UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DESARROLLO DE UN SISTEMA DE VIGILANCIA CON SERVIDOR VPN RASPBERRY PI Y APP PARA MÓVIL CON ANDROID STUDIO

Jesús Marín Rodríguez

Tutor: Ángel Héctor García Miquel

Cotutor: Francisco José Martínez Zaldívar

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-20

Valencia, 2 de abril de 2020

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

En este proyecto se ha desarrollado un sistema de video vigilancia utilizando una Raspberry Pi como servidor. El servidor tendrá configuración de Red Privada Virtual (VPN). El sistema debe tener la posibilidad de conexión a internet mediante conexión inalámbrica o por cable a un router con acceso a internet. El control de las cámaras se llevará a cabo desde cualquier terminal móvil con sistema operativo Android mediante una aplicación diseñada con el IDE Android Studio. El sistema deberá tener la posibilidad de conectarse a las cámaras en tiempo real, así como tener un sistema de alarma al detectar presencia (cambio del entorno), avisando mediante un email y/o con una alarma sonora en el móvil. Cuando se detecte presencia se tomarán imágenes y/o video con la cámara. Estas imágenes se guardarán en una carpeta de la Raspberry Pi, así como se subirán a un servidor al que se pueda acceder desde la aplicación para ver las imágenes.

Resum

Desenvolupament d'un sistema de vídeo vigilància utilitzant una Raspberry Pi com a servidor. El servidor tindrà configuració de Xarxa Privada Virtual (VPN). El sistema ha de tindre la possibilitat de connexió a internet mitjançant connexió sense fils o per cable a un router amb accés a internet. El control de les càmeres es durà a terme des de qualsevol terminal mòbil amb sistema operatiu Android mitjançant una aplicació dissenyada amb l'IDE Android Studio. El sistema ha de tindre la possibilitat de connectar-se a les càmeres en temps real, així com tindre un sistema d'alarma al detectar presència (canvi de l'entorn), avisant mitjançant un correu electrònic i / o amb una alarma sonora en el mòbil. Quan es detecte presència es prendran imatges i / o vídeo amb la càmera. Aquestes imatges es guardaran en una carpeta de la Raspberry Pi, així com es pujaran a un servidor al que es puga accedir des de l'aplicació per a veure les imatges.

Abstract

In this project has been developed a video surveillance system using a Raspberry Pi as a server. The server will have Virtual Private Network (VPN) configuration. The system must have the possibility of connecting to the Internet via wireless or cable connection to a router with internet access. The control of the cameras will be carried out from any mobile terminal with Android operating system through an application designed with the IDE Android Studio. The system must have the possibility of connecting to the cameras in real time, as well as having an alarm system when detecting presence (changing the environment), warning by email and / or with a sound alarm on the mobile. When presence is detected, images and / or video will be taken with the camera. These images will be saved in a Raspberry Pi folder, as well as uploaded to a server that can be accessed from the application to view the images.

Índice

Capítulo 1. Introducción

Nos encontramos en una época en la que vivir sin hacer uso de la tecnología parecería prácticamente imposible, aunque lo realmente asombroso es el ritmo exponencial en que se producen los avances tecnológicos.

Estamos acostumbrados a que salgan novedades tecnológicas día sí y día también, y la mayoría de estas novedades tienen como objetivo mejorar la calidad de vida de las personas.

Este proyecto va a estar centrado en el aprovechamiento de los recursos tecnológicos con el fin de ofrecer mayor seguridad a los ciudadanos, ya que por mucho que queramos, no es posible contar con un cuerpo de seguridad en cada punto de la localidad con el fin de vigilar y controlar los actos vandálicos que se producen en las diferentes localidades de nuestra geografía, y más concretamente, en los pequeños pueblos de nuestro país, que están bastante más desprotegidos debido a su menor número de habitantes.

Hasta no hace muchos años, los ayuntamientos de las pequeñas localidades contaban con un pequeño cuerpo de policías locales que no disponían de los suficientes medios para combatir la actividad delictiva que se produce casi a diario, debido a que se ven obligados a desplegar a sus agentes a lugares donde se cometen dichos delitos dejando desprotegidos otros lugares en consecuencia. Esto se acentúa más aún en los pequeños municipios, donde se cuenta con un cuerpo de Guardia Civil para cada 5 o 6 municipios, siendo prácticamente imposible tener controlado todo el perímetro de todos ellos continuamente.

