Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERIA DE BILBAO

INDUSTRIA INGENIARITZA TEKNIKOKO ATALA
SECCIÓN INGENIERÍA TÉCNICA INDUSTRIAL

GRADO EN INGENIERÍA EN INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

2015 / 2016

ZAINTZA PI
Sistema de detección de intrusos mediante cámara

MEMORIA

DATOS DE LA ALUMNA O DEL ALUMNO		DATOS DEL DIRECTOR O DE LA DIRECTORA	
NOMBRE	AITOR	NOMBRE	OSCAR
APELLIDOS	ECHEZARRAGA PORTO	APELLIDOS	CASQUERO OYARZABAL
DNI		DEPARTAMENTO	INGENIERÍA DE SISTEMAS Y AUTOMÁTICA
FDO.:		FDO.:	
FECHA:		FECHA:	

Anexo II

1. RESUMEN

Con la finalidad de evitar sistemas de vigilancia caros, la idea es construir un sistema de vigilancia mediante video con nuestros medios, de manera sencilla y barata, y cumpliendo todas las funciones más relevantes como una captura de imágenes, un sensor de movimiento, un sistema de avisos y una manera remota de controlar el sistema y gestionar los avisos. El sistema que queremos construir deberá ser accesible desde cualquier punto mediante un dispositivo Android que cuente con conexión de datos o WiFi. Además, el sistema deberá de ser capaz de almacenar las capturas que haga en una cuenta Dropbox.

El dispositivo Android contará con una serie de funcionalidades accesibles. Por un lado, desde el dispositivo tendremos la opción de revisar capturas antiguas almacenadas en la cuenta de Dropbox, con la posibilidad de eliminarlas −tanto del dispositivo como de la cuenta−. También podremos solicitar una imagen en tiempo real de lo que la cámara de vigilancia está viendo en ese momento. Además, podremos decidir si queremos que el sensor de movimiento se ponga en modo automático para la detección de intrusos, o si por el contrario lo queremos en modo manual donde el usuario decide (mediante la opción anterior) cuándo queremos que nos enseñe una imagen; Por ejemplo, para la vigilancia de bebes no querríamos que nos mande una imagen cada vez que se mueve, sino revisar su estado de vez en cuando, cuando nos parezca oportuno. Otra de las funcionalidades, aunque más orientada a la configuración, será la opción de modificar desde qué red queremos acceder al sistema, para usar una IP interna (si estamos dentro de la misma red que el sistema), o para usar la dirección DNS configurada para acceder al sistema (si estamos fuera de la red). Para finalizar con las funcionalidades, el dispositivo contará con alertas. Cuando el sistema está en modo automático y detecta un movimiento, envía una alerta −a modo de notificación− al dispositivo Android asociado a la misma cuenta Dropbox que el sistema. Esta acción se realiza con el fin de que el usuario pueda abrir la notificación y visualice la imagen detectada por el sistema, y de este modo, pueda decidir cómo proceder. Además, con la intención de ampliar la accesibilidad, la notificación lleva incorporado un largo periodo de vibración para que −tanto usuarios sin problemas de audición como aquellos con problemas de audición− queden alertados de la intrusión.

Para completar el proyecto, primero hemos desarrollado la idea de forma conceptual mediante gráficos, para asegurarnos que cumplíamos de forma lógica con las funcionalidades y que estas eran viables. Después de diseñar y analizar las tareas y aplicaciones a desarrollar, comenzamos con la programación de manera modula con el fin de mejorar la usabilidad del código y que, además, resulte de fácil lectura.

2. ÍNDICE DE CONTENIDOS

3. ÍNDICE DE IMÁGENES

4. ÍNDICE DE TABLAS

5. INTRODUCCIÓN

La finalidad de este Trabajo Fin de Grado (TFG) es desarrollar un sistema de vigilancia mediante video para la detección de intrusos. En este sistema se contemplan los siguientes apartados:

- **Hardware instalado en la ubicación a vigilar, basado en una plataforma Raspberry Pi (RPi).** El sensor de presencia se conectará a los pines GPIO de la RPi y la cámara al puerto CSI. Además, las llamadas procedentes de fuera de la red de la RPi en formato REST tendrán una DNS dinámica y un servidor web al que mandar las órdenes.
- **Servicio Dropbox utilizado como SaaS (Software as a Service)**. Dropbox se utilizará como contenedor privado de imágenes, sacadas por la RPi.
- **Cliente: Aplicación Android**. Este cliente notificará del movimiento mediante alerta, dará acceso a la imagen, y permitirá gestionar tanto la alerta como la imagen. Además, el usuario podrá elegir entre un control manual o uno automático.

Funcionamiento:

- **Modo automático.** Cuando el usuario sale de casa activa el sistema mediante la aplicación Android. Cuando se detecta la presencia de un intruso, la cámara saca una foto y la sube a la cuenta Dropbox del usuario. Cuando esto sucede, el usuario recibe una notificación que le permite visualizar la imagen sacada por cámara.
- **Modo manual.** El usuario tiene la posibilidad de solicitar a la RPi que saque una foto en el momento que lo desee.

Diagrama de bloques:

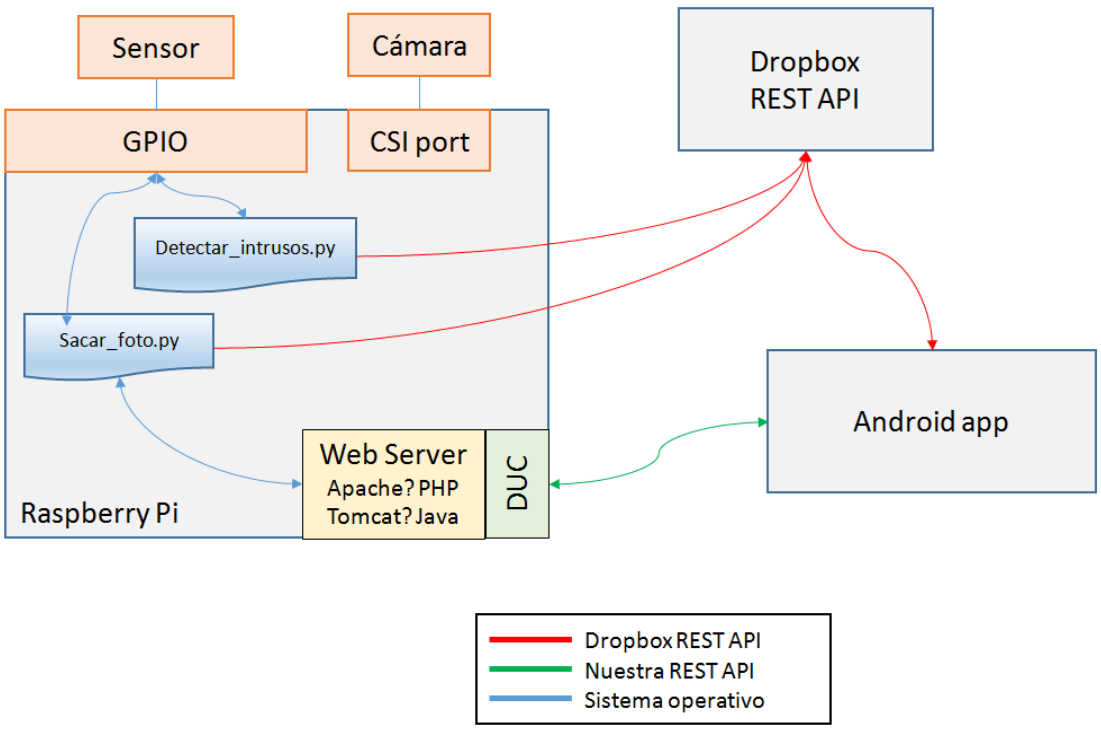


Ilustración 1. Diagrama de bloques

6. PLANTEAMIENTO INICIAL

El proyecto tiene como objetivo realizar un sistema de video vigilancia con costes mínimos y calidad óptima, y que además cuente con la posibilidad de controlar el sistema desde un dispositivo Android.

Para el proyecto se ha utilizado la RPi dado que éste es un ordenador de plataforma reducida OSHW de bajo coste y que permite abordar proyectos como éste relacionados con el Internet de las Cosas (IoT, por sus siglas en inglés).

En primer lugar, el proyecto ha involucrado un proceso de aprendizaje instalación de una plataforma RPi. Se revisaron las funcionalidades posibles, tanto del Hardware como de los distintos Software, así como los Sistemas Operativos disponibles para instalar en él. El proyecto también ha involucrado informarse y aprender acerca del control de la cámara que va conectada al puerto CSI mediante scripts de Python, y acerca del control del sensor de movimiento conectado vía pines de la GPIO de la RPi mediante scripts Python.

A continuación, se han configurado la RPi para conectarse vía WiFi y el dominio DNS necesarios para conectarnos desde fuera de la red. En este paso hemos personalizado los puertos del router mediante Port Forwarding con el fin de dirigir las conexiones desde el exterior a los puertos de interés. Para finalizar, hemos configurado la RPi para que tanto el script de Python que controla el sensor de movimiento como el encargado de sacar una foto sean capaces de subir una imagen a una cuenta de Dropbox y almacenarla en ella. La configuración de esta cuenta se realizará mediante una web que desarrollaremos.

A continuación, se procede a explicar el diseño y desarrollo de la aplicación Android, para la cual necesitaremos una amplia formación.

En primer lugar, hemos diseñado sobre papel un boceto inicial de la aplicación y lo hemos trasladado a un entorno informático de diseño. En este caso, se emplea Ninjamock, ya que permite mejorar y asentar las funcionalidades y conexiones entre elementos. Los colores se eligen en base a un estándar [18] para que ningún usuario tenga problemas con la lectura y siguiendo un patrón a la hora de diseñar las paginas. En segundo lugar, hemos realizado un diagrama de clases y otro de secuencia, analizando de este modo las funcionalidades y variables necesarias, así como el orden de ejecución. El diseño se llevará a cabo mediante Visual Paradigm. Finalmente con todo bien estructurado comenzaremos la programación en Android de nuestro proyecto.

Una vez finalizados los pasos anteriores, hemos comenzado con el periodo de pruebas. Este periodo se ha utilizado para observar y corregir los posibles errores que pueden haber pasado desapercibidos con anterioridad por diversos motivos, como por ejemplo la falta de conocimiento. Los comentarios de los usuarios que han probado la aplicación nos ha llevado a mejorar el diseño.

En el proyecto se prevé el riesgo de tener unos conocimientos en Android limitados, con lo que la formación y las ideas en cuanto al diseño de las clases podrían verse afectadas. Se espera que se necesitará una gran cantidad de horas para la formación con el fin de desarrollar la idea lo mas a fin posible a lo planteado. Como planteamiento inicial de los bloques a realizar, se prevé una cuantía de horas detalladas en la siguiente tabla:

Tabla 1. Planificación de horas por bloques

Bloque	Horas
SW de Raspberry Pi	30h
Cámara	4h
PIR	8h
Python Cámara	4h
Python PIR	4h
Python Dropbox-uploader	10h
Proyecto Dropbox	2h
Python Cámara + PIR + Dropbox	4h
Elegir tipo de servidor web	4h
Configuración servidor web	20h
Configuración DUC	4h
Aplicación Android	90h
Pruebas	40h
Documentación	60h
Reuniones con el tutor	20h
TOTAL	304h

Estos bloques se dividen en tareas, y las horas previstas incluirán las horas de formación y las posibles dificultades y problemas que podrían plantearse.

Dado que para la resolución del TFG necesitaremos un software y un hardware concretos, hemos optado por un software gratuito para evitar compras y gastos innecesarios y así poder ajustarnos mejor al presupuesto.

Se prevé el siguiente software:

Tabla 2. Software previsto

Software	Función
PuTTY	Conexión SSH con la Raspberry Pi para control remoto sin necesidad de entorno visual
DUC NO-IP	DNS para IP externa para la RPi
WebIDE	Entorno remoto de programación para la RPi
GitHub	Almacenamiento de código Android
Bitbucket	Almacenamiento de código de WebIDE
Android Studio	Programación de aplicación Android
Dropbox	Creación de proyecto y almacenamiento de documentos
Dropbox-uploader	Conexión entre la RPi y Dropbox
Tomcat8	Servidor Web
VNC	Visualización remota del escritorio de la RPi

Se usará el siguiente hardware:

Tabla 3. Hardware previsto

Hardware	Función	Dónde comprar
Raspberry Pi	Plataforma OSHW sobre la que montaremos el proyecto	https://www.sparkfun.com/products/12994
Cámara	Sacar las fotos	https://www.sparkfun.com/products/11868
Adaptador WiFi	Conexión sin cables a la red	http://www.cetronic.es/sqlcommerce/disenos/plantilla1/seccion/producto/DetalleProducto.jsp?idIdioma=&idTienda=93&codProducto=255191102&cPath=1358
PIR	Sensor de movimiento	https://www.sparkfun.com/products/8630
Jumper	Conexión de pines de PIR con RPi	https://www.sparkfun.com/products/9140
Breadboard	Pruebas de conexiones de pines	https://www.sparkfun.com/products/9567

Para el desarrollo del TFG, se tuvieron que tener en cuenta los antecedentes, es decir, conocer lo que el mercado ofrece en el momento de comenzar el proyecto, y analizar qué es lo que nosotros podíamos aportar como novedad.

7. ANTECEDENTES

Tal y como se ve en la tabla 4, en el mercado hay proyectos desarrollados con algunas características similares a Zaintza Pi, pero ninguno de ellos integra todas las funcionalidades que ofrece este proyecto en un solo producto. Además, los proyectos que son más completos tienen como desventaja un precio elevado en comparación con el que desarrollaremos nosotros.

Ponemos en esta comparativa un par de modelos de la misma compañía (i.e., D-Link), para analizar las mejoras que hacen de una versión otra, así como dos modelos de otras compañías con el fin de comparar las diferencias entre ellas.

Las dos versiones de D-Link (i.e., 8201 y 9301) tienen un sensor y cámara de buena calidad, pero observamos que las alertas solo son vía email al usuario, sin dar acceso al sistema desde el exterior. La mejora de una a otra es que el sensor permite menos luminosidad a la hora de detectar movimiento. Por otro lado, aunque el modelo más nuevo vuelve a la conexión de transformador, nosotros opinamos que esta conexión es una desventaja mientras que la conexión USB es un avance. Aunque ambas son opciones muy completas, creemos que la mayor desventaja de estas es que no teniendo

acceso remoto al sistema, el almacenamiento de las imágenes se haga en la tarjeta del propio dispositivo.

Conceptronic integra una mejora en comparación con los modelos de D-Link y es que el almacenamiento se hace en la nube, con lo que aun siendo las alertas vía email y no tengamos acceso al sistema desde el exterior, si tendremos posibilidad de revisar capturas anteriores.

El último modelo que observamos es la Cámara IP Cloud, que como mejora de los anteriores, integra un software compatible con Android y IOS, dando así acceso al sistema de forma remota y permitiendo también acceder a capturas anteriores dado su almacenamiento en la nube.

Zaintza Pi, integra las funcionalidades que nos parecen claves, es decir, un sensor por infrarrojos, una alimentación por USB, un control remoto vía aplicación Android y un almacenamiento en Dropbox.

Tabla 4. Comparativa de mercado

	D-Link 8201	**D-Link 9301**	**Conceptronic**	**Cámara IP Cloud**	**Zaintza Pi**
Sensor	CMOS	CMOS	CMOS	CMOS	PIR
Alimentación	USB	Transformador	Transformador	Transformador	USB
Almacenamiento	Tarjeta	Tarjeta	Nube	Nube	Dropbox
Alertas	Correo	Correo	Correo	Aplicación propia	Aplicación propia
Acceso remoto	No	No	No	Si	Si
Configuración de red	Si	Si	Si	Si	Si

8. CAPTURA DE REQUISITOS

El proyecto tendrá que cumplir ciertos requisitos que son imprescindibles a la hora de desarrollar un sistema de vigilancia. Estos requisitos se describen a continuación.

Primero, un sistema de vigilancia quedaría muy restringido si no tuviese la posibilidad de tomar imágenes. Segundo, no tendría mucho sentido un sistema de vigilancia que obligue al usuario a estar en el mismo sitio que el aparato; Esto nos lleva al siguiente requisito, el de proporcionar un modo de controlar remotamente el aparato; En nuestro caso, sería una aplicación Android desde la cual tendríamos acceso al sistema. Tercero, hemos observado que es muy importante que un sistema de vigilancia sea capaz por sí mismo de detectar intrusos, por lo que incluir un sensor es otro requisito del sistema de vigilancia.

Finalmente el proyecto deberá ser lo más barato posible y al mismo tiempo cumplir los requisitos así como dar acceso a la mayor parte de los interesados. Es por este motivo por el que hemos optado por Android y Dropbox, ya que ambas son las plataformas más usadas en su sector y son de uso gratuito, evitando así la necesidad de licencias para el desarrollo Android y de cuentas de pago para el almacenamiento de Dropbox. Hay que

tener en cuenta que Dropbox tiene versión de pago y que la versión gratuita tiene límite de almacenamiento. No obstante, la capacidad de ésta versión gratuita es suficiente para satisfacer nuestras necesidades de almacenamiento.

En conclusión, el proyecto deberá 1) abarcar la mayor parte del mercado Android, 2) desarrollar el sistema de vigilancia de la manera más rentable, y 3) ofrecer detección de intrusos, captura de imágenes, almacenamiento en Dropbox y una aplicación Android para controlar remotamente.

