarritutako simulatzaile batekin erabiltzeko aukera ere eskaintzen du. [44]

3.2 Antzeko proiektuak

Proiektuaren garapenarekin hasi baino lehen, beharrezkoa da GrAL honek aurkezten duen arazoari soluzio bat eman diezaioketen proiektuak aztertzea, dagoeneko existitzen dena ez berrasmatzeko helburuarekin. Hauek ikertu ondoren, ikusi da ez dagoela planteatu den aplikazioaren funtzionalitateak eskaintzen dituen proiekturik. Hala ere, 2018an Eder Pérez ikasleak garatu zuen GrALak proiektu honekiko antzekotasuna izan dezake, eta Ros-Mobile Android aplikazioak ere ROS↔Android komunikaziorako beharko ditugun ideia batzuk eskaini dizkigu.

3.2.1 *Construcción de un sistema de navegación para interiore*s, Eder Pérez

2017-2018 ikasturtean Eder Pérezek garatu zuen GrALak proiektu honekiko antzekotasunak ditu. Ederren proiektuan Android aplikazio bat gauzatu zen, ROSekin komunikazioa burutzen zuena, eta aplikazioaren bidez robotak erabiltzaile bat solairu batean zehar gidatzen zuen. Haatik, Ederren aplikazioaren helburua ez zen, inondik ere, GidaBot sistemarekin integratzea eta bere interfazea Android sistema eragilera moldatzea.

Gainera, Ederrek 2018 urtean sortutako aplikazioak web-zerbitzari zentral bat erabiltzen zuen ROS sistemarekin komunikatzeko, eta gure aplikazioaren helburua, aldiz, aplikazioa sarearen bidez ROS sistemarekin zuzenean komunikatzea izango da. Modu horretan, erabiltzailearen *smartphone*a robot batera konektatuko da, eta robotari mezuak zuzenean bidaliko dizkio, GidaBot sisteman roboten arteko komunikazioan gertatzen den bezala.

3.2.2 *Ros-Mobile*

2020 urtean, Lübeckeko Unibertsitateko Nils Rottmann irakasleak eta Nico Studt ikasleak, beste hainbat ikaslerekin batera *ROS-Mobile: An Android application for the Robot Operating System*[68] argitaratu zuten. *ROS-Mobile* Android sistema eragilerako garatutako aplikazio bat da, ROS sisteman oinarritutako robot mugikorren kontrol eta bisualizazio dinamikorako sortuta. Aplikazioaren kodearen gehiengoak *MVVM* edo *Model-View-ViewModel* arkitektura jarraitzen duenez, aplikazioa oso egonkorra da.

Aplikazioak funtzionalitate ugari ditu, eta garatzaileek oso eguneratuta mantentzen dute [67]. Hala ere, aplikazioak RSAIT taldearen arazoa ez du konpontzen; hau da, ez du Gida-

Bot sistemarekin integraziorik eskaintzen, robotaren ibilbide guztia mugikorretik bertatik kontrolatzeko. Hala ere, nahiz eta GidaBoten erabilpen-kasurako balio ez duen, aplikazioaren arkitekturaren eredu modura erabili ahalko dugu; are gehiago, aurrerago Arkitekturaren atalean ikusiko denez, proiektu honetan garatuko den aplikazioak *MVVM* arkitektura ere jarraituko du.

4. KAPITULUA

Teknologiak

Proiektu handiekin lan egiten dugunean, berebiziko garrantzia dauka lan egiteko tresna egokiak hautatzeak. Izan ere, gure lan-fluxua mantsotzen duen tresnaren bat hautatzen badugu, epe-luzean denbora asko galduko dugu programak eragindako zailtasunengatik, eta denbora hori ezingo diogu inplementazio edota memoriaren garapenari esleitu. Hori dela eta, proiektuaren garapenerako tresnen-hautaketa egokian denbora inbertitzea interesgarria izango da, gure proiektuaren estimazioak betetzen lagunduko baitigu.

Beraz, atal honetan erabili den software guztia deskribatuko da, Android aplikazioa garatzeko erabili den Android Studio garapen-ingurunetik hasita, StarUML modelatze-aplikazioraino. Tresna bakoitzaren deskribapen labur bat egingo da, eta merkatuko beste aukeren gainetik zergatik hautatu den azalduko da.

4.1 Android Studio

Android Studio IntelliJ IDEA[37] garapen-ingurunean oinarritutako Android garapenerako Garapen Integratuko Ingurune ofiziala da (ingelesez *Integrated Development Environment, IDE*). IntelliJren editore eta garapen-tresnez gain, Android Studiok, Android aplikazioak garatzerako prozesuan laguntzen duten tresna gehigarriak ere eskaintzen ditu, hots, *Gradle*n[51] oinarritutako eraikuntza-sistema (*build system*), Android gailuak emulatzeko aukera ematen duen emuladore azkar eta aberatsa, edota *Google Cloud Platform*, *Google Cloud Messaging* eta *App Engine*-rekin integratzeko aukera.[57]

Proiektu honen garapenean zehar Android Studio 4.1.2 bertsioa erabili da, Ubuntu 18.04 sistema eragilean instalatu dena.

4.2 ROS[1]

The Robot Operating System (ROS), robotei zuzendutako softwarea garatzeko *framework* bat da. Sistema eragile batetik espero genitzaken zerbitzuak eskaintzen ditu, besteak beste, hardware abstrakzioa, maila baxuko gailu-kontrola, prozesuen arteko emisioak igarotzea eta paketeen kudeaketa. Era berean, tresnak eta liburutegiak eskaintzen ditu, ordenagailu anitzetatik kodea idatzi, exekutatu eta kudeatzeko. [17]

Izan ere, Aurrekarien kapituluan aipatu den bezala, GidaBot nabigazio-sistemak ROS erabiltzen du sistemako roboten kontrol eta kudeaketarako. Proiektuaren garapenerako ROS *Melodic* bertsioa erabili da, baina beste hainbat bertsio daude eskuragarri, *Indigo* eta *Kinetic*, besteak beste. [18]

4.3 Gazebo

Robotekin lan egiten duen garatzaile ororentzat funtsezkoa da simulatzaile on bat izatea. Ondo diseinatutako simulatzaile batek, algoritmoak azkar probatzeko, robotak diseinatzeko, erregresio-probak egiteko, eta adimen artifizialak entrenatzeko aukera ematen du, ingurumen errealistak erabiliaz. Beraz, Gazebok garatzaileen behar horiek asetzeko, robot-taldeak barru eta kanpoko ingurumenetan simulatzeko aukera ematen du. Gazebo simulatze-ingurunea Apache 2.0[29] lizentziapean argitaratuta dago. [30]

4.4 LaTeX

LaTeX kalitate handiko dokumentuak sortzeko sistema da. Gehienetan dokumentu tekniko eta zientifikoetarako erabiltzen da, baina ia edozein dokumentu mota argitaratzeko erabil daiteke. Ez da testu-prozesadore bat; izan ere, egileei dokumentuaren aurkezpenaz ahaztu eta eduki egokia idazten zentratzen laguntzen die.

[1] Atal honetan ROS ingurunea oso laburki azaltzen da, baina irakurleak beharko balu, ROS eranskinean ROSen osagai nagusiak aztertzen dira.

Gradu Amaierako Lan honen memoria idazteko erabili da LaTeX, baina *TexMaker*[8] edota *LyX*[26] bezalako editore lokalak erabili ordez, Overleaf[39] izeneko online editorea erabili da.

4.4.1 Overleaf

Overleaf *startup* eta gizarte-ekintzailetza bat da, hainbat lankidetza-tresna moderno eskaintzen dituena, zientzia eta ikerketa-lanak azkarrago eta irekiago garatzen laguntzeko. Izan ere, eskaintzen dituen tresnen artean erabiliena dena LaTeX editorea da. Aipatzekoa da ere, plataforma ordainpekoa dela bi kolaboratzaile baina gehiagoko proiektuentzat, baina proiektu honen zuzendariek e-mail berdinarekin lan egin dutenez, doako bertsioarekin nahikoa izan da.[39]

Proiektu honetako memoriaren idazte-prozesua asko erraztu du plataforma honek, Google Drivek eta antzeko kolaborazio-plataformek eskaini ditzaketen tresnen antzekoak baititu, batez ere dokumentuen iruzkinekin lotuta.

4.4.2 PgfGantt

Pgfgantt LaTeX paketeak Gantt diagramak sortzeko ingurune bat eskaintzen du. Sarean mota honetako diagramak sortzeko aukera asko dauden arren, horietako asko ordainpekoak dira, edota irudiak esportatzerakoan kalitate urriko irudiak ematen dituzte. Hori dela eta, LaTeX-en bertan Gantt diagramak sortzeko aukera ematen duen tresna hau erabili da memoriaren garapenerako.[72]

4.4.3 Dirtree

Memoriaren hainbat zatitan, batez ere Informazio-sistemaren deskribapenean, sistemaren fitxategi-egitura modu grafiko batean deskribatzeko beharra zegoen. Sistema eragileko fitxategi-hautatzailearen pantailaren argazki bat jartzea zen aukera bat, baina Dirtree paketea erabiltzea aukera hobea iruditu zitzaigun. Izan ere, Dirtree paketeak fitxategi-egiturak bistaratzeko aukera ematen du sintaxi oso sinple baten bitartez.[9]

4.5 Terminator

ROS sistemarekin lan egiterakoan, exekuzio guztiak linuxeko *bash*-aren bitartez egin dira, eta hainbat lan-fluxutarako terminal askorekin lan egin behar da aldi berean. Ataza horiek burutzeko, oso komenigarria izaten da leiho berean hainbat terminal izatea errazten duen aplikazio bat erabiltzea. Arazo horri aurre egiteko, Terminator-ek leiho desberdinak sareta-moduko batean antolatzeko aukera ematen digu, eta horrek, ROS sistemaren eta Android aplikazioaren komunikazioa probatzeko prozesua erraztu du, batez ere.[7]

4.6 Notion

Notion hainbat zerbitzu eskaintzen dituen plataforma da, oro har, datu-baseak, oharrak, wikiak, egutegiak, oroigarriak eta abar gordetzeko aukera ematen duena.[79] Proiektu honetan, tresna hau ataza bakoitzaren dedikazioaren jarraipena egiteko erabili da. Horretarako, plataforman bertan taula bat sortu da, eta GrALeko atazaren batean lan egiten zen bakoitzean, atazaren hasiera eta amaiera ordua bertan ezarri dira. Honi esker, esan daiteke Jarraipen eta kontroleko kapituluan aurkeztuko diren dedikazioak nahiko zehatzak izan direla.

4.7 Marvel App

Interfaze grafikoaren prototipaketa egiteko erabili den tresna da. Enpresetan, interfazeen diseinuak sortzeko lan postu espezifikoak egon ohi dira (*UI/UX designer*), eta beraz, ugariak dira diseinuak sortzeko aukera ematen diguten tresnak merkatuan, *InVision*, *Adobe XD* edota *Framer* direlarik ezagunenak.[75] Hala ere, tresna hauek enpresei zuzenduta daudenez, gehiengoak ordainpekoak dira eta ikaste-kurba handia izan ohi dute.

Hortaz, nahiz eta merkatuan interfaze prototipaketa burutzeko aukera hobeagoak dauden, proiektu honen interfazearen prototipoa osatzeko Marvel App diseinatze-plataforma erabili da. Alde batetik, doakoa delako, eta bestetik, hirugarren mailako Pertsonen eta Konputagailuen arteko Elkarrekintza ikasgaian erabili zelako, eta beraz, logikoa denez, denbora gutxiago beharko zelako honen bidez prototipoa osatzeko.

4.8 Git

Git bertsio-kontrolerako kode irekiko doako sistema da. Bere helburua, proiektu txiki zein handien bertsio-kontrola azkar eta modu eraginkor batean egitea da.[10]

Git funtsezkoa izan da proiektuaren inplementazio-fasean, eskaintzen duen bertsio-kontrol errazari esker, edozein momentutan kodean atzera egin ahal izateko. Horretarako, kodean funtzionalitate berriak inplementatu eta probatu ondoren, ohikoa izaten zen *commit* bat egitea.

Aplikazioaren iturri-kodearen bertsio ezberdinak hodeian gordetzeko Github[33] erabili da, batez ere Android Studiorekin integrazioa eskaintzen duelako. Beraz, inplementazioaren hasieran *repository* edo biltegi bat sortu zen, eta bertan iterazio bakoitzaren hasieran *branch* berri bat hasten zen, aurrekoaren emangarriaren kodea eskuragarri mantentzeko eta memoriaren idazketa errazteko ondoren.

4.9 Inkscape

Proiektuaren inplementazioan, Donostiako Informatika Fakultateko planoaren etiketen bigarren bertsioa sortzeko, *.png* formatuko irudiak *.svg* formatura bektorizatu ziren, vectorizer.io plataformaren bitartez. Izan ere, planoen irudiak bektore-formatuan izateak irudiei aldakortasuna ematen die, beharren arabera tamaina ezberdineko *.png* fitxategietara eskalatzeko aukera ematen baitute, kalitaterik galdu gabe. Gainera, *svg* formatuko irudien gainean etiketak ezartzeko Inkscape tresna erabili da, bektorizatutako irudiak editatzeko aukera ematen duen doako software librea. [25]

4.10 StarUML

StarUML software-modelatze plataforma bat da, batez ere UML diagramak sortzera bideratuta. Hamaika diagrama-mota ezberdin sortzeko aukera eskaintzen du, batik bat klase diagramak, sekuentzia diagramak eta erabilpen-kasu diagramak, proiektuaren memoria honetan erabili direnak soluzioaren diseinu eta soluzioaren garapenaren atalean. StarUMLk ordainketa-planak eskaintzen ditu, baina erabilpen profesionaletara bideratuta daude. Proiektu honetan starUMLren doako lizentzia erabili da programaren 4.0.1 bertsioarekin, gaur arte bertsio eguneratuena dena. [43]

Aipatzekoa da ere, beste motatako diagramak osatzeko draw.io plataforma erabili dela. LDE diagrama edota Sistemaren Arkitekturaren diagrama, esaterako, azken tresna honen bidez sortu dira.

4.11 Timeshift

Timeshiftek Windowseko *Restore* eta MacOSko *Time Machine* tresnen antzeko funtzioa betetzen du Linuxen. Timeshiftek gure sistema eragilea babesten du, sistemaren *snapshot* inkrementalak denbora-tarte erregularretan sortuz. *Snapshot* edo sistemaren "argazki" horiek aurrerago berreskura daitezke, sisteman egon diren azken aldaketak baliogabetuz. [76]

Tresnaren helburu nagusia da sistema-fitxategiak eta konfigurazioak soilik babestea. Hori dela eta, Informazio-sistemen atalean eta arriskuen kudeaketan aipatu den modura, GidaBoten eta Android Studioko konfigurazio fitxategi guztien babeskopiak gordetzeko Timeshift erabili da, ezusteko aldaketa edo erroreren bat egonez gero atzera egin ahal izateko.

4.12 Mega.nz

Timeshift konfigurazio-fitxategien babeskopia lokalak egiteko erabili da, baina proiektuarekin erlazionatutako eduki-fitxategiak hodeian gordetzeko mega.nz edo mega.io plataforma erabili da. Linuxerako mahaigainerako aplikazioa ere eskaintzen du, eta honen bidez eduki-fitxategien kopiak hodeian gorde dira. Gainera, $50TB$ko doako kuota eskaintzen du, nahikoa eta sobera dena GrAL honen edukiarentzat. [41]

4.13 Vegas Pro

Proiektuaren defentsarako aplikazioaren demo bat prestatu da, bideo formatuan. Bideo horretarako aplikazioa fakultatean probatu da robot fisikoekin, eta bi solairuko bidai bateko roboten ikuspuntuak grabatu dira, gailu mugikorreko pantailaz gain. Ondoren, bideo-zati horiek guztiak elkartzeko, MAGIX Vegas Pro bideo-editorea erabili da, 2019an argitaratu zen 17. bertsioan. [34]

Tresna hau merkatuko beste aukeren gainetik hautatu da, egile honek aurretik beste bideo-edizio proiektuetarako erabili duelako, eta ondorioz beste alternatibek baina lan-fluxu azkarragoa eskaintzen diolako.

5. KAPITULUA

Eskakizunen bilketa

Atal honetan aplikazioaren eskakizun edo betekizunen bilketa aurkeztuko da. Honen helburua aplikazioaren betekizun funtzional eta ez funtzionalak modu argi batean definitzea izango da. Izan ere, atal hau proiektuaren arrakastarako funtsezkoa izango da, azken produktuak erabiltzaileen beharrak asetzeko beharrezkoa baita garatu beharreko sistemaren betebeharrak oso ondo ulertzea.

5.1 Erabilpen kasuak

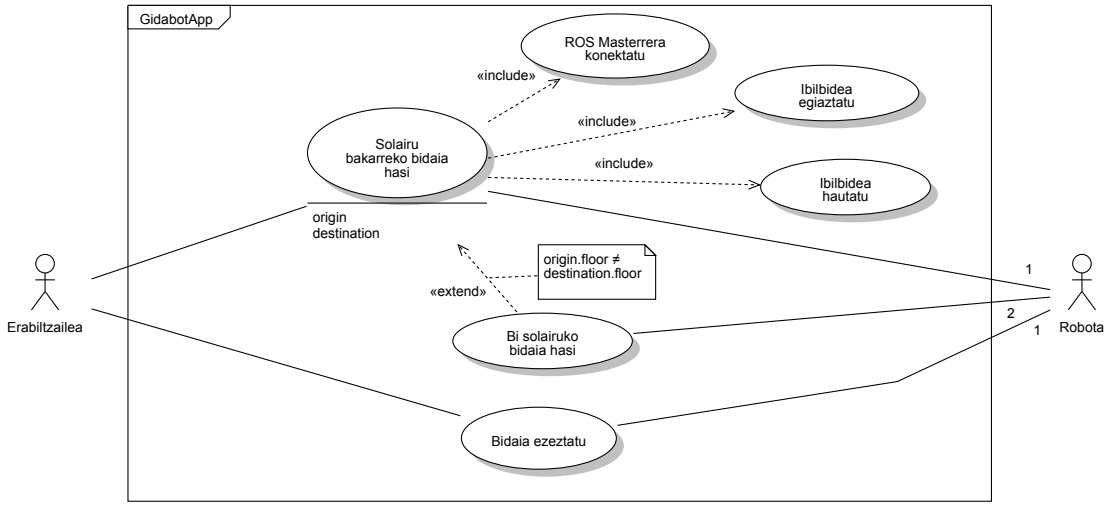
Erabilpen kasu batek ekintza edo jarduera bat deskribatzen du. Helburuetan deskribatutako funtzionalitateak garatu ahal izateko, beharrezkoa da aurretik erabiltzaileak zein ekintza burutu ahalko dituen definitzea. Hori dela eta, proiektuko funtzionalitateak asetzen dituen eredua garatu da, 5.1. irudian ikus daitekeen moduan.

Bide batez, Proiektuaren irismenean aipatu denez, aplikazioaren bereizgarri nagusia ez da erabilpen kasuen ugaritasuna, erabilpen kasu bakoitzak inplementatzeko daukan konplexutasuna baizik, eta beraz, garatuko den sistemak ez ditu erabilpen kasu asko izango.

Sistemako erabilpen kasu nagusia “Bidaia hasi” izango da. Bertan, erabiltzaileak Fakultateko zein kokalekutan dagoen zehaztuko du, eta dagokion solairuko robota erabiltzailearengana hurbilduko da, ondoren, erabiltzailea zehaztutako helmugara gidatzeko. Hala ere, gerta daiteke erabiltzailearen helmuga beste solairu batean egotea, eta horregatik erabilpen kasu honek hedapen-puntu bat izango du; erabiltzaileak zehaztutako jatorria eta

helburua, hain zuzen. Hedapen-puntu horren arabera, solairu bakarreko edo bi solairuko bidaia egikarituko da.

Hari beretik, erabiltzaile batek bidaia hasten duenean, bidaia hori ezeztatzeko aukera izan beharko du, eta horretarako definitu da "Bidaia ezeztatu" erabilpen kasua.



5.1 Irudia: Erabilpen kasuen eredua

5.1.1 Aktoreen deskribapena

Erabilpen kasu bateko aktoreak sisteman parte hartzen duten aktore oro dira. Gure aplikazioaren kasuan, aktore hauek erabiltzaile arrunta eta GidaBot sistemako robotak izango dira.

Erabiltzailea: Sistemako aktore nagusia. Erabiltzaile batek interfaze grafikoaren bidez ibilbide bat zehaztuko du, eta robotak zehaztutako helbururaino gidatuko du.

Robota: Sistemako bigarren mailako aktorea izango da, eta aplikazioari zerbitzu bat emango dio: GidaBot sistemaren bitartez erabiltzailea zehaztutako helbururaino gidatuko du. Erabiltzaileak zehazten duen ibilbidearen arabera, bi robotek parte hartuko dute ibilbidean.

5.1.2 ROS Masterrera konektatu

Erabiltzaileak aplikazioa abiaraziko du, baina aplikazioak robotekin komunikatu ahal izateko ROS Master batekin konektatuta egon behar du. Horretarako, abiatu bezain laster ROS Masterraren IP helbidea eta portua sartzeko eskatuko zaio erabiltzaileari.

Aurrebaldintzak

1. Erabiltzaileak ROS Masterraren URIa daki (IP helbidea eta portua), edo ROS Masterraren IP helbide eta portua dituen QR kodea dauka eskuragarri

Gertaera fluxua

1. Erabilpen kasua hasten da erabiltzaileak GidaBotApp aplikazioa abiarazten duenean
2. Erabiltzaileak ROS Masterraren IP helbidea sartuko du
3. Erabiltzaileak “Connect” botoia sakatuko du
4. Konexioa ezarriko da eta “Connected” mezua agertuko da pantailan

Fluxu alternatiboa

“Cancel” botoia sakatu. Erabiltzaileak “Cancel” botoia sakatuz gero, erabiltzaileari jakinaraziko zaio aplikazioa berrabiarazi behar duela eta aplikazioa itxi egingo da.

QR bidez konexioa ezarri. Bigarren pausuan, IP helbidea eskuz sartu ordez “Read QR-Code” botoia sakatuz gero, Masterraren IP eta portua dituen QR kodea irakurriko da, eta sistemak QR horri dagokion IP helbidera ezarriko du konexioa.

Konexioaren ukoa. Bigarren pausoan ezarritako IP-a edo portua ez badira zuzenak, “Unable to communicate with master” mezua agertuko da pantailan.

Postbaldintzak

ROS Masterrarekin konexioa ezarri da, edo aplikazioa itxi egin da.

5.1.3 Solairu bakarreko bidaia hasi

Erabiltzaileak aplikazioan non dagoen eta nora joan nahi duen zehaztu beharko du, robota beregana hurbildu ahal izateko. Behin bi kokalekuak zehaztuta, solairu horretako robota gerturatuko zaio, zehaztutako helbururaino gidatzeko.

Aurrebaldintzak:

Erabiltzailea ROS Masterrera konektatuta dago.

Gertaera fluxua

1. Erabiltzaileak ibilbidea hautatuko du (*origin* eta *destination*)
2. Erabiltzaileak “Bidaia hasi” botoia sakatuko du
3. Sistemak baliozko ibilbidea dela egiaztatuko du: jatorria eta helmuga ez daude hutsik
4. Robotari jatorrira (*origin*) joateko eskaera bidaliko zaio
5. Robota jatorrira iritsiko da
6. Erabiltzaileari galdetuko zaio ea hurrengo helburura (*destination*) jarraitu nahi duen.
7. Erabiltzaileak “Ados” botoian sakatuko du
8. Robotari helmugara (*destination*) joateko eskaera bidaliko zaio
9. Robota helmugara iritsiko da
10. Helmugara iritsi dela jakinaraziko zaio erabiltzaileari, mezu bat pantailaratuz

Fluxu alternatiboa

Ibilbide okerra. Erabiltzaileak ezarritako ibilbidearen eremuren bat hutsik badago, pantailan mezu bat agertuko da egoeraren berri jakinarazteko, eta eskatuko zaio falta diren eremuak betetzeko.