Gracias al avance de las nuevas tecnologías, se puede contar con la videovigilancia, que se está convirtiendo en el mejor aliado para los cuerpos de protección civil.

Debido a esto, cada vez es más habitual que los municipios y, en general cualquier administración pública, se planteen realizar inversiones en la instalación de cámaras de seguridad con el fin de tener un mayor control de lugares públicos como parques, edificios públicos, principales vías, centros comerciales, ..., que suelen ser los lugares donde mayor vandalismo callejero se produce. También es una buena medida de controlar el tráfico, sirviendo como medida para evitar accidentes y contribuyendo a la seguridad vial.

A parte del vandalismo y los problemas con el tráfico comentados, otra de las principales causas que llevan a la instalación de estos sistemas de videovigilancia por parte de las administraciones es la prevención de actos delictivos como es el caso de los robos. Es cierto que un sistema de videovigilancia no garantiza que no se produzcan estos robos, pero a lo que sí nos van a ayudar es a la identificación de lo ocurrido debido a la calidad de las imágenes y vídeos que seremos capaces de captar de la escena. En este apartado se va a centrar el proyecto que a continuación se presenta.

Capítulo 2. Objetivos

Provengo de un pequeño pueblo situado en la provincia de Albacete de unos 300 habitantes, donde la mayoría de la población de mi pueblo y de los pueblos de alrededor viven de la agricultura y la recolecta de aceituna para la producción del aceite.

Por desgracia, al tratarse de pueblos tan pequeños, no disponen de cuerpos de seguridad ni de ningún tipo de vigilancia, por lo que los robos de herramienta de trabajo (motosierras, vareadores, desbrozadoras, sopladoras, tractores, …) y materia prima también se ha visto incrementado en los últimos años.

También, desde pequeño me han llamado la atención los coches clásicos de los años 90, pero debido a su edad, su sistema de seguridad es fácilmente franqueable, lo que provoca que sean los más fáciles de robar. Estamos en una época donde los robos de vehículos están a la orden del día, y ahora que por fin he conseguido adquirir uno de ellos, me preguntaba cómo podría evitar, en la medida de lo posible, que sea robado.

Por estas razones quería implementar un sistema de videovigilancia desarrollado por mí con el que pudiese controlar si alguien entra en el garaje, en casa, o en la parcela donde se encuentra la cosecha y/o la herramienta.

Con el proyecto desarrollado, se recibirá una alerta en el momento en que nuestro sistema detecte la presencia de alguna persona, y en ese instante, tomará fotos y guardará un vídeo del tiempo en que la persona aparezca en la imagen.

Al desarrollar también una aplicación para teléfonos móviles, de una forma sencilla contamos con la posibilidad de visualizar el vídeo en directo, aunque no se detecte movimiento podremos comprobar en cualquier momento que todo está en su sitio.

Este sistema nos permitirá tener vigilada la vivienda de una forma muy sencilla, así como una plaza de garaje, un comercio, etc.

Se trata de un sistema que, al dotarle de la propiedad de autonomía, puede ser ubicado en el campo, o vigilando una nave donde se tenga la herramienta alejada de la población, ya que, con la placa solar, la batería y el teléfono móvil con SIM, funcionará de igual forma que conectado al router dentro de casa. Esto nos permitiría vigilar zonas muy vulnerables a sufrir robos al estar alejadas de la población.

Con esta propiedad no estaríamos limitados solamente a lugares donde llegue la señal Wi-Fi, sino que podríamos vigilar zonas situadas a kilómetros de nuestro hogar.

Es también una buena opción para las personas que tienen apartamentos en la costa alejados de su habitual lugar de residencia, o casas en el campo, chalets, …, donde no acuden con gran frecuencia y les gustaría comprobar que no ha entrado nadie a invadir su vivienda.

Este proyecto presenta una forma económica de tener un poco vigiladas las cosas de las que vive la gente en los pequeños pueblos de nuestro país, que no poseen las ventajas de las que se dispone en las ciudades, y sus comercios y viviendas no están lo suficientemente protegidas a robos como se cree.

Capítulo 3. Metodología

El proyecto lo comencé adquiriendo en primer lugar el hardware necesario para la realización del mismo. En primer lugar, consiguiendo la Raspberry PI 3B junto con su correspondiente cámara (Raspberry PI Camera V2). A partir de ahí, comencé instalando el sistema operativo y conectando y habilitando el módulo de la cámara.