9. ANALISIS Y DISEÑO

El diseño de la aplicación de Android se ha desarrollado en 7 pantallas mediante la aplicación Ninjamock [20] :

1. Inicio
2. Pantalla principal
3. Configuración de la conexión
4. Listado de capturas
5. Captura, desde el botón de sacar foto
6. Captura, desde elemento de la lista
7. Lista de capturas, después de borrar una captura

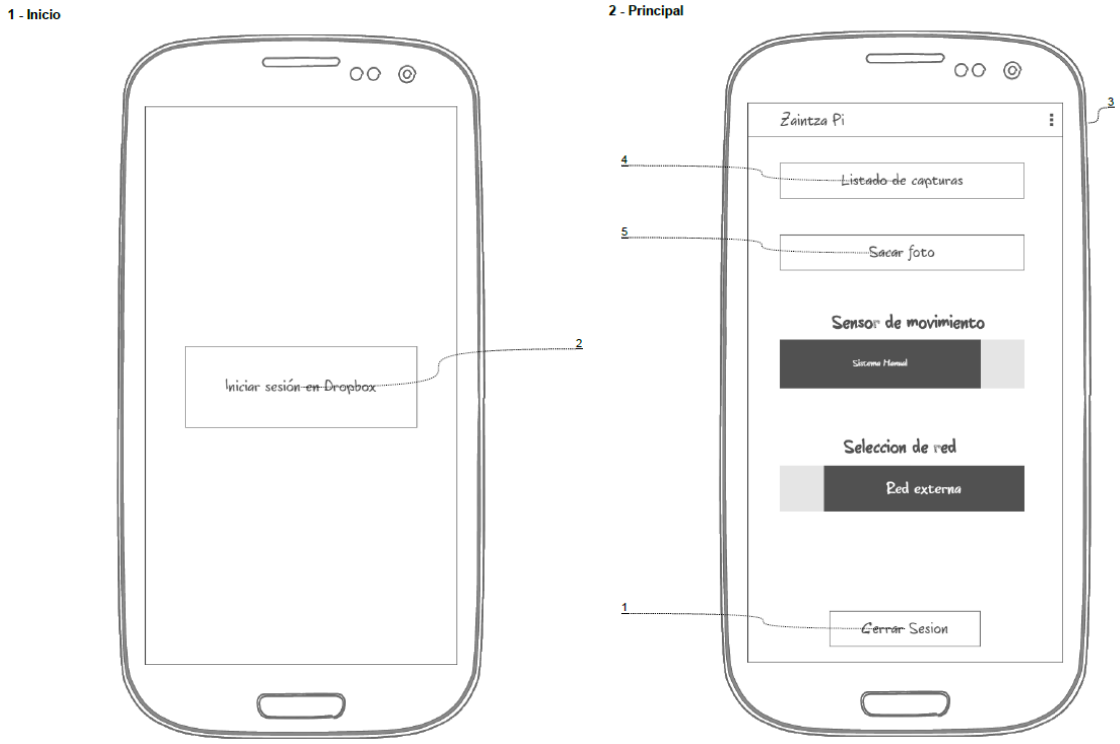


Ilustración 2. Pantalla inicio y Ilustración 3. Pantalla principal

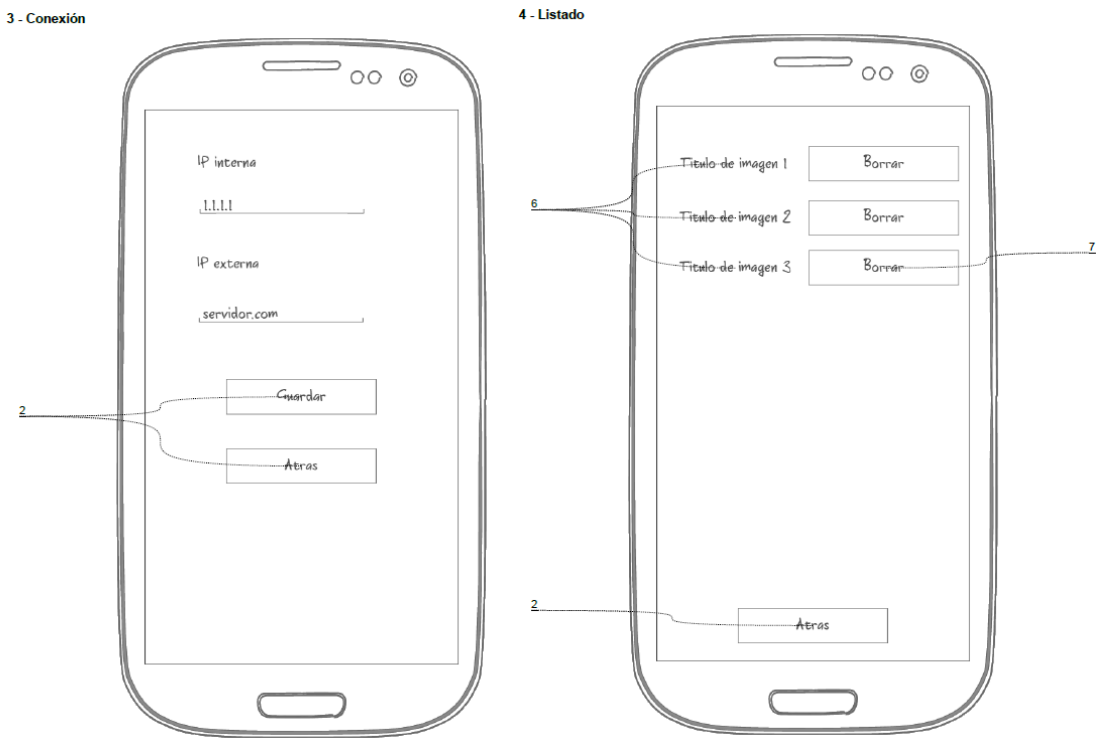


Ilustración 4. Pantalla conexiones y Ilustración 5. Pantalla listado

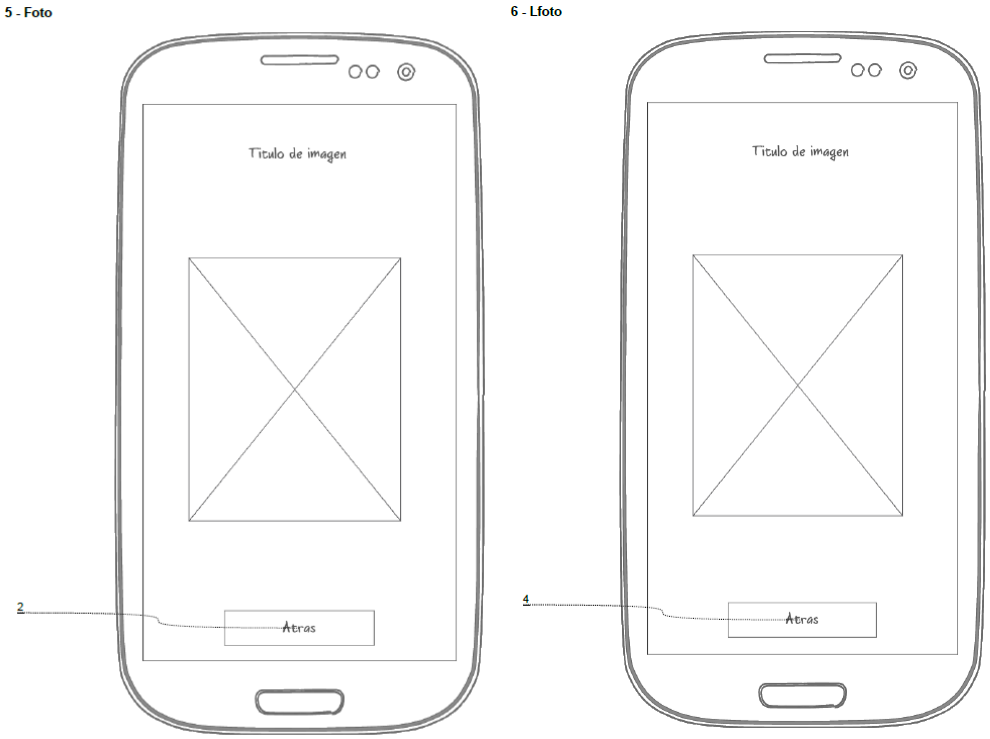


Ilustración 6. Pantalla foto y Ilustración 7. Pantalla foto desde listado

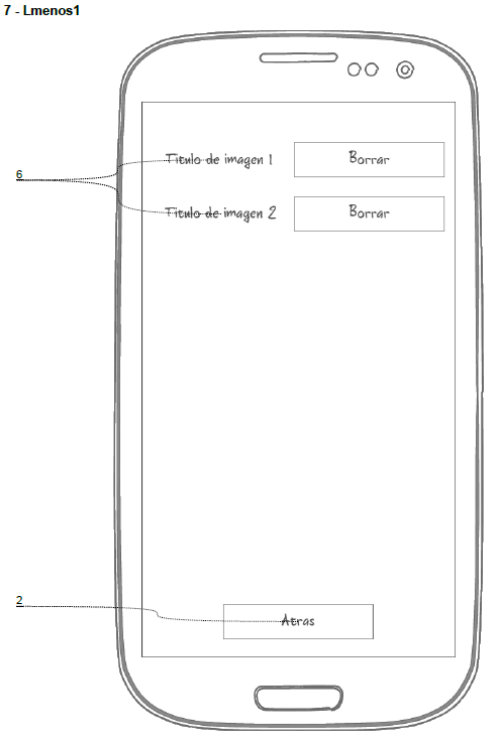


Ilustración 8. Pantalla listado con 1 elemento menos

Diagrama de clases:

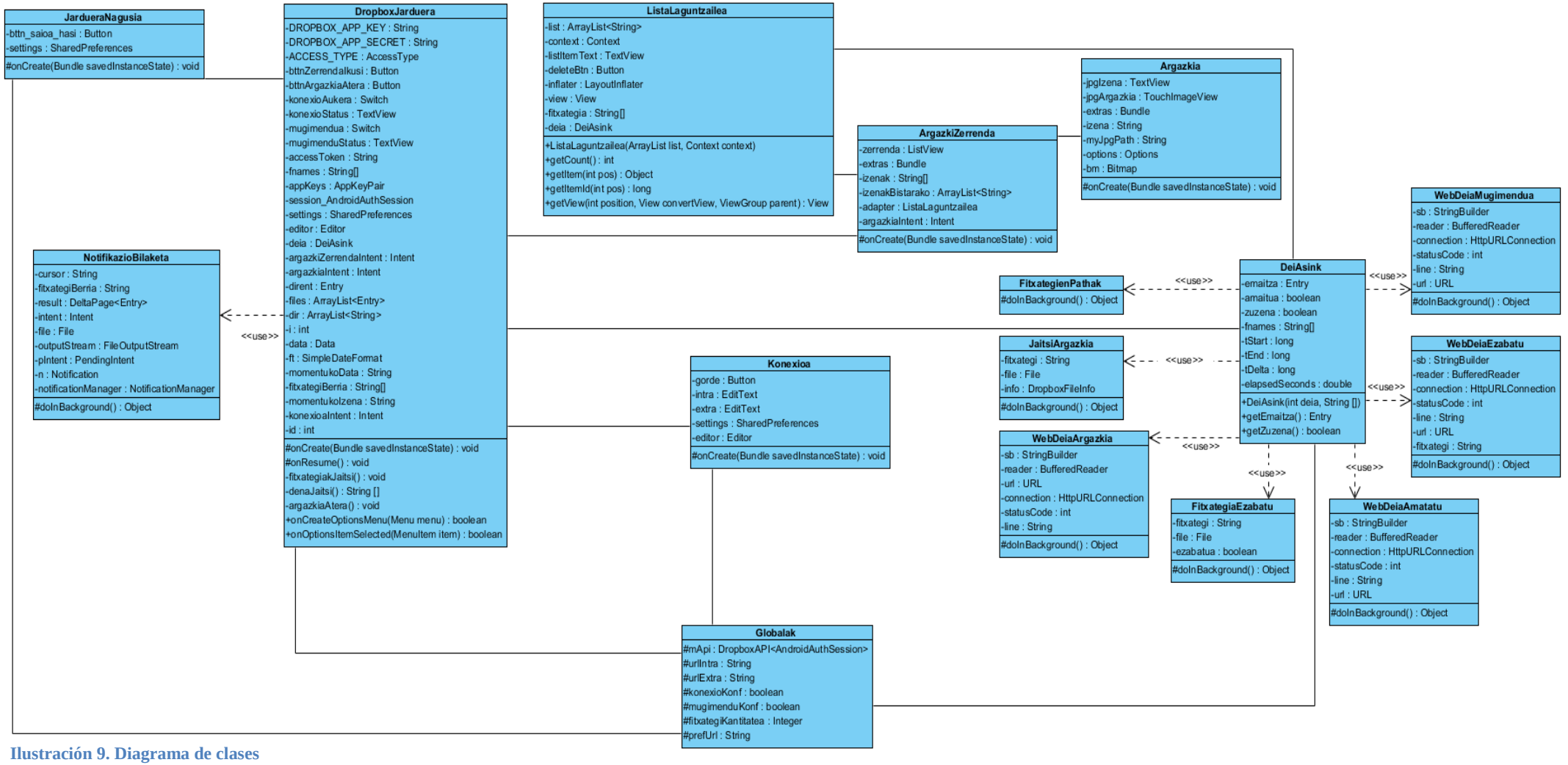


Ilustración 9. Diagrama de clases

Diagrama de secuencia:

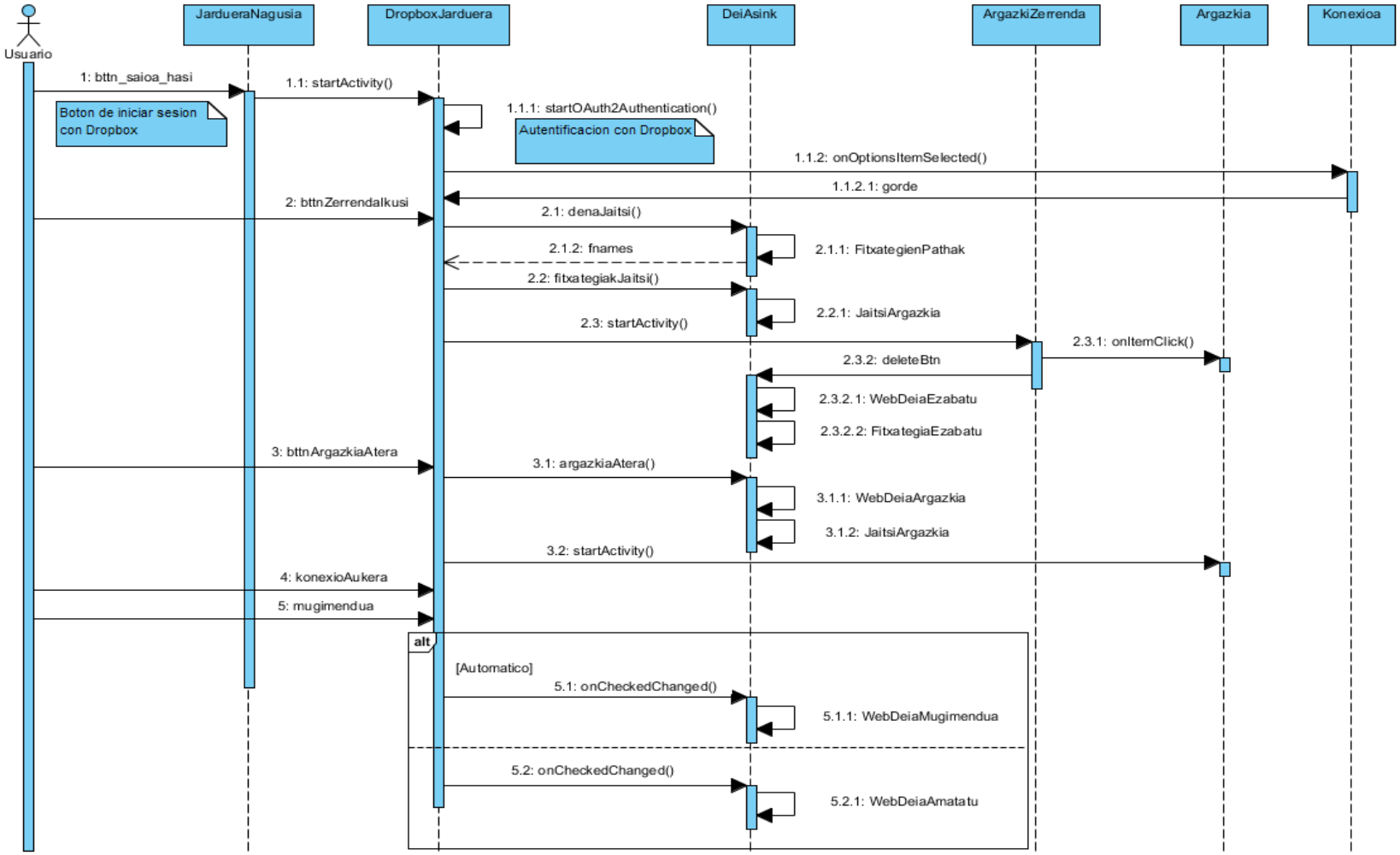


Ilustración 10. Diagrama de secuencia

10. DESARROLLO

Iniciamos el proyecto por la parte hardware. Primero, observamos la variedad de sistemas que se le pueden meter a la OSHW RPi y, en función de lo analizado, nos decantamos por un Sistema operativo (SO) libre. Este, integra RPi y Debian, facilitando así la comodidad de trabajar en él. El SO elegido es Raspbian. Dado que vamos a empezar a probar cómo funciona el sistema, lo primero para las primeras pruebas e instalaciones para utilizar la RPi con comodidad es conectarle la pantalla por HDMI, el teclado, el ratón, y una conexión por cable de Ethernet. A posteriori, observamos que para aprovechar más recursos de la RPi sería más rentable que el sistema no tuviera entorno gráfico, ya que de este modo, el sistema sería más ligero. Además hemos decidido usar el sistema mediante SSH con un picho WiFi, y así solo tener el cable de alimentación como limitación de movilidad.