Erabiltzaileak bidaia ezeztatu. Robotari jatorrira joateko eskaera bidaltzen zaion unetik (4. pausoa) bidaia ezeztatzen bada, erabiltzaileak ibilbidea berriz ezarri beharko du. Modu berean, 7. pausoan erabiltzaileak ez badu "Ados" botoia sakatzen, bidaia ezeztatu egingo da.

Barne-fluxuak

Ibilbidea hautatu.

1. Erabiltzaileak zein solairutan dagoen adieraziko du, interfazetik solairuetako bat hautatuz (0,1,2 edo 3)
2. Erabiltzaileak non dagoen zehaztuko du (*origin*)
3. Erabiltzaileak nora joan nahi duen adieraziko du (*destination*).

Ibilbidea egiaztatu. Sistemak egiaztatuko du ezarritako jatorria eta helmuga eremuak ez daudela hutsik

Bidaia ezeztatu.

1. Erabiltzaileak "Bidaia ezeztatu" botoia sakatuko du
2. Robotari bidaia ezeztatzeko eskaera bidaliko zaio
3. Robota gelditu egingo da

Bi solairuko bidaia. 7. pausoan, helmugaren solairua eta jatorriko solairua desberdinak badira, fluxua Bi solairuko bidaiara pasako da.

Postbaldintzak

Erabiltzailea helmugara iritsi da, bidaia ezeztatu da, edo helmuga beste solairu batean dago.

5.1.4 Bi solairuko bidaia hasi

Erabiltzaileak hautatutako jatorria eta helmuga ez badaude solairu berdinean, erabiltzailearen solairuan dagoen robotak erabiltzailea igogailu/eskaileretara eramango du, eta beste solairuko robotak igogailu/eskaileretatik helmugaraino.

Aurrebaldintzak

Erabiltzailearen solairuko robotak erabiltzailea jatorriraino eraman du (Solairu bakarreko bidaiaren 7. puntua arte), eta jatorria eta helmuga solairu desberdinetan daude.

Gertaera fluxua

1. Sistemak galdetuko du ea eskaileretatik edo igogailutik igo nahi duen
2. Jatorriko solairuko robotak erabiltzailea eskailera edo igogailuraino eramango du
3. Sistemak jakinaraziko dio erabiltzaileari zein solairutara jo behar duen
4. Helmugako solairuko robota eskailera edo igogailura gerturatuko da
5. Erabiltzailea helmuga-solairura igo (edo jaitsiko) da
6. Erabiltzaileak robotaren pantailan “Ados” botoia sakatuko du
7. Helmugako solairuko Robotak erabiltzailea helmugaraino eramango du

Fluxu alternatiboa

Bi solairuko bidaia ezeztatu

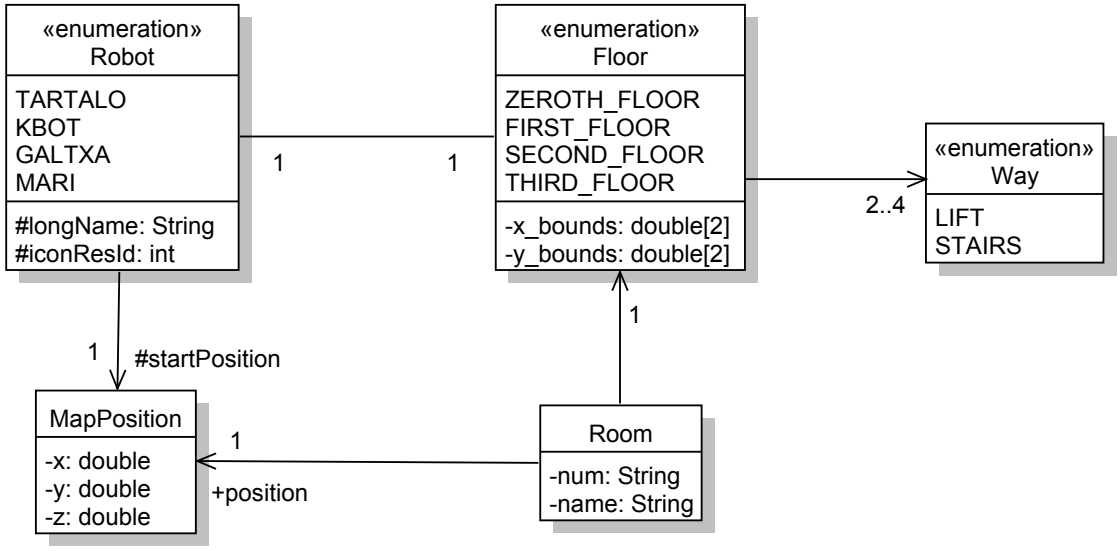
1. Erabiltzaileak “Bidaia ezeztatu” botoia sakatuko du
2. Robotei bidaia ezeztatzeko eskaera bidaliko zaie
3. Robotak gelditu egingo dira

Postbaldintzak

Erabiltzailea helmugara iritsi da edo bidaia ezeztatu du.

5.2 Domeinua

Domeinuaren eredu batek, interesekoa den domeinu bateko klase kontzeptualen edo benetako munduko objektuen irudikapen bisuala adierazten du. GidaBotApp aplikazioaren domeinua 5.2 irudian islatu da, non aplikazioak erabiliko dituen mundu errealeko entitate nagusiak irudikatzen diren: Robota, Solairua (*Floor*), mapako posizioa (*MapPosition*), gela (*Room*) eta norabidea (*Way*), hain zuzen.



5.2 Irudia: Domeinuaren eredua

5.2.1 *Robot*

Robot entitateak GidaBot sistemako robot bat irudikatzen du. GidaBot nabigazio-sisteman lau robot zehatz erabiltzen direnez, entitatea *enum* edo *enumeration* moduan adierazi da, lau instantzia horiek, eta soilik horiek existituko direlako gure sisteman: Tartalo, Kbot, Galtxa eta Mari robotak, hain zuzen. Hots, robot bakoitzak hasierako posizio bat gordeko du atributu modura, baita bere izen luzea (*longName*) ere: Galtxaren kasuan Galtxagorri, eta Mariren kasuan Marisorgin. Gainera, robot bakoitzak bere ikonoaren baliabidearen identifikatzailea izango du (*resource id*), aplikazioak robot bakoitzaren ikonoa mapan erakutsi ahal izateko.

5.2.2 *Floor*

Solairu edo *Floor* entitateari dagokionez, solairu bakoitzak Informatika Fakultateko solairu bat adieraziko du. GidaBot sisteman solairu bakoitzean robot finko bat dago (beheko solairuan Tartalo, edota lehenengoan Kbot, esaterako), eta beraz, gure sistemako solairuek ere robot bakarra izango dute esleituta. Kasu honetan entitatea ere *enum* moduan adierazi da, lau solairu horiek (bakarrik) egongo baitira operatibo gure sisteman: beheko solairua, lehena, bigarrena eta hirugarrena. Hala ere, Informatika Fakultatean egon badago beste solairu bat: entreplanta, baina bertara ez denez robotik iristen, aplikazioren garapenerako ez da aintzat hartu.

5.2.3 *Room*

Fakultateko gela edo kokaleku esanguratsuenak *Room* entitatearekin adieraziko dira: Kafetegia, Fakultateko Sarrera Nagusia edota RSAITen laborategia, esaterako, gure sistemako gelak izango dira. Logikoa denez, gela bakoitzak posizio bat izango du mapan: *MapPosition* entitatea.

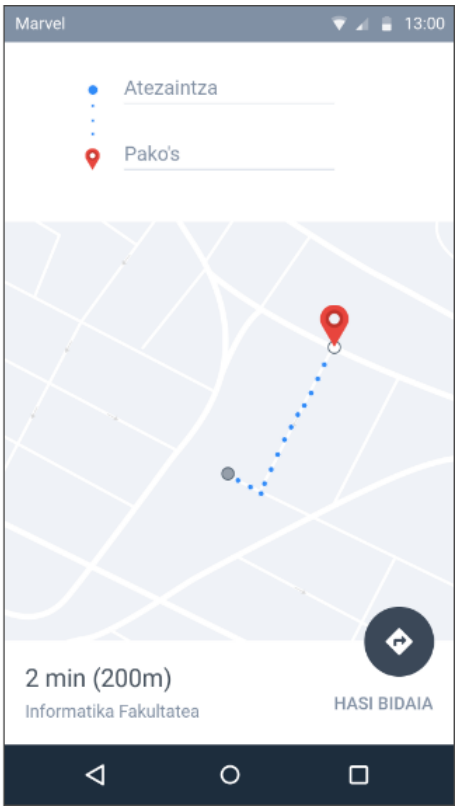
5.2.4 *MapPosition*

MapPosition entitateak mapako posizio bat irudikatuko du. Robotak eta gelak mapako posizio batean egongo dira, eta posizio horietako bakoitzak x, y eta z koordenatuak izango ditu. Gainera, robot bakoitzak hasierako posizio desberdin bat izango du.

5.3 Interfazearen prototipaketa

Proiektu honetako eskakizunen bilketarako erabili den beste teknika bat interfaze grafikoaren prototipaketa da. Prototipoari esker ikusiko da analistaren eta bezeroaren ideiak bat datozen, eta prototipoari behatuz, bezeroak azpimarratu ahalko du zer aldaketa egin nahi dituen, proiektuaren inplementazioarekin hasi baino lehen. Modu honetan, diseinu eta garapenean zehar iradokizun horiek kontuan izango dira.

Teknologien atalean aipatu den legez, Marvel App erabili da interfazearen prototipoa osatzeko: 5.3 irudian interfazearen pantailetako bat ikus daiteke.



5.3 Irudia: Interfazearen prototipoaren "Hasi bidaia" pantaila

(Prototipo interaktibo osoa esteka honetan eskuragarri)

5.3.1 Prototipaketaren ondorioak

Prototipoa bezeroari erakutsi ondoren, bezeroak hainbat iradokizun edo hobekuntza proposatu zituen interfazearen inguruan:

- Erabiltzaile-izena sartzea ez da beharrezkoa, sistema erabiltzaile anonimoek ere erabili ahalko dutelako.
- Helmuga zerrenda batetik hautatzen den modura, erabiltzaileak jatorria ere zerrenda batetik hautatzea, mapatik zuzenean hautatu ordez.
- Robotak egingo duen ibilbide osoa mapan ez erakutsi (5.3. irudian ikusten den modura). Izan ere, denbora gehiegi hartuko lukeelako hori inplementatzeak, eta aplikazioaren funtzionamendu egokirako ez da beharrezkoa.
- Aplikazioaren kontrol guztiak pantaila bakarrean agertzea: hots, non zauden (jatorria), nora zoazen (helmuga) *drop-down list* moduko zerrenden bidez hautatzea,

eta pantaila berean erabiltzaileak mapa ikustea, bere burua ingurunean kokatu ahal izateko.

- Pantaila nagusi horretan “Cancel” botoi bat gehitzea, erabiltzaileak bidaia ezeztatu ahal izateko.

6. KAPITULUA

Soluzioaren diseinua

Behin eskakizunen bilketa eginda eta aplikazioak bezeroaren beharrak asetzen dituela bermatuta, soluzioaren diseinua garatuko da. Lehenik eta behin aplikazioak izango duen arkitektura definituko da, ondoren aplikazioaren klase diagrama aurkeztuko da, eta azkenik erabilpen kasu bakoitzaren sekuentzia diagramak azalduko dira. Bide batez, aplikazioaren inplementazioa modu iteratiboan garatu denez, atal honetan soilik garapeneko azken iterazioko aplikazioaren, hau da, produktu finalaren egitura aurkeztuko da.

6.1 Arkitektura

Mahaigaineko aplikazioek kasu gehienetan sarrera-puntu bakar bat izan ohi dute, eta behin exekutatuta programa bakar eta independiente bezala exekutatzen dira. Android aplikazioek, ordea, egitura askoz ere konplexuagoa dute. Hots, mugikorrerako aplikazio orok osagai ugari ditu: besteak beste, jarduerak (*activity*), zatiak (*fragments*), zerbitzuak, eduki-hornitzaileak eta broadcast hartzaileak (*broadcast receivers*). [54]

Kontuan izanik erabiltzaile batek denbora-tarte txiki batean app asko erabiltzen dituela, aplikazioak fluxu eta eginkizun mota ezberdinetara egokitu behar dira. Esaterako, erabiltzaile batek bere gustuko argazki bat sare sozial baten aplikazioan partekatu nahi badu, aplikazioak honako ekintzak burutuko beharko ditu:

1. Aplikazioak gailuko kamera botatzeko deia bidali, eta Android sistemak kamera

aplikazioa botako du, erabiltzaileak argazkia atera dezan. Beraz, une honetan sare sozialaren aplikazioa utzi du, baina bere esperientziak egokia izaten jarraitzen du.

2. Kamera aplikazioak beste *intent* batzuk abiaraziko ditu, esaterako, fitxategi-hautatzailea, eta honek ere beste aplikazio bat jaurti dezake argazkia hautatzeko.

3. Azkenik, erabiltzailea sare sozialeko aplikaziora bueltatuko da, eta argazkia partekatuko du.

Prozesuaren edozein unetan, erabiltzaileak dei edo jakinarazpen bat jaso dezake, eta argazkia partekatzearen prozesua eten. Eten horren ostean, erabiltzaileak espero du, bere sare sozialeko aplikaziora bueltatu eta argazkia partekatzeko prozesuarekin jarraitzeko aukera izatea. Jokabide hau oso ohikoa da gailu mugikorretan, eta beraz, gure aplikazioak zuzen maneiatu beharko ditu mota hauetako fluxuak.

Gainera, telefono mugikorren baliabideak mugatuak direnez, Android sistema eragileak edozein momentutan zenbait aplikazio-prozesu hil ditzake, prozesu berriak exekutatu ahal izateko. Hori dela eta, Android aplikazioetan garrantzitsua da bista-geruzan ez informazio ezta egoerarik ere gordetzea, sistema eragileak gure aplikazioaren bistak desordenaturik egikaritzea erabaki baitezake, edota edozein unetan etetea.

Baina, nola diseinatuko dugu gure aplikazioa ezin badugu ikuspegi edo bista geruzan informaziorik gorde? Bada, horretarako Kezka-banaketa edo *Separation of Concerns, SoC* jarraituz. [35] Ohiko akatsa izaten da Android aplikazioetan bista-klaseetan (hots, *Activity* eta *Fragment*) kode guztia idaztea, baina klase hauek soilik erabiltzailearen interfazearekin lotutako logika inplementatu beharko lukete. Modu berean, garrantzitsua da ere erabiltzailearen interfazeak eredu edo *Model* batetik hartzea informazioa: *Model* edo ereduak aplikazio baten datuak maneiatzeaz arduratzen diren osagaiak dira, eta ez daukate aplikazioaren bista geruzako objektuekiko menpekotasunik.

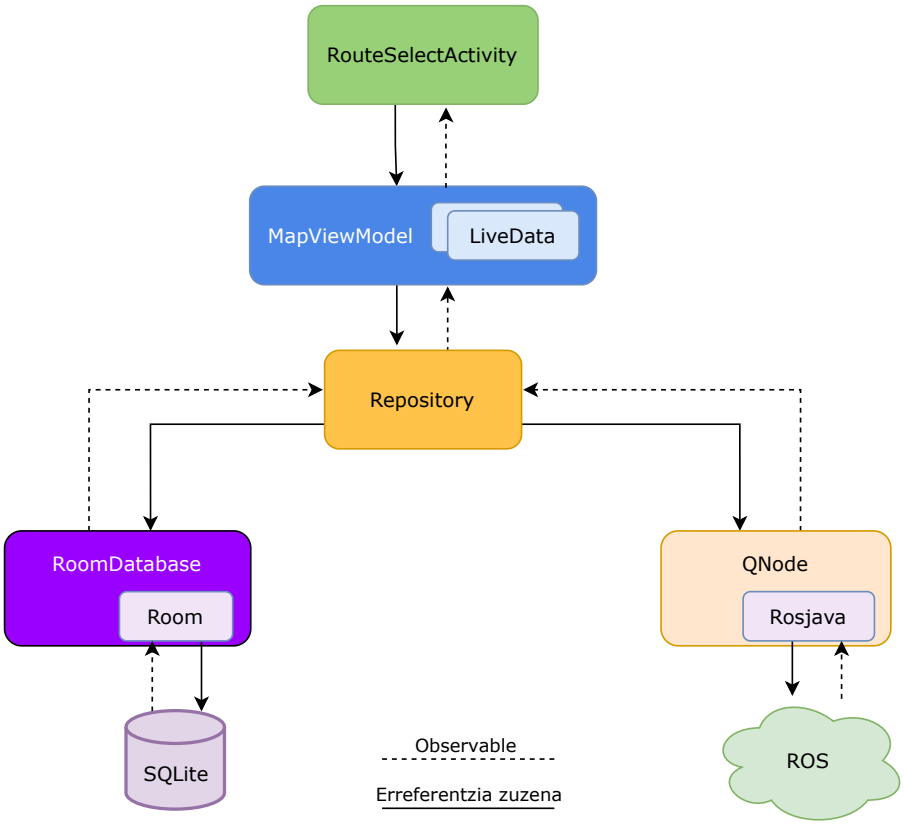
Hau guztia kontuan izanik, Android garatzaileen dokumentazioan *Model-View-ViewModel* (MVVM) arkitekturarekin lan egitea gomendatzen dute Android aplikazio bat garatzerakoan.[54]

6.1.1 MVVM

MVVM edo “Bista-Eredu-BistaEredu” arkitektura softwarea garatzeko arkitektura-patroi bat da. Arkitektura honek hiru osagai nagusi ditu, izenean bertan ageri direnak: Bista (*View*), Eredua (*Model*) eta BistaEredua (*ViewModel*). Arkitektura-patroi honen helburu nagusia erabiltzailearen interfazearen (Bista) geruza negozioaren logikaren geruzatik

(Eredua) bereiztea da. Horretarako, BistaEredu edo *ViewModel* geruzak aurkezle lana egiten du, eredutik jasoko dituen datuak bistari erakutsiz, honek azkenik informazioa erabiltzaileari aurkezteko. [78]

Android Developersen aurkezten zaigun arkitektura oinarritzat hartuz, 6.1 diagraman GidaBotApp aplikazioaren arkitektura azaltzen da. Irudian beha daitekeenez, arkitekturako osagai bakoitzak soilik bere azpian dagoen osagaiarekiko menpekotasuna dauka, eta goiko klaseak behekoaren inguruko informazioa jasotzen du *Observer* patroiaren bitartez. Salbuespen bakarra *Repository* klasea izango da, zeinek bi iturritatik jasoko dituen datuak, bai datu-eredu iraunkorretik, baita ROSetik ere, hau da, GidaBot sistemako robotetatik.



6.1 Irudia: Aplikazioaren arkitektura

Arkitektura-diseinu honi esker, erabiltzaileak esperientzia koherente eta atsegin bat izango du gure aplikazioa erabiltzean. Nahiz eta erabiltzailea aplikaziora bueltatzen den hainbat minuturen ostean, informazioa lokalki mantenduko da. Gainera, bueltatzen den momentuan erakusten den informazioa zaharkituta badago, *Repository* klaseak informazioa eguneratu egingo du.

6.2 Android bertsioa

Aplikazioaren garapenarekin hasi aurretik, helburu edo *target* izango den Android bertsioa kontuan izatea garrantzitsua da, hau da, Android Studion *compileSdkVersion*, *targetSdkVersion* eta *minSdkVersion* balioak definitzea. *CompileSdk* bertsioaren bitartez, *Gradle*ari[51] esango diogu zein Android SDK-rekin konpilatu behar duen aplikazioa, eta horrek berebiziko garrantzia izango du aplikazioaren garapenean. Izan ere, konpilatuko dugun Android bertsioaren arabera aplikazioaren inplementazioa aldatuko baita, Android bertsio oso zahar bat hautatzen badugu, ez dugu aukerarik izango MVVM patroia jarraitzeko beharrezkoak diren tresnak[55] hautatzeko aukerarik izango, edota sistema eragileko baliabide batzuk atzitzeko. Horrez gain, *targetSdkVersion* eta *minSdkVersion* parametroekin adieraziko diogu aplikazioak zein Android bertsio-tartetan ibil daitekeen funtzionalitaterik galdu gabe. [38]

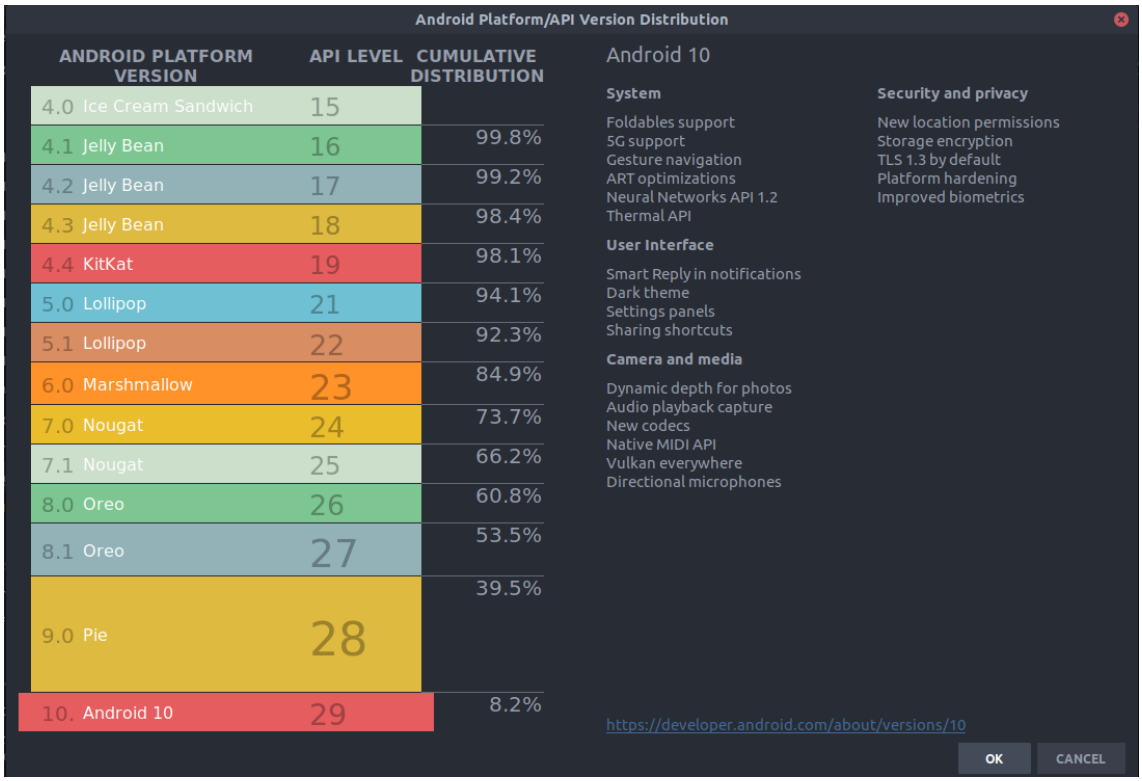
Gure aplikazioaren Android bertsioa hautatzen laguntzeko, Android Studiok proiektu berri bat sortzean 6.2 irudian ikusten diren datuak erakusten dizkigu. Datu horien arabera, Android 10 konpilatze-bertsioa hautatuz gero, gailuen %8.2ak soilik erabili ahalko luke gure aplikazioa. Bestalde, API 28 maila hautatzen badugu, (*9.0 Pie*), erabiltzaileen %39.5ak exekutatu ahalko luke aplikazioa. Izan ere, Android garapenaren arazo nagusia atzerako bateragarritasuna da, API maila bakoitzean funtzionalitate berriak txertatzen baitira.