Una vez tenía todo esto en funcionamiento, me puse a investigar cómo convertir la Raspberry en un servidor VPN, con el fin de darle la mayor seguridad posible al sistema de videovigilancia.

Este proceso, que a priori puede parecer no muy complicado, me llevó varias semanas de trabajo debido a que siempre se producían errores que me obligaban a reinstalar el sistema operativo y comenzar de nuevo.

Una vez conseguí tener mi servidor VPN correctamente instalado, tocaba comprobar su correcto funcionamiento. Para ello, desde mi terminal móvil Android, busque algunas aplicaciones que me permitiesen conectarme a un servidor VPN. Encontré varias, y me aseguré probando con todas que el proyecto estaba correctamente desarrollado hasta ese punto.

Tras ello, tocaba convertir nuestro servidor VPN en un detector de movimiento. Esta tampoco fue una tarea sencilla, pero poco a poco se consiguió completar y verificar.

Teniendo el detector de movimiento correctamente configurado, tocaba realizar un script en Python que se encargase de adjuntar imágenes tomadas al detectarse dicho movimiento y enviarlas por correo electrónico al usuario.

Con todo ello, la parte de trabajo con la Raspberry estaría completada, y entonces tocaba ponerse a desarrollar una aplicación que pudiesen utilizar fácilmente los usuarios para Android en la que plasmar todo este trabajo.

Debido a que el año anterior se cursó una asignatura de Android Studio, se partía con unos ciertos conocimientos acerca de este entorno de desarrollo. Esta tarea no se complicó en exceso, pero llevó su tiempo ya que hacer una aplicación para teléfono móvil desde cero no es una tarea sencilla.

En esta parte, se hizo uso de GitHub para controlar las distintas versiones que se iban creando de la aplicación a medida que se iba avanzando en su desarrollo. También me aseguraba no perder el trabajo en caso de algún imprevisto, ya que tenía toda la aplicación almacenada en la nube.

Cuando la aplicación fue completada, se probó todo en conjunto, es decir, a visualizar en la aplicación el vídeo en directo capturado por el servidor VPN, así como la recepción de correos electrónicos al detectarse movimiento y demás funcionalidades.

Tras esto, para ir un paso más allá, se pensó en dotar a nuestro sistema de la propiedad de "autonomía", es decir, convertir nuestro proyecto en un proyecto autosuficiente. Para ello, se hizo un estudio de consumo y se buscó una batería y una placa solar con la que alimentar el sistema en zonas donde no llegue la corriente eléctrica.

También debíamos abastecer a nuestro servidor de conexión a la red, ya que sino no podríamos conectarnos a el ni recibir correos electrónicos de alerta. Inicialmente se pensó en emplear un módulo GSM/GPRS, pero tras varios días de pruebas, se me ocurrió utilizar un antiguo móvil como Punto de Acceso, cosa que simplificaría bastante la tarea.

Con esto, le daríamos a nuestro proyecto la característica de sistema autosuficiente.

Capítulo 4. Trabajo con la Raspberry PI

4.1 Raspberry PI

Comenzando con una breve descripción sobre qué es una Raspberry PI, comentar que se trata de una placa computadora de bajo coste. De forma resumida podríamos decir que es un mini-ordenador. Esta placa fue desarrollada en el Reino Unido en el año 2011.

El objetivo inicial por el que fue desarrollado era para incentivar el aprendizaje de la informática en escuelas e institutos, ya que tiene únicamente los accesorios necesarios para que un ordenador funcione de la forma más básica posible. Por lo tanto, tendríamos un ordenador muy sencillo por un precio que ronda los 40 euros.

Se le conoce también como una maravilla en miniatura, debido a que, a pesar de su tamaño y precio, tiene un poder de cálculo notable, cosa que nos permite hacer millones de aplicaciones.

En este proyecto, la transformaremos en un servidor VPN con el que visualizaremos vídeo en directo de algunas cámaras, tomaremos fotos cuando se detecte movimiento y enviaremos correos con dichas imágenes a un usuario.

4.2 Virtual Private Network

En primer lugar, comenzaré con la definición de red privada virtual o Virtual Private Nertwork (VPN), que es una tecnología de red de computadoras que permite una extensión segura de la red de área local (LAN) sobre una red pública o no controlada, como es el caso de Internet.