Las primeras pruebas en la RPi se realizaron con el SO, analizando los programas que incluía y la organización de las carpetas y los usuarios para poder trabajar desde SSH, conociendo esta estructuración.

10.1. WiFi

Lo primero que nos interesa es no tener la RPi conectada por cable Ethernet al router, ya que por distancia el entorno de trabajo quedaba alejado. De este modo, lo que hicimos fue formarnos para poder instalar el adaptador de WiFi y así, desplazarnos luego a nuestro entorno para trabajar más cómodamente.

La instalación del pincho es sencilla, simplemente tenemos que insertarlo en una entrada de USB, pero aun quedaría la configuración para establecerle redes a las que conectarse. En nuestro caso, nos pareció interesante establecer una IP estática para poder localizar la RPi dentro de la red de manera sencilla y sin perder la IP con el reinicio del router. Para ello, modificamos los ficheros “wpa_supplicant.conf” y “interfaces”, localizados en “/etc/wpa_supplicant/” y “/etc/network/” respectivamente.

En el primer fichero (i.e., “wpa_supplicant.conf”), definiremos las redes a las que nos interese poder conectar la RPi, cada una con un valor en el parámetro id_str para poder localizarlas luego en el segundo fichero. La configuración de cada red tendrá que seguir el siguiente ejemplo (Ver Ilustración 11):

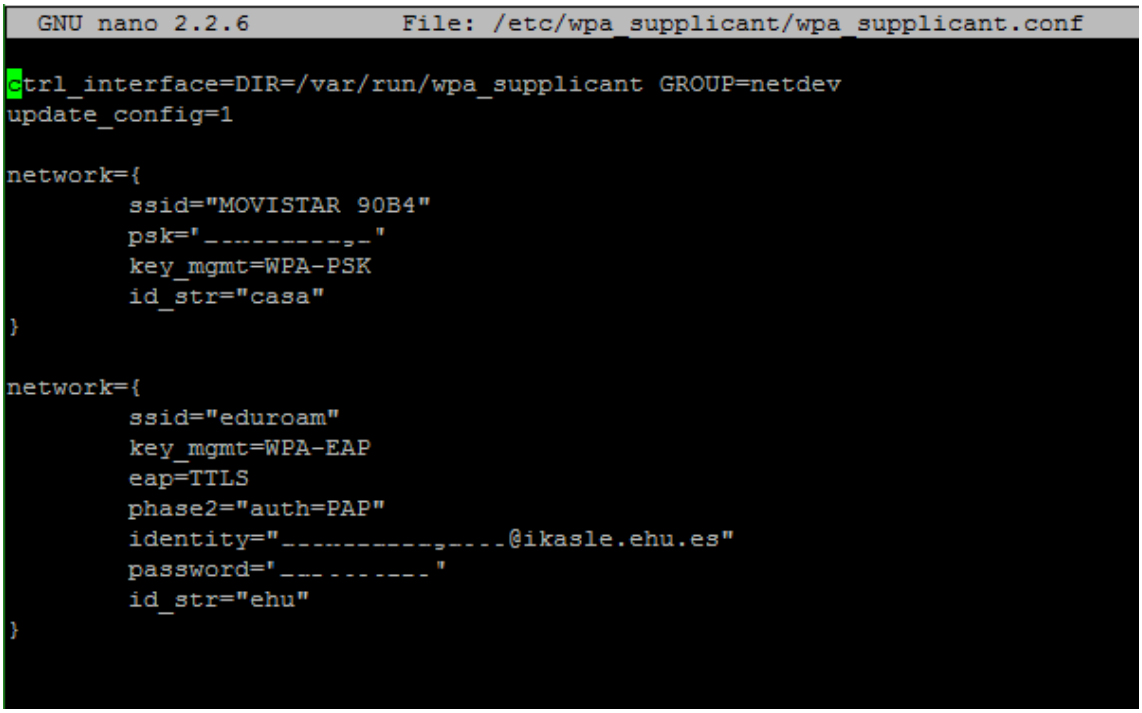


Ilustración 11. WiFi fichero 1

En el segundo fichero (i.e., "interfaces"), estableceremos las propiedades de los siguientes tipos de redes: local (localhost), cableada (eth0, eth1… en el caso de RPi solo habría el 0, ya que no tiene más entradas de red Ethernet) y WiFi (wlan0, wlan1… dependiendo de cuantos adaptadores de conexión se conecten). En nuestro caso, como se ha mencionado anteriormente, nos interesaba una IP estática para la conexión de "casa" tanto para localizarla nosotros en la red la RPi, como para hacer luego el Port Forwarding para la conexión desde fuera de la red. Elegimos la IP 192.168.1.55. Seguimos el ejemplo adjuntado (Ver Ilustración 12) para la configuración de las redes. En el caso de establecer IP estática, la configuración requiere más datos, pero en la de "ehu" simplemente se le dice que le pida la IP al DHCP.

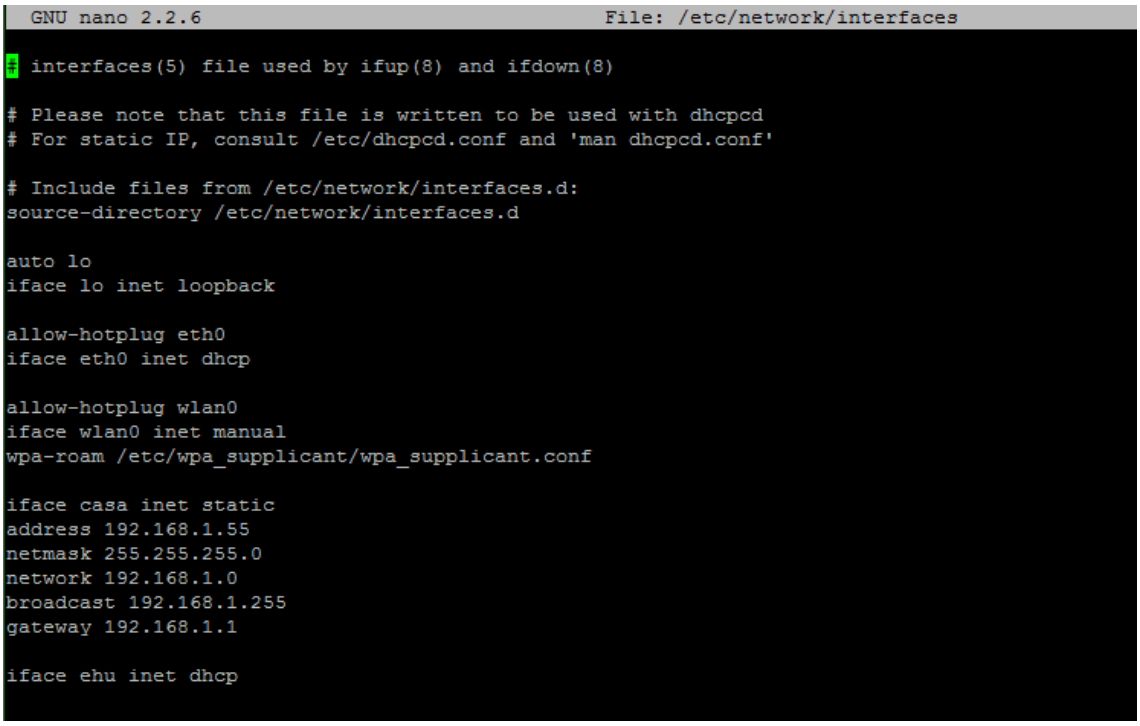


Ilustración 12. WiFi fichero 2

Con el objetivo de informar acerca de cómo funciona la configuración de red de la RPi: en el fichero "interfaces", se indica cómo se debe arrancar cada interfaz: la primera línea indica que se debe arrancar automáticamente en caliente; la segunda (i.e., iface wlan0 inet manual) que no se le debe asignar IP (lógico, porque todavía no se ha autenticado en la red a nivel físico); la tercera indica que se debe usar un comando externo (i.e., wpa-roam) para gestionar la autenticación con la información indicada en "wpa_supplicant.conf". Desde este punto se pasa al "wpa_supplicant", se realiza la autenticación en la red y se vuelve a interfaces a la sección señalada como id_str, donde se indica que la IP se debe asignar de forma estática según la información que se detalla a continuación.

El hecho de establecer una IP estática le impide a la RPi conectarse a la red con los nombres de las páginas. Por este motivo, necesitaríamos es conocer las IP de las páginas a las que queremos acceder. Esto sucede debido a no pedirle la IP al DHCP y definir nosotros una estática. Por ello, con el fin de solucionar esto, añadiremos en el fichero "resolv.conf" localizado en "etc" la IP del DNS de Google con IP 8.8.8.8. Usaremos el siguiente comando:

```
echo "nameserver 8.8.8.8" | sudo tee /etc/resolv.conf
```

Pero éste solo evitaría el problema mientras la RPi no se reinicia, ya que con cada reinicio este fichero se vacía. Por ello, después de comprobar que no tendría efecto negativo en nada, tomamos la decisión, de bloquear el fichero para protegerlo de modificaciones con el siguiente comando:

```
sudo  chattr +i /etc/resolv.conf
```

Una vez hemos realizado lo anterior, reiniciamos la RPi y quitamos el cable de Ethernet, dejando sólo el adaptador puesto. Es recomendable hacer toda esta configuración entera desde el principio, ya que una vez hecho esto ya no necesitamos tener conectados pantalla, teclado y ratón, y podremos conectarnos vía SSH mucho más cómodamente. A pesar de que al principio usamos VNC para trabajar en el escritorio remotamente por miedo a la pantalla negra de SSH, fuimos pasándonos poco a poco al control mediante SSH, a medida que conseguíamos más conocimientos y que resultaba más fácil usar los comandos, hasta el punto en el que decidimos que no era necesario para el desarrollo del proyecto el VNC.

Esta configuración de red y el hecho de establecer una IP estática son unos pasos que hay que dejar bien establecidos al inicio, ya que nosotros lo aprendimos a base de perder la IP de la RPi en la red varias veces y teniendo que volver a enchufarle pantalla, ratón, teclado y cable de Ethernet otra vez para saber cuál era la nueva IP que había cogido o si estaba conectada a la red siquiera. Para la búsqueda de la IP, en la red encontramos una aplicación para el móvil llamada “Fing”, la cual facilitaba en redes pequeñas el descarte de IP para encontrar la nuestra, pero resultaba muy tedioso si no se conocían las IP conectadas o si había muchas en el momento de la búsqueda.

10.2. VNC

Segundo en la configuración de la RPi vemos útil la visualización remota del escritorio. A pesar de ello, tal y como se ha mencionado anteriormente, luego vimos que no sería necesario. No obstante, por si resultase interesante para próximos desarrolladores, a continuación se indican los pasos a seguir porque vemos que sería el momento idóneo para decidir si instalarlo o no, dependiendo de cómo nos manejemos con la consola de comandos.

Nos conectaremos con PuTTY mediante SSH a la RPi para poner el siguiente comando:

```
sudo apt-get install tightvncserver
```

Mediante este comando instalaremos el servidor de VNC para poder conectarnos desde en cliente de VNC desde otro ordenador sabiendo la IP de la RPi y el puerto que estableceremos al arrancar el servidor. En el siguiente comando se indica cómo arrancar el servicio y que el número 1 puesto después de los dos puntos, es el puerto que luego necesitaremos recordar para la conexión remota:

```
vncserver :1 -geometry 1920x1080 -depth 24
```

La configuración y el servicio estarían listos, pero como información adicional, hay que tener en cuenta que la contraseña del VNC se podría modificar sin saber la anterior mediante el comando `vncpasswd` y siguiendo los pasos que nos va mandando la consola.

10.3. WebIDE

Tercero, nos parece interesante dejar preparado en este punto el entorno sobre el que remotamente podremos programar los scripts de Python, guardarlos y ejecutarlos sobre

la RPi. Esto lo haremos mediante WebIDE, que a su vez nos pedirá que nos registremos en Bitbucket para el almacenamiento de lo que desarrollemos. Bitbucket es un servicio como GitHub que nos permite hacer versiones de nuestro código y almacenarlo en la nube, pudiendo acceder a él desde cualquier conexión.

Dado que al principio no encontrábamos algo que se asemejase a lo que buscábamos, una manera remota de programar y ejecutar en la RPi scripts de Python, la elección de este entorno nos llevo mucho tiempo de investigación y búsqueda en la red. Al final, conseguimos éste, (i.e., WebIDE) encontrado en "Adafruit", una empresa que diseña y fabrica sus propios circuitos OSHW y vende productos a terceros (como la RPi y Arduino) y fabrica accesorios para ellos. Siguiendo los pasos de la URL[1] conseguimos dejar preparado el entorno. Aun así, indicaremos los pasos para facilitar la lectura de aquellos que no se manejen en inglés.

Conectados mediante SSH descargamos y ejecutamos el instalador mediante el siguiente comando:

```
curl https://raw.githubusercontent.com/adafruit/Adafruit-WebIDE/alpha/scripts/install.sh | sudo sh
```

Una vez realizado éste paso, reiniciamos la RPi y cuando esta esté lista, ponemos en el navegador de otro ordenador, que esté en la misma red que la RPi, la IP interna que le pusimos de estática a la RPi. Por defecto, se pondrá en el puerto 80 de la RPi, pero como desde fuera de la red no querremos que el entorno de desarrollo sea accesible, ponemos mediante Port Forwarding que el puerto 80 desde el exterior apunte al 8080, será donde estará el servidor web a configurar más adelante.

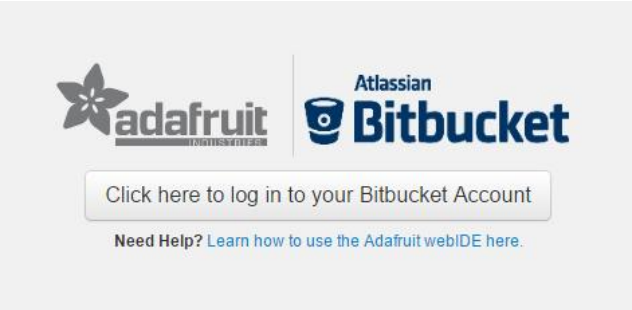


Ilustración 13. Página inicial después de instalar WebIDE

Cuando hayamos entrado en el navegador, en el IP de la RPi nos pedirá que nos registremos en Bitbucket. Esta página es como GitHub, nos permite almacenar, hacer revisiones y ramas de nuestro proyecto. Una vez registrados, iremos a nuestro perfil y elegiremos "configurar la cuenta" (Bitbucket settings), luego en la columna de la izquierda nos desplazaremos hasta la sección de "gestión de acceso" (Access management) y haremos clic en la opción de "OAuth". A continuación, clicaremos en la opción de "añadir un nuevo consumidor" (Add consumer) y rellenaremos el formulario que aparecerá. Este proceso nos dará dos claves, una clave publica y una privada.

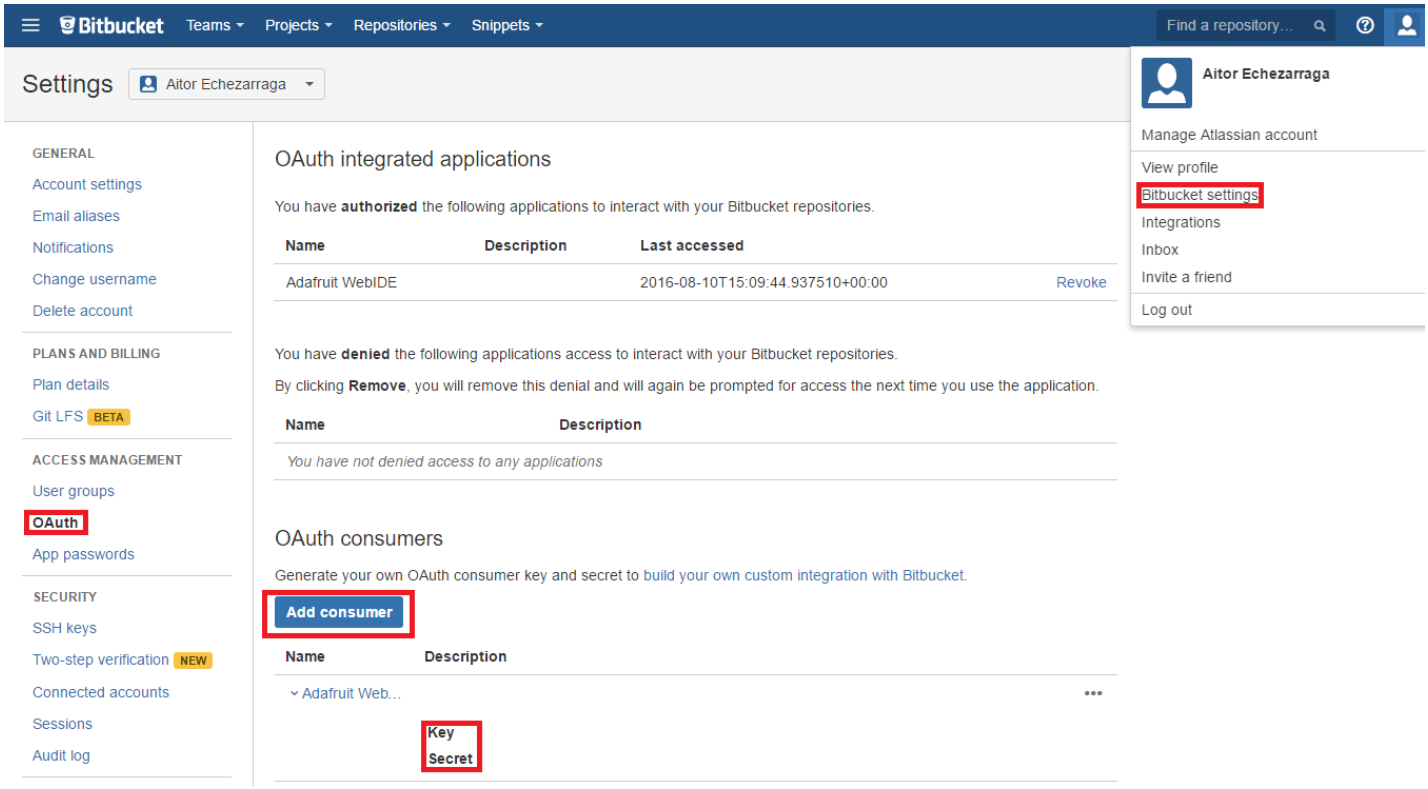


Ilustración 14. Pasos a seguir en Bitbucket

Deberemos llevar estas claves a la primera página, donde entramos con la IP de la RPi, para conceder el acceso a nuestro Bitbucket desde WebIDE.