Hori horrela, proiektu honetan garatuko den aplikazioa Android APIaren 29 mailan (Android 10) konpilatuko da, batez ere egile honen gailu mugikorrak erabiltzen duen Android bertsioa delako. Hala ere, Android bertsio zaharragoa daukaten mugikorrek aplikazioa exekutatu ahal izateko, Androidek eskaintzen dituen bateragarritasun-klaseak erabiliko dira aplikazioan, *AppCompatActivity*[50], esaterako.

6.3 Klase diagrama

6.3. irudian GidaBotApp aplikazioaren klase diagrama azaltzen da. Bertan, aplikazioaren egitura nagusia ikus daiteke, non aplikazioa osatzen duten klase nagusiak irudikatu diren. Klase bakoitza bere paketearen barruan kokatu da, bost direlarik aplikazioaren pakete edo geruza nagusiak: data, domain, view, viewmodel eta repository.

Hori horrela, aplikazioan parte hartzen duten klase nagusiak ondokoak dira:



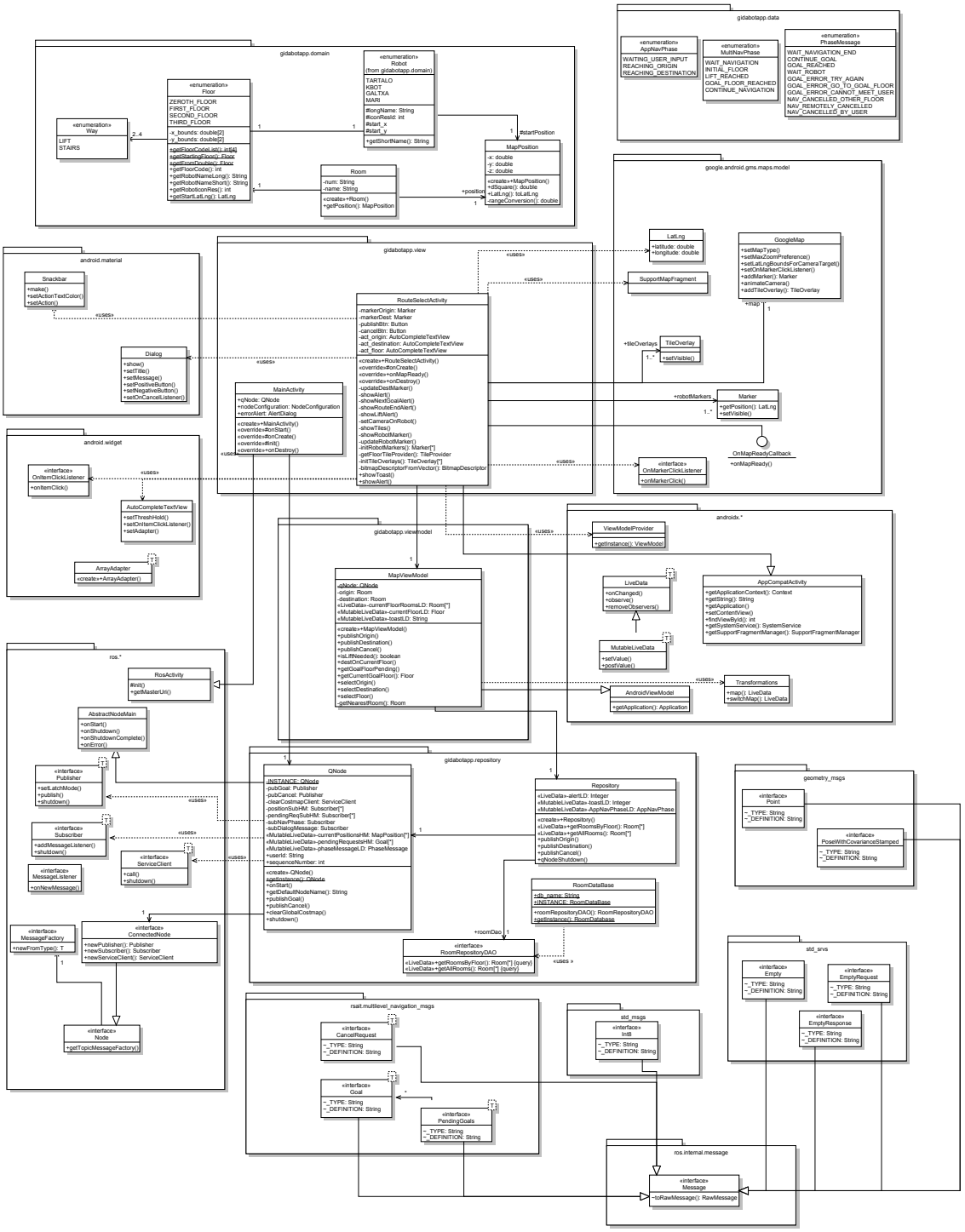
6.2 Irudia: Android bertsioen banaketa 2021ko maiatzan

- **MainActivity:** RosActivity klasea hedatzen duen bista. Klase honetan, QNode komunikazio-nodoaren bitartez ROS sistemarekin komunikazioa ezartzen da.
- **RouteSelectActivity:** Erabiltzaileak ibilbidea hautatuko duen bista, oro har aplikazioari funtzionalitatea ematen diona. Bertan, erabiltzaileak zehaztuko du non dagoen eta nora iritsi nahi duen, eta ibilbidea hasi zein ezeztatzeko aukera izango du.
- **MapViewModel:** Aplikazioaren egoera gordeko duen ViewModel geruza. Bertan, erabiltzaileak sartutako input oro gordeko da, eta Repository eta bista geruzen arteko zubi lana egingo du ere.
- **Repository:** Aplikazioari datu-geruzara sarbidea emango dion klasea. Honek ROS komunikazio-nodora zein datu-base iraunkorrekin elkar eragiteko metodoak ditu.
- **QNode:** ROS sistemarekin komunikazioa burutuko duen klasea. Bertan, robotari helburu bat zehazteko, bidaia hasteko eta bertan behera uzteko metodoak daude. Gainera, robotetik informazioa jasoko du, eta bere gaineko klaseei erakutsi *LiveData* moduan.
- **RoomDatabase:** Datu-basearen instantzia gordeko duen klasea.

- **RoomRepositoryDAO:** DAO interfazeak datu-basetik gelak lortzeko metodoak definitzen ditu.
- **AppNavPhase:** Aplikazioaren egoera adieraziko duen *enum* motako klasea. Aplikazioak hiru egoera posible izango ditu, erabiltzailearen datu-sarrera itxaroten (*WAITING_USER_INPUT)*, jatorrira iristen (*REACHING_ORIGIN*) eta *REACHING_-DESTINATION*, robota helmugara iristen ari denerako.

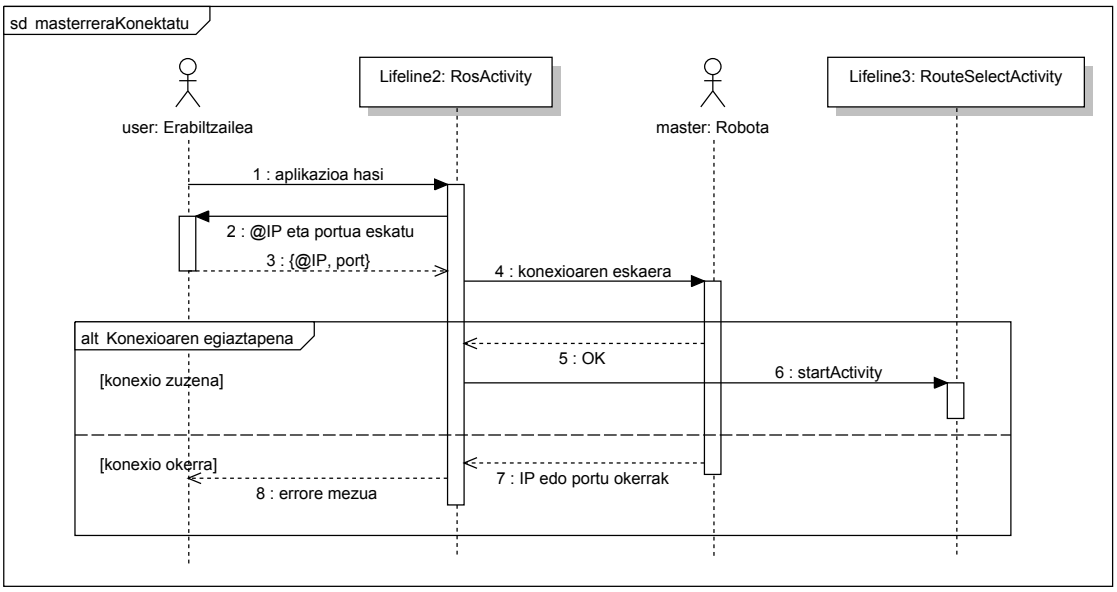
6.4 Sekuentzia diagramak

Definitu diren sekuentzia-diagramen bidez, erabilpen-kasu bakoitzean parte hartzen duten aktore eta aplikazioko osagaien arteko interakzio sekuentziala adierazi da. Modu honetan, behin diagramak osatuta, errazagoa izango da aplikazioaren kodea idaztea, aplikazioaren logikaren aldetik ikusita behintzat.

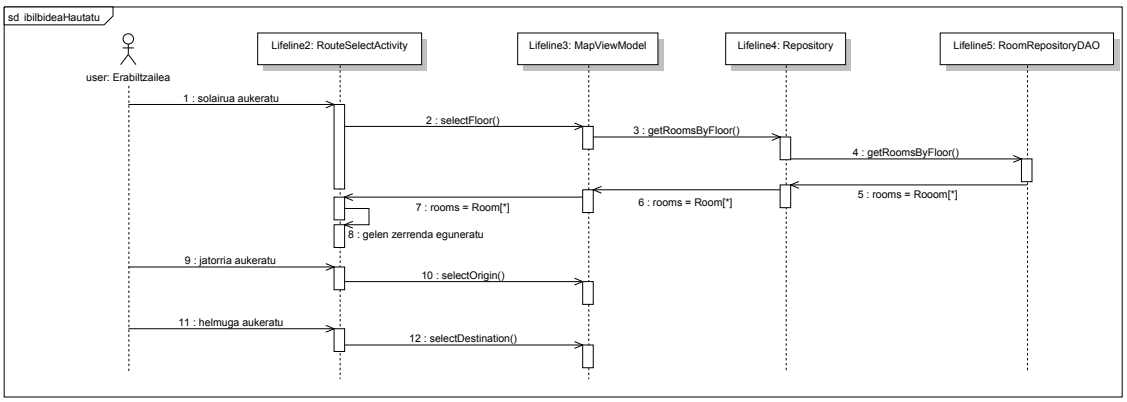


6.3 Irudia: Klase diagrama

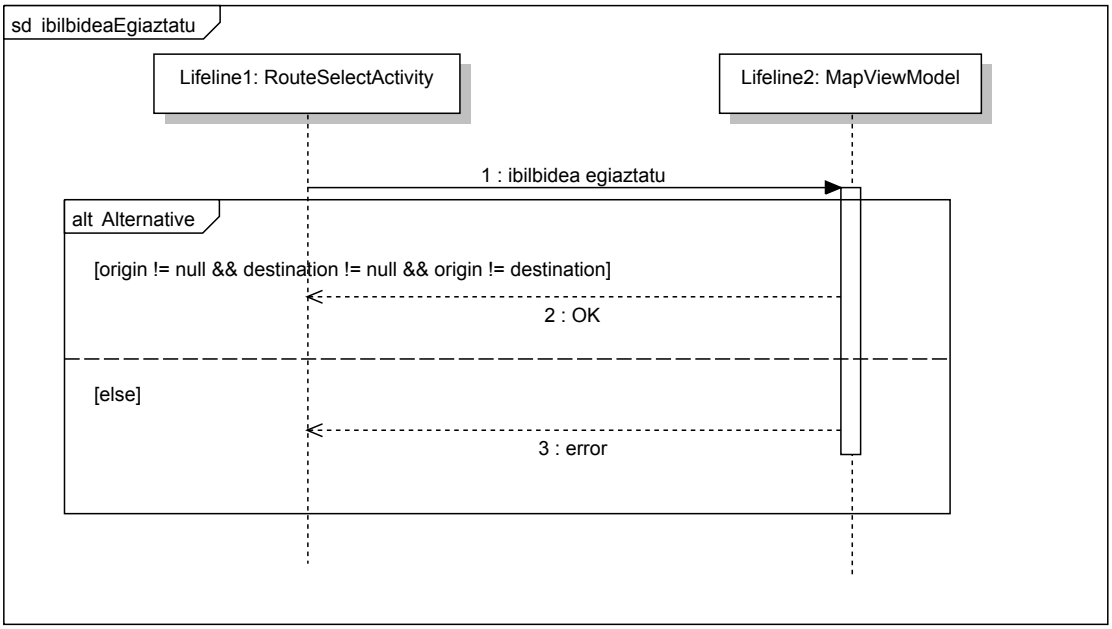
(Diagrama tamaina osoan esteka honetan)



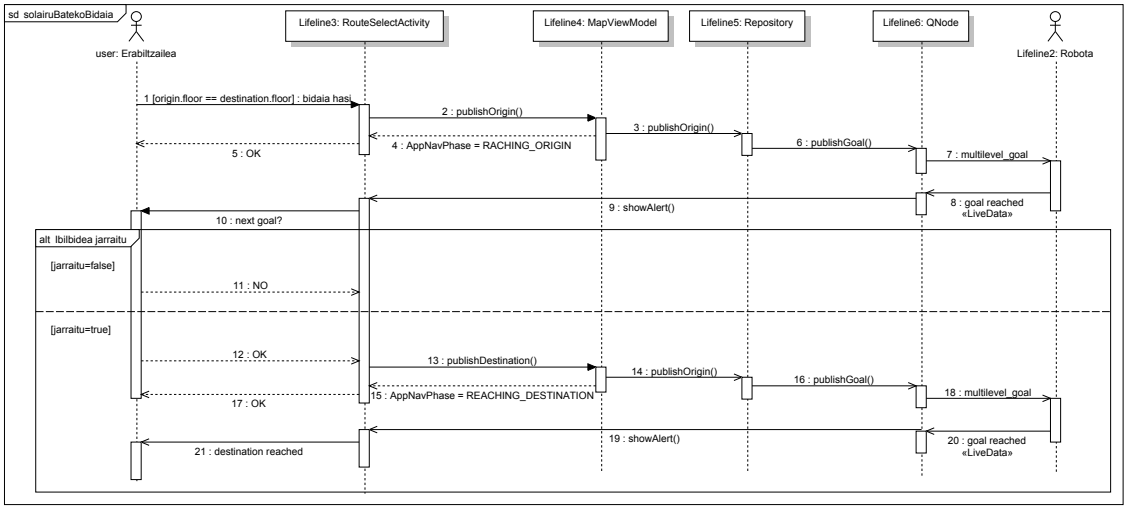
6.4 Irudia: Sekuentzia diagrama: ROS masterrera konektatu



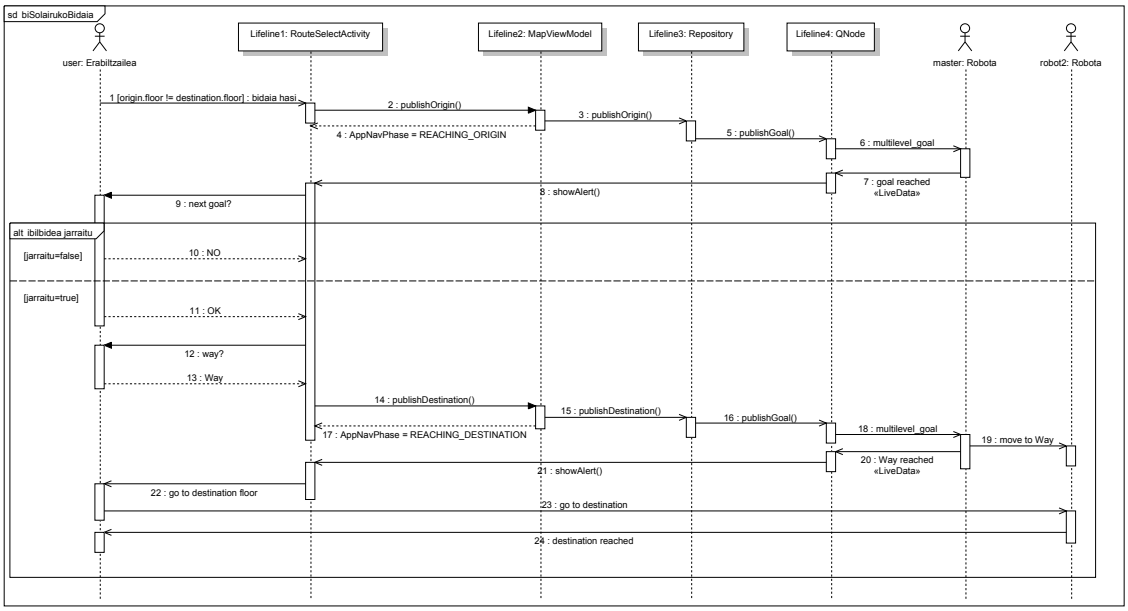
6.5 Irudia: Sekuentzia diagrama: ibilbidea hautatu



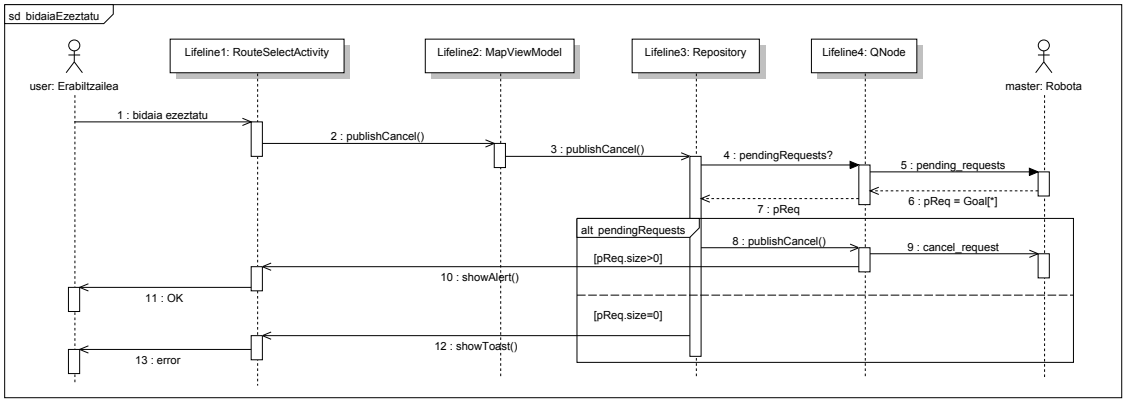
6.6 Irudia: Sekuentzia diagrama: ibilbidea egiaztatu



6.7 Irudia: Sekuentzia diagrama: Solairu bateko bidaia hasi



6.8 Irudia: Sekuentzia diagrama: Bi solairuko bidaia hasi



6.9 Irudia: Sekuentzia diagrama: Bidaia ezeztatu

(Tamaina osoko sekuentzia diagramak esteka honetan)

7. KAPITULUA

Soluzioaren garapena

Kapitulu honetan, diseinatutako soluzioaren inplementazioa kronologikoki azalduko da. Horretarako, garapen-iterazio bakoitzari atal bat esleituko zaio, eta iterazio bakoitzean azalduko da zein zen aplikazioaren egoera iterazioa hasi baino lehen, baita iterazioaren amaieran ere, bezeroaren *feedback*arekin batera. Gainera, hartutako erabakiak azaldu eta arrazoituko dira, baita garapenean zehar izandako arazoak aurkeztu ere.

7.1 GidaBot sistemaren egitura

GidaBotApp aplikazioaren inplementazioa ulertu ahal izateko, beharrezkoa da GidaBot sistemaren arkitektura eta funtzionamendua ulertzea. Horregatik, atal honetan GidaBot nabigazio-sistemaren egitura orokorra azalduko da, GidaBot-en osagai nagusiak aurkeztuz, eta GidaBot-eko roboten arteko mezuen transmisioa nola burutzen den azalduz.

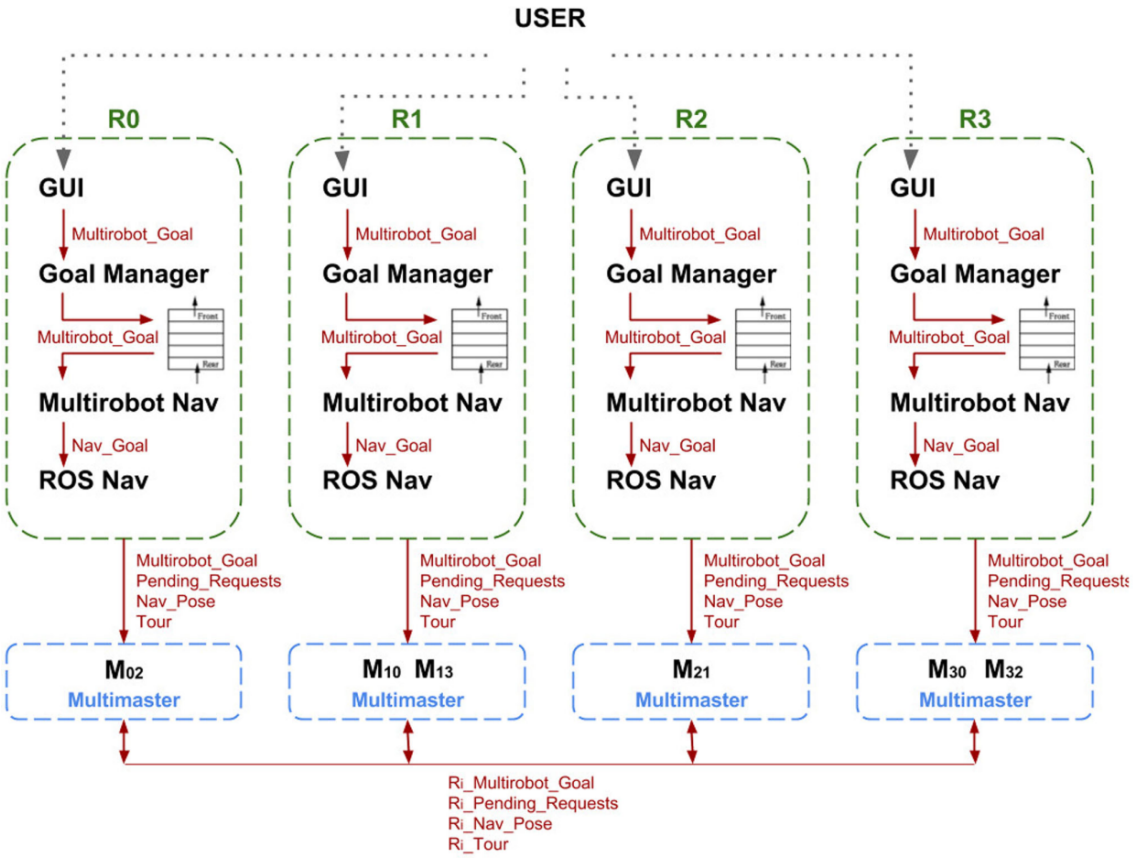
7.1.1 Mezuak eta mezuen transmisioa

Aurrekarien kapituluan aipatu den modura, GidaBot nabigazio-sistema elkar komunikatzen diren hainbat roboten bidezko nabigazio atazak lankidetzan burutzeko diseinatuta dago. Roboten komunikaziorako diseinatutako hurbilketa hori, informazio espezifikoaren hartu-emanean oinarritzen da.