Esta configuración permite que la computadora en la red envíe y reciba datos sobre redes compartidas o públicas como si fuera una red privada con toda la funcionalidad, seguridad y políticas de gestión de una red privada.

En la imagen que se muestra a continuación, se describe muy bien lo anteriormente comentado.

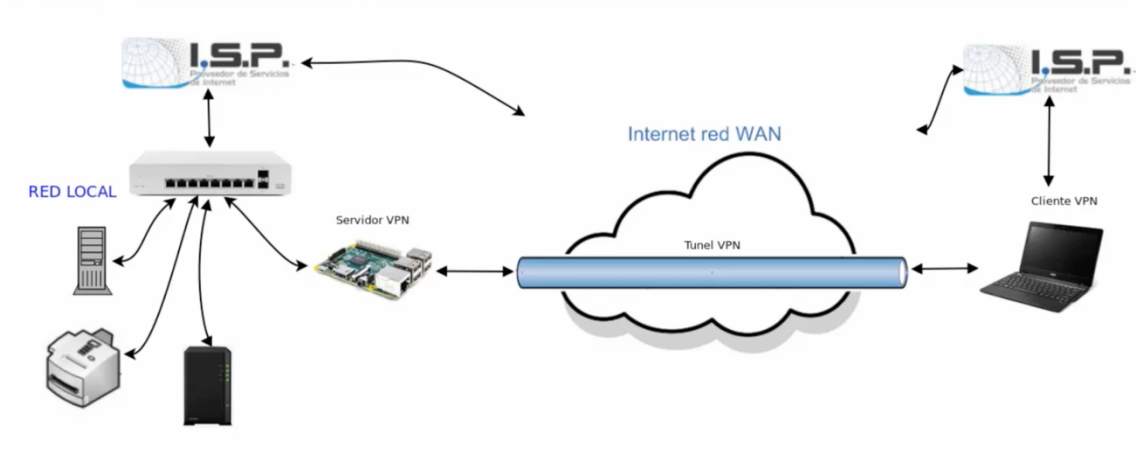

Figura 1. Esquema Virtual Private Network.

La imagen anterior nos muestra un esquema con lo que buscamos, que es acceder a nuestra red local desde el cliente, como si se dispusiese de un cable que conectase dicho cliente con el switch de la red local. Esto físicamente es imposible, ya que necesitaríamos un cable de kilómetros de longitud, por eso se le llama red virtual. Se hace uso de Internet para llevar a cabo esta conexión, conexión que se hace a través de un "túnel", por eso también se le llama red privada. Los datos que circulan por este túnel que se crea, están encriptados. Sólo desde el cliente, con el software apropiado, así como los certificados y claves, se puede acceder a nuestra red local.

La red local consta de un switch con conexión a Internet gracias a un proveedor de servicios, y varios dispositivos conectados a ese switch (impresoras, equipos, teléfonos móviles, …). A ese

switch es donde se conectará la Raspberry PI 3, que en este proyecto va a actuar de servidor VPN, tras la instalación del software correspondiente.

En el otro extremo, tenemos el cliente, que es quién desea conectarse a la red local desde fuera de ella, como ya se ha comentado.

Las principales ventajas del empleo de esta estructura son las siguientes:

- Confidencialidad, integridad y seguridad de los datos.
- Sencillez y reducción en los costos.
- Protege la información más relevante de tu equipo.
- Un servidor VPN oculta la dirección IP real, lo que nos lleva a la siguiente ventaja.
- Dar acceso a contenido no disponible en tu región, ya que podemos evitar las restricciones geográficas.
- Conexiones sencillas entre lugares lejanos.
- Facilidad de creación de canales privados.

Algunos de los ejemplos más habituales del empleo de estas redes privadas virtuales los comento a continuación:

- La posibilidad de conectar dos o más sucursales de una empresa usando como vínculo Internet.
- Permitir a los miembros de un equipo técnico la conexión desde su casa al centro de trabajo, lo que se conoce como acceso remoto a la oficina para teletrabajar.
- Que un usuario pueda acceder a su equipo doméstico desde cualquier lugar remoto con conexión, como por ejemplo un hotel.
- Evitar hackeos y robo de información por parte de nuestro proveedor de Internet.
- También es una manera de esquivar la censura.
- Garantizar seguridad al conectarnos a redes WiFi públicas, redes muy poco recomendables debido a la facilidad que existe a la hora de obtener información por parte de los ISP o usuarios malintencionados conectados a esa red.