Llegados a este punto, tendremos un entorno en el que hacer pruebas con los scripts de Python que crearemos más adelante con el fin de ejecutarlos en la RPi remotamente y de manera intuitiva y sencilla.

10.4. Cámara

Dado que la configuración de la cámara es más sencilla que la del sensor PIR, como cuarto punto en la configuración de la RPi, nos parece interesante continuar con la cámara. De este modo,podremos empezar a probar unos sencillos programas como el "Hola Mundo" para ver que las cosas funcionan bien.

La cámara se conecta en el puerto CSI de la RPi, que es el puerto ubicado al lado de la entrada de Ethernet. Esto se muestra en la siguiente imagen (Ver ilustración 15) dentro de un cuadrado rojo:

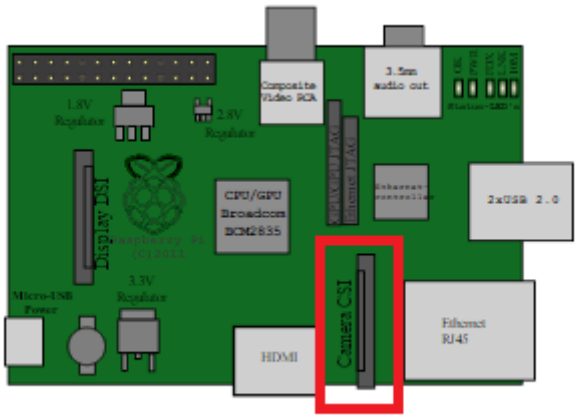


Ilustración 15. Raspberry Pi cámara [2]

Una vez conectada la cámara a la RPi, iremos al menú de configuración mediante el siguiente comando:

```
sudo raspi-config
```

En el menú que aparecerá después de ejecutar el comando, iremos al apartado de habilitar cámara llamado "Enable Camera", tal y como veremos más adelante en las imágenes (Ilustraciones 16 y 17). Este apartado será el que, después de seleccionarlo, nos abrirá otra ventana donde seleccionaremos "habilitar" (Enable). Una vez hecho esto, le daremos a "terminar" (Finish) en la primera pantalla, y cuando nos pregunte si queremos "reiniciar" (Reboot), ahora, le diremos a que "sí" (yes).

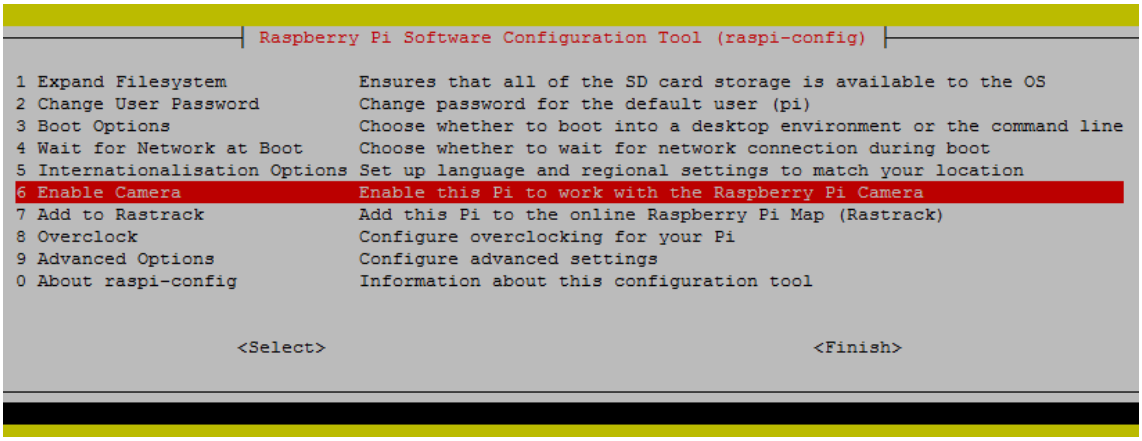


Ilustración 16. Configuración cámara 1

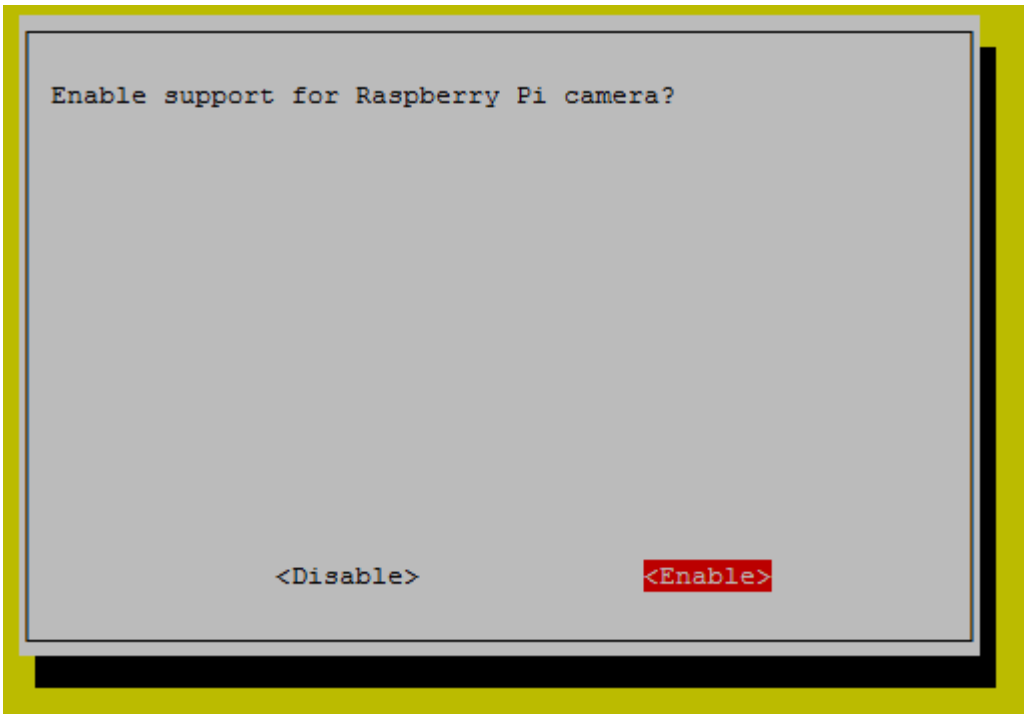


Ilustración 17. Configuración cámara 2

Llegados a este punto, tendremos la cámara en funcionamiento. Para asegurarnos su buen funcionamiento, lanzaremos el siguiente comando y veremos si la luz led roja que tiene la cámara se enciende .Si es así, podremos seguir con las pruebas de programación en scripts de Python.

```
raspistill -o image.jpg
```

Para comenzar con las pruebas, instalaremos la librería necesaria para controlar la cámara desde un script de Python. Esta programación la realizaremos en el WebIDE. Crearemos un proyecto donde iremos metiendo todas las pruebas y dentro crearemos la primera prueba que realizaremos para sacar una foto y almacenarla; esta prueba la meteremos en un fichero con terminación ".py" para que el sistema entienda que se trata de un script de Python.

La librería necesaria la instalaremos mediante el siguiente comando:

```
sudo apt-get install python-picamera python3-picamera python-rpi.gpio
```

Con el comando ejecutado solo tendremos que añadir a nuestro script de Python la importación de esta librería (import picamera) al inicio del fichero. Nosotros decidimos que las capturas realizadas por la cámara se guardarían en una carpeta del escritorio, y después de varias pruebas, nos quedamos con el código de la siguiente imagen (Ver ilustración 18), en el que se inicia la captura, se esperan 5 segundos, se le dice a la cámara que realice la captura y donde guardarla y finalizaremos la captura.

```
import time
import picamera

with picamera.PiCamera() as camera:
    camera.start_preview()
    time.sleep(5)
    camera.capture('/home/pi/Desktop/kapturak/image.jpg')
    camera.stop_preview()
```

Ilustración 18. Python cámara

10.5. Sensor PIR

Realizado esto podremos pasar con el quinto paso en la configuración de la RPi, que será la configuración y el testeo de el sensor de movimiento PIR.

El sensor PIR es un elemento que detecta cambios en la radiación infrarroja y que se usa para detectar presencia, ya que todo cuerpo vivo o maquina desprende calor.

Esta configuración fue de complejidad superior, por lo que tuvimos que realizar más pruebas hasta quedarnos con el código con mayor eficiencia y con el que más sentido tenia para nuestro sistema de vigilancia.

El primer paso será localizar los pines de la RPi que nos interesan para la conexión del sensor. El sensor PIR disponía de tres pines, uno de alimentación, uno de toma tierra y otro de señal de salida por pulsos. Sabiendo esto, había que encontrar cuáles eran los pines que les correspondían en la RPi. Tal y como vemos en la siguiente imagen (Ver ilustración 19), elegimos el pin 2 para el cable rojo del sensor PIR que conectaría la alimentación de 5V que necesitaría, elegimos el pin 6 para la toma tierra necesaria para el sensor conectado por cable negro, y conectamos el cable amarillo de señal de salida del sensor PIR al pin 12 de la RPi, con función de modulación por ancho de pulsos (PWM), que recibirá los pulsos del sensor y los convertirá a una señal analógica a gestionar mediante un script de Python teniendo en cuenta el BroadcomPin, que en el caso del pin 12 es el 18.

El BroadcomPin son una serie de pines que enlazan directamente con ciertos pines del microprocesador Broadcom de la RPi y que permiten ampliar la funcionalidad de la placa conectándola al mundo físico. Hay dos esquemas de numeración para estos pines, el físico y el lógico. Aunque nosotros usamos el último, también se puede usar el primero para referenciar los pines.

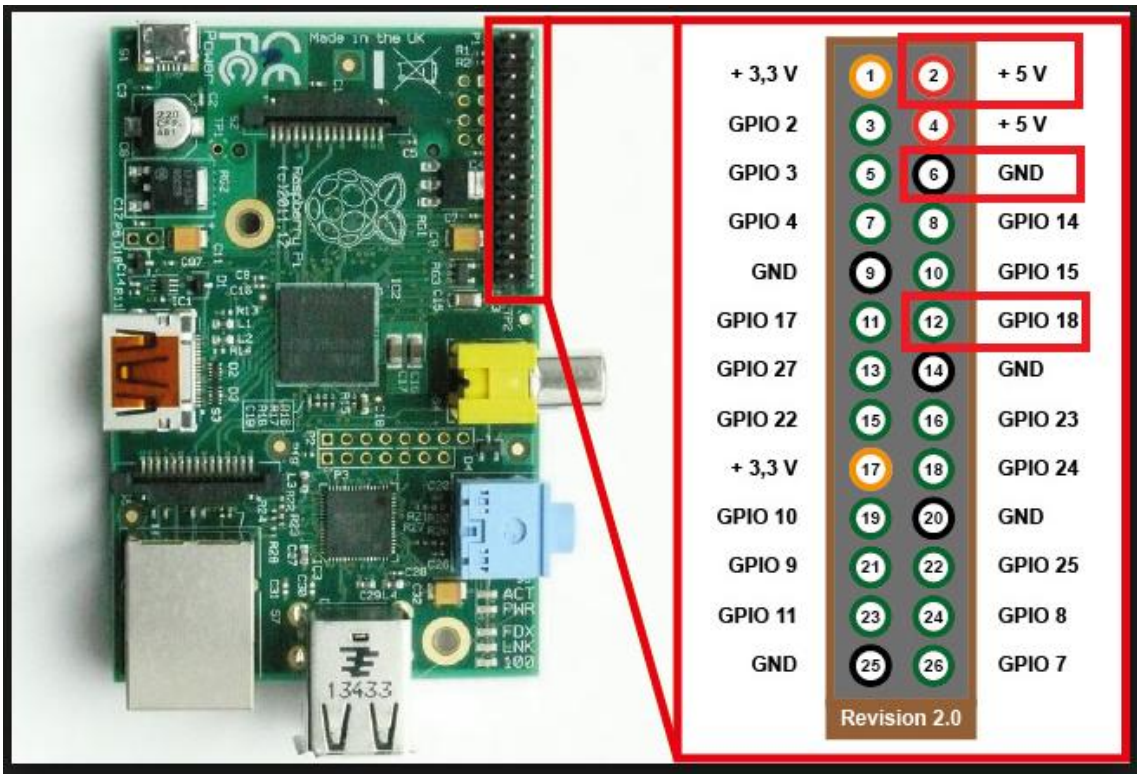


Ilustración 19. Raspberry Pi GPIO [3]

Antes de comenzar con las pruebas, es importante destacar que al principio conectábamos el sensor mediante una placa de pruebas (Breadboard), ya que el sensor traía un conector para alargar sus pines (aunque con terminación macho). Debido a que, al igual que el sensor, la RPi solo tenía pines macho, decidimos comprar unos Jumper macho-hembra para alargar los pines de la RPi a la placa de pruebas y poder hacer las conexiones. Sin embargo, terminadas las pruebas, observamos que era muy inestable tener los pines conectados mediante la placa de pruebas y que, además, ocupaba demasiado. Por lo tanto, optamos por comprar unos Jumper hembra-hembra para conectar directamente los pines macho del sensor PIR a los pines macho de la RPi.

Aclarado este punto podemos empezar con la parte de las pruebas del sensor. Dado que el sensor usa los pines GPIO de la RPi, necesitaremos añadir las importaciones necesarias al inicio. Además, imprimiremos por pantalla un texto (mediante un bucle) para comprobar que funciona bien cuando el sensor detecta movimiento. Dejamos un tiempo de dos segundos entre movimiento y movimiento, para que se vea claro que con cada movimiento se sacará un texto. Pero si el movimiento durase más de dos segundos, volvería a salir otra vez.

```
import time
import RPi.GPIO as io
io.setmode(io.BCM)

pir_pin = 18

io.setup(pir_pin, io.IN)

while True:
    if io.input(pir_pin):
        print("PIR ALARM!")
    time.sleep(2)
```

Ilustración 20. Python PIR

10.6. Dropbox y Dropbox-Uploader

Una vez la cámara y el sensor estén configurados, podremos pasar al sexto paso. En este paso configuraremos Dropbox, crearemos una aplicación y configuraremos Dropbox-Uploader para la subida de capturas a la cuenta configurada.

Algo principal para el proyecto es la creación de la aplicación en el entorno para desarrolladores de Dropbox. Para esto necesitaremos una cuenta Dropbox. Esta cuenta la podremos usar posteriormente para el almacenamiento de imágenes, ya que también será necesaria.

Primero, crearemos una cuenta en Dropbox [4], y después, iremos al entorno de desarrolladores [5] para la creación de la aplicación.

Una vez en el entorno de desarrolladores, iremos a la sección de “mis aplicaciones” (My apps) y le daremos a “crear una nueva aplicación” (Create app). Entre las opciones de configuración de la app, seleccionaremos el tipo de aplicación que usara las API de Dropbox, la cual tendrá acceso a ficheros y almacenes de datos y sólo tendrá acceso a los ficheros creados dentro de su propia carpeta. Seleccionaremos, a continuación, un nombre para la app y aceptaremos los términos y condiciones de Dropbox. Finalmente, le daremos a crear la aplicación.