Mezu horiek robot guztientzat eskuragarri egongo dira, eta ondoren, robot bakoitzak era-

bakiko du jasotako informazioa garrantzizkoa den, edo ez. Hala ere, ROS ez dago sistema banatuetarako diseinatuta, non informazioa entitate guztien artean partekatzea beharrezkoa den. Horregatik, *multimaster* paketeari esker, GidaBot sistemako robot bakoitzak beste robotetako *topic* eta zerbitzuak ikus ditzake (ikus topic eta zerbitzuak ROS eranskinean).

Modu horretan, sistemako robot bakoitzak bere ROS Masterra dauka, eta robot guztiak elkar konektatuta daude, latentzia baxuko mezuak transmititzeko aukera ematen duen komunikazio-grafo oso bat osatuz. Adibide modura, 7.1 irudian GidaBot sistemaren arkitektura azaltzen da lau robotekin. Aipatzekoa da, azpiko M_{ij} notazioak *multimaster*-a i eta j robotekin komunikatzen dela adierazten duela. *Multimaster* nodoaren bitartez bi robot konektatzeko, bi robotetako batean nodoa exekutatzea nahikoa da; ondorioz, n sistemako robot-kopurua izanik, $\frac{n(n-1)}{2}$ nodotan exekutatu beharko dugu *multimaster* nodoa. Beraz, irudian ikusten denez, 4 robot konektatzeko 6 *multimaster* beharko ditugu.

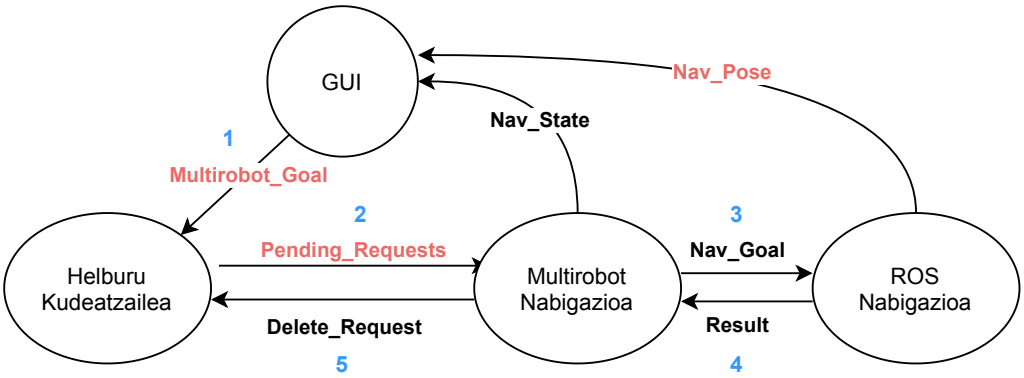


7.1 Irudia: GidaBot sistemaren multirobot arkitektura

7.1.2 GidaBot-en osagaiak

GidaBot-en robot anitzeko gidatze-sistemak lau osagai nagusi ditu:

1. Erabiltzailearen interfaze grafikoa (*GUI*a)
2. ROS nabigazio gunea
3. Robot anitzeko azpisistema, bi nodoz osatuta: multirobot nabigazio nodoa eta helburu-kudeatzailea
4. Komunikazio azpisistema



7.2 Irudia: GidaBot sistemako nodoen arteko komunikazioa

Zerrendatutako osagai guztiak GidaBot-eko robot bakoitzean exekutatzen dira. 7.2 irudian elementu guzti horien arteko komunikazioa adierazten da. Loturen alboko zenbakiek helburu bat prozesatzeko ordena izendatzen dute, hau da, nabigazioko nodoen arteko komunikazio-fluxu sekuentziala. Aipatzekoa da ere, gorri koloreko testuak roboten artean partekatzen diren mezuak adierazten dituela.

Erabiltzaile batek helmuga batera iritsi behar dela baieztatzen duenean, *multirobot_goal* bat sortu eta robot guztietara bidaltzen da. Ondoren, helburu-kudeatzaileek helburu hori beraien *pending_requests*-etara gehituko dute, baina soilik dagokion robotak ibilbidean parte hartu behar duenean. Gainera, multirobot nabigazio nodoak burutu gabeko eskaeren zerrenda etengabe konprobatzen duenez, zerrenda aldatzen den bakoitzean ROS nabigazio-prozesua aktibatuko du.

7.1.3 Proiektuan erabilitako *topic* eta zerbitzuak

Garatuko den aplikazioa GidaBot-eko robotekin komunikatzeko gai izateko, behar-beharrezkoa da zein mezu zein *topic*-era bidali behar dugun jakitea. Hori dela eta, 7.1 taulan GidaBot sistemako robotei aginduak bidaltzeko erabili beharko ditugun argitaratzaile, hartzaile eta zerbitzuak adierazten dira.

Mota	***Topic***	**Mezu mota**
Argitaratzailea (*Publisher*)	*/multilevel_goal* */cancel_request*	*Goal* *CancelRequest*
Hartzailea (*Subscriber*)	*/tartalo/amcl_pose* (*) */tartalo/pending_requests* (*) */dialog_qt_message*	*PoseWithCovarianceStamped* *PendingGoals* *Int8*
Zerbitzu-bezeroa (*Service Client*)	*/move_base/clear_costmaps*	*EmptyRequest*, *EmptyResponse*

7.1 Taula: Aplikazioak erabili beharko dituen *topic*-ak

(*) Posizioaren eta burutu gabeko eskaeren kasuan, robot bakoitzaren hartzaileak behar dira, hala nola, Marisorginen kasuan */mari/amcl_pose*, eta Kbotenean */kbot/amcl_pose*.

7.2 Lehenengo iterazioa

Proiektuaren inplementazioaren hasieran, ROS eta Android Studio proiektu batean erabiltzen ziren lehenengo aldia zenez, erabaki zen lehen iterazioan aplikazio sinple bat garatzea ROS eta Android sistemen arteko komunikazioa burutuko zuena.

Hori dela eta, lehen iterazioari dedikatu zitzaion denboraren zati handi bat, produktu finala garatzeari baino, formakuntzari esleitu zitzaion. Izan ere, garrantzitsua zen bi sistemen oinarriak ondo ulertzea, ondoren garatuko zen aplikazioaren eraikitze prozesuan zehar kontzeptuak argi izateko.

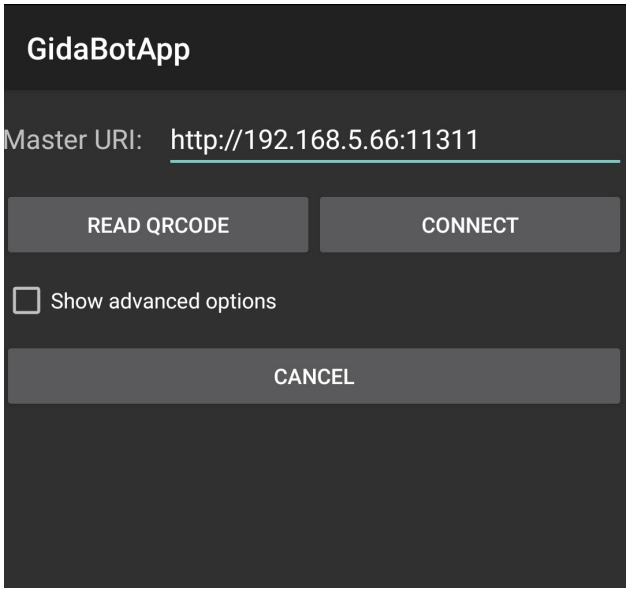
7.2.1 Pubsub

ROS eta Android arteko komunikazioa burutzeko informazioa bildu ostean, ikusi zen aukerarik zentzudunena *Rosjava* modulua erabiltzea zela, honek *android_core* modulua gehitzeko aukera ematen baitu, eta horren bidez Android eta ROSen arteko komunikazioa

burutzekoa. Beraz, ROS Wikiko tutorialetako pausuak jarraituz, *Rosjava* eta *android_core* moduluak instalatu ziren. [16][15]

Behin beharrezko paketeak instalatuta, ROS Wikiko tutorialean azaltzen zen *pubsub* eredua [12] erabili zen ROS eta Android sistemen arteko komunikazioa burutzeko. Software arkitekturaren munduan, *pubsub* deritzogu *publish-suscribe* edo argitaratzaile-harpidedun patroiari. Patroi honetan, mezuaren igorleak hartzaileari zuzenean mezuak bidali ordez, bidalitako mezuak hartzaileen ezagutzarik ez duten klaseetara bidaltzen ditu. Era berean, hartzaile edo harpidedunak klase bat edo gehiagorenganako interesa adierazten dute, eta soilik interesatzen zaizkien mezuak jasotzen dituzte, zein argitaratzaile dauden jakin gabe. [80]

Pubsub eredu hori Android Studion inportatzeak errore asko ekarri zituen. Izan ere, eredua ROS Indigo bertsiorako garatuta zegoen, eta azken *commit*a 2015 urtekoa zen. Geroztik Android Studiok zenbait gauza aldatu ditu, batez ere Gradle[51] bertsioa. Beraz, alternatiba baten bilaketan, pubsub eredu baten bertsio eguneratuago bat aurkitu zen, ROS Kinetic bertsiora eguneratuta zegoena, eta ondorioz, 2018 urtekoa.[13]



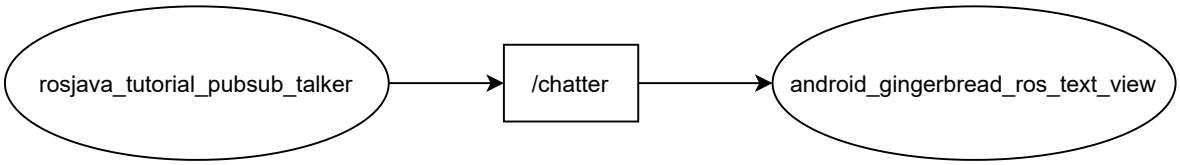
7.3 Irudia: ROS Masterrarekin konexioa zehazteko pantaila

Eredu berri hau Android Studion inportatu ondoren, proiektuaren Gradle fitxategiak sinkronizatuta, *RosActivity* klasea hedatzen duen *MainActivity* klasea izango dugu, ROS sistemari mezuak bidali eta jasotzeko gai izango dena.

MainActivity klaseak *RosActivity* klasea hedatzen duenez, aplikazioa exekutatzen denean, ROS Masterraren IP helbidea sartzeko eskatuko zaio erabiltzaileari (ikus 7.3 irudia). Sar-

tutako IP helbidea eta portuak zuzenak badira, aplikazioak ROS Masterrarekin konexioa hasiko du, eta *MainActivity*iko *init*() metodoa exekutatuko da. Metodo honek *NodeMainExecutor* motako objektua jasoko du, eta betearazle edo *executor* horri esker, Masterrera iristen diren mezuak entzun eta bidali ahalko ditugu.

Talker klasea gure pubsub-eko nodo argitaratzailea izango da. Bere *loop()* metodoa etengabe exekutatuko du, eta iterazio bakoitzean "Hello World!" mezu bat argitaratuko du */chatter* nodora, bere *Publisher*aren bitartez.



7.4 Irudia: Pubsub ereduaren komunikazio-grafoa

Ereduko mezuen hartzailea, ordea, *MainActivity*n izango dugu, eta */chatter topic*-era iristen diren mezuak *RosTextView* objektuan erakutsiko ditu, hau da, erabiltzailearen interfazean (7.4 irudian nodoen komunikazio-grafoa adierazten da).

Hortaz, behin gure aplikaziotik ROSekin komunikatzeko gai garela, daukagun pubsub eredu hau hedatu beharko dugu, GidaBot nabigazio-sistemak erabiltzen dituen mezu-motak jaso eta argitaratu ditzan.

7.2.2 QNode

C++ lengoaian idatzita dagoen GidaBot sistemak *QNode* izeneko nodoa erabiltzen du robotekin komunikatzeko, hau da, robotei mezuak bidali eta robotengandik mezuak jasotzeko. Nabigazio-sistema originalak erabiltzen duen klasea *rsait_common_packages/.../multilevel_navigation_qt/src/qnode.cpp* fitxategian erazagutzen da, eta komunikazioa burutzen du bertan sortzen diren *publisher* eta *subscriber*en instantziekin. Hauei esker, robotei mezuak bidaltzen zaizkie, eta mezu horien bidez adieraz diezaiokegu robotari zein helmugaraino mugitu behar duen. Era berean, robotak nabigazioaren inguruko informazioa bidaliko dio *QNode* nodoari, esaterako, momentuan zein posiziotan dagoen.

Hortaz, gure aplikazioan *QNode* klase honen Android bertsioa sortu beharko dugu ROSekin komunikazioa burutzeko, eta behin klase hori erazagututa, erabiltzaileari informazioa erakutsi *MainActivity* bistaren bitartez, honek robotari eskaerak burutu ahal izateko.

Aipatu bezala, Pubsub ereduan erabilitako *Talker* klasearen antzeko zerbait izango da gure *QNode*a, baina mezu gehiago bidaliko ditu, baita nodotik bertatik mezuak jaso ere.

Behin komunikazio-nodoak *topic*-etan mezuak egoki jaso eta bidaltzen dituela ziurtatuta, erabiltzailearen interfazea garatzeko ordua zen, honek aplikazioaren bidez eskaerak burutu ahal izateko era grafiko batean.

7.2.3 ListView

Lehenengo iterazio honetan ez zitzaion interfazeari garrantzi handirik eman. Izan ere, iterazio honen helburua bakar-bakarrik funtzionalitateak aztertzea zen, eta, beraz, ahalik eta interfaze sinpleena garatu zen. Ondorioz, nahikoa zen erabiltzaileak jatorri eta helmugako kokalekuak hautatzeko bi zerrenda jarri eta bidaia hasteko botoi batekin.

Hori horrela, *MainActivity* klasean bi *ListView* izango ditugu, eta *Button* motako botoi bat. *Room* klasearen pare bat instantzia sortu, zerrenda batean sartu, eta *ArrayAdapter*en bidez zerrenda bakoitzean kargatuko ditugu gelak. Gainera, *ListView* bakoitzari *OnItemClickListener* gertaera-entzule bat gehituz, jakingo dugu erabiltzaileak zein jatorri eta helmuga hautatu dituen.

Azkenik, “Hasi Bidaia”, hau da, *publishBtn* botoia sakatzean zehaztutako jatorria eta helmuga *qNode*-ra bidaliko dira (*publishGoal*() metodoaren bitartez). Ondoren, erabiltzaileari jakinaraziko ibilbidea argitaratu den edo ez, *Toast*[64] motako mezu bat erakutsiz.

7.2.4 XMLPullParser

Aplikazioaren helburu nagusia erabiltzaileak Informatika Fakultate barruan gidatzea da. Ondorioz, moduren batean Fakultateko gela nahiz toki esanguratsuenen guztien zerrenda erakutsi beharko zaio erabiltzaileari — hala nola, idazkaritza, kopistegia, eta atezaintza — berak zerrendatik jatorria eta helmuga hautatu ahal izateko.

GidaBot aplikazioak lekuen datuak *room_names.xml* fitxategitik lortzen ditu, non Fakultateko gela bakoitzarentzat *room* objektu bat dagoen. Bertan, gelaren solairua, zenbakia, izena eta posizioa adierazten dira, bakoitzari bere etiketa esleituaz.

```
1 <?xml version="1.0"?>
2 <rooms>
3     <room>
4         <floor>0</floor>
```

```
5         <num>000</num>
6         <name>Fakultateko sarrera nagusia</name>
7         <x>3.5503</x>
8         <y>-18.4937</y>
9         <yaw>1.5708</yaw>
10    </room>
11    <!-- ... -->
12 </rooms>
```

7.1 Kodea: *room_names.xml* fitxategia

Lehenengo iterazio honetan XML fitxategi hau erabiliko dugu gure aplikazioan gela guztiak kargatzeko, baina horretarako *XML Parser* bat beharko dugu. Eskuragarri dauden *Parserrak* behatuz, ikusiko dugu orokorrean bi parser-mota nagusi daudela: DOM parserrak batetik, eta SAX parserrak bestetik. Labur esanda, DOM parserrek XML dokumentuaren zuhaitzaren irudikapena memorian gordetzen du, eta horren bidez aplikazioari metodoak eskaintzen dizkio zuhaitzean zehar nabigatzeko. SAX motako parser APIek, aldiz, gertaerak erabiltzen dituzte; gertaera horiek maneiatuz (elementuen hasiera eta amaiera, esaterako) gure aplikazioa XML fitxategiko informazioa irakurtzeko gai izango da. Hori dela eta, SAX parserrek normalean ez dute XML dokumentu osoaren zuhaitzaren irudikapena memorian gordetzen. [42, *3-9 About DOM and SAX APIs*]

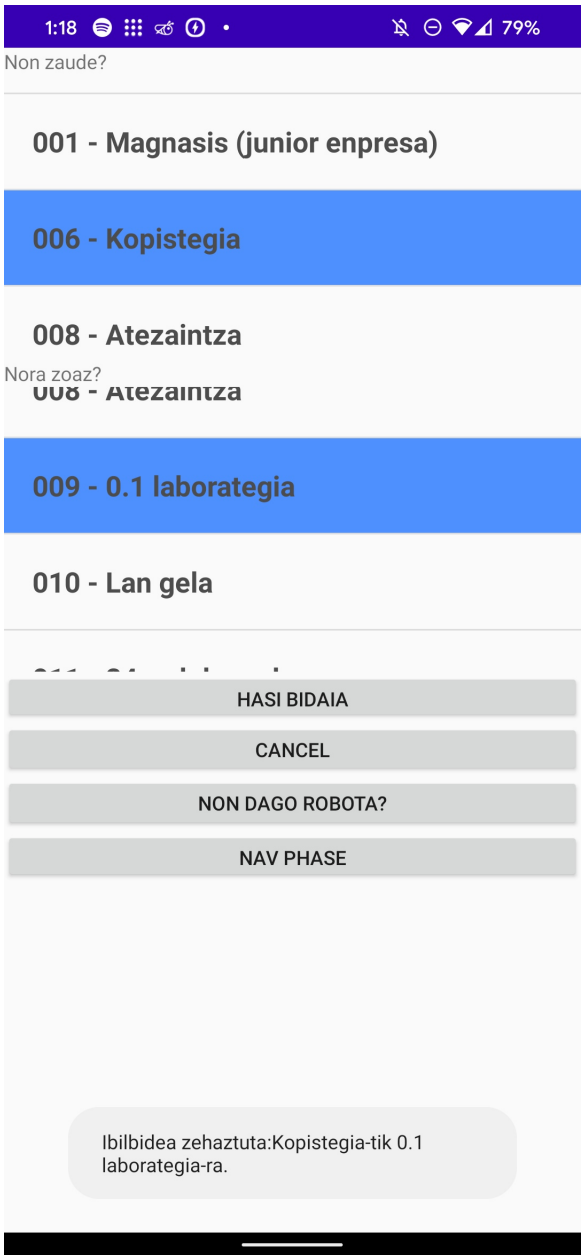
Hau guztia kontuan izanik, SAX parserra erabiliko dugu gelak aplikaziora kargatzeko, aipatu bezala honek ez baitu XML zuhaitz osoa memorian kargatuko. Hala eta guztiz ere, aplikazioaren azken helburua izango da gela guzti horiek datu-base iraunkor batera eramatea.

Berriz ere Android Developers dokumentaziora jotzen badugu, ikusiko dugu gida bat dagoela XML datuak analizatzeko edo *parse*atzeko, Javako *XMLPullParser* liburutegia erabiliaz.[58] Beraz, bertako argibideak jarraituz, gure XML dokumentutik gelak lortzeko parserra eraikiko dugu, *RoomXmlParser* klasea, hain zuzen.

7.2.5 Emangarria

Behin aurreko atazak amaitu eta probatuta, lehenengo iterazioaren helburuak asetuta zeuden: aplikazioak proiektuaren irismen minimoa gainditzen zuen. Hau da, aplikazioaren bidez erabiltzailea solairu bakar bateko helburu batera iristeko gai izatea, robota lehenik erabiltzailearen kokalekura (jatorrira) hurbilduz, eta ondoren hautatutako helmugara.

Lehenengo iterazioaren emaitza 7.5 irudian ikus daiteke[1].



7.5 Irudia: Lehenengo iterazioa. Emangarria

[1]Lehenengo iterazioaren emaitzaren iturri-kodea: https://github.com/i0su/gidabotApp/tree/it1_end

7.2.6 *Feedback*a

Lehenengo iterazioa amaitu ostean, bezeroari emangarria entregatu zitzaion, eta hauek izan ziren bezeroaren eskakizunak hurrengo iteraziorako:

- GidaBot-en bertsio originalaren interfaze antzeko bat sortzea aplikaziorako, hau da, pantaila bakar batean kontrol guztiak izango dituena.
- Interfaze grafikoan Fakultatearen mapa bat azaltzea, non robotaren uneko posizioa zein den adieraziko den, eta Fakultateko gela guztien kokapena azalduko den testu moduan. Gainera, ahal izanez gero mapan zoom egin ahal izateko aukera gehitzea.
- Aplikazioak arkitektura-patroiren bat jarraitzea.
- Aplikazioak gelen datuak datu-base iraunkor batetik hartzea.

7.3 Bigarren iterazioa

Behin bezeroak lehenengo iterazioko feedbacka emanda, bigarren iterazioan bezeroaren eskakizun berriak inplementatu ziren. Horretarako, noski, aurretik eskakizunen analisia egin zen, eta funtzionalitate berrien diseinua.

7.3.1 AndroidX

Duela urte batzuk, Android garatzaileek Android Support liburutegia erabiltzen zuten bertsio ezberdinetako gailuen arteko bateragarritasuna bermatzeko. Baina, Android API 28 bertsiotik aurrera (Android 9.0), *Android Jetpack*[49] liburutegi-suitea argitaratu zen, non *Support* bertsio berriagoak dauden, hainbat osagai berriagorekin batera (*Activity*, *Fragment*, edota *CameraX*, kasu). Bide batez, Android Jetpackek erabiltzen dituen artefaktu guztiak *AndroidX* izen-eremuaren barruan daude.

Beraz, gure aplikazioak MVVM arkitektura jarraitu ahal izateko, Android Jetpack edo AndroidX-en parte diren Androideko Arkitektura Osagaiak edo *Android Architecture Components* erabili beharko ditugu. Izan ere, Androiden arkitektura-bilduma liburutegi-multzo bat da, aplikazio sendo, testagarri eta mantenugarriak sortzea erraztuko diguna:[47]

- Androiden Bizi-zikloa ezagutzen duten osagaiak[55] erabiliko ditugu: *AppCompatActivity*.
- *ViewModel* klasean bista-geruzak beharko duen informazio garrantzitsua gordeko dugu: *keep the view dumb*.
- *LiveData* objektuei esker datu-basea aldatzen denean bista-geruza eguneratuko da.
- *Room* liburutegiak SQLiteren gaineko abstrakzio geruza bat eskainiko digu.