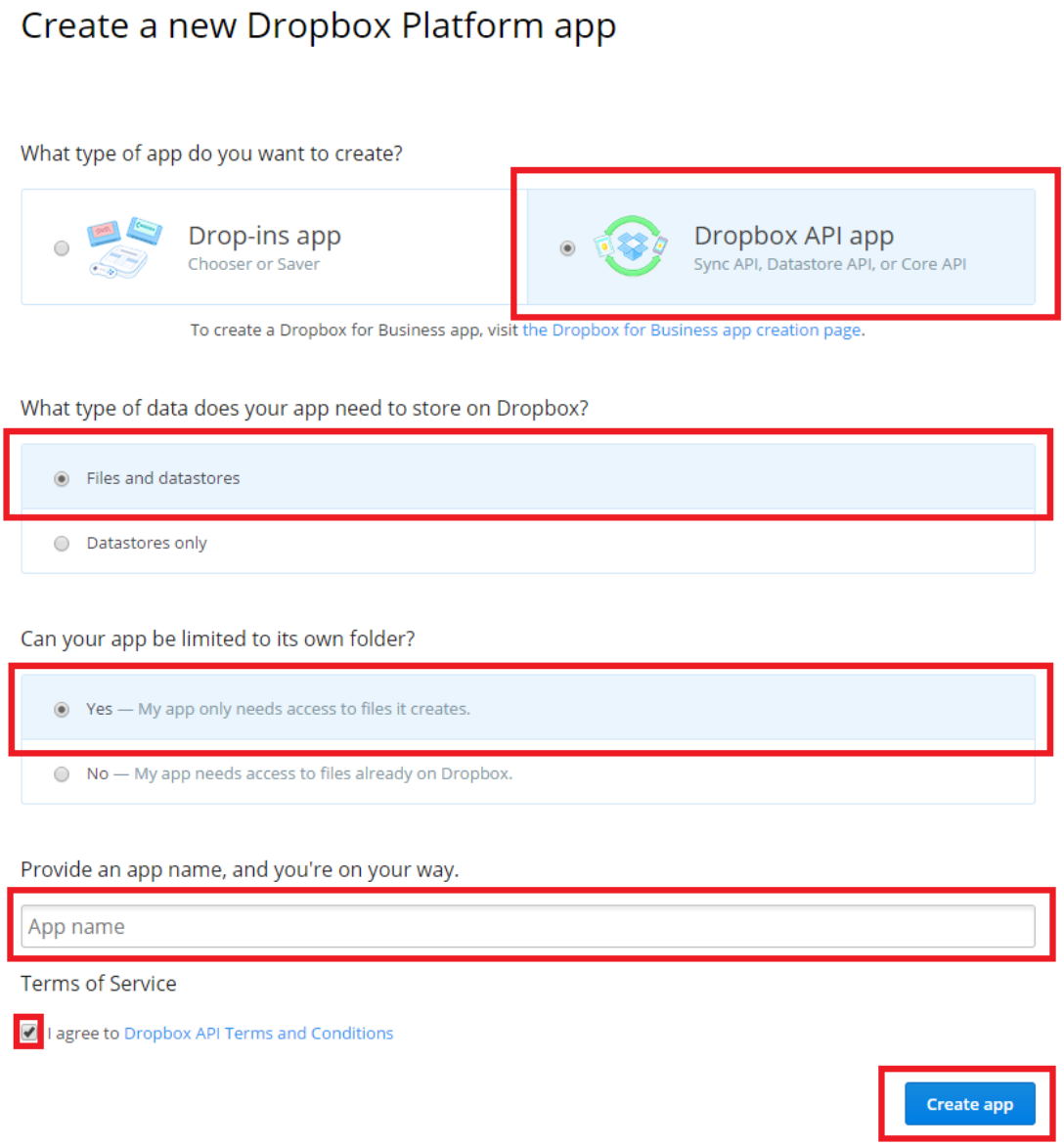


Ilustración 21. Creación app Dropbox

Creada la aplicación, veremos que dentro de la sección de mis aplicaciones tendremos la nueva creada. La abriremos y la mantendremos abierta, ya que necesitaremos las llaves de acceso, tanto la pública como la secreta (App key y App Secret), creadas para nuestra aplicación en la configuración de Dropbox-Uploader con el que entraremos ahora.

Abriremos la terminal de la RPi y ejecutaremos los siguientes dos comandos. El primero para instalar git-core, el que nos permitirá la clonación de proyectos de GitHub, y el segundo para subir imágenes a una cuenta de Dropbox a configurar en la primera ejecución, que nos clonara a la RPi el proyecto Dropbox-Uploader [6] . Esta configuración de la cuenta a la que subir los ficheros, más adelante y una vez instalado el servidor web, será un proceso que se podrá hacer mediante el navegador con unas páginas web y un script propio que hemos desarrollado ad-hoc, para facilitar la configuración.

```
sudo apt-get install git-core

git clone https://github.com/andreafabrizi/Dropbox-Uploader.git
```

Una vez finalizada la instalación y la clonación del proyecto, tendremos en nuestra RPi una carpeta llamada "Dropbox-Uploader". Mediante la consola, nos colocaremos en la carpeta mediante el comando cd y ejecutaremos el script "dropbox_uploader.sh", tal y como veremos en los siguientes dos comandos.

```
cd Dropbox-Uploader

./dropbox_uploader.sh
```

Una vez ejecutado el script, nos pedirá las claves que nos proporcionó el entorno desarrollador de Dropbox a la hora de crear la aplicación. Por lo tanto, iremos al navegador donde teníamos abierta la aplicación y copiaremos la app key y después la app secret; después, le diremos mediante la letra "a" que la aplicación solo tendrá acceso a los ficheros de la app, y una vez comprobado que todo está correcto, le diremos mediante la letra "y" que los datos están correctos.

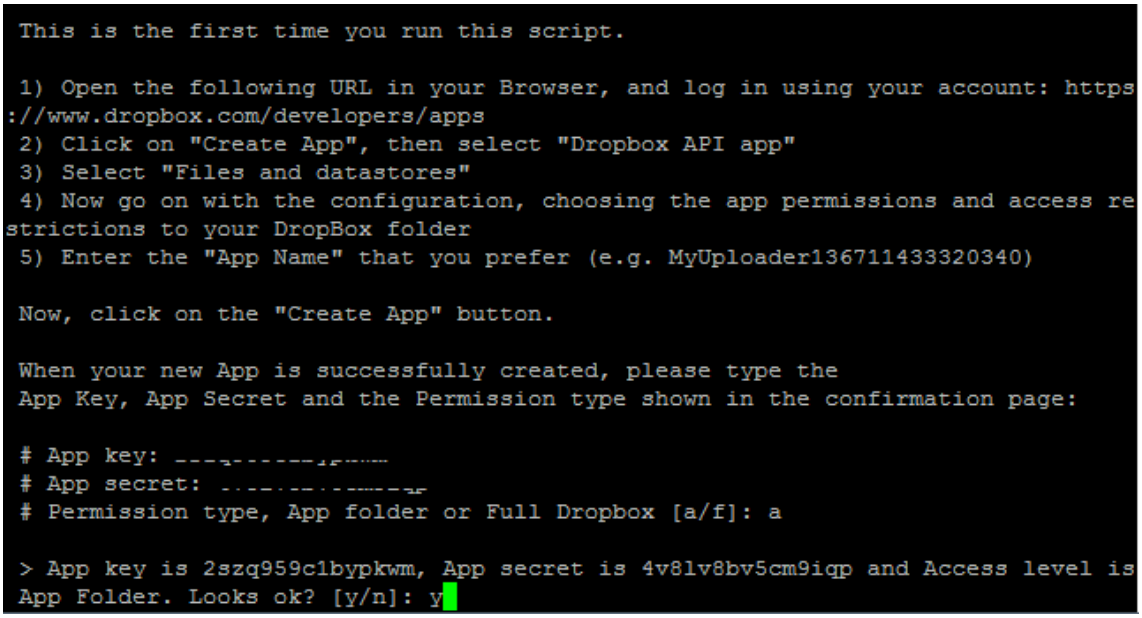


Ilustración 22. Configuración Dropbox-Uploader 1

Cuando acabe la comprobación nos dará una URL que tendremos que copiarla al navegador y asegurarnos que estamos registrados con la cuenta a la que nos interesa subir las fotos. En caso de ser así, en la página, la cual se abrirá con la URL que nos dio el script desde terminal, le daremos a "permitir", y cuando nos diga que el paso se realizó correctamente, volveremos a la terminal y pulsaremos "intro".

```
> Token request... OK

Please open the following URL in your browser, and allow Dropbox Uploader
to access your DropBox folder:

--> https://www.dropbox.com/1/oauth/authorize?oauth_token=

Press enter when done...
```

Ilustración 23. Configuración Dropbox-Uploader 2

Si todo ha ido correctamente, nos saldrá lo siguiente y podremos pasar al siguiente paso.

```
> Access Token request... OK

Setup completed!
```

Ilustración 24. Configuración Dropbox-Uploader 3

Éste proceso es el que define Dropbox para la conexión de un cliente que no es web ni móvil mediante OAuth, un protocolo de delegación de autentificación y autorización.

Como información adicional antes de continuar con el siguiente paso, creemos relevante indicar que a nosotros nos pareció interesante cambiar la imagen de la aplicación de Dropbox y crear un icono [7] que usaríamos posteriormente para la aplicación Android, para que a los usuarios de Dropbox y de la aplicación Android les resultase más visual saber los permisos de qué aplicación van a aceptar y cuál es la aplicación que van a usar.

Ilustración 25. Logo Zaintza Pi

10.7. Python

Llegados a este punto, tendremos Dropbox-Uploader configurado y la parte hardware ya instalada y configurada, por lo que en este séptimo paso realizaremos más pruebas con la cámara y Dropbox-Uploader en los scripts de Python. Acabaremos este paso con un código que integrará la cámara y el sensor con la subida del fichero a Dropbox.

Hicimos un par de pruebas con los pines de la RPi y la placa de pruebas, por ejemplo sacando fotos cuando se pulsaba un botón, o creando bucles de imágenes controlando los tiempos de captura, simplemente para tener un mayor control de las posibilidades.

Después fuimos integrando todo, primero la cámara con el Dropbox-Uploader, y luego añadiéndole el sensor PIR al conjunto.

Se añade el código en forma de ilustraciones, y se proporciona una explicación a continuación: Debido a que nos interesa que las imágenes no se sobrescriban, la mejor idea que se nos ocurrió fue crear una función que capturase la fecha del momento. También se incluyeron los segundos para así poder también conocer rápidamente cuándo fue el momento de la captura. Como hemos incluido los segundos, y sabiendo que cuando introdujéramos el sensor tendríamos que dejar unos segundos de espera entre captura y captura (para que el mismo movimiento no lanzase un montón de capturas en una fracción de segundo), nos evitamos el problema de que dos imágenes tengan el mismo nombre. Las imágenes se guardan en una carpeta de la RPi que luego se usará para subir a Dropbox desde ella, y en ambos sitios se guardará con el nombre del momento en el que se realizó la captura.

```
import time
import picamera
import time
import datetime
from subprocess import call

def getFileName():
    return datetime.datetime.now().strftime("%Y-%m-%d_%H.%M.%S.jpg")

with picamera.PiCamera() as kamera:
    argazki_izena=getFileName()
    fileName = '/home/pi/Desktop/kapturak/' + argazki_izena
    kamera.start_preview()
    kamera.capture(fileName)
    kamera.stop_preview()
    print("Argazkia aterata"+ fileName)
    photofile = "/home/pi/Dropbox-Uploader/dropbox_uploader.sh upload "+fileName+" "+argazki_izena
    call ([photofile], shell=True)
```

Ilustración 26. Python cámara con Dropbox-Uploader

Después de enlazar la cámara y el Dropbox-Uploader, nos quedaría añadirle la parte del sensor PIR, y para esto había que hacer modificaciones. Ahora no se iba a ejecutar el script de Python cada vez que quisiéramos hacer una captura, ya que el sensor decidiría el momento de realizar una captura. De este modo, el script de Python se mantiene en ejecución dentro de un bucle, con el fin de que esté continuamente detectando movimiento. A pesar de que este procedimiento no fuera el idóneo, ya que presenta la limitación de tener que matar el Python cada vez que quisiéramos parar de detectar movimiento, no encontramos otra solución viable y tuvimos que dejarlo así. De todas maneras, como el sistema lo hemos montado nosotros, sabemos con seguridad que no habrá otro script de Python en ejecución en ningún momento; la solución fue sencilla: Queremos que el Python de detección de movimiento no esté siempre activo y queremos poder pararlo desde una aplicación Android, lo que implica no saber el ID del proceso relacionado a nuestro detector, pero sabiendo que no hay mas, mataremos todo los scripts de Python activos. Nos aseguraremos así, tanto de que el sistema deja de detectar, como de que no se solapen dos procesos por intento de ejecutar un script de Python que se encuentra actualmente en ejecución.

Cuando estábamos probando el sensor, observamos que podíamos darle dos maneras de funcionar a nuestro sistema: 1) sacaba una foto cada vez que detectaba movimiento pero no sacaba otra hasta que ese movimiento no se finalizaba, y ya que en nuestro sistema de vigilancia nos parecía lógico sacar más de una foto aunque el movimiento fuese el mismo, decidimos descartarla. Nos interesaría por ejemplo ver que es lo que el intruso estaba haciendo, no solo ver que el intruso había entrado en el campo visual de la cámara. Esta opción estaría bien como una segunda funcionalidad, pudiendo en vez de hacer una captura, hacer varias y montar una secuencia, o grabar un video, para así tener un control de lo que ha pasado durante el transcurso de cada movimiento. 2) La otra opción saca una foto cada diez segundos en el caso de que hubiera movimiento detectado en ese momento; es decir, aunque el movimiento fuese el mismo, nos sacaría una captura cada diez segundos para ver qué es lo que el intruso está haciendo. Ya que esta es la opción por la que optamos es la que tiene la parte de la cámara y la subida a Dropbox incluida, la otra opción esta en el código (Ver ilustración 27) para otros desarrolladores o mejoras posteriores.

```
import RPi.GPIO as GPIO
import picamera
import time
import datetime
from subprocess import call

def getFileName():
    return datetime.datetime.now().strftime("%Y-%m-%d_%H.%M.%S.jpg")

kamera = picamera.PiCamera()
sensorPin = 18

GPIO.setmode(GPIO.BCM)
GPIO.setup(sensorPin, GPIO.IN)#, pull_up_down=GPIO.PUD_DOWN)

'''
prevState = False
currState = False

while True:     #Mogimendua detektazen duenean detektazen ez duenerarte ez da berriro martxan jartzen.
    time.sleep(0.1)
    prevState = currState
    currState = GPIO.input(sensorPin)
    if currState != prevState:
        newState = "HIGH" if currState else "LOW"
        print "Motion Detected"
'''
while True:
    if GPIO.input(sensorPin):     #10 segunduro argazki bat ateratzen du sentsoreak mugimendua somatzen badu
        print("Mugimendua dago!")
        argazki_izena=getFileName()
        fileName = '/home/pi/Desktop/kapturak/' + argazki_izena
        kamera.start_preview()
        kamera.capture(fileName)
        kamera.stop_preview()
        print("Argazkia aterata"+ fileName)
        photofile = "/home/pi/Dropbox-Uploader/dropbox_uploader.sh upload "+fileName+" "+argazki_izena
        call ([photofile], shell=True)
        time.sleep(10)
```

Ilustración 27. Python cámara con Dropbox-Uploader y sensor PIR

10.8. Servidor Web

Comenzaremos el octavo paso de configuración de la RPi, instalando el Tomcat 8, y finalizaremos este paso y la configuración de la RPi, explicando los Servlets y páginas web que hemos desarrollado para la integración con la aplicación Android y comodidad de usuario en cuanto a la configuración de la cuenta Dropbox en la RPi.

Tal y como se ve en el bloque de diagrama (Ilustración 1), teníamos la duda de qué servidor web íbamos a instalar, Tomcat o Apache. Debido a que conocíamos el servidor web Tomcat y que estábamos informados de que trabajaba con Servlets y en Java, nos decantamos por este, ya que la formación necesaria para cumplir nuestros objetivos partiría de una buena base adquirida en el grado.

Seguiremos unos sencillos pasos para la instalación de Tomcat 8; 1) actualizaremos el sistema para tener los últimos paquetes y no tener nada obsoleto, 2) instalaremos Java ya que los Servlets de Tomcat se compilan con Java, 3) instalaremos el propio Tomcat 8, y 4) lo encenderemos.

```
sudo apt-get update

sudo apt-get install oracle-java8-jdk

sudo apt-get install tomcat8

sudo service tomcat8 start
```

Si todo ha salido bien, desde un navegador de un ordenador que se encuentre en la misma red que la RPi, pondremos como dirección la IP estática que definimos en la configuración del WiFi y como WebIDE se encuentra en el puerto 80, que suele ser por defecto el de los servidores web, Tomcat se habrá colocado en su segundo puerto por defecto que es el 8080. Por lo tanto, con la IP estática y el puerto 8080 tendríamos que ver en el navegador la siguiente imagen (Ver ilustración 28):

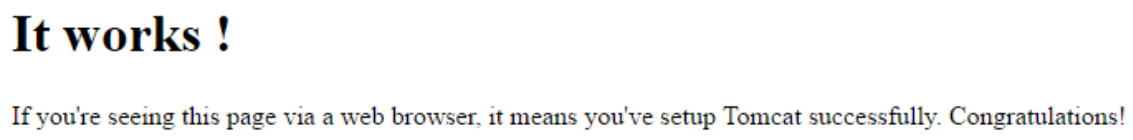


Ilustración 28. Tomcat 8 funcionando

El último comando que ejecutamos sirve para arrancar el servicio del servidor, pero también se podría parar o reiniciar el servicio, sustituyendo “start” por “stop” o “restart” respectivamente.