7.3.2 MVVM

Lehenengo iterazioan garatu zen *MainActivity* klaseak *RosActivity* klasea hedatzen zuen, eta *RosActivity* klaseak bere aldetik *Activity* klasea. Aldiz, hautatutako arkitektura inplementatzeko *LifecycleOwner* motako klase bat beharko dugu, oro har *Fragment* edota *AppCompatActivity* izango dena. Beraz, egoera honetan bi aukera genituen: *RosActivity* klaseak *AppCompatActivity* klasea hedatzea, edo *lifecycleowner* izango den beste activity bat sortzea, eta behin ROS Masterrera konexioa ezarrita (hots, horretarako behar dugulako *RosActivity*a), activity berri horretara bidaltzea fluxua.

Edonola ere, *RosActivity* klasea aldatzeko, konpilatutako klase bat zenez (*.class* fitxategia), jatorrizko klasea aldatu eta *android_core* modulua birkonpilatu beharko litzateke. Honen birkonpilazioa exekutatzerako orduan, konpilatzaileak hainbat errore ematen zituen, eta erabaki zen momentuz bigarren konponbidea hautatzea, eta aurrerago denborarik balego, arazo honi aurre egitea.

Singleton patroia

*RosActivity*aren konponbideari aurre egiterakoan, beste arazo bat aurkeztu zen: *QNode* klasea. Lehenengo iterazioko soluzioaren bidez, *QNode* klasea *MainActivity* klasean instantziatzen zen, baina ondoren ezin zuten beste klasek erabili. Haatik, gure programan *QNode* instantzia bakarrak existitu beharko luke, eta beste klase guztiek *QNode* instantzia berdina ikusi beharko lukete uneoro. Ondorioz, *QNode* klaseak *singleton* patroia jarraituko du.

```
1 //QNode.java
2 public class QNode extends AbstractNodeMain {
3     private static QNode INSTANCE = null;
4     //...
```

```
5       private QNode() {...}
6       public static synchronized QNode getInstance(){
7           if(INSTANCE == null){
8               INSTANCE = new QNode();
9           }
10          return INSTANCE;
11      }
12      //...
13  }
14
15  //MainActivity.java
16  public class MainActivity extends RosActivity {
17      private qNode qNode;
18      @Override
19      protected void onCreate(Bundle savedInstanceState) {
20          super.onCreate(savedInstanceState);
21          this.qNode = QNode.getInstance();
22      }
23      //...
24       protected void init(NodeMainExecutor nodeMainExecutor) {
25          nodeConfiguration = NodeConfiguration.newPublic(getRosHostname());
26          nodeConfiguration.setMasterUri(getMasterUri());
27          nodeMainExecutor.execute(qNode, nodeConfiguration);
28       }
29  }
```

7.2 Kodea: QNode, Singleton patroia inplementatuaz

Ikusten denez, *MainActivity* klasean *QNode* klasea instantziatzen da *onCreate*() gertaeran, eta ondoren *init*() metodoan *nodeMainExecutor*-aren bidez Masterrera konektatzen da. Modu honetan, jada *qNode* instantzia Masterrera konektatuta egongo da, eta beste edozein klasek ROSekin komunikatzeko aukera izango du, klasearen instantzia horren bitartez.

Room database

Aurreko iterazioan, *RoomXmlParser* klaseari esker Fakultateko gela guztiak kargatzen genituen gure aplikazioan. Iterazioaren amaieran, ordea, bezeroaren eskakizun berri bat egon zen: aplikazioak datu-base bat erabiltzea. Gainera, errendimendu aldetik ez zen oso eraginkorra aplikazioa exekutatzen zen bakoitzean XML dokumentu osoa parseatzea. Beraz, arazo horri aurre egiteko datu-base erlazional bat erabiltzea erabaki zen. Android aplikazioen kasuan, datu iraunkorrak modu erlazional batean gordetzeko hainbat aukera eskaintzen dira, beharren arabera.

Hots, gure aplikazioak erabiltzaile desberdinen artean partekatuko den datu-base zentral

bat behar badu, *Firebase*k[28] aukera emango digu hodeian datu-base bat kokatzeko. Baina, gure aplikazioaren kasuan bezala datu estatikoak gorde nahi baditugu, SQL datubase lokal bat erabiltzearekin nahikoa izango dugu. Hari beretik, *SQLite*[60] liburutegiak datu lokalak gordetzeko aukera emango digu, baina *Room*[59] liburutegiak *SQLite*ren gaineko abstrakzio maila bat eskainiko digu, konpilatze-fasean SQL kontsultak egiaztatuz, eta datu-base migrazioak zein anotazioen bidezko entitateen erazagupena eskainiz.

Ondorioz, Room liburutegia erabiliko da aplikazioko datu-base erlazionala sortu eta kudeatzeko. Aplikazioan *Room* liburutegia erabili ahal izateko, hiru osagai sortu beharko ditugu:

- Datu-basearen instantzia gordeko duen *RoomDatabase* klasea, datu-basearen konexioaren sarbide izango dena.
- Fakultateko gela bakoitza irudikatzen duen *Room* klasea, datu-entitate izango dena.
- *RoomRepositoryDAO* klasea, aplikazioari datu-baseko datuak kontsultatzeko metodoak eskainiko dizkiona.

```
1  // Room.java
2  @Entity (tableName = "rooms")
3  public class Room {
4      @PrimaryKey @NonNull
5      private final String num;
6
7      private final double floor;
8      private final String name;
9
10     @Embedded
11     private final MapPosition position;
12     // ...
13 }
14
15 // RoomDatabase.java
16 @Database(entities = {Room.class}, version = 1)
17 public abstract class RoomDatabase extends androidx.room.RoomDatabase {
18     private static final String db_name = "Rooms.db";
19     private static RoomDatabase INSTANCE;
20     public abstract RoomRepositoryDAO roomRepositoryDAO();
21
22     public static RoomDatabase getInstance(final Context context){
23         if(INSTANCE == null){
24             synchronized (Room.class) {
25                 if(INSTANCE == null){
26                     INSTANCE = androidx.room.Room.databaseBuilder(context,
27                             RoomDatabase.class, db_name)
```

```
                            .createFromAsset("database/"+db_name)
                            .build();
                    Log.i("DAO", "Database instantiated");
                }
            }
        }
        return INSTANCE;
    }
}

// RoomRepositoryDAO.java
@Dao
public interface RoomRepositoryDAO {
    @Insert
    void insertAll(List<Room> rooms);

    @Query("SELECT * FROM rooms WHERE floor = :floor")
    public LiveData<List<Room>> getRoomsByFloor(double floor);

    @Query("SELECT * FROM rooms")
    LiveData<List<Room>> getAllRooms();
}
```

7.3 Kodea: Room liburutegiarekin osatutako datu-basea

Ikusten denez, *RoomDatabase* klaseak *singleton* patroia jarraitzen du, *QNoderekin* gertatzen den modura, aplikazio osoan datu-base instantzia bakarra egongo baita.

Datu-basera gela guztiak sartzeko, *room_names.xml* fitxategitik gelak kargatu, eta *insertAll()* metodoarekin *rooms* taulan datu guztiak txertatu ziren. Behin osatuta, lortutako *Rooms.db* fitxategi informazioduna *assets/* karpetan gorde zen. Modu honetan, datu-basea instantziatzerakoan, fitxategi horretatik datu guztiak kargatuko dira, eta ez dugu XML fitxategia gehiago beharko.

ViewModel

Aurretik aipatu bezala, *ViewModel* geruzaren helburua izango da *view* edo ikuspegi-geruza ergel mantentzea, hau da, bertan informazio garrantzitsurik ez gordetzea.

Hori lortzeko, erabiltzaileak hautatzen dituen jatorri eta helmuga bistan gorde ordez, *MapViewModel* klase berrian gordeko ditugu, *selectOrigin()* eta *selectDestination()* metodoen bitartez, hurrenez hurren.

Gainera, datu-basetik lortutako informazioa bistaratzeko, kontuan izan beharko dugu datu- baseak *LiveData<>* objektuak erakusten dituela, eta hauek modu asinkronoan tratatu

beharko ditugula. *LiveData* objektu bat *observable* motako klase bat da, zeinek edozein motatako objektuak gordetzen dituen. Honek esan nahi du, *RouteSelectActivity* bistatik *ViewModel*ak eskaintzen dizkigun *LiveData* objektuak behatuko (*observe*) ditugula, eta hauek aldatzen direnean, ekintzaren bat burutu. [2]

Hiru geruzen arteko interakzioa hobeto ulertzeko, azter dezagun erabiltzaileak solairua hautatzen duen erabilpen-kasua:

```
1  //RouteSelectActivity.java
2  public class RouteSelectActivity extends AppCompatActivity{
3      private MapViewModel viewmodel;
4      protected void onCreate(Bundle savedInstanceState) {
5          super.onCreate(savedInstanceState);
6          setContentView(R.layout.activity_route_select);
7
8          viewModel = new ViewModelProvider(this, ViewModelProvider.AndroidViewModelFactory.
       getInstance(this.getApplication())).get(MapViewModel.class);
9          //...
10         viewModel.getCurrentFloorRooms().observe(this, new Observer<List<Room>>() {
11             @Override
12             public void onChanged(List<Room> rooms) {
13                 ArrayAdapter<Room> adapterRooms = new ArrayAdapter<>(getApplicationContext(), R.
       layout.spinner_item, R.id.spinnerText, rooms);
14                 spinnerNon.setAdapter(adapterRooms);
15                 spinnerNora.setAdapter(adapterRooms);
16             }
17         });
18     }
19     //...
20     spinnerFloor = findViewById(R.id.spinnerFloor);
21     //...
22     spinnerFloor.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
23             @Override
24             public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
25                 viewModel.selectFloor(position);
26             }
27             //...
28         }
29
30 // MapViewModel.java
31 public class MapViewModel extends AndroidViewModel {
32     private RoomRepository roomRepository;
33     private final LiveData<List<Room>> currentFloorRooms;
34     //...
35     public MapViewModel(@NonNull Application application) {
36         roomRepository = new RoomRepository(application.getApplicationContext());
37         //...
38         this.currentFloorRooms = Transformations.switchMap(currentFloor, new Function<Floor,
       LiveData<List<Room>>>() {
39             @Override
```

```
40                  public LiveData<List<Room>> apply(Floor floor) {
41                      return roomRepository.getRoomsByFloor(floor);
42                  }
43          });
44          //...
45      }
46      //...
47  }
```

7.4 Kodea: Aplikazioaren komunikazioa LiveData objektuen bidez

Jatorriko gela beti erabiltzaileak hautatzen duen solairuan egongo da, eta beraz, jatorria hautatzeko gelen zerrendan soilik uneko solairuko gelak erakutsi beharko ditugu. Horretarako, erabiltzaileak solairu bat hautatzen duenean, *viewModel*ean solairu hori hautatzen da $selectFloor()$ setterraren bidez. *ViewModel*ean, bere aldetik, *Transformations*[65] klasearen $switchMap()$ metodoaren bitartez, solairua aldatzen den bakoitzean, *Repository*tik (hau da, datu-basetik), solairu horri dagozkion gelak lortzen dira. Azkenik, bistatik *LiveData* den *currentFloorRooms* objektua behatuko da, eta hau aldatzen den bakoitzean jatorriko gela hautatzeko zerrenda eguneratuko da.

MVVM arkitekturaren portaera honi esker lortzen dena, erabiltzaileak solairu bat hautatzen duen bakoitzean, solairu horri dagozkion gelak datu-basetik zerrendara kargatzea modu asinkrono batean. Gainera, prozesu osoan aurkezpen geruzan ez da informazio kritikorik gorde, eta horrekin lortu dugu erabiltzaileak edozein unetan aplikazioa utziko balu, bueltatzean gure sistemak operatibo jarraitzea.

7.3.3 Mapa

Fakultatearen mapa eraikitzeko, hainbat estrategia desberdin baloratu ziren. Horietako bat, fakultateko planoa irudi modura kargatzea zen, ondoren irudi horren gainean roboten irudiak erakustea, eta roboten irudi horiek denbora errealean eguneratzea. Bestalde, beste aukera bat zen mapak erakusteko liburutegiren bat erabiltzea, eta horren gainean fakultateko mapa zein robotak gainjartzea. Azken aukera hau zentzudunena iruditu zitzaigun, irudiekin lan egiteak beheko mailako funtzio gehiago hasieratik programatzea eskatuko lukeelako.

Aplikazioan mapak erakusteko liburutegien artean, Google-ren *Maps SDK for Android*[22] erabiltzea erabaki zen, aplikazioaren beharrentzat egokien moldatzen zena baitzen. Beste aukera batzuen artean, ordainpekoa den MapBox[40] genuen, baita OpenStreetMapen Android bertsioak [11] ere, baina azken hauek ez zituzten gure beharrak guztiz asetzen.

Google Maps SDK

Aipatu bezala, Google-ren *Maps SDK for Android* tresnak aplikazioaren behar guztiak asetzen zituen. Batetik, Fakultateko planoen irudia mapan gainjartzeko aukera ematen du, eta gainera mapa horretan *Marker*ak gehitzeko aukera eskaintzen du, roboten denbora-errealeko posizioa adierazteko. Are gehiago, mapa *fragment* baten barruan sartzeko aukera eskaintzen du, eta horren bidez aplikazioaren kontrol guztiak pantaila berean izan ahalko ditugu, hau da, mapa, jatorria eta helmuga hautatzeko zerrendak, eta bidaia hasi eta ezeztatzeko botoiak.

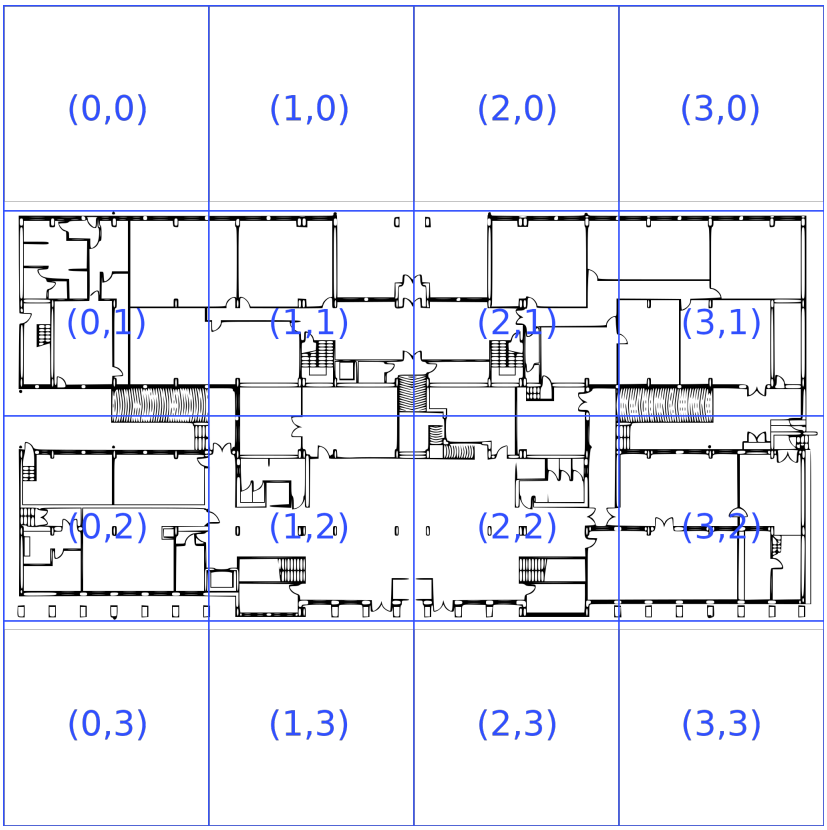
Liburutegia erabili ahal izateko, Google Cloud plataforman *API KEY* edo API-gako bat eskatu beharko dugu. Erabileraren arabera, gakoa eskatzeko ordainpeko plan[45] bat kontratatu beharko dugu, batez ere Googleren mapa-datuak behar badugu, edota *streetview* modua erabiliko badugu. Hau ez da gure kasua, eta beraz, gure erabilerarako nahikoa izango dugu doako bertsioarekin.

Tile Overlay

Fakultateko planoa mapan gainjartzeko, liburutegiak *TileOverlay* klasea eskaintzen digu. Honen bitartez, mapa huts bat sortu ahalko dugu, eta solairu bakoitzaren planoaren irudia mapan bertan ezarri. Maparen gainean zooma erabili ahal izateko, zoom maila bakoitzeko irudia lauki-multzo bateko sareta batean kargatzen du, eta mapa bat beste zoom maila batera mugitzen denean, APIak zehazten du zein ataza bete behar diren zoom horretako irudiak lortzeko. [21]

Zoomaren 0. mailan, mundu osoa kargatuko da irudi bakarrean, zoom 1. mailan $2x2$ tamainako sareta izango dugu, eta 2. mailan $4x4$koa. Hortaz, zoom maila bakoitzean, saretaren tamaina $2^z x 2^z$-koa izango da, non z zoom maila den. Gainera, ipar-mendebaldeko karratua beti $(0,0)$ koordenatua izango dugu, eta x balioa mendebaldetik ekialdera handituko da, eta y balioa, aldiz, iparraldetik hegoaldera. Adibide modura, 7.6.irudian zoom 2. mailako sareta adierazten da, non 16 lauki dauden.

Badaukagu zoom bakoitzeko irudia nola kargatu mapan, baina orain planoa zati horietan banatzea falta zaigu. Horretarako, irudia lauki-multzo desberdinetan zatitu beharko dugu, hau da, adibidean adierazi den bezala planoa 16 zatitan banatu (zoom 2. mailaren kasuan). Hau eskuz egiteak denbora gehiegi hartuko liguke, eta aukera onena zatiketa hori script baten bidez egitea izango litzateke. Horren haritik, egileak esker onak eman nahi dizkio



7.6 Irudia: Beheko solairuko maparen sareta, zoom 2. mailan

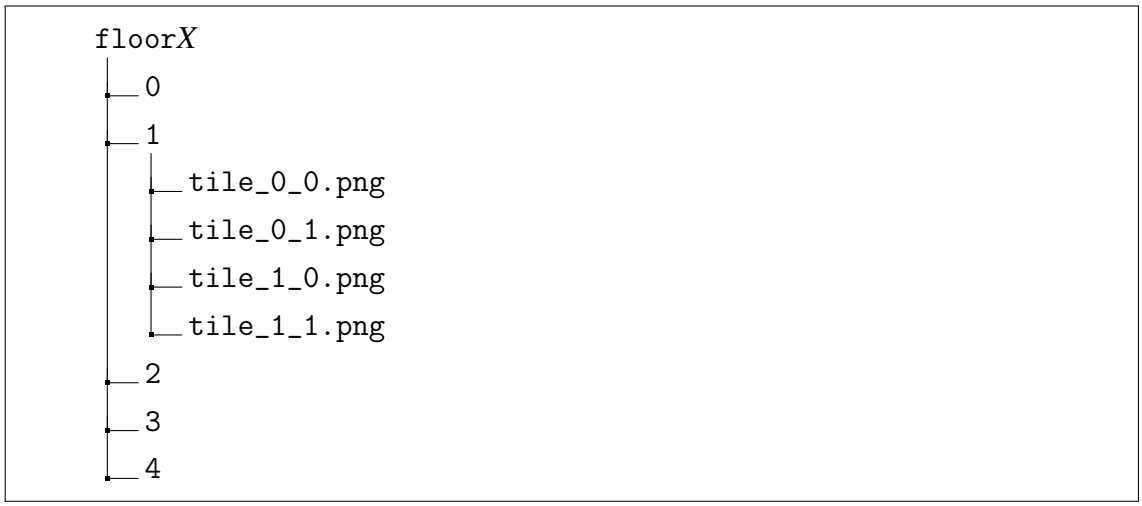
github/jahed erabiltzaileari, berak sortutako *maptiles*[1] scriptak gure aplikazioaren behar horiek asetu baitzituen.

Hortaz, *maptiles script*a Fakultateko solairu bakoitzaren irudian exekutatuko dugu, eta exekuzioa amaitzean solairu bakoitzaren karpetak 7.7. irudiko egitura izango du. Bertan, azpi-karpeta bakoitzak zoom-maila bat adieraziko du. Fitxategi horiekin, mapak zoomaren mailaren arabera azpikarpeta batetik edo bestetik hartuko ditu.

Markers

Robotaren denbora-errealeko posizioa mapan erakusteko, *Maps SDK* liburutegiak eskuragarri duen *Marker* klasea erabiliko da. *Marker*ek edo markatzaileek mapako kokaleku bakarrak adierazten dituzte. Gure mapako markatzaileei posizioa alda diezaiokegu, baita erakusten duten ikonoa ere. Gainera, markerrek *InfoWindow* izeneko leiho bat daukate, non objektuaren inguruko informazio gehigarria erakusten duten. [20]

GidaBot sistemako roboten uneko posizioa erakusteaz gain, markerren bidez Fakultateko



7.7 Irudia: Solairu bateko sareta-fitxategien egitura

gelen izenak ere erakutsiko dira. Izan ere, erabiltzaileak soilik hutsa ikusiko balu soilik, beharbada berarentzat zaila izango litzateke ingurunean kokatzea, batez ere Fakultatea ezezaguna izanik. Hortaz, *Marker* bat sortuko dugu gela bakoitzaren izenarekin, eta dagokion posizioan kokatu.

Mapako posizioaren doiketa

Hala ere, GidaBot sistemako posizioa gure aplikazioan zehatz erakutsi ahal izateko, posizioen balioak doitu beharko dira. Izan ere, *Maps SDK* liburutegiko mapako posizioak latitude eta longitudearekin adierazten baitira, eta GidaBot sisteman, ordea, x eta y koordenatuekin.

Hori horrela, posizioak (x, y) koordenatuetatik koordenatu geografikoetara pasatzeko, hau da, (lat, lng), bi adierazpen-sistemen mugak jakin behar dira. Hortaz, GidaBot sistemako x eta y mugak jakin beharko ditugu, baita mapako latitude eta longitude mugak ere.

Latitude eta longitudeari dagokionez, maparen gainean hainbat proba egin ondoren, ondorioztatu zen latitudearen balioak $latBounds = [-65, 65]$ artean zeudela, eta longitudearenak, aldiz, $longBounds = [-180, 180]$ artean. Latitudearen balioak muga geografikoak baino txikiagoak dira (-90 eta 90), planoa laukizuzena denez planoaren altuera maximoa mugaren azpitik dagoelako.

Ordea, GidaBot-en X eta Y koordenatuen mugak jakiteko, balio zehatzak lortzea ezinezkoa zen (hau da, ez zegoen muga horiek zehazten zituen fitxategi edo baliorik), eta beraz, zuzendariekin kontsultatu ostean, erabaki zen gutxi gorabeherako balioak hartzea

Rviz-etik. Hori dela eta, beheko solairuko (eta ostean, gainontzeko solairuetako) robotaren mapako posizioa ez da guztiz zehatza izango: beheko solairuan, x ardatzaren tarteak $xBounds \approx [-30, 37.5]$ dira, eta y ardatzarenak $yBounds \approx [-21.6, 9.3]$; latitude eta longitudeak lortzeko bi tarteen arteko heinaren bihurketa bat egingo dugu (*rangeConversion*):

$$longitude = rangeConversion(xBounds, longBounds, x) \tag{7.1}$$

$$latitude = rangeConversion(yBounds, latBounds, y) \tag{7.2}$$

7.3.4 Emangarria

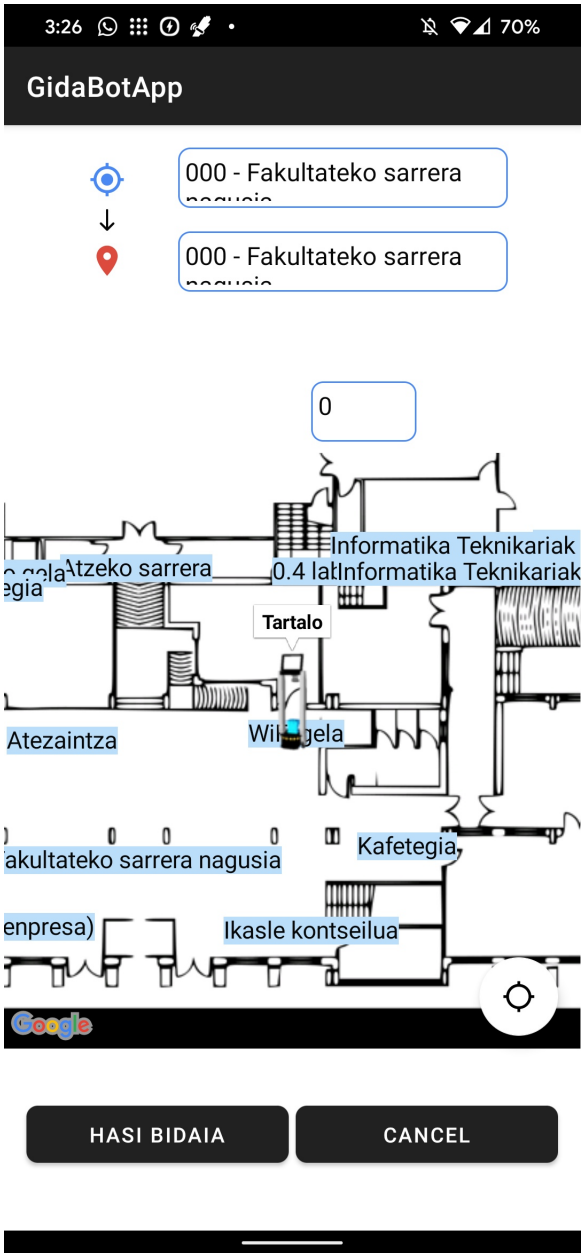
Aurreko atazak amaitu eta probatu ostean, bigarren iterazioaren amaierara iritsi zen proiektua. Izan ere, bezeroaren eskakizun berriak aseturik zeuden, eta ondorioz, bezeroaren *feedback*a jasotzeko ordua zen. 7.8 irudian bigarren iterazioaren emaitza ikus daiteke[2].

7.3.5 *Feedback*a

Bigarren iterazioko emaitza bezeroari entregatu ostean, hauek izan ziren bezeroaren betekizunak hurrengo iteraziorako:

- Aplikazioak bi solairu desberdineko deiekin funtzionatzea. Hau asmo handieneko helburua zen, eta ez genekien ziur lor zitekeenik.
- Gelen etiketen posizioa aldatzea mapan. Bigarren iterazioan gelen izena gela bakoitzaren posizioan erakusten zen, eta zenbait gelen etiketek elkar gainjartzen zuten.
- *Drop-down list*etatik, hau da, gelen zerrendetatik jatorri eta helburua hautatzerakoan, gelaren izena eskuz idazteko aukera egotea (bilatzaile moduko batekin). Ginera, *dropdown-list* bakoitzaren alboan etiketa bat azaltzea, erabiltzaileak jakiteko zein eremuri dagokion (hots, jatorria, helmuga edota solairua).
- Helburuak bi fasetan bidaltzea. Hau da, lehenik erabiltzailearen jatorria bidaltzea robotari, eta robota erabiltzailearengana iristean, galdetzea ea hautatutako helmugara joan nahi duen.

[2]Bigarren iterazioko emaitzaren iturri-kodea: https://github.com/i0su/gidabotApp/tree/it2_end



7.8 Irudia: Bigarren iterazioa. Emangarria

- Maparen korritze-horizontala blokeatzea mapa amaitzean. Hau da, bigarren iterazioko emangarrian mapak korritze infinitua onartzen zuen; ezker edo eskuin aldeetara etengabe mugi zitekeen, nahiz eta mapa amaitu.

7.4 Hirugarren iterazioa

Bigarren iterazioaren inguruko *feedback*a jaso ondoren, hirugarren eta azken iterazioarekin hasteko ordua zen. Bagenituen bezeroak ezarritako eskakizun berriak, eta horien arabera, azkeneko iterazio honen helburu nagusia izango zen aplikazioaren bitartez erabiltzailea bi solairuetako bidaiak egiteko gai izatea.

7.4.1 Aldaketak domeinuan

Aplikazioaren diseinu-fasean, pentsatu zen gela bakoitzaren solairua adierazteko *floor (double)* atributu batekin nahikoa izango zela. Dena den, inplementazioan zehar ohartu ginen solairu bakoitzarekin erlazionatutako datuak egon bazeudela (hots, solairu bakoitzaren x eta y mugak). Ondorioz, erabaki zen solairu bakoitza *Enum* batekin adieraztea, eta solairu bakoitzarekin $1:1$ erlazioa izango zuen *Robot* entitatea ere definitzea. Modu honetan, kodea ulergarriagoa izango zen, eta, esaterako, robot bakoitzaren irudia *ViewModel* klasean gorde ordez, *Robot* klase horretan gordeko zen.

Egia da, aldaketa hauen beharrak, esan nahi zuela hasierako diseinua ez zela guztiz egokia izan. Ordea, kontuan izan behar da, garapen-inguruneak egilearentzat guztiz berriak izanik, nahiz eta egileak objektuei orientatutako aplikazioak garatzen esperientzia izan, oso zaila zela aplikazioa hasieratik ondo diseinatzea.

Horrekin guztiarekin, aldaketen ostean honako *Floor* eta *Robot* klaseak genituen gure aplikazioan:

```
1  // Floor.java
2  public enum Floor {
3      ZEROTH_FLOOR    (TARTALO, -30,    37.5,  -21.6,   9.3),
4      FIRST_FLOOR     (KBOT,    -60.82, 2.38,  -1.82,  28.42),
5      SECOND_FLOOR    (GALTXA,  -14.61, 47.0,  -41.91,-17.34),
6      THIRD_FLOOR     (MARI,    -8.6,   54.25, -14.6,  14.27)
7      ;
8      private final Robot robot;
9      private final double[] x_bounds;
10     private final double[] y_bounds;
```

```
11
12      private static final List<String> CODES;
13      static {
14          CODES = new ArrayList<>();
15          for(Floor f: Floor.values()){
16              CODES.add(String.valueOf(f.ordinal()));
17          }
18      }
19
20      Floor(Robot robot, double x_min, double x_max, double y_min, double y_max)
21      {
22          this.robot = robot;
23          this.x_bounds = new double[]{x_min,x_max};
24          this.y_bounds = new double[]{y_min,y_max};
25      }
26
27      public LatLng getStartLatLng(){
28          return new MapPosition(robot.start_x, robot.start_y, 0).toLatLng(this);
29      }
30      public static List<String> getFloorCodeList(){
31          return Collections.unmodifiableList(CODES);
32      }
33      //...
34  }
35
36  // Robot.java
37  public enum Robot{
38      // StartPoint => 000 - Fakultateko Sarrera Nagusia
39      TARTALO ("Tartalo",     R.drawable.tartalo_small,3.5503, -18.4937),
40      // StartPoint => 122 - Dekanotza
41      KBOT    ("Kbot",        R.drawable.kbot_small,  -16.2700, 4.42812),
42      // StartPoint => Ezkerreko eskailerak (2nd floor)
43      GALTXA  ("Galtxagorri", R.drawable.galtxa_small, 5.2744, -35.9580),
44      // StartPoint => 302 - DCI, Egokituz
45      MARI    ("Marisorgin",  R.drawable.mari_small,  -0.3069, -8.7494)
46      ;
47      protected String longName;
48      protected int iconResId;
49      protected double start_x, start_y;
50
51      Robot(String longName, int iconResId, double start_x, double start_y){...}
52
53      protected String getShortName(){
54          return this.name().toLowerCase();
55      }
56  }
```

7.5 Kodea: Floor eta Robot klaseak, domeinuaren aldaketen ostean

Aipatzekoa da, robot bakoitzaren ikonoa *resource id*[46] modura gordetzen dela. Izan ere, Android aplikazioetan *res/* karpetan gordetzen ditugun baliabideak lortzeko, kontestua

edo *Context*[52] klasea erabiltzen da. Beraz, kontestua normalean bista-geruzak soilik izango duenez, hau da, *lifecycleowner*-ak, bertatik lortuko da *id* horri dagokion irudia *BitmapDescriptorFactory.fromResource(id)* metodoa erabiliaz.

Era berean, aplikazioak pantailaratzen dituen string guztiak *res/values/strings.xml* fitxategian definitu dira, eta testuren bat pantailaratu behar den bakoitzean, *Context.getString()* metodoaren bidez *id* horri dagokion testua lortuko da.

7.4.2 Multilevel

Aplikazioaren solairu bakarreko bertsioan, robotaren posizio bakarra pantailaratu behar zuen aplikazioak. Erabiltzaileak bi solairuko bidaiak egiteko, ordea, robot guztien posizioak jaso beharko dira ROS Masterretik, eta beraz, *QNode* komunikazio-nodoan hartzaile edo listener berri bat definitu beharko dugu robot bakoitzeko. Hori horrela, kodea laburragoa eta errepikakorra ez izateko, *HashMap* egitura bat sortu da, non indizeak solairuak diren (*Floor*), eta solairu bakoitzari dagokion robotaren posizio-listenerra definitzen den.

Hortaz, bista-geruzatik (*RouteSelectActivity*) *qNode*an definitu den *positionSubHM HashMap*-eko *MutableLiveData* bakoitza behatuz (*observe*), mapan robot bakoitzaren denbora-errealeko posizioa erakutsi ahalko da. Hari beretik, maparen lehenengo bertsioan Tartalo bakarrik erakusten zenez (beheko solairuko robota), bistan ere robot bakoitzaren posizioaren *Marker*-arentzat *HashMap* bat izango dugu, baina kasu honetan *HashMap<Floor, Marker>* motakoa, *QNode*tik jasotako posizioen arabera eguneratuko dena.

7.4.3 Material Design

Behin aplikazioaren funtzionalitate nagusiek egoki funtzionatzen zutela egiaztatuta, erabiltzailearen interfazeari edo *GUI*ari ukitu hobe bat ematea erabaki zen, horretarako *Material Design* estiloa jarraituz. *Material* Googlek sortutako diseinu-sistema da, kalitatezko Android, IOS, Flutter eta web aplikazioak sortzeko prozesua errazteko. [24]

Android aplikazioetan Material Design estiloa jarraitu ahal izateko, Android Jetpack suitearen parte diren Material Design Osagaiak edo *Material Design Components*-ak[23] eskuragarri daude, zeintzuk gure aplikazioari ukitu garbiago eta modernoago bat emango dioten.

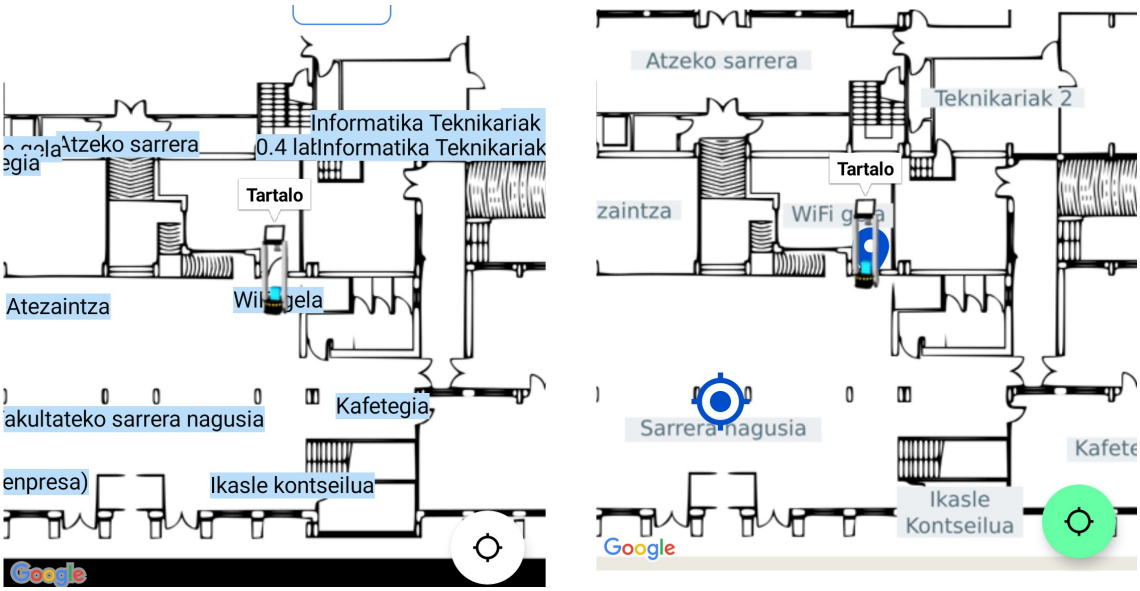
Gainera, aurretik erabiltzen ziren *Spinner*ak[62] *Dropdown menu*ekin[19] ordezkatuz, be-

zeroaren bi eskakizun berri beteko dira, bata jatorri edo helburua testu bidez sartu ahal izatea, eta eremu bakoitzaren alboan etiketa bat erakustearena bestea.

7.4.4 Maparen etiketak aldatzea

Bigarren iterazioko eskakizun berrietako bat izan zen, mapan erakusten ziren gelen etiketen posizioa aldatzea. Hortaz, mapako gelen etiketak bakoitzaren koordenatuetan erakutsi ordez, GidaBot sisteman erabiltzen diren moduko etiketak erabiltzea eskatzen zen, hau da, gela bakoitzaren barruan, etiketak elkar ez gainjartzeko.

Hori lortzeko, aukera bat zen orain arte egiten zen bezala gela guztien etiketak Marker baten bidez adieraztea, markatzaile horiek exekuzio-hasieran sortzen direlarik. Beraz, aldaketa horiek inplementatzeko etiketen posizio berriak XML edo datu-baseko taula berri batean gorde beharko lirateke, eta inplementazio-denbora gehiegi hartuko luke horrek. Ondorioz, erabaki zen gelen etiketak mapako planoetako irudietan zuzenean jartzea. Izan ere, gelen posizioak estatikoak izanik, ez zuen zentzu handirik aplikazioa exekutatzen zen bakoitzean prozesatze-denbora xahutzeak etiketa horiek sortzen. Adibide gisa, 7.9. irudian ikus daiteke etiketen aldaketa beheko solairuan.



7.9 Irudia: Bigarren eta hirugarren iterazioetako etiketen alderaketa

7.4.5 *APK*ren sorrera eta azken aldaketak

Lehen iterazioetan aplikazioa Android Studioko debuggerraren bitartez probatzen zen, hau da, gailu fisikoa ordenagailura konektatuz, eta aplikazioa zuzenean exekutatuz. Dakigunez, Android aplikazio gehienak APK (*Android Application Package*) formatuan zabaltzen dira[3], eta beraz, azken pausua izango da aplikazio-paketea sortzea, azken probak gailua ordenagailura konektatuta egon gabe egin ahal izateko.

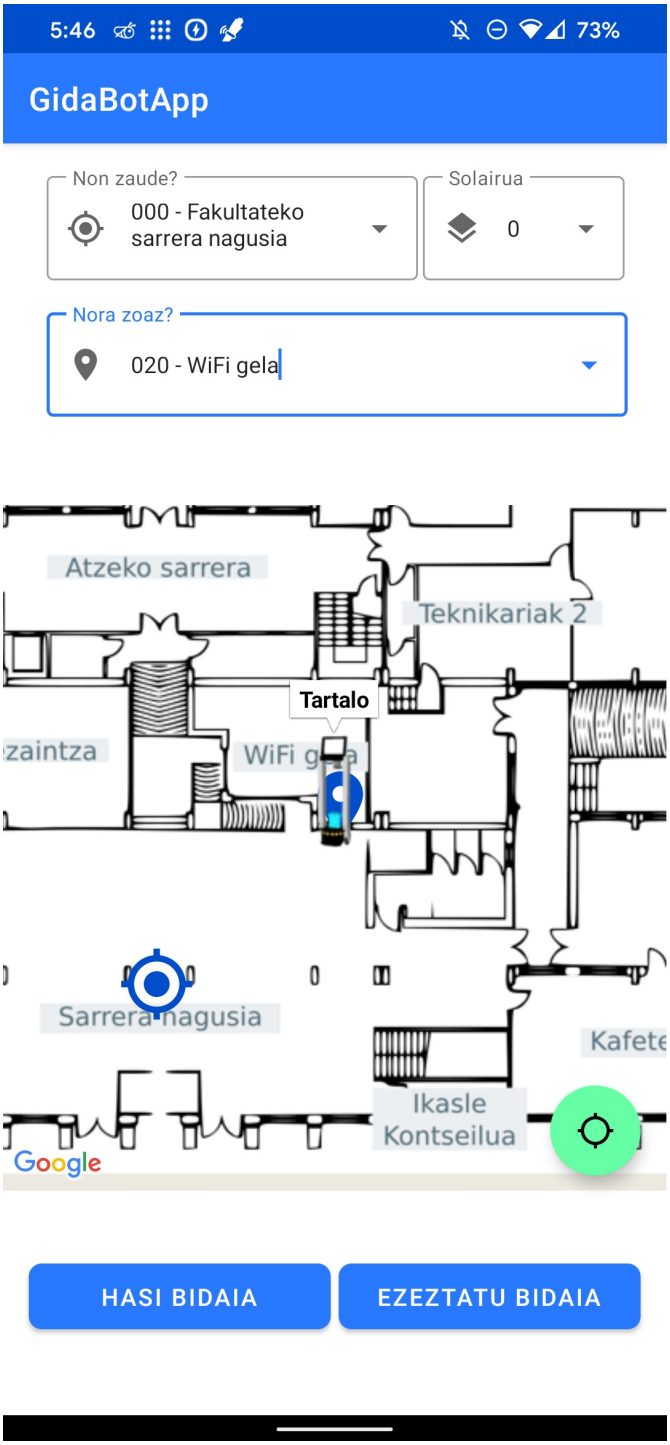
Beraz, aplikazioaren pakete sinatua sortu nahi dugunez, gako bat sortu beharko dugu, eta APK paketea sinatu gure gakoarekin. [61] Behin hau eginda, *GidaBotApp.apk*[4]fitxategia sortuko zaigu, eta hau instalatuz edozein erabiltzailek GidaBot sistema erabili ahalko du bere gailu mugikorretik.

7.4.6 Emangarria

Behin aplikazioak bezeroaren eskakizun berri guztiak asebetetzen zituela, hirugarren eta azken iterazioa amaitu egin zen. Egia da oraindik ere hobekuntza batzuk egin zitezkeela, baina baloratu zen jada aplikazioak kalitate ona zuela, eta ondorioz, hobe zela memoria eta beste atazei esleitzea geratzen zen denbora. Azkenik, 7.10 irudian azken iterazioko emangarriaren interfaze-grafikoa ikus daiteke.

[3]Hala ere, Google Play zerbitzura igo nahiko bagenu, Android App Bundle modura paketatu beharko genuke.

[4]*GidaBotApp.apk*: `https://github.com/i0su/gidabotApp/releases/latest`



7.10 Irudia: Hirugarren iterazioa. Emangarria

8. KAPITULUA

Jarraipen eta kontrola

Kapitulu honetan GrAL honen garapenean zehar irismenak jasan dituen aldaketak azalduko dira, baita ataza bakoitzaren dedikazioak plangintzarekiko sortu duen desbiderapena ere. Gainera, arriskuen eta kalitatearen inguruko kontrolerako burututako atazak azalduko dira.

8.1 Irismenaren desbiderapena

Proiektuaren irismena definitzean azaldu zen modura, Gradu Amaierako Lan honek izaera ikertzailea duenez, proiektuaren hasieratze-fasean ez genekien lanak zehaztutako helburu guztiak beteko zituen. Horren ondorioz, proiektuak arrakasta izateko bete beharko zuen irismen minimoa definitu zen.

Gainera, proiektuaren garapena estrategia arinak jarraituz burutu denez, iterazio bakoitzaren amaieran bezeroak eskakizun berriak zehazten zituen. Horietako eskakizun batzuk aplikazioari gehitu zaizkio, baina beste batzuk denbora faltagatik baztertu behar izan dira.

Erabiltzaile konkurrenteak

Lehenengo iterazioaren amaierako bileran, planteatu zen ea zein izan beharko litzatekeen aplikazioaren portaera hainbat erabiltzailek batera jarduten bazuten, robotei eskaerak modu konkurrente batean burutuz. Izan ere, erabiltzaile batek “Bidaia hasi” botoia sakatzen

duenean, robotari soilik erabiltzaile horren jatorrira joateko agindua bidaltzen zaio. Modu horretan, erabiltzailearengana iristean erabiltzaileak bidaia jarraitu nahi badu, helmugara joateko agindua bidaltzen zaio robotari. Portaera hori egokia eta zuzena da soilik erabiltzaile batek elkarreragiten badu robotarekin.

Aldiz, hainbat erabiltzaile badaude aplikazioa konkurrenteki erabiltzen, gerta daiteke robota erabiltzaile batengana joaten hastea, eta bidean dagoelarik beste erabiltzaile batek beste eskaera bat burutzea. Hori gertatzen denean, robotak aginduak ilara batean jasotzen dituenez, FIFO (*First-In, First-Out*) moduan kudeatuko dira iristen diren eskaerak. Beraz, robota lehenengo erabiltzailearen kokalekura iristen denean, jarraian bigarren erabiltzaileak zehaztutako kokalekura hurbilduko da. Portaera hau ez da erabiltzaileak espero duena, ezta GidaBot nabigazio-sistemarengandik espero dena ere, azken finean robotak erabiltzaileengana hurbildu bai, baina ez dituztelako haien helmugetara gidatuko.

Portaera oker hori konpondu nahian, planteatu zen erabiltzaileak eskaera bat burutzean robotari zuzenean bi kokalekuak bidaltzea (jatorria eta helmuga). Modu horretan, robotaren FIFO portaerak jarraituko zuen, baina kokaleku-mailako FIFOa izan ordez, ibilbide-mailakoa. Hortaz, estrategia berria aplikazioan inplementatu eta probatu zen, baina estrategia berri honek arazoak sortzen zituen ibilbidean bi robotek parte hartu behar zutenean (bi solairuko bidaien helburua). Ondorioz, bezeroarekin adostu ondoren erabaki zen erabiltzaile konkurrenteak kontuan ez izatea, eta aplikazioak erabiltzaile bakarrarekin funtzionatzea, baina solairu bakarreko eta bi solairuetako bidaietan.

Mapan bidea irudikatzea

Proiektuaren hasiera-fasean ezarri zen erabiltzaile berri bat iristen bazen eta solairu horretako robota okupatuta bazegoen, aplikazioak hautatutako helmugaraino iristeko jarraitu beharreko bidea erakutsiko ziola mapa batean.

Bigarren iterazioa amaitzean, bezeroarekin eztabaidatu ostean, erabaki zen funtzionalitate hau baztertzea. Izan ere, mapako bi kokalekuren arteko bidea zehazteko robotaren planifikatzaile globalak egiten duen lanaren antzeko zerbait egin beharko genuke, esaterako $A*$ edota *Wavefront* algoritmoak erabiliz.[4] Hori horrela, estimatu zen algoritmo horiek erabiliz mapa batean ibilbideak bistaratzeak denbora gehiegi emango lukeela, eta ondorioz, irismenetik kanpo utzi zen funtzionalitate hori.

Gainera, aurreko atalean azaldu den bezala, lehenengo iterazioaren amaieran erabaki zen aplikazioaren garapena erabiltzaile bakar baten funtzionamendurako egokitzea beste era-

biltzaile posibleak kontuan izan gabe. Hortaz, nolanahi ere, maparen bidea irudikatzeak ez luke zentzu handirik izango aplikazioa erabiltzaile bakar batentzat pentsatuta balego.