En este punto nos parece interesante explicar cómo dejar el servicio de Tomcat preparado para que si se reinicia la maquina, el servicio se arranque por sí solo otra vez, sin tener que abrir la terminal para lanzarlo. Primero, tendremos que crear el fichero de arranque en “/etc/init.d/” llamado “tomcat” con el código de la imagen (Ver ilustración 29), donde corregiremos en caso de ser incorrecto la dirección del Tomcat 8 que está definida en la variable CATALINA_HOME.

```
#!/bin/bash
#
# tomcat
#
# chkconfig:
# description:  Start up the Tomcat servlet engine.

# Source function library.
. /etc/init.d/functions


RETVAL=$?
CATALINA_HOME="/usr/share/tomcat8"

case "$1" in
 start)
        if [ -f $CATALINA_HOME/bin/startup.sh ];
          then
        echo $"Starting Tomcat"
            /bin/su tomcat $CATALINA_HOME/bin/startup.sh
        fi
    ;;
 stop)
        if [ -f $CATALINA_HOME/bin/shutdown.sh ];
          then
        echo $"Stopping Tomcat"
            /bin/su tomcat $CATALINA_HOME/bin/shutdown.sh
        fi
    ;;
 *)
    echo $"Usage: $0 {start|stop}"
    exit 1
    ;;
esac

exit $RETVAL
```

Ilustración 29. Configuración Tomcat 8 para autoarranque

Después, añadiremos el link del fichero a la línea de ejecución para que al arrancar la RPi se ejecute nuestro código. La carpeta "/etc" dentro tiene unas carpetas que empiezan por "rc" y tienen un numero seguido. Estas carpetas dentro tienen ficheros que son enlaces de los ficheros de arranque de servicios. Cuando la RPi se arranca, va ejecutando los ficheros de esas carpetas en orden, respetando los niveles de ejecución. Después de búsquedas y lecturas, encontramos la página que nos ayudó a decidir dónde colocarla y qué nombre darle [8]. El comando para dejar el link de nuestro fichero es el siguiente:

```
sudo ln -s /etc/init.d/tomcat /etc/rc5.d/S71tomcat
```

Le daremos permisos a la carpeta donde guardábamos las capturas en la RPi, con el objetivo de que el usuario Tomcat tenga acceso para crear y editar.

Ahora empezaremos con la aplicación que tendrá nuestros Servlets dentro. La estructura que tendrá será la siguiente:

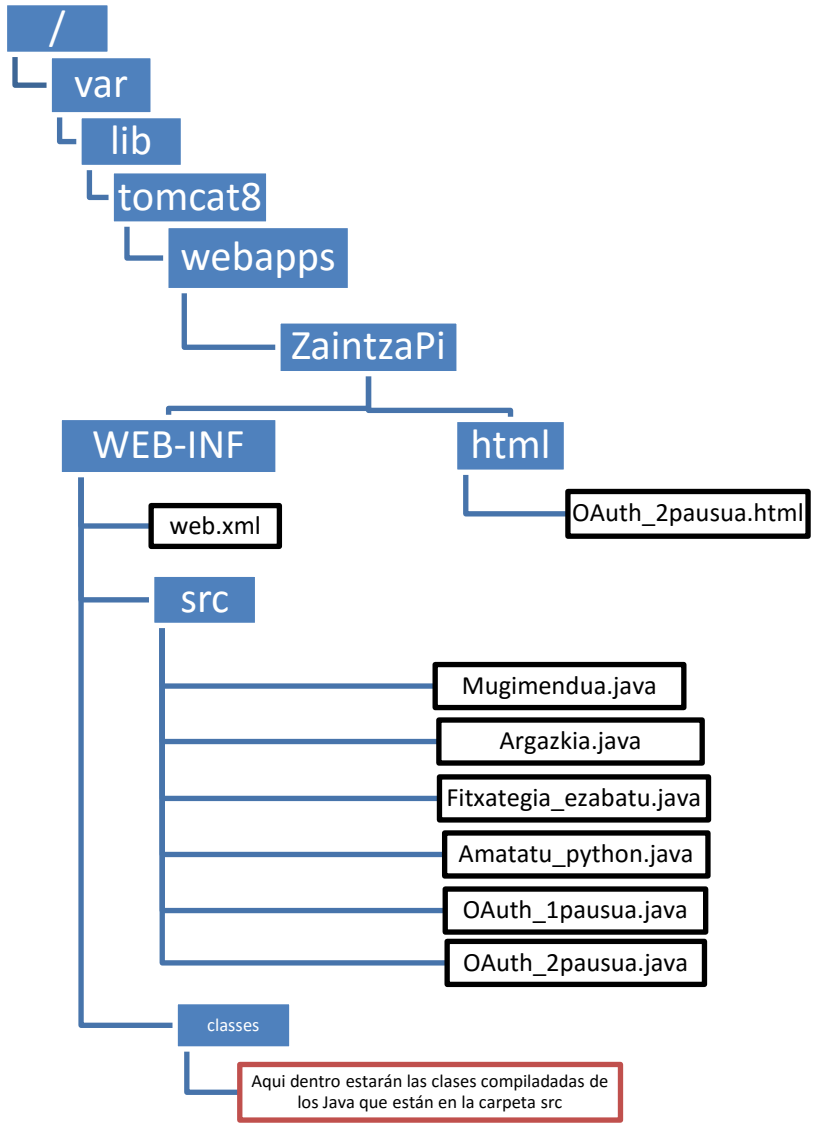


Ilustración 30. Estructura servidor web

Nuestra aplicación se llamara ZaintzaPi, igual que la aplicación Android, Zaintza por ser la palabra para vigilancia en euskera y Pi por la integración con la Raspberry Pi. Dentro de la aplicación tendremos dos carpetas, una será para HTML, que en nuestro caso sólo cuenta con una página web para el final de la configuración entre Dropbox-Uploader y nuestra aplicación Dropbox. Y la otra carpeta, WEB-INF, para los Servlet que tendrán que estar definidos en el fichero web.xml dentro de esta carpeta.

Adjuntamos la siguiente imagen (Ilustración 31) para proporcionar una idea de la manera en la que hay que definir los Servlet dentro del fichero web.xml.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

        <servlet>
        <servlet-name>Mugimendua</servlet-name>
        <servlet-class>Mugimendua</servlet-class>
        </servlet>

        <servlet-mapping>
        <servlet-name>Mugimendua</servlet-name>
        <url-pattern>/servlet/ZaintzaPi_Mugimendua</url-pattern>
        </servlet-mapping>

        <servlet>
        <servlet-name>Argazkia</servlet-name>
        <servlet-class>Argazkia</servlet-class>
        </servlet>

        <servlet-mapping>
        <servlet-name>Argazkia</servlet-name>
        <url-pattern>/servlet/ZaintzaPi_Argazkia</url-pattern>
        </servlet-mapping>

</web-app>
```

Ilustración 31. Ejemplo web.xml

Cada Servlet tendrá 2 bloques, tal y como se ve en la imagen. En el ejemplo tenemos la definición de dos Servlet, uno “Mugimendua” y otro “Argazkia”.

En el primer bloque, se le da un nombre al Servlet y se le dice cuál será la clase que tiene que usar de la carpeta de clases, los nombres no tienen por qué ser iguales. La clase la conseguiremos compilando el Java mediante el siguiente comando:

```
sudo javac -cp /usr/share/tomcat8/lib/servlet-api.jar -d ../classes Mugimendua.java
```

Este ejemplo se tendrá que lanzar estando en la carpeta src, y de este modo, se creará el fichero “.class” para el Java “Mugimendua.java”.

El segundo bloque define la URL que se usará para ejecutar este Servlet desde el navegador, el nombre del Servlet tendrá que coincidir con el nombre definido en el primer bloque.

Así un Servlet quedará definido cuando tengamos una URL relacionada a la clase del código en Java desarrollado, que se ejecutará al abrir la URL.

10.9. Servlets

A continuación se explican uno a uno la razón y uso de cada Servlet, que se han realizado para la gestión del sistema desde la aplicación Android y para la configuración de la cuenta Dropbox para Dropbox-uploader:

1. Mugimendua.java

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.swing.*;

public class Mugimendua extends HttpServlet{
        public void doGet (HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
                System.out.println("--->Hasiera Mugimendua Servlet");

                runBackgroundTask();

                System.out.println("--->Amaiera Mugiemdua Servlet");


        }

        private void runBackgroundTask() {
          new SwingWorker<Void, Void>() {
            protected Void doInBackground() throws Exception {
                 String pythonScriptPath = "/home/webide/repositories/my-pi-projects/Zaintza_Sistema/Kamera_PIR_Dropbox.py";

                String[] cmd = new String[3];
                cmd[0]="sudo";
                cmd[1]="python";
                cmd[2]= pythonScriptPath;

                ProcessBuilder rt = new ProcessBuilder(cmd);
                Process pr = rt.start();
                return null;
            }
          }.execute();
        }
}
```

Ilustración 32. Ejemplo Java para ejecutar Python

Este Java, al igual que el resto que hemos implementado (exceptuando los de OAuth), lo único que hace es simular la ejecución de un comando por consola. En este caso, lanzamos el comando para que arranque el script de Python que tenemos creado para el sensor de movimiento PIR integrado con la cámara y el Dropbox-uploader.

Se usará para encender el sistema automático de detección.

2. Argazkia.java

Este Java es idéntico al anterior, con la única diferencia de que en este caso lanzamos el script de Python que tenemos sin el sensor PIR, el que sólo integra la cámara con el Dropbox-Uploader.

Se usará para cuando estamos en modo manual y queremos sacar una captura.

3. Fitxategia_ezabatu.java

El siguiente Java, se usará para que, cuando desde la aplicación Android se elimine una captura de la lista de capturas (que vera tomadas por la cámara y almacenadas en la carpeta de la aplicación en su cuenta de Dropbox), también se elimine de la RPi y así no se tengan imágenes en la RPi que el usuario a decidido borrar de Dropbox mediante la aplicación Android.

Ya que el nombre almacenado en la carpeta de la RPi y en Dropbox será el mismo, le mandaremos en la llamada desde Android el nombre de la captura que se va a borrar para que se borre también de la RPi.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

import java.util.Enumeration;

public class Fitxategia_ezabatu extends HttpServlet{
        public void doGet (HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
                System.out.println("--->Hasiera Fitxategia_ezabatu Servlet");

                Enumeration<String> parameterNames = request.getParameterNames();
                while (parameterNames.hasMoreElements()) {
                    String paramName = parameterNames.nextElement();
                    String[] paramValues = request.getParameterValues(paramName);
                    for (int i = 0; i < paramValues.length; i++) {
                        String paramValue = paramValues[i];
                        String fitxategiPath="/home/pi/Desktop/kapturak/"+paramValue;

                        String[] cmd = new String[3];
                        cmd[0]="sudo";
                        cmd[1]="rm";
                        cmd[2]=fitxategiPath;

                        ProcessBuilder rt = new ProcessBuilder(cmd);
                        rt.redirectErrorStream(true);
                        Process pr = rt.start();

                        BufferedReader bfr = new BufferedReader(new InputStreamReader(pr.getInputStream()));
                        String line = "";
                        while((line = bfr.readLine()) != null) {
                                System.out.println(line);
                        }
                    }
                }

                System.out.println("--->Amaiera Fitxategia_ezabatu Servlet");


        }
}
```

Ilustración 33. Ejemplo Java para ejecutar comando de borrado

4. Amatatu_python.java

Tal y como se comento en la creación de los Python, al integrar el sensor PIR necesitabamos un bucle que se quedara continuamente iterado para detectar movimiento. No obstante, queríamos poder encender y apagar la detección continua. De este modo, lo que hace este java es lanzar un comando por consola. Un comando, peligroso por su repercusión, pero controlado porque sabemos que sólo estará ese Python encendido en el caso de haber alguno.

```
sudo killall python
```

5. OAuth_1pausua.java y OAuth_2pausua.java

```java
import java.io.InputStreamReader;

import java.net.URL;
import java.net.URLEncoder;

import javax.net.ssl.HttpsURLConnection;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebServlet("/OAuth_1pausua")
public class OAuth_1pausua extends HttpServlet {
	private static final long serialVersionUID = 1L;

    public OAuth_1pausua() {
        super();
    }


	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		HttpSession session = request.getSession(true);

		String uri = "https://api.dropboxapi.com/1/oauth/request_token";

		URL obj = new URL(uri);
		HttpsURLConnection conn = (HttpsURLConnection) obj.openConnection();

		conn.setRequestMethod("POST");
		conn.setRequestProperty("User-Agent", "Tomcat");

		String app_key = "----------";
		String app_secret = "----------";
		conn.setRequestProperty("Authorization", buildInitialOAuthHeader(app_key, app_secret));
		System.out.println("Request URI: " + uri);

		int responseCode = conn.getResponseCode();
		System.out.println("Response code: " + responseCode);

		BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
		String inputLine;
		StringBuffer erantzuna = new StringBuffer();
		while ((inputLine = in.readLine()) != null) {
			erantzuna.append(inputLine);
		}
		in.close();

		String edukia = erantzuna.toString();
		System.out.println("Response body: " + edukia);

		String oauth_token = edukia.split("&")[1].split("=")[1];
		String oauth_token_secret = edukia.split("&")[0].split("=")[1];
		System.out.println("oauth_token: " + oauth_token);
		System.out.println("oauth_token_secret: " + oauth_token_secret);
		session.setAttribute("oauth_token", oauth_token);
		session.setAttribute("oauth_token_secret", oauth_token_secret);

		String base_uri = "https://www.dropbox.com/1/oauth/authorize";
		//String oauth_callback = "https://echezaservidor.ddns.net/ZaintzaPi/servlet/ZaintzaPiKonfOauth2";
		String params = "oauth_token=" + oauth_token;// + "&oauth_callback=" + URLEncoder.encode(oauth_callback, "UTF-8");

		uri = base_uri + '?' + params;
		response.sendRedirect(uri);
	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		doGet(request, response);
	}

	public static String buildInitialOAuthHeader(String key, String secret)
	{
	    StringBuilder buf = new StringBuilder();
	    buf.append("OAuth ");
	    buf.append(" oauth_consumer_key=\"").append(key).append("\"");
	    buf.append(", oauth_signature_method=\"PLAINTEXT\"");
	    buf.append(", oauth_signature=\"").append(secret).append("&\"");
	    return buf.toString();
	}
}
```

Ilustración 34. OAuth paso1

```java
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.URL;
import java.net.URLEncoder;

import java.io.File;
import java.lang.ProcessBuilder.Redirect;

import javax.net.ssl.HttpsURLConnection;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebServlet("/OAuth_2pausua")
public class OAuth_2pausua extends HttpServlet {
	private static final long serialVersionUID = 1L;
    public OAuth_2pausua() {
        super();
    }

	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		HttpSession session = request.getSession(true);

		String uri = "https://api.dropboxapi.com/1/oauth/access_token";

		URL obj = new URL(uri);
		HttpsURLConnection conn = (HttpsURLConnection) obj.openConnection();

		conn.setRequestMethod("POST");
		conn.setRequestProperty("User-Agent", "Tomcat");

		String app_key = "----------";
		String app_secret = "----------";
		String oauth_token = (String) session.getAttribute("oauth_token");
		String oauth_token_secret = (String) session.getAttribute("oauth_token_secret");
		conn.setRequestProperty("Authorization", buildOAuthHeader(app_key, app_secret, oauth_token, oauth_token_secret));
		System.out.println("Request URI: " + uri);

		int responseCode = conn.getResponseCode();
		System.out.println("Response code: " + responseCode);

		BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
		String inputLine;
		StringBuffer erantzuna = new StringBuffer();
		while ((inputLine = in.readLine()) != null) {
			erantzuna.append(inputLine);
		}
		in.close();

		String edukia = erantzuna.toString();
		System.out.println("Response body: " + edukia);

		oauth_token = edukia.split("&")[1].split("=")[1];
		oauth_token_secret = edukia.split("&")[0].split("=")[1];
		System.out.println("oauth_token: " + oauth_token);
		System.out.println("oauth_token_secret: " + oauth_token_secret);

		String fitxategiPath="/usr/share/tomcat8/.dropbox_uploader";
		String appkey="APPKEY=2szq959c1bypkwm";
		String appsecret="APPSECRET=4v81v8bv5cm9iqp";
		String access="ACCESS_LEVEL=sandbox";
		String token="OAUTH_ACCESS_TOKEN="+oauth_token;
		String token_secret="OAUTH_ACCESS_TOKEN_SECRET="+oauth_token_secret;

		new ProcessBuilder("echo", appkey).redirectOutput(new File(fitxategiPath)).start();
		new ProcessBuilder("echo", appsecret).redirectOutput(Redirect.appendTo(new File(fitxategiPath))).start();
		new ProcessBuilder("echo", access).redirectOutput(Redirect.appendTo(new File(fitxategiPath))).start();
		new ProcessBuilder("echo", token).redirectOutput(Redirect.appendTo(new File(fitxategiPath))).start();
		new ProcessBuilder("echo", token_secret).redirectOutput(Redirect.appendTo(new File(fitxategiPath))).start();

		response.sendRedirect("/ZaintzaPi/html/OAuth_2pausua.html");