Bi solairuko bidaia

Aplikazioaren garapenaren hasieran bi solairuko bidaiak burutzea asmo handiko helburu modura ezarri zen. Hau da, uste genuen nahikoa denbora hartuko zuela aplikazioak solairu bakar batean funtzionatzeak, baina hala eta guztiz ere, hirugarren iterazioan ikusi zen nahikoa denbora zegoela aplikazioaren bitartez bi solairuko bidaiak burutzeko. Hortaz, azkenean lortu da aplikazioaren bitartez erabiltzaileek pisu anitzeko ibilbideak burutzea, baina ez soilik aplikaziotik. Izan ere, Ondorioen atalean hobeto azalduko denez, egungo GidaBoten egoeraren ondorioz, erabiltzaileak helmugako solairura jaitsi edo igotzen direnean, solairu horretako robotaren pantailatik jarraitu beharko dute bidaia, hau da, ezingo da bidaia osoa aplikaziotik egin %100ean.

8.2 Plangintzaren desbiderapena

Atal honetan, proiektuaren plangintzan egindako atazen denboraren estimazioa eta atazen dedikazioa alderatuko da. Horretarako, 8.1 taulan lan-pakete bakoitzeko atazak burutzeko behar izan den denbora adierazi da.

Taulan ikus daitekeenez, oro har, Jarraipen eta Kontrol, Aplikazio eta Plangintza burutzeko atazei hasieran planifikatutakoa baina denbora gehiago eskaini behar izan zaie. Gainera, proiektuaren amaieratze-faseko paketeak izan dira desbiderapen positiboa izan dutenak, hau da, estimatutakoa baina denbora gutxiago eskatu dutenak; Memoria, Dokumentazio Teknikoa eta Defentsa.

Desbiderapen orokorrari behatuz, ikus daiteke estimatutakoa baina 16 ordu gehiago behar izan dugula proiektua aurrera eramateko. Kontuan izanik planifikazioan atazak gainetik estimatu zirela, ematen du gehiegizko tarte hori ez dela nahikoa izan, eta eskuarki ataza guztiek pentsatzen zena baina aski denbora gehiago eman dutela.

Ataza	Estimazioa	Dedikazioa	Desbiderapena
Plangintza (P)	**17**	**25**	**+8**
P.1	4	7	+3
P.2	10	15	+5
P.3	3	3	0
Jarraipen eta Kontrola (JK)	**11**	**22**	**+11**
JK.1	3	3	0
JK.2	-	2	+2
JK.3	-	2	+2
JK.4	8	15	+7
Formakuntza (F)	**75**	**72**	**-3**
F.1	35	20	-15
F.2	20	22	+2
F.3	20	30	+10
Eskakizunen Bilketa (EB)	**12**	**15**	**+3**
EB.1	5	8	+3
EB.2	4	4	0
EB.3	3	3	0
Soluzioaren Diseinua (SD)	**18**	**22**	**+4**
SD.1	3	10	+7
SD.2	5	6	+1
SD.3	10	6	-4
Aplikazioa (A)	**100**	**107**	**+7**
A.1	15	7	-8
A.2	70	90	+20
A.3	5	4	-1
A.4	10	6	-4
Memoria (M)	**75**	**71**	**-4**
M.1	70	65	-5
M.2	5	6	+1
Dokumentazio Teknikoa (DT)	**15**	**5**	**-10**
DT.1	10	3	-7
DT.2	5	2	-3
Defentsa (D)	**18**	**18**	**0**
D.1	3	3	0
D.2	10	8	-2
D.3	5	7	+2
GUZTIRA	**341**	**357**	**+16**

8.1 Taula: Atazen dedikazioa eta planifikazioarekiko desbiderapenak

(*Ataza bakoitzaren dedikazio-taula esteka honetan)

8.3 Arriskuen jarraipena

Atal honetan, 2.4 partean aztertu ziren arriskuen jarraipen eta kudeaketa azalduko da. Horretarako, 8.2 taulan arrisku bakoitzaren jarraipena zein izan den adierazi da. Orokorrean, esan daiteke proiektuak ongi jasan dituela arrisku guztiak, eta ez dela proiektuaren arrakasta kolokan jarri duen mehatxurik egon.

8.4 Kalitatearen jarraipena

Kalitatearen kudeaketari dagokionez, proiektuaren irismeneko kapituluan kalitatea kudeatzeko burutuko diren bi proba-motak aurkeztu dira, hau da, barne zein kanpo probak. Beraz, atal honetan azaldu da nola burutu diren proba horiek proiektuaren garapenean.

Barne Probak

Barne probei dagokionez, Android Developers plataforman Android aplikazioen probak automatizatzeko gida zabal bat eskaintzen da, *AndroidX Test API*aren bitartez aplikazioko proba-kasuak automatizatzeko.[53] Hala ere, denbora falta zela eta, egileak berak eskuz exekutatu behar izan ditu probak. Horrek, ondorioz, esan nahi du kasuren batean aplikazioak funtzionamendu desegoki bat izan dezakeela, oso zaila baita eskuzko proben bidez kasu guztiak barne hartzea.

Kanpo Probak

Burutu diren kanpo probak aztertuz, Kalitatearen kudeaketa atalean zehaztu zen modura, iterazio bakoitzaren amaieran bezeroari aplikazioaren erakusketa egin zaio, eta aukera izan du aplikazioa probatu ahal izateko. Modu horretan, printzipio arinak jarraituz iterazioaren *feedback*a eman, eta hurrengo iterazioan iradokizun horien araberako aldaketak egin dira. Gainera, aplikazioa amaituta zegoenean, azken proba bat burutu da Informatika Fakultatean robot fisikoekin, aplikazioaren bideo-demoa grabatu ahal izateko.

Arriskua	Tratamendua	Jarraipena
A1	Arindu	ROSen bertsio-eguneraketa batek GidaBotek beharrezkoak zituen pakete batzuk ezabatu zituen, baina pakete horiek berriz instalatzearekin arazoa konpondu zen.
A2	Arindu	Plangintzaren desbiderapenean ikusi denez, aplikazioaren lan-paketea desbiderapen gehien izan duena da. Hala ere, desbiderapen horrek proiektuan ez du eragin handia izan.
A3	Arindu	Lehenengo iterazioan proiektuaren irismen minimoa bete zen. Ondorioz, lehenengo iterazioan jada arriskua ekidin egin zen.
A4	Arindu	Arriskuen kudeaketa planean adierazi bezala, aplikazioaren kodearen eta konfigurazio-fitxategien segurtasun-kopiak periodikoki burutu dira. Bi astean behin, babeskopia horiek modu egokian gordeta zeudela egiaztatzen zen. Horri esker, garapenaren amaieran iterazio bakoitzeko kodea berreskuratu ahal izan da, memoriaren idazketa errazteko.
A5	Onartu	Ez zuzendarien ezta egile honen erabilgarritasun egoera ez da aldatu proiektuaren garapenean zehar, eta beraz, arriskuak ez du lanean inpakturik izan.
A6	Onartu	COVID 19ak ez du inpakturik izan proiektuan, taldeko inor ez baita gaixotu proiektuaren bizi-ziklo osoan. Are gehiago, izatekotan, arrisku positiboa izan du, pandemiaren egoerak hobera egiteak bilera presentzialen aukera ahalbidetu baitu.
A7	Onartu	Egile honen eramangarriak ez du arazorik izan proiektuaren garapen osoan. Ondorioz, arriskuak ez du inpakturik izan.

8.2 Taula: Proiektuko arriskuen jarraipena

8.5 Komunikazio-sistemak

Komunikazio sistemei dagokionez, Proiektuaren irismenean azaldu den modura, e-posta izan da zuzendarien (bezeroen) eta egile honen arteko komunikabide nagusia. Gainera, komunikazioen historiaren berrikuspena errazteko, GrALarekin lotutako komunikazio guztiak hari bakar batean egin dira, hau da, e-posta berri bat bidali behar zenean hari horretara txertatzen zen erantzun modura, aurreko mezu guztiak ikusi ahal izateko begirada batez.

Bilerei dagokionez, espero zen bezala bilerak *BlackBoard Collaborate* plataformaren bidez egin dira. Horrez gain, pandemiaren egoerak zertxobait hobera egin duenez, hainbat aurrez-aurreko bilera egiteko aukera izan dugu baita ere. Hau proiektuarentzat onuragarria izan da, batez ere aplikazioaren funtzionamendua bezeroari zuzenean erakusteko aukera eman duelako.

9. KAPITULUA

Ondorioak

Aplikazioaren garapena amaitu ostean, proiektuari itxiera emateko beharrezkoa da lortutako produktuaren kalitatea epaitzea hasieratze-fasean ezarritako irizpideen arabera. Horretarako, kapitulu honetan produktuaren emaitza baloratuko da, eta garatze-prozesuan ikasitako lezioak ere adieraziko dira. Azkenik eta amaitzeko, denbora faltagatik aplikazioan inplementatu ezin izan diren funtzionalitate berri edo aldaketak adieraziko dira, etorkizunean hobekuntza modura gehitu litezkeenak.

9.1 Lortutako emaitza

Garapenaren ostean lortutako produktua irismenean zehaztutako irizpideen arabera ebaluatzeko, irizpide horien araberako azken probak egingo zaizkio aplikazioari.

Horretarako, 9.1 taulan, aplikazioa amaitzean egikaritu diren azken probak adierazten dira. Ezker zutabean probaren identifikatzailea adierazten da, eta ondoan proba bakoitzak zein irizpide aztertzen d(it)uen. Azkenik, portaeraren zutabean probaren emaitza azaltzen da.

Taulan ikusten denez, aplikazioak hasieran ezarritako irizpide ia guztiak bete ditu, bete ez duen bakarra (edo probatu ezin izan dena) azken Android bertsioan egoki funtzionatzen duen aztertzea izanik. Hortaz, ezin dugu esan aplikazioaren kalitatea bikaina dela, ez dituelako ezarritako guzti-guztiak bete. Ordea, baiezta dezakegu aplikazioaren kalitatea **ona** eta **bikaina**ren artekoa dela, ona izateko beharrezkoak ziren KO-1 eta MP1..3 bete baititu,

Proba	Irizpideak	Deskribapena	Portaera
CR-3	VX-N1 VX-N2	Aplikazioaren pantaila eta elkarrizketa-leiho bakoitzean atzera-botoia sakatu	Egokia
CR-1	VX-N3	Aplikazioaren pantaila eta elkarrizketa-leiho bakoitzean hasiera-botoia sakatu	Egokia
CR-9	VX-S2	Aplikazioak erakutsi ditzakeen jakinarazpen guztiak egikaritu. Jakinarazpenetan klik egin eta erakusten dituen botoietan klik egin	Egokia
SC-4	FN-P2	Aplikazioak behar dituen baimen guztiak berrikusi, bai *manifest* fitxategian, exekuzio-garaian, edota konfigurazioetan ere.	Egokia
CR-0	PS-T1	Azken Android bertsioarekin aplikazio osoa nabigatu. Bai pantaila, leiho, konfigurazio eta fluxuak, baita konfigurazio guztiak ere.	Okerra (*)
MN-1	KO-1 MP-1 MP-3	Solairu bakarreko bidaia burutu GidaBot sisteman	Egokia
MN-2	MP-2	Bi solairuko bidaia burutu GidaBot sisteman	Egokia

9.1 Taula: Aplikazioaren azken kalitate-probak

(*) Ezin izan da probatu, egileak nahiz zuzendariek ez zeukatelako azken Android bertsioa zuen gailu fisikorik eskuragarri.

eta gainera, hautatu diren gainontzeko irizpide estandarrak betetzen dituelako, *PS-T1* izan ezik.

9.2 Ikasitako lezioak

Garapenean zehar hartutako erabaki onek, baina batez ere erabaki txarrek (edota hartu ez diren erabakiek) beti ematen digute zerbait ikasteko aukera. Beraz, burututako akatsak etorkizuneko proiektuetan berriz ere ez errepikatzeko, atal honetan GrALean ikasitako lezioak errepasatuko dira.

9.2.1 Bertsio kontrola

Nahiz eta aurreko proiektuetan bertsio-kontroleko programekin lan egin izan, proiektu hau orain arte egile honek burutu duen handiena izanik, proiektuak bertsio kontrol on bat mantentzearen garrantzia azpimarratu du. Izan ere, bai garapeneko zein dokumentazioko ataletan, askotan egin da aurrera eta atzera aplikazioaren bertsioetan. Garapeneko atalean, esaterako, funtzionalitate berriak inplementatzerakoan behin baino gehiagotan egin behar izan da bertsio egonkor batera atzera, garatzen ari zen kodeak errore gehiegi sortzen baitzituen. Bestalde, dokumentazioko fasean lehenengo eta bigarren iterazioetako kodea berreskuratu ahal izan da bertsio kontrolari esker, eta horrek dokumentazio lana asko erraztu du.

Hortaz, ikaspen bezala ondorioztatzen da aplikazio bat garatzea helburu duen software proiektu orok bertsio kontrolerako tresnaren bat erabili beharko lukeela.

9.2.2 Instalazioak

GrAL honetan garatu den aplikazioa inplementatu ahal izateko hainbat instalazio jarraitzea beharrezkoa izan da: ROSen oinarrizko instalazioa batetik, *rosjava_core* modulua bestetik, eta azkenik *android_core* modulua. ROSen oinarrizko instalazioak ez zuen arazo gehiegirik ekarri, baina rosjava eta android moduluen instalazioak arazoak sortu zituzten. Batez ere, *android_core* moduluaren kasuan, moduluaren bertsio oso zaharkitu bat instalatu zen, eta Android Studion proiektuak irekitzerakoan errore asko jaurtitzen zituen. Hainbat saiakera eta denbora galdu ostean errore guzti horiek konpontzen saiatzen, ikusi zen beste bertsio eguneratuago bat zegoela, eta horrek egoki funtzionatzen zuela.

Ondorioz, honakoa da proiektuko instalazioen ikasketa: proiektu bat burutzeko edozein software instalatu behar denean, saiatu bertsio eguneratuena instalatzen, edota taldearen behar teknologikoetara gehien hurbiltzen dena. Horretarako, softwarearen dokumentazioa aztertu. Dokumentazio-orri bat baino gehiago badauka, guztiak irakurri eta ondorioztatu zein den taldearen beharretara gehien hurbiltzen dena.

9.3 Etorkizuneko hobekuntzak

Denbora faltaren edota beste kausa batzuen ondorioz, aplikazioan ezin izan dira bezeroak proposatutako hainbat funtzionalitate inplementatu. Hori dela eta, atal honetan hobekuntza modura aurkeztuko dira inplementatu ezin izan diren funtzionalitate berri edo aldaketa horiek.

9.3.1 Hizkuntza aldaketa

Aplikazioaren kodea hizkuntza aldaketarako prest dago; falta den bakarra da pantailaratzen diren testu eta mezu guztiak nahi diren hizkuntzetara itzultzea, eta hizkuntza bakoitzerako *stings.xml* fitxategi bat sortzea. Izan ere, aplikazio osoan zehar *resource id*-ekin lan egiten da, eta bista-geruzak mezu bat inprimatzerakoan *id* horri dagokion testua lortzen du. Horri esker, edozein momentutan *Locale*a (hizkuntza) aldatzen bada, bista-geruzak testua hizkuntza horri dagokion *strings.xml* fitxategitik hartuko du, eta erabiltzaileak hizkuntza-aldaketa ikusiko du. [63]

9.3.2 Bi solairuko bidaiak

Bi solairuko bidaiak inplementatu ostean, konturatu ginen erabiltzailea helmuga-solairura iristean ezingo zuela ibilbidea aplikaziotik bertatik jarraitu (espero lezakeen bezala), baizik eta robotetik bertatik egin beharko zuela bidaiaren azken zatia. Arazo hau konpontzeko, GidaBot sistemaren egungo egitura aldatu beharko litzateke, eta ez da horretarako denborarik egon.

Izan ere, erabiltzailea aplikaziotik robot bakar batera konektatzen da, eta nahiz eta aplikazioak robot horri soilik eskaerak egiten dizkion, sistema gai da jakiteko helmuga beste solairu batean baldin badago erabiltzaileak solairu-aldaketa bat egin behar duela. Beraz,

erabiltzaileari nabigazioaren inguruko jakinarazpenak erakusteko, GidaBotek mezuak bidaltzen ditu *topic* batera, */dialog_qt_message topic*-era hain zuzen. Solairu bakar bateko bidaietan arkitektura horrek ongi funtzionatzen du, baina bi solairuetakoan erabiltzaileari ez zaizkio helmugako solairuko roboteko jakinarazpenak iristen, bera jatorriko solairuko robotera konektatuta baitago. Ondorioz, ezin du aplikaziotik bidea jarraitu, eta roboteko pantailatik jarraitu beharko du.

Hori horrela, robot bakoitzak bere */dialog_qt_message topic*-a izango balu, arazoa konponduko litzateke. Hau da, Tartaloren kasuan, esaterako, */tartalo/dialog_qt_message*, edota Kbotenean */kbot/dialog_qt_message*. Modu horretan, aplikazioa *topic* guzti horietara harpidetuko da, eta erabiltzaileari uneko solairuaren araberako jakinarazpenak erakutsiko zaizkio.

9.3.3 *RosActivity*

Aurretik azaldu den modura (ikus 7.2.1 atala), gure aplikazioak *RosActivity* klaseari esker ezartzen du robotarekin konexioa, honek eskaintzen baitu ROS sistemarekin konektatzeko interfazea. Klase honek GidaBotApp aplikazioaren garapena erraztu du, baina egia da nahiko zaharkitua dagoela, eta aldaketa batzuk behar dituela (ikus garapeneko 7.3.2 atala). Hari horretatik, klaseak duen arazo nagusia da *Activity* klasea hedatzen duela, eta horren ondorioz ezin dugula MVVM-ko bista geruza modura erabili. Horregatik, aplikazioan (soluzio modura) *RosActivity* klasea soilik ROS Masterrarekin konexioa egiteko erabiltzen da, eta ondoren *RouteSelectActivity* klaseari pasatzen zaio lekukoa.

Azaldu den bezala, *Activity* bakarretik guztia egin ahal izateko, *RosActivity* klaseak *AppCompatActivity* klasea hedatu beharko luke, baina klasea birkonpilatzeak erroreak ematen zituen. Ondorioz, ematen du arazoa konpontzeko soluzio aproposena *RosActivity* klasea ez erabiltzea dela, eta guk geuk inplementatu eta kudeatzea ROS sistemarekiko konexioak (baina noski, horrek ere lan gehiago ekarriko luke).

9.3.4 Java bertsioa

Proiektuan erabili den *RosActivity* klasea Java 7 (JDK 7) bertsioan garatuta dago, eta ondorioz, aplikazioaren garapenaren hasieratik Java 7 bertsioarekin lan egin zen. Hari horretatik, iturri-kodean klase — eta metodo — anonimo ugari erabili dira, eta garapenaren amaieran (kodea errefaktorizatzerakoan) ikusi zen Javako lambda funtzioak[27] erabiliz

klase anonimo horien kodea labur eta ulergarriagoa izan zitekeela. Ordea, lambda adierazpenak Java 8 bertsioan erazagutu ziren, eta erroreak ekiditeko erabaki zen Java 7 bertsioan jarraitzea. Ondorioz, etorkizuneko hobekuntza bezala geratzen da aplikazioaren java bertsioa eguneratzea, eta horrekin batera iturri-kodean lambda adierazpenak erabiltzea.

9.3.5 Wifiaren deskonexioa

Erabiltzaile bat GidaBotApp erabiltzen dagoelarik bere gailuari Wifi konexioa deskonektatzen bazaio, robotarekiko konexioa galduko du, aplikazioa berriz jaurti beharko duelarik. Hau ekiditeko, Wifia deskonektatzen den gertaera tratatu beharko genuke, erabiltzaileari aukera emanez wifira berriz konektatzeko, edota zuzenean aplikazioa birjaurtiz.[56]

9.3.6 Lokalizazio automatikoa

Proiektuaren irismenean aipatu den modura, garatu den aplikazioak ez du GPS seinalea erabiltzen erabiltzailea lokalizatzeko, GPS seinalea ezin daitekeelako barruko inguruneetan erabili. Izan ere, horregatik erabiltzaileak GidaBotApp aplikazioaren bitartez robotari bere jatorrizko kokalekua adierazi behar dio.

Hala ere, erabiltzailea automatikoki lokalizatzeko, Barneko Posizionamendu-sistema (*Indoor Positioning Systems*, *IPS*) bat erabili liteke. *IPS* baten bidez, robotak erabiltzailea zehazki non dagoen jakin ahalko luke, eta erabiltzaileak robotari soilik nora iritsi nahi duen zehaztea. Besteak beste, Wifian oinarritutako posizionamendu-sistema (*Wi-Fi-based Positioning System*, *WPS*), Irrati-frekuentziaren bidezko identifikazioa (*Radio Frecuency IDentification*, *RFID*) edota *RSSI* (*Received Signal Strength Indication*) motako teknologiek barrualdeetan lokalizazioa eskuratzeko aukera eskaintzen dute. [77]

Zerrendatutako *IPS* teknologietatik *WPS* teknologia aipagarriena da. Izan ere, beste teknologiek eraikinean (kasu honetan, Informatika Fakultatean) teknologia berriak instalatzea eskatzen dute, baina *WPS*n oinarritutako soluzioen bidez Wifi sarrera puntuen bitartez erabiltzaileak barrualdeetan kokatzen dira. Badira jada Wifi posizionamendu sistemaren bitartez erabiltzaileak eraikin barruetan edota lurpean kokatzeko aukera ematen duten proiektuak, hala nola, *SubPos*[74][73]. Ondorioz, halako proiekturen bat erabiliz erabiltzailea automatikoki kokatzeko aukera egongo litzateke, aplikazioa maila handi batean hobetuz.

9.3.7 Erabiltzaile konkurrenteak

GidaBot sistemaren berezko izaeraren ondorioz, garatu den aplikazioak ez ditu hainbat aldibereko erabiltzaileren eskaerak kudeatzen (ikus 8.1 atala). Arazo nagusia, GidaBot sistemak helburuak pilaratzen dituela da, eta GidaBotApp aplikazioak, ordea, ibilbideak pilaratzea eskatzen duela. Hau da, egungo egoeran, Kafetegian dagoen erabiltzaile batek Kopistegira joateko eskaera egiten badu, robota Kafetegira joango da, eta iristean Kopistegira joateko deia bidaliko zaio. Ordea, erabiltzailearen jatorrira (hau da, Kafetegira) iritsi baino lehen beste erabiltzaile batek robotari Atezaintzan dagoela adierazten badio, Kafetegira iritsi ondoren robota Atezaintzara joango da, lehen erabiltzailearen ibilbidea osatu gabe geratuko delarik.

Beraz, GidaBot sisteman aldaketa horiek egin ondoren, aplikazioa ere bi erabiltzaile desberdinen deiak aldi berean kudeatzeko egokitu ahalko litzateke. Gainera, behin robotek bi helburu jarraian egoki kudeatzen dituztelarik, aplikazioan aurpegi-antzemate (*face recognition*) modulua gehitzeko aukera izango genuke.

Izan ere, *face recognition* modulua gehituta daukan GidaBot-en bertsioan, solairu aldaketa bat dagoenean lehenengo robotak erabiltzaileari argazki bat ateratzen dio, bigarren robotak ibilbidea erabiltzaile berdinak jarraitzen duela egiaztatzeko. Horretarako, bigarren robotak erabiltzaileari beste argazki bat atera, eta lehenengo robotak egindako argazkiarekin bat badator, ibilbidearekin jarraitzen du.

Hori horrela, proposatzen den azken hobekuntza, argazkiak robotek atera ordez, aplikaziotik bertatik ateratzea da.

Eranskinak

A. ERANSKINA

GrALaren hasierako proposamena

Titulua	GidaBotApp: GidaBot nabigazio sistemaren bidez erabiltzailea barruko inguruneetan bideratzeko aplikazio baten garapena eta integrazioa.
Indibiduala / Taldean	Indibiduala
Espezialitatea	Software Ingeniaritza
Azalpena	GidaBot, RSAIT taldeak garatu duen gida sistema bat da. Sistema hau, elkarlanean gida lanak aurrera eramateko gai diren lau robotez osatzen da, eta erabiltzaileak, eraikin barnean puntu batetik bestera autonomoki eramateko diseinaturik dago. Proiektu honen helburua, GidaBot sisteman integratuko den aplikazio bat garatzean datza. Aplikazio horren bitartez erabiltzaileak GidaBot sistemarekin elkarreragin ahalko du, sistemari gida eskaerak eginez aplikazioaren bitartez.
Lantaldea	Robotika eta Sistema Autonomoen Ikerketa Taldea (RSAIT)

Egin beharreko lanak	Proiektu honetan garatu beharreko atazak ondorengoak dira: 1) GidaBot sistema hausnartu. 2) GidaBot sistemara dispositibo mugikor bat gehitu eta bien arteko komunikazioa gauzatu. 3) Android aplikazio bat garatu GidaBot sistemari eskaerak burutuko diona.
Helburuak	GidaBot sistemari laguntzeko aplikazio bat garatu.
Lanerako materiala	1. ROS 2. Android Studio 3. Java
Orduak guztira	300
Iraupena	2020/09/14-2021/07/23
Zuzendaria	JAUREGI IZTUETA, EKAITZ ekaitz.jauregi@ehu.eus
Zuzendarikidea	RODRIGUEZ RODRIGUEZ, IGOR igor.rodriguez@ehu.eus

B. ERANSKINA

ROS: *The Robot Operating System*

Memorian azaldu den modura, GidaBot sistemako robotek ROS frameworka erabiltzen dute, eta horri esker robot bakoitzak zehazten zaion helmugara gida dezake erabiltzailea, eta helmuga beste solairu batean badago beste robotekin komunikatu erabiltzaileak solairu-aldaketa egin ahal izateko. Teknologien atalean jada ROS meta-sistema eragilearen sarrera bat egin da, baina atal honetan ROSen kontzeptu nagusiak azalduko dira modu labur batean, irakurleak sistema ezagutzen ez badu memoria hobeto ulertu ahal izateko. Gainera, GidaBot eta GidaBotApp aplikazioen funtzionamendua ahalbidetzen duten osagaien adibideak emango dira. [14]

B.1 Fitxategi-sistema mailako kontzeptuak

Fitxategi-sistemaren mailan, ROSek diskoan gordetzen dituen baliabideak izango ditugu; besteak beste, paketeak eta mezu — eta zerbitzu — motak.

B.1.1 Paketeak

ROSek softwarea antolatzeko daukan unitate nagusia dira. ROS pakete batek hainbat prozesu izan ditzake bere barnean (nodoak), liburutegiak, datu-multzoak, konfigurazio fitxategiak, eta abar. Paketeak ROSen eraiki eta argitaratu daitezkeen elementu atomikoenak dira; hau da, sortu eta argitara daitekeen unitaterik txikiena.

Pakete bakoitzak *manifest* deituriko fitxategi bat dauka (*package.xml*), non paketearen inguruko informazioa ematen den (bertsioa, izena, deskribapena, dependentziak, eta abar).

Gainera, meta-pakete deituriko pakete-multzoak ere existitzen dira, elkarrekiko erlazioa duten pakete-multzo bat adierazten dutenak. Izan ere, GidaBotek hainbat meta-pakete erabiltzen ditu, esaterako, *rsait_common_packages* edota *tartalo*, azken honek Tartalo robotak behar dituen pakete guztiak biltzen dituelarik.

B.1.2 Mezu-motak

ROSek erabiltzen dituen mezu-mota bakoitza *.msg* fitxategi batean definitzen da, eta bertan mezu horrek garraiatuko duen informazio-mota(k) adierazten d(ir)a. Modu horretan, mezu-motak paketeetan gordeko dira, *my_package/msg/MyMessageType.msg* izango delarik pakete bateko mezuen kokalekua. Esaterako, GidaBotek erabiltzen dituen *Goal* mezuak *multilevel_navigation_messages/Goal.msg* fitxategian definitzen dira.

B.1.3 Zerbitzu-motak

Era berean, zerbitzuak *.srv* fitxategietan definituko dira, baina kasu honetan *my_package/srv/MyServiceType.srv* kokalekuan.

B.2 Konputazio-grafo mailako kontzeptuak

ROSen konputazio-grafoa, parez-pareko sare bat da, non datuak prozesatzen dituzten ROSen prozesu guztiak elkartzen diren. Konputazio-grafoko kontzeptu nagusiak Masterra, Zerbitzaria, mezuak, zerbitzuak eta *topic*-ak dira, batik bat.

B.2.1 Nodoak

ROS ingurunean nodoak kalkuluak burutzen dituzten prozesuak dira. ROS sistemaren helburua modularra izatea da, eta ondorioz, robot baten kontrol-sistema batek, normalean, nodo asko erabiltzen ditu. Adibide modura, nodo batek laser-sentsore bat kontrola dezake, beste batek robotaren gurpilen abiadura, eta azken batek bideen planifikazioa burutu. Oro har, ROS nodoak ROS bezero-liburutegiak (*ROS client library*) erabiliz garatzen dira,

hots, *roscpp* edota *rospy*. Gainera, GidaBotApp aplikazioan Masterrarekin komunikazioa burutuko duen *QNode* klasea ere nodo bat izango da.

B.2.2 Masterra

ROS Masterrak gainontzeko Konputazio-grafoaren gaineko bilaketa eskaintzen du. ROS Masterrari esker, grafoko nodoek elkar ikus dezakete, mezuak trukatu eta zerbitzuak jaurti.

Hori horrela, GidaBot nabigazio-sisteman robot bakoitzak bere ROS Masterra izango du, eta, beraz, GidaBotApp aplikazioa Master batera konektatzen dela esaten dugunean, sistemaren robotetako batekin konektatzen dela esan nahi dugu.

B.2.3 Mezuak

Nodoek elkarren artean komunikatzeko mezuak erabiltzen dituzte. Mezu bat datu-egitura bat besterik ez da, mezuaren zehaztapen-fitxategian zehazten diren datu-motak garraiatzen dituena. Mezuek datu-mota primitiboak onartzen dituzte (osokoak, boolearrak, koma mugikorreko zenbakiak, etab), baina kabiatutako egiturak baita arrayak ere onartzen dituzte. Esaterako, GidaBotek erabiltzen dituen *Goal* mezuek honako egitura dute:

```
## Goal.msg
uint32 goal_seq
float32 initial_floor
geometry_msgs/Point initial_pose
float32 goal_floor
geometry_msgs/Point goal_pose
bool intermediate_robot
float32 intermediate_floor
string way
string start_id
string goal_id
string language
# User
string user_name
sensor_msgs/Image user_image
```

B.1 Kodea: *Goal* mezuen erazagupena

B.2.4 *Topic*-ak

ROS sisteman mezuak garraio-sistema baten bidez bidaltzen dira, argitaratzaile/harpidedun semantikekin. Nodo batek mezu bat bidaltzeko mezu hori buzoi edo *topic* batera argitaratuko du. Beraz, *topic* bat, mezuaren edukia identifikatzeko erabiliko den izena da. Modu horretan, datu jakin batzuk interesatzen zaizkion nodo bat, datu horiek dituen *topic*era harpidetuko da.

Gai bakar baterako argitaratzaile eta harpidedun ugari egon daitezke, eta nodo batek *topic* ugaritan argitaratu dezake eta harpidedun izan. Arkitektura hau jarraituz, argitaratzaile eta hartzaileek ez dute elkar ezagutzen, informazioaren ekoizpena eta kontsumoa bereizten direlarik.

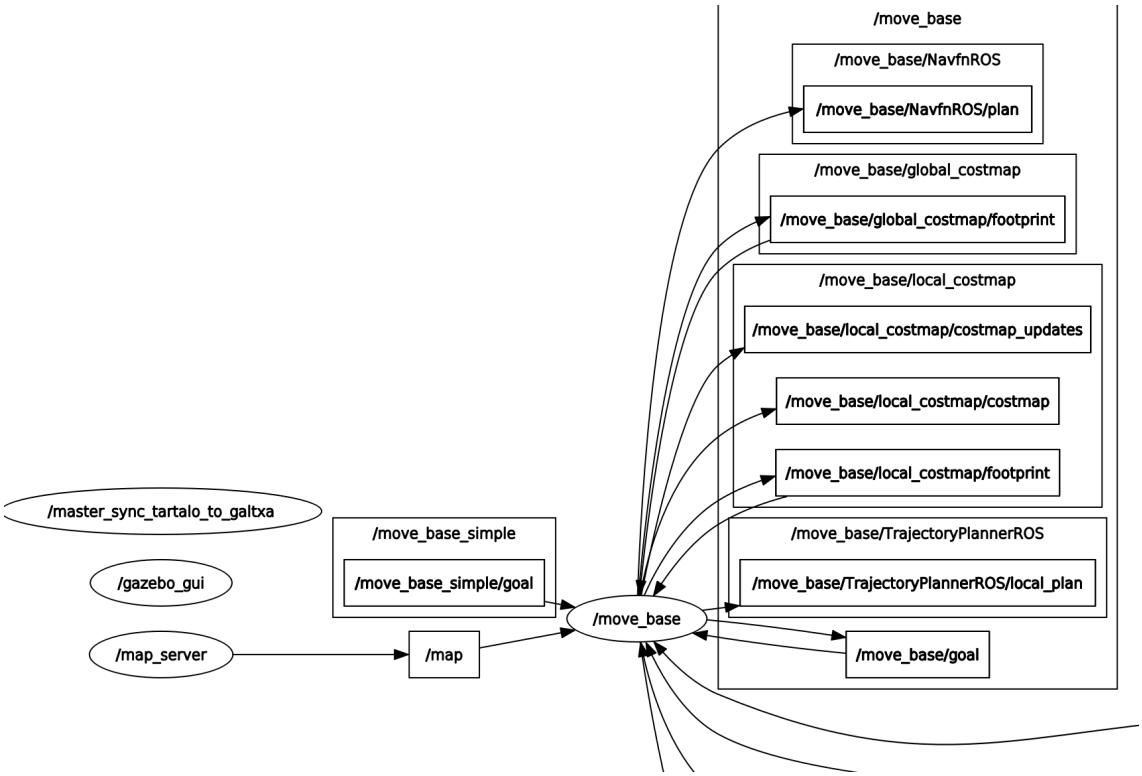
Adibide modura, GidaBoten */multilevel_goal topic*-ak *Goal* motako mezuak jasotzen ditu, eta bertara mezu horiek formatu egokian argitaratuz, lortzen da Android aplikaziotik robotari aginduak bidaltzea kokaleku batera mugitzeko.

B.2.5 Zerbitzuak

Argitaratzaile/harpidedun eredua oso komunikazio paradigma malgua da, baina garraio modu hori eskaera/erantzun elkarrekintzetan ez da oso egokia. Sistema banatuetan komunikazio-mota hori askotan behar izaten denez, ROSek zerbitzuak definitzen ditu elkarrekintza horietarako. Zerbitzuak mezu-egitura bikote bat dira, bata eskaera, eta erantzuna bestea. Nodo batek zerbitzu bat eskaintzen du izen batekin, eta bezero batek zerbitzua erabiltzen du, mezu bat bidali eta erantzunaren zain geratuz. GidaBoten kasuan, zerbitzuak batik bat erabiltzen dira planifikatzaile-globalaren buferra garbitzeko, */move_base/clear_costmaps topic*-era mezu huts bat bidaliz.

B.2.6 RQt

ROSek RQt izeneko tresna eskaintzen du, zeinek interfaze baten bitartez, uneko Konputazio-grafoa irudikatuko duen. Esaterako, makina batean Tartalo robota jaurtitzen badugu, B.1.irudian adierazten den grafoa lortuko dugu.



B.1 Irudia: Tartalo robotaren ROS konputazio-grafoaren zati bat

B.3 Rviz

ROS aplikazioen 3D bisualizazioa ahalbidetzen duen tresna bat da Rviz. Tresna honek, robotaren modeloaren ikuspegia eskaintzen du, robotaren sentsoreen informazioa jasotzen du, eta atzemandako irakurketak erakusten ditu. Gainera, kamara, laser, eta 2 zein 3 dimentsiotako gailuen datuak erakuts ditzake. [71]

B.4 Lengoaiarekiko independentzia

ROS ingurunearen helburuetako bat, edozein programazio-lengoaia modernotan txertatzeko erraza izatea da. Hori dela eta, gaur egun Python, C++ eta Lisp lengoiaetan inplementatuta dago, eta Java zein Lua lengoaietarako liburutegiak eskaintzen ditu.

Proiektu honetarako interes gehien liburutegia Javakoa da, memorian zehar aipatu den bezala, Javan oinarritutako *rosjava_core* eta *android_core* liburutegiei esker garatu baita GidaBotApp aplikazioa.

B.4.1 *rosjava_core*

*Google*k eta *Willow Garage*k garatuta, Rosjava ROSen Javako lehen inplementazio purua da. Rosjavak Android eta ROS ingurunean ointarritutako roboten integrazioa ahalbidetzen du, *android_core* paketearen bidez.

B.4.2 *android_core*

ROSeko *android_core* paketea, Androiden ROS aplikazioak garatzeko baliagarriak diren osagai eta adibideen bilduma bat da. Oro har, pakete honek *Android View* eta *NodeMain* kontzeptuak konbinatzeko eredua definitzen du, datuetan oinarritutako osagaiak (*data driven components*) definituz, hala nola, *RosTextView*. Gainera, paketean *RosActivity* klasea erazagutzen da, Android aplikazioetatik ROS Master batekin konexioa ezartzea errazten duena.

Bibliografia

[1] Jahed Ahmed. maptiles. https://github.com/jahed/maptiles, 2020.

[2] Jose Alcérreca. Viewmodels and livedata: Patterns + antipatterns. https://medium.com/androiddevelopers/21efaef74a54, 2017.

[3] Agile Alliance. Extreme programming. https://www.agilealliance.org/glossary/xp/.

[4] Aitzol Astigarraga Pagoaga and Elena Lazkano Ortega. *Robot mugikorrak. Oinarriak*. Udako Euskal Unibertsitatea, 2011.

[5] Atlassian. What is agile? https://www.atlassian.com/agile.

[6] Atlassian. What is kanban? https://www.atlassian.com/agile/kanban.

[7] Stephen Boddy. Terminator — documentation. https://terminator-gtk3.readthedocs.io/en/latest/, 2017.

[8] Pascal Brachet. Texmaker — free cross-platform latex editor. https://www.xm1math.net/texmaker/.

[9] Jean-Côme Charpentier. dirtree —- display trees in the style of windows explorer. https://ctan.org/pkg/dirtree.

[10] Software Freedom Conservancy. git –local-branching-on-the-cheap. https://git-scm.com/.

[11] OpenStreetMap contributors. Android wiki. https://cloud.google.com/maps-platform/pricing.

[12] ROS contributors. Android tutorial pubsub (indigo). https://github.com/rosjava/android_core/tree/indigo/android_tutorial_pubsub.

[13] ROS contributors. Android tutorial pubsub (kinetic). https://github.com/rosjava/android_core/tree/kinetic/android_tutorial_pubsub.

[14] ROS Wiki contributors. Ros — concepts. http://wiki.ros.org/ROS/Concepts, 2010.

[15] ROS Wiki contributors. Android installation - ros development environment. http://wiki.ros.org/android/Tutorials/kinetic/Installation, 2016.

[16] ROS Wiki contributors. Rosjava source installation. http://wiki.ros.org/rosjava/Tutorials/kinetic/Source%20Installation, 2017.

[17] ROS Wiki contributors. Ros — introduction. http://wiki.ros.org/ROS/Introduction, 2018.

[18] ROS Wiki contributors. Ros — distros. http://wiki.ros.org/Distributions, 2021.

[19] Google Developers. Dropdown menus. https://www.material.io/components/menus/android.

[20] Google Developers. Maps sdk — markers. https://developers.google.com/maps/documentation/android-sdk/marker.

[21] Google Developers. Maps sdk — tile overlays. https://developers.google.com/maps/documentation/android-sdk/tileoverlay.

[22] Google Developers. Maps sdk for android overview. https://developers.google.com/maps/documentation/android-sdk/overview.

[23] Google Developers. Material components. https://material.io/components?platform=android.

[24] Google Developers. Material design — introduction. https://material.io/design/introduction.

[25] Inkscape developers. Inkscape overview. https://inkscape.org/about/.

[26] LyX developers. Lyx — the document processor. https://www.lyx.org/.

[27] Oracle Java Documentation. Lambda expressions. https://medium.com/androiddevelopers/a098a0341ebd.

[28] Firebase. Read and write data on android. https://firebase.google.com/docs/database/android/read-and-write.

[29] Apache Software Foundation. Apache license, version 2.0. https://www.apache.org/licenses/LICENSE-2.0.html.

[30] Open Source Robotics Foundation. Gazebo simulator. http://gazebosim.org/.

[31] Martin Fowler, Jim Highsmith, et al. The agile manifesto. *Software Development*, 9(8):28–35, 2001.

[32] Freepik. Project cover designed by freepik. http://www.freepik.com/.

[33] Inc. Github. Github — where the world builds software. https://github.com/about.

[34] MAGIX Software GmbH. Vegas pro — features. https://www.vegascreativesoftware.com/us/vegas-pro/features/#productMenu.

[35] Walter L Hürsch and Cristina Videira Lopes. Separation of concerns. 1995.

[36] Project Management Institute. *La guía de los fundamentos para la dirección de proyectos (Guía del PMBOK)*. Project Management Institute, 2017.

[37] JetBrains. Intellij idea overview. https://www.jetbrains.com/help/idea/discover-intellij-idea.html, 2021.

[38] Ian Lake. Picking your compilesdkversion, minsdkversion, and targetsdkversion. https://medium.com/androiddevelopers/a098a0341ebd.

[39] Writelatex Limited. Overleaf, online latex editor. https://es.overleaf.com/about.

[40] Mapbox. Maps sdk for android. https://docs.mapbox.com/android/maps/guides/.

[41] MEGA. mega.nz. https://mega.nz/storage.

[42] Jack Melnick. *Oracle XML Developer's Kit Programmer's Guide, 10g Release 1 (10.1)*. Oracle, 2003.

[43] Ltd. MKLabs Co. Staruml overview. http://staruml.sourceforge.net/docs/user-guide(en)/ch01.html.

[44] Oihane Parra, Igor Rodriguez, Ekaitz Jauregi, Elena Lazkano, and Txelo Ruiz. Gidabot: A system of heterogeneous robots collaborating as guides in multi-floor environments. *Intelligent Service Robotics*, 12(4):319–332, 2019.

[45] Google Cloud Platform. Pricing that scales to fit your needs. https://cloud.google.com/maps-platform/pricing.

[46] Android Open Source Project. Accessing your app resources. https://developer.android.com/guide/topics/resources/providing-resources#Accessing.

[47] Android Open Source Project. Android architecture components. https://developer.android.com/topic/libraries/architecture.

[48] Android Open Source Project. Android core app quality guidelines. https://developer.android.com/docs/quality-guidelines/core-app-quality.

[49] Android Open Source Project. Android jetpack. https://developer.android.com/jetpack.

[50] Android Open Source Project. Appcompatactivity. https://developer.android.com/reference/androidx/appcompat/app/AppCompatActivity.

[51] Android Open Source Project. Configure your build. https://developer.android.com/studio/build.

[52] Android Open Source Project. Context. https://developer.android.com/reference/android/content/Context.

[53] Android Open Source Project. Fundamentals of testing. https://developer.android.com/training/testing/fundamentals.

[54] Android Open Source Project. Guide to app architecture. https://developer.android.com/jetpack/guide.

[55] Android Open Source Project. Lifecycle components. https://developer.android.com/topic/libraries/architecture/lifecycle.

[56] Android Open Source Project. Manage network usage. https://developer.android.com/training/basics/network-ops/managing.

[57] Android Open Source Project. Meet android studio. https://developer.android.com/studio/intro.

[58] Android Open Source Project. Parse xml data. https://developer.android.com/training/basics/network-ops/xml.

[59] Android Open Source Project. Save data in a local database using room. https://developer.android.com/training/data-storage/room.

[60] Android Open Source Project. Save data using sqlite. https://developer.android.com/training/data-storage/sqlite.

[61] Android Open Source Project. Sign your app. https://developer.android.com/studio/publish/app-signing.

[62] Android Open Source Project. Spinners. https://developer.android.com/guide/topics/ui/controls/spinner.

[63] Android Open Source Project. Support different languages and cultures. https://developer.android.com/training/basics/supporting-devices/languages.

[64] Android Open Source Project. Toasts overview. https://developer.android.com/guide/topics/ui/notifiers/toasts.

[65] Android Open Source Project. Transformations. https://developer.android.com/reference/androidx/lifecycle/Transformations.

[66] Igor Rodriguez, Unai Zabala, Pedro A. Marín-Reyes, Ekaitz Jauregi, Javier Lorenzo-Navarro, Elena Lazkano, and Modesto Castrillón-Santana. Personal guides: Heterogeneous robots sharing personal tours in multi-floor environments. *Sensors*, 20(9), 2020.

[67] Nils Rottmann, Nico Studt, Floris Ernst, and Elmar Rueckert. Ros-mobile. https://github.com/ROS-Mobile/ROS-Mobile-Android, 2020.

[68] Nils Rottmann, Nico Studt, Floris Ernst, and Elmar Rueckert. Ros-mobile: An android application for the robot operating system. *arXiv preprint arXiv:2011.02781*, 2020.

[69] Adena Schutzberg. Ten things you need to know about indoor positioning. https://www.directionsmag.com/article/1598, 2013.

[70] Scrum.org. What is scrum? https://www.scrum.org/resources/what-is-scrum.

[71] Amazon Web Services. Aws robomaker: Developer guide. https://docs.aws.amazon.com/robomaker/latest/dg/aws-robomaker-dg.pdf#simulation-tools-rviz.

[72] Wolfgang Skala. pgfgantt —- draw gantt charts with tikz. https://ctan.org/pkg/pgfgantt.

[73] SubPos.org. Subpos android api. https://github.com/subpos/subpos_android_api.

[74] SubPos.org. Subpos positioning system. https://hackaday.io/project/4872-subpos-positioning-system.

[75] Maruti Techlabs. 11 best prototyping tools for ui/ux designers — how to choose the right one? https://medium.theuxblog.com/c5dc69720c47, 2018.

[76] teejee2008. Timeshift. https://teejee2008.github.io/timeshift/.

[77] Wikipedia contributors. Indoor positioning system — Wikipedia, the free encyclopedia. https://w.wiki/3VZ6, 2021. [Online; accessed 16-June-2021].

[78] Wikipedia contributors. Model–view–viewmodel — Wikipedia, the free encyclopedia. https://w.wiki/3NUH, 2021. [Online; accessed 23-May-2021].

[79] Wikipedia contributors. Notion (productivity software) — Wikipedia, the free encyclopedia. https://w.wiki/3Qhn, 2021. [Online; accessed 30-May-2021].

[80] Wikipedia contributors. Publish–subscribe pattern — Wikipedia, the free encyclopedia. https://w.wiki/3NUG, 2021. [Online; accessed 23-May-2021].