	}

	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		doGet(request, response);
	}

	public static String buildOAuthHeader(String app_key, String app_secret, String token_key, String token_secret)
	{
	    StringBuilder buf = new StringBuilder();
	    buf.append("OAuth ");
	    buf.append("oauth_token=\"").append(token_key).append("\"");
	    buf.append(", oauth_consumer_key=\"").append(app_key).append("\"");
	    buf.append(", oauth_signature_method=\"PLAINTEXT\"");
	    buf.append(", oauth_signature=\"").append(app_secret).append("&").append(token_secret).append("\"");
	    return buf.toString();
	}
}
```

Ilustración 35. OAuth paso 2

Estos dos Servlet son para facilitar la configuración de Dropbox-Uploader, para que en vez de seguir los pasos que seguimos con la primera configuración, tengamos desde el navegador accesible el proceso mediante dos URL.

Después de analizar el código de Dropbox-Uploader, nos dimos cuenta de que creaba un fichero con la configuración, y que para poder cambiar de usuario abría que localizarla, borrarla y volver a lanzar el script como si fuese la primera vez que se configuraba.

Mediante estos dos Servlet, simulamos la interacción entre la consola y Dropbox, mandándole las claves necesarias y creando el fichero de configuración con los token de acceso que el proceso devuelve. Así, el usuario podría cambiar de cuenta para el almacenamiento de imágenes de manera fácil.

Las URL serán las siguientes, sustituyendo IP y puerto por los correspondientes. El usuario ira a la primera URL y iniciando sesión con el usuario que le interesa aceptará los permisos, y cuando Dropbox le diga que se realizó correctamente, irá a la siguiente URL y esta le mandará a la HTML que definimos para que el usuario sepa que la configuración se ha realizado correctamente.

IP:Puerto/ZaintzaPi/servlet/ZaintzaPiKonfOauth1

IP:Puerto/ZaintzaPi/servlet/ZaintzaPiKonfOauth2

10.10. Conexión externa

El proceso de configuración de la RPi ya estaría finalizado, pero aun nos faltaría el poder acceder desde una red externa a la RPi, ya que si no, el sistema de vigilancia sólo se podría controlar desde la misma red en el que este se encontrase.

Por lo tanto, crearemos un dominio para la IP externa de nuestro router con el objetivo de poder acceder a este desde el exterior sin tener que memorizar los números y por si la IP cambiara por algún motivo. Para este cometido optamos por DUC (Dynamic DNS Update Client) de NO-IP que nos brindará un dominio gratuito y de una manera muy sencilla. Crearemos una cuenta en su página web [9] y en la sección de “Hostnames” crearemos una nueva, añadiendo aquí la IP externa que tenemos en ese momento en nuestro router, para eso miraremos en la página de myip [10].

Para la configuración de la RPi de la versión para Raspberry Pi de DUC seguiremos el tutorial que la propia página de NO-IP nos proporciona [11], ya que es muy sencillo carece de explicación alguna.

Teniendo un dominio para poder acceder desde fuera de la propia red de la RPi (algo interesante para poder seguir trabajando mediante SSH desde fuera y para configurar la pagina web al puerto 80 de nuestro dominio) vamos a hacer Port Fortwarding para dirigir las conexiones desde el exterior.

En nuestro caso, con el objetivo de dotarle al sistema de un mínimo de seguridad, no usaremos el puerto habitual 22 para el SSH y pondremos otro mapeado al 22 de la IP

estática de la RPi. Añadiremos el del ejemplo (Ver ilustración 36) para que en el caso de que no se ponga un puerto al escribir el dominio (que es el puerto por defecto 80) se dirija al 8080 de la RPi y así enseñarnos la página de Tomcat 8.

El mapeado se hace estando en la misma red que el router, poniendo en un navegador la IP 192.168.1.1. Esto abrirá un menú de configuración que depende de cada router, por lo indicamos que por norma general, hay un apartado llamado "Port Forwarding" o "Port mapping table", donde añadiremos los mapeados de los puertos que nos interesan. A continuación se puede observar un ejemplo en el que redirijo el puerto de la web..

Port Mapping Table List

Sel.	#	Entry	Application	External Port	Server IP Address	Internal Server Start Port	Interface Name	Remote Host
○	5.	Enable	HTTP (TCP 80)	TCP 80	192.168.1.55	8080	Internet	

Ilustración 36. Port Forwarding

Llegados a este punto, ya tendremos todo lo relacionado con la RPi realizado, y sólo nos quedará la aplicación Android que explicaremos pondremos a continuación.

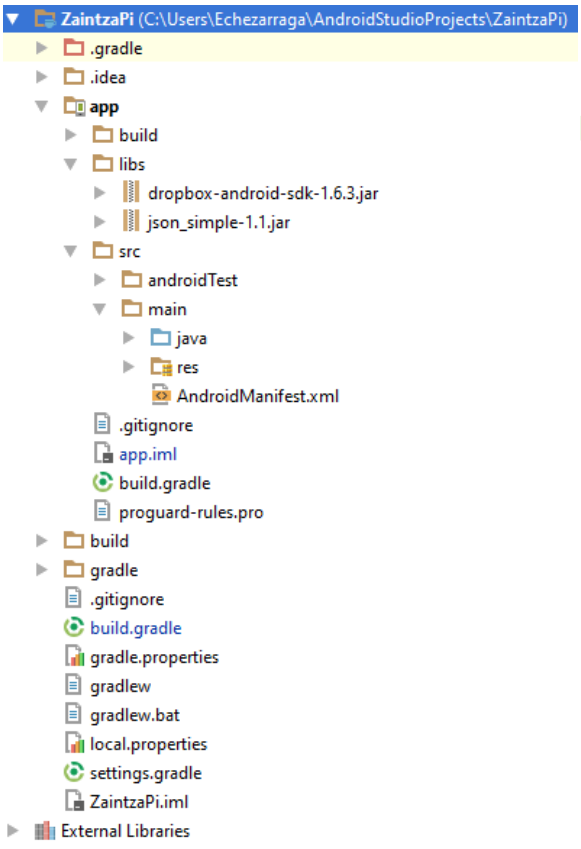
10.11. Android

Teniendo en el apartado de análisis y diseño la clase de diagrama y el diseño de la aplicación, explicaremos para qué sirve cada clase de diagrama y cuál es la función de cada método. De todos modos, el código [12] estará accesible en GitHub para otros desarrolladores o futuras mejoras.

Para el desarrollo de la aplicación, en Android usamos Android Studio [13], que es un entorno gratuito muy completo y usado por la mayoría de creadores de aplicaciones. A este le enlazamos GitHub [14] , ya que el propio Android Studio en la parte de configuración facilita dicho proceso; lo usamos para no perder nada de los progresos que íbamos haciendo y tener revisiones para poder volver a situaciones en las que nos encontrábamos anteriormente. Además, esto supone una ventaja para los casos en los que sea necesario poder trabajar desde varios sitios teniendo disponible en ese lugar siempre lo último que se consigue, por ejemplo en el caso de tener que compaginar estudio y trabajo..

A continuación se explican una serie de detalles sobre el proyecto y sobre los ficheros que componen la estructura de Zaintza Pi. Asimismo, mostraremos unas capturas de cómo fue el resultado final del proyecto, y terminaremos comentando las clases principales y sus funciones.

Primero, veremos mediante estas dos imágenes (Ilustraciones 37 y 38) la estructura del proyecto con el fin de ubicarnos cuando los ficheros se vayan explicando. La primera imagen es la vista proyecto de Zaintza Pi, y la segunda, es la carpeta "src" de la primera imagen ampliada con la vista app:



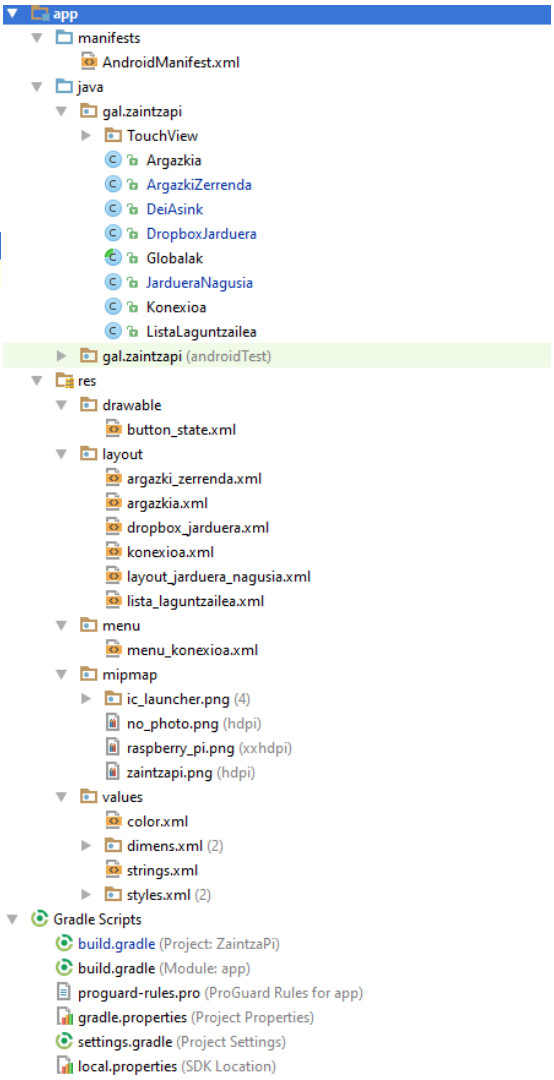


Ilustración 37. Vista proyecto en Android Studio y Ilustración 38. Vista aplicación en Android Studio

Ya que la mayoría del proyecto se basa en la interacción con Dropbox, tenemos que añadir a las librerías la API [16] de éste para Android. Debido a que cuando empezamos con el desarrollo sólo estaba la versión 1 de la API, esta es la que se ha mantenido. No obstante, pero como mejora, en un futuro se recomienda actualizar Zaintza Pi a la versión 2 de la API de Dropbox.

En el manifiesto que se llamará "AndroidManifest.xml" es donde irán declaradas todas las actividades que vamos a tener; Es decir, donde definiremos los permisos que la aplicación necesitará para que el usuario sepa a qué tendrá acceso Zaintza Pi, y definiremos también el nombre de la aplicación así como el diseño que tendrá. Por ejemplo, la definición de la actividad para la autentificación con la cuenta de Dropbox se definirá aquí, tal y como se explica en la página de descarga de la librería.

Los permisos de la aplicación son cinco: tendremos acceso a internet, acceso al estado de la conexión, acceso a la escritura en la tarjeta y también lectura, para descargar las capturas de Dropbox al móvil, poder verlas desde Zaintza Pi y poder borrarlas, y por

último acceso a la función de vibración del dispositivo, para las notificaciones de usuarios con discapacidad auditiva.

El proyecto consta de dos "build.gradle", ambos creados por el propio Android Studio. El que nos interesa es el que tiene el identificador añadido de "Module: app", tal y como se ve en la segunda imagen (Ilustración 38) de la estructura del proyecto. En este fichero definiremos las dependencias, por ejemplo a la librería de la API de Dropbox que incluimos en las librerías.

También definiremos las versiones de Android para las que nuestra aplicación estará disponible. En nuestro caso, cuando tomamos la decisión, Lollipop era la última versión en el mercado, pero viendo el progreso que tenía el mercado de Android sabíamos que para cuando terminásemos el desarrollo el mercado jugaría a nuestro favor, por lo que decidimos limitar la aplicación a la versión 5.0 y 5.1 que corresponderían a las API 21 y 22. Ahora que la aplicación está finalizada y teniendo en el mercado el siguiente sistema operativo de Android podríamos añadir la siguiente API. Sin embargo, con la versión 6.0 de Android la manera de pedir al usuario los permisos cambia, y ahora tienen que ser en mitad de la ejecución.

Tal y como se ve en la siguiente imagen obtenida desde ticplace [17] , se puede apreciar que fue un acierto la elección de la versión Android, ya que solo la versión 5.0 y 5.1 ocupan más del 30% de la cuota del mercado actual y con vista al futuro ese porcentaje incrementará mientras las versiones anteriores van quedándose obsoletas.

Version	Codename	API	Distribution
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.7%
4.1.x	Jelly Bean	16	9.0%
4.2.x		17	12.2%
4.3		18	3.5%
4.4	KitKat	19	36.1%
5.0	Lollipop	21	16.9%
5.1		22	15.7%
6.0	Marshmallow	23	0.7%

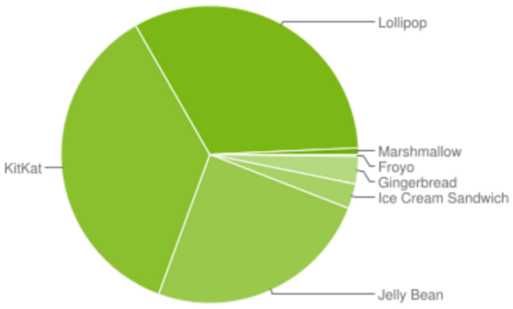


Ilustración 39. Cuota del mercado de versiones de Android

10.11.1. Pantallas

Antes de enseñar las pantallas de Zaintza Pi, indicaremos los colores que hemos seleccionado y la razón. Para evitar que algunos usuarios tuviesen dificultades con la

visibilidad de la aplicación, usamos una rueda de color [18] para elegir colores que tuviesen altos contrastes y que no impidiesen la visibilidad a aquellos usuarios con deficiencias visuales, todo ello siguiendo un patrón de colores verdes para dar la sensación al usuario de seguridad y de tranquilidad.

A continuación se explican cada una de las pantallas.

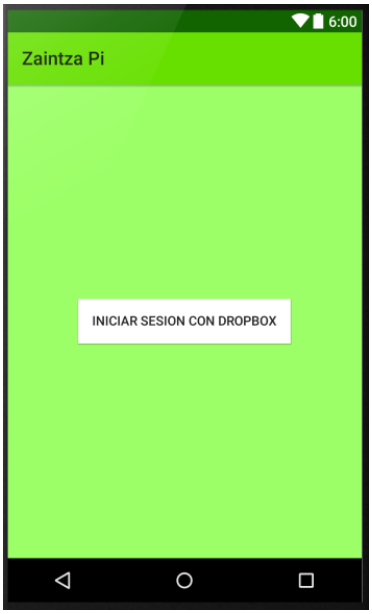


Ilustración 40. Pantalla inicial

layout_jarduera_nagusia.xml

Esta es la pantalla de inicio de la aplicación, una vez hacemos clic en el botón nos mandará al navegador a la página de Dropbox, donde tendremos que autentificarnos y permitir la aplicación Zaintza Pi de Dropbox para poder seguir.

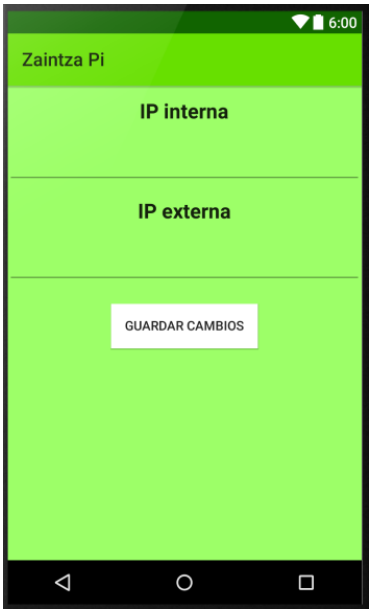
Ilustración 41. Pantalla principal

dropbox_jarduera.xml

Esta es la pantalla principal de la aplicación, donde tendremos toda la gestión de la aplicación. Aquí llegaremos una vez habiéndonos autentificado.

Podremos sacar una lista de las capturas que hay en la cuenta de Dropbox, podremos sacar una captura en el momento, podremos activar y desactivar el sensor de movimiento, y podremos cambiar la configuración de la red por si estas dentro la misma red que la RPi o estas desde fuera de la red.

A pesar de que en la pantalla no se visualice, desde esta pantalla podremos acceder a la siguiente, que es la de configuración de red, mediante un botón en la barra de la cabecera.



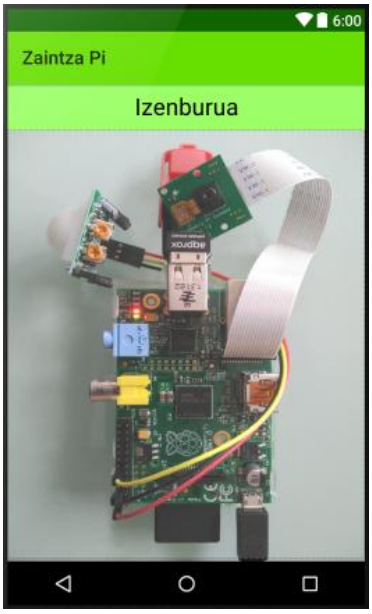
konexioa.xml

Esta es la pantalla mencionada de configuración de red, a la cual se llega a través de un botón de menú en la cabecera de la pantalla anterior.

Aquí configuraremos tanto la IP interna estática que tiene la RPi, como la IP externa que le hemos dado en forma de dominio.

Una vez guardados los cambios, nos llevará de vuelta a la pantalla anterior.

Ilustración 42. Pantalla de conexiones



argazkia.xml

Aquí tenemos la pantalla que se mostrara tanto si le pedimos en la pantalla principal que tome una captura a la RPi como si seleccionamos un ítem de la lista de capturas que se muestra en la siguiente pantalla.

Se le ha añadido la opción de poder agrandar y empequeñecer la imagen, ya que es un sistema de vigilancia para poder hacer zoom y ver con mayor exactitud lo que nos interesa de la imagen. Esta opción se le ha podido dar haciendo la imagen del tipo TouchView [19] , que está disponible en GitHub, reutilizando el código disponible y cogiendo sólo la parte que nos interesaba para conseguir la funcionalidad mencionada.

Ilustración 43. Pantalla de captura

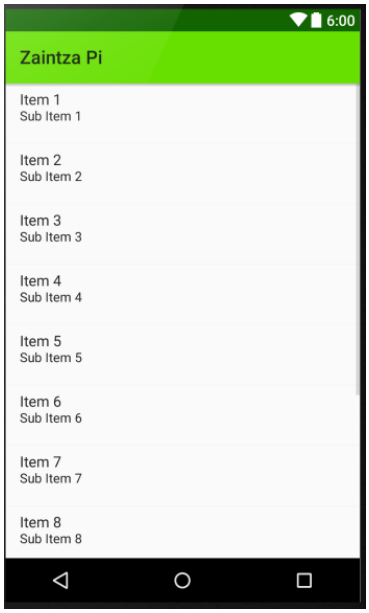


Ilustración 44. Pantalla de listado

argazki_zerrenda.xml

En esta pantalla veremos una lista de las capturas que tiene el usuario en su carpeta de la aplicación Zaintza Pi en su cuenta de Dropbox.

Cada ítem de la lista será seleccionable, lo que nos llevara a la pantalla anterior de la captura, y también tendrá un botón para borrar la captura.

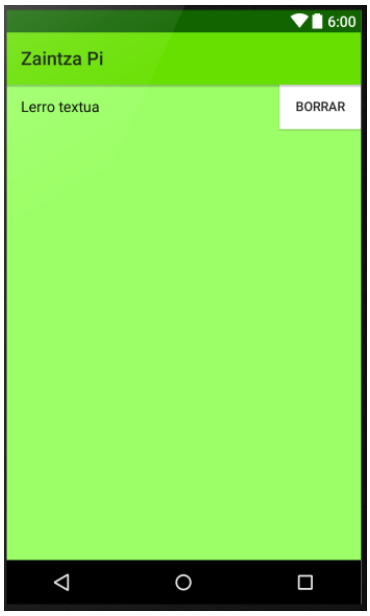


Ilustración 45. Pantalla de elemento del listado

lista_laguntzailea.xml

Este sería el formato de cada ítem de la lista anterior. Debido a que queremos que cada ítem tenga un botón de borrado, en vez de usar una vista predefinida, también hemos tenido que diseñar esta vista.

10.11.2. Programación

Una vez explicadas las pantallas, el siguiente paso es informar acerca de las clases y las funcionalidades.

El usuario abrirá la aplicación y entrará en la primera pantalla de la clase "JardueraNagusia", donde tendrá un botón de inicio de sesión con Dropbox y donde se recogerán datos almacenados en la clase "Globalak" (los cuales serán datos definidos para el primer uso o almacenados de usos anteriores del usuario para mantener sus configuraciones realizadas). Los datos recopilados en este momento serán las URL de conexión interna y externa, la selección de estado del switch de selección de red y el

estado del sistema de detección de movimiento, bien para estado automático como para manual.

En todo momento de la aplicación, el usuario podrá usar el botón de retroceso del propio dispositivo para retroceder a la actividad anterior.

Cuando el usuario haga clic en el botón de "inicio de sesión" con Dropbox se lanzará la siguiente actividad de la clase "DropboxJarduera" que será la pantalla principal de Zaintza Pi. Antes de que el usuario pueda ver la pantalla, el usuario será dirigido al proceso de autentificación de Dropbox mediante el navegador, enviándole las claves para que Dropbox las asocie con Zaintza Pi, donde el usuario permitirá el acceso de la aplicación Dropbox.

Una vez permitido el acceso, volverá a la aplicación y tendrá las siguientes funcionalidades a su disposición:

Primero, tendrá en la barra superior una opción de menú de configuración de conexiones, que en el caso de seleccionarla nos dirigirá a la actividad de la clase "Konexioa", donde podremos definir la IP interna y externa de la RPi para las llamadas posteriores a esta. En el caso de que no esté bien configurado o de que el switch de selección de red (que luego veremos) esté en mala posición, nos saldrá un pop-up de que ha habido problema con la conexión. Una vez le demos a guardar, nos devolverá a la actividad anterior.

Segundo, también tendremos la opción de generar un listado con las capturas de Dropbox: al darle al botón del listado este hará una llamada asíncrona de la clase "DeiAsink" del tipo "FitxategienPathak" que recogerá los nombres de las capturas de la carpeta de la aplicación, y mediante la llamada asíncrona del tipo "JaitsiArgazkia" nos descargará las imágenes que no tengamos en el dispositivo de las que hay en la cuenta de Dropbox. Después, nos creará la vista con las capturas, cada elemento de la lista de la clase "ArgazkiZerrenda" será del tipo "ListaLaguntzailea" para poder añadir el botón de borrado a cada uno de los ítems.

Cada elemento de la lista tiene dos funcionalidades dependiendo de que se pulse el botón de borrado o el nombre de la captura. Si pulsamos en el botón de borrar, se harán tres acciones: 1) se eliminará la imagen del dispositivo en el caso de que se hubiese descargado; 2)se hará una llamada asíncrona de la clase "DeiAsink" del tipo "WebDeiaEzabatu" para ejecutar el Servlet de la RPi llamado "Fitxategia_ezabatu.java" llamando a su URL correspondiente y pasándole como parámetro el nombre del fichero para borrar, y así borrando también de la RPi la imagen que borramos de la lista; 3) Por último, la imagen se borrará de la cuenta Dropbox mediante la clase asíncrona "DeiAsink" del tipo "FitxategiaEzabatu". En cambio, si pulsamos el nombre de la captura, nos creará una actividad de la clase "Argazkia" donde nos mostrará el título de la captura y la captura, siendo esta de tipo "TouchView" para poder agrandar o empequeñecer.

Tercero, si volvemos a la actividad de "DropboxJarduera", tendremos la funcionalidad de sacar una captura en el momento. Al pulsar este botón estaremos entrando en el control manual de la RPi, con lo que si el control automático para el sensor del movimiento estaba activado, se apagará y pasará control manual. Seguidamente, creará una llamada asíncrona de la clase "DeiAsink" del tipo "WebDeiaArgazkia" que ejecutará el Servlet, el cual a su vez lanza el script de Python para hacer una captura y subirla a Dropbox. A continuación, creará otra llamada asíncrona de la clase ya mencionada del tipo "JaitsiArgazkia", que descargará la imagen que acabamos de subir a Dropbox a nuestro dispositivo. Finalmente, se buscará en el dispositivo, para enseñar en una nueva actividad de la clase "Argazkia", una captura que tenga una fecha posterior a la pulsación del botón.

Por último tendremos dos swicth para selecciones de estado en la actividad de "DropboxJarduera", uno para seleccionar el estado de la red (es decir, si estamos dentro de la misma red que la RPi o si estamos accediendo desde fuera de esa red), y otro para activar el sensor de movimiento y poner el sistema en modo automático, o para ponerlo en modo manual. Al entrar en modo automático, encenderemos la funcionalidad de notificaciones de la clase "NotifikazioBilaketa" para que si el sensor detecta movimiento estando la aplicación cerrada, se cree una notificación de alerta con la opción de ir directo a la clase "Argazkia", donde se enseñará la captura que hizo saltar el mecanismo. La alerta será sonora, tendrá vibración para los que tengan discapacidades auditivas, y encenderá el LED de notificaciones en caso de disponer de tal opción.

Este último switch de configuración de modo activo o manual, creará una llamada asíncrona de la clase "DeiAsink" del tipo "WebDeiaMugimendua" y otra del tipo "WebDeiaAmatatu". Cada vez que se active se lanzará el primero, el cual ejecutará el Servlet que ejecuta el Python en el que está integrado cámara, sensor y Dropbox-uploader. En caso de desactivación, se lanzará el segundo que ejecuta el Servlet que mata los scripts de Python activos en la RPi por motivos previamente explicados.

De esta forma, se finaliza el desarrollo de la aplicación Android de Zaintza Pi.

11. VERIFICACIÓN Y EVALUACIÓN

Separaremos las verificaciones y evaluaciones en dos partes, por un lado, los Python y Servlet, y por el otro lado las funcionalidades de la aplicación Android. Es relevante informar de que una vez el proyecto ha finalizado, todas las tareas realizan lo esperado.

Tabla 5. Pruebas de Python y Servlets

Elemento a probar	Tarea a realizar
Python Cámara + Dropbox-Uploader	Sacar una foto y subirla a Dropbox
Python Cámara + PIR + Dropbox-Uploader	Detectar movimiento, sacar una foto y subirla a Dropbox
Servlet Captura	Ejecutar Python Cámara + Dropbox-Uploader
Servlet Movimiento	Ejecutar Python Cámara + PIR + Dropbox-Uploader

Servlet Borrar	Recibir como parámetro y borrar fichero de RPi
Servlet Apagar Python	Matar Pythons activos
Servlet configuración paso 1	Pasos de OAuth: Conseguir token de acceso pasando llaves de aplicación y usarlos para la autentificación
Servlet configuración paso 2	Aceptada la autentificación guardar el fichero de configuración para Dropbox-uploader

Tabla 6. Pruebas Android

Elemento a probar	Tarea a realizar
Botón inicio sesión	Enviar al usuario a Dropbox para dar permisos a la aplicación
Botón listado	Enseñar listado de capturas
Botón captura	Sacar una captura y enseñarla
Menú conexión	Abrir menú de configuración de red
Botón guardar	Guardar los cambios
Switch movimiento	Activar y desactivar sensor de movimiento
Switch red	Cambiar configuración de red
Notificaciones	Notificación de movimiento con el sensor en modo automático
Llamada asíncrona a API de Dropbox para listado de capturas	Conseguir listado de capturas de la cuenta de Dropbox
Llamada asíncrona a API de Dropbox para descargar imagen	Descarga imagen de Dropbox a dispositivo
Llamada asíncrona a API de Dropbox para eliminar imagen	Borrar imagen de la cuenta de Dropbox
Llamada asíncrona a servidor web de la RPi para sacar foto	Ejecutar Python de sacar foto en la RPi
Llamada asíncrona a servidor web de la RPi para encender sensor	Ejecutar Python de encender sensor en la RPi
Llamada asíncrona a servidor web de la RPi para borrar imagen	Borrar imagen de RPi
Llamada asíncrona a servidor web de la RPi para matar Pythons	Matar Python activos en la RPi

12. CONCLUSIONES Y TRABAJO FUTURO

La estimación del tiempo para cada tarea fue realizada antes del proyecto, y se ha podido mantener así en gran medida, ya que la mayoría de tareas han tomado aproximadamente el tiempo estimado. No obstante, la parte de Android ha tomado más tiempo del esperado ya que la falta de conocimientos delimitaba los progresos. Es decir, en cuanto al tiempo de planificación asignado para la tarea de programación de Android, esta nos resultó muy difícil, costaba hacer progresos mínimos dado que la formación en dicho entorno era escasa. A pesar de haber realizado la asignatura optativa del cuarto curso del grado, los conocimientos que conseguimos fueron muy útiles pero distaban de poder ayudar tanto como necesitaba para el desarrollo de Zaintza Pi, y por ello, se alargo el tiempo de planificación para esta tarea. GitHub ayudó mucho pudiéndome dejar tener en el trabajo las últimas modificaciones que había realizado en casa, y aprovechar los descansos para seguir intentando avanzar. Para finalizar Zaintza

Pi, tuvimos que informarnos a través de lecturas y formarnos realizando MOOCs [15] que nos ayudasen con la programación y en conseguir un resultado como el planteado.

Como resultado, en la siguiente tabla se aprecia el desvío de horas de las calculadas inicialmente.

Tabla 7. Desglose de horas por bloques

Bloque	Horas Estimadas	Horas Reales
SW de Raspberry Pi	30h	25h
Cámara	4h	4h
PIR	8h	7h
Python Cámara	4h	4h
Python PIR	4h	4h
Python Dropbox-uploader	10h	8h
Proyecto Dropbox	2h	2h
Python Cámara + PIR + Dropbox	4h	4h
Elegir tipo de servidor web	4h	2h
Configuración servidor web	20h	15h
Configuración DUC	4h	4h
Aplicación Android	90h	130h
Pruebas	40h	40h
Documentación	60h	60h
Reuniones con el tutor	20h	20h
TOTAL	304h	329h

Dado que el proyecto se comenzó a desarrollar hace tiempo, en el transcurso de este se han sacado nuevas versiones, como por ejemplo la API de Dropbox, que se ha actualizado a la versión 2, o el nuevo sistema operativo de Android. Por este motivo, y añadido a que en el transcurso del desarrollo del proyecto íbamos percibiendo que habría sido interesante crear o modificar ciertas partes, hemos dejado estas como recomendaciones de tareas para el futuro. Estas tareas son las siguientes.

Sería interesante desarrollar una parte web para usuarios sin dispositivos Android y para gestionar desde un navegador el sistema de vigilancia.

Se podría, usando el fichero de literales de Android Studio, traducir la aplicación y poder tenerla para distintos idiomas.

La aplicación se vería enriquecida, si en lugar de capturar únicamente foto, se podría añadir la captura de video, reciclando el script de Python que capturaba imágenes sólo cuando detectaba movimiento y no capturaba otro hasta que ese movimiento finalizaba. Se podría grabar un video mientras detectase un movimiento, por ejemplo reutilizando lo que tenemos, o sacando capturas mientras el movimiento es el mismo y juntándolas todas como un clip cuando este finalizase.

Esta sería una actualización más que necesaria para el proyecto, con vista al futuro, ya que la API que Dropbox usaba hasta el momento era la versión 1, pero en este tiempo han sacado una segunda versión y le han puesto fecha de caducidad a la primera.

Cuando se comenzó con el desarrollo la API v2 estaba recién salida y nos pareció más sensato trabajar con la versión estable, que era la API v1.

Otra actualización necesaria, aunque esta sería por temas de seguridad, sería modificar el código de Dropbox-uploader para que en vez de OAuth 1 use la versión nueva OAuth 2 para la autentificación entre RPi y Dropbox. Por la magnitud del proyecto no nos pareció adecuado modificar el código obtenido para realizar esta modificación, porque nos parecía una tarea sumamente compleja modificar el código para que en vez de usar OAuth 1 usase la segunda versión, ya que abría que entrar al detalle a ver todo lo que la aplicación hacia y seguramente reescribir la mayor parte del código.

Debido a que ha salido la versión 6.0 de Android, habría que actualizar los permisos de la app si se tuviese interés en abarcar el mercado hacia el futuro. Con la nueva actualización, los permisos de las aplicaciones se piden en plena ejecución y no son parámetros definidos como hasta ahora. Por lo que, sabiendo lo mucho que nos había costado llegar a donde estábamos en el desarrollo de la aplicación y sin comprender muy bien como habría que modificar el código para la 6.0, decidimos bloquear su uso a sistemas Android 5.0 y 5.1.

13. BIBLIOGRAFÍA

[1] URL de la página de Adafruit donde se detallan los pasos a seguir para la instalación de WebIDE.

https://learn.adafruit.com/webide/overview

[2] Imagen para señalar lugar de cámara en la RPi

https://es.wikipedia.org/wiki/Raspberry_Pi

[3] Imagen de GPIO de la RPi

http://www.recantha.co.uk/blog/?p=13491

[4] Página de Dropbox donde crearemos la cuenta de usuario para almacenamiento, y la que usaremos para el entorno de desarrollo a la hora de crear una aplicación.

https://www.dropbox.com

[5] Página del entorno de desarrollo de Dropbox donde crearemos la aplicación.

https://www.dropbox.com/developers/apps

[6] Dropbox-Uploader es un proyecto de Andrea Fabrizi publicado como público en GitHub y el que nos permite la subida de ficheros mediante script y usando Oauth1 a una cuenta configurada en la primera ejecución del script.

https://github.com/andreafabrizi/Dropbox-Uploader

[7] El logotipo de Zaintza Pi que se usará para la aplicación Android y Dropbox, se usó mediante las imágenes de las siguientes dos páginas:

http://www.taringa.net/post/celulares/18733208/Mis-Aplicaciones-Android-Para-Vos.html

http://www.danasojkj.top/raspberry-pi-logo-vector/

[8] Páginas de información para los servicios y niveles de ejecución a la hora de automatizar el arranque del servicio de Tomcat 8.

https://www.linux.com/news/introduction-services-runlevels-and-rcd-scripts

http://www.raibledesigns.com/tomcat/boot-howto.html

[9] Página de NO-IP donde crearemos la cuenta para DUC NO-IP.

https://www.noip.com/

[10] Página de para saber la IP externa de nuestro router.

http://myip.es/

[11] Pasos a seguir para la instalación de DUC en la Raspberry Pi.

http://www.noip.com/support/knowledgebase/install-ip-duc-onto-raspberry-pi/

[12] Código de la aplicación Android Zaintza PI.

https://github.com/aechezarraga003/ZaintzaPi

[13] Android Studio para el desarrollo de aplicaciones Android.

https://developer.android.com/studio/index.html

[14] GitHub para almacenar los progresos de Android Studio.

https://github.com/

[15] MOOC de Android.

https://courses.edx.org/courses/course-v1:UAMx+Android301x+3T2015/info

[16] Librería Dropbox API v1 para Android.

https://www.dropbox.com/developers-v1/core/sdks/android

[17] Cuota de mercado Android por versiones.

http://www.ticplace.es/cuota-de-mercado-android-por-versiones/

[18] Rueda de color para evitar dificultades con la visibilidad de la aplicación

http://gmazzocato.altervista.org/colorwheel/wheel.php

[19] TouchView para poder hacer zoom en las imágenes.

https://github.com/Dreddik/AndroidTouchGallery/tree/master/library/src/ru/truba/touchgallery/TouchView

[20] Ninjamock para el prototipo de la aplicación Android.

http://ninjamock.com/