Todos estos ejemplos son utilizando Internet como infraestructura.

Los pasos para convertir nuestra Raspberry PI 3 en un servidor VPN se presentan a continuación.

4.2.1 Instalar RASPBIAN y actualizar a la última versión

Comenzaremos con la instalación del sistema operativo Raspbian en nuestra Raspberry PI 3. Lógicamente, se busca la última versión disponible para la instalación. Esta instalación se llevará a cabo con NOOBS. NOOBS (New Out Of Box Software) es un administrador de instalación de sistema operativo sencillo para Raspberry.

Por lo tanto, descargamos la versión de NOOBS que se muestra a continuación, que es la última versión hasta ahora, con fecha del 10 – 07 – 2019.

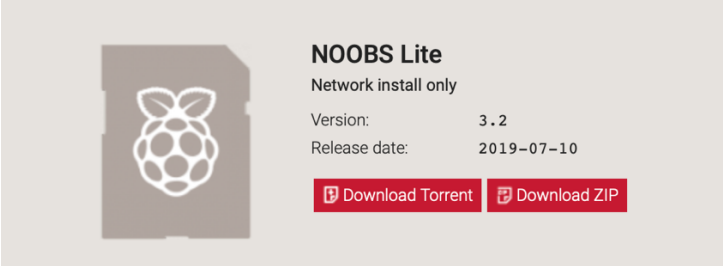

Figura 2. Descarga NOOBS Lite.

Una vez descargado NOOBS, necesitaremos una tarjeta microSD para introducir el instalador en nuestra Raspberry PI.

En primer lugar, formatearemos la tarjeta microSD, y posteriormente, descomprimir el fichero con extensión .zip que acabamos de descargar en la tarjeta.

Hecho esto, la instalación del sistema operativo es una tarea bastante sencilla.

Introducimos la tarjeta microSD en la Raspberry PI, la conectamos a la corriente y a un monitor mediante HDMI, y esperamos a que se nuestra máquina inicie.

Cuando la Raspberry PI inicia, aparece un menú como el que se muestra en la siguiente imagen, donde tenemos un listado de los diferentes sistemas operativos disponibles.

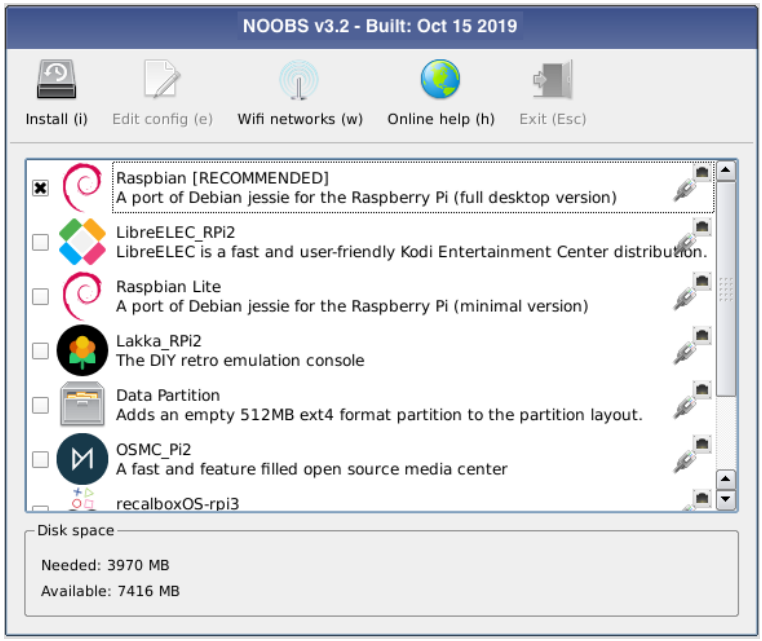

Figura 3. Instalación Raspbian (1).

Tan solo tenemos que elegir el sistema operativo que queremos instalar en nuestra Raspberry (Raspbian en este caso) y pulsar la tecla (i) para instalar, como nos dice la opción de arriba a la izquierda.

Seguidamente comienza el proceso de instalación del sistema operativo seleccionado, por lo que tendremos que esperar a que esto acabe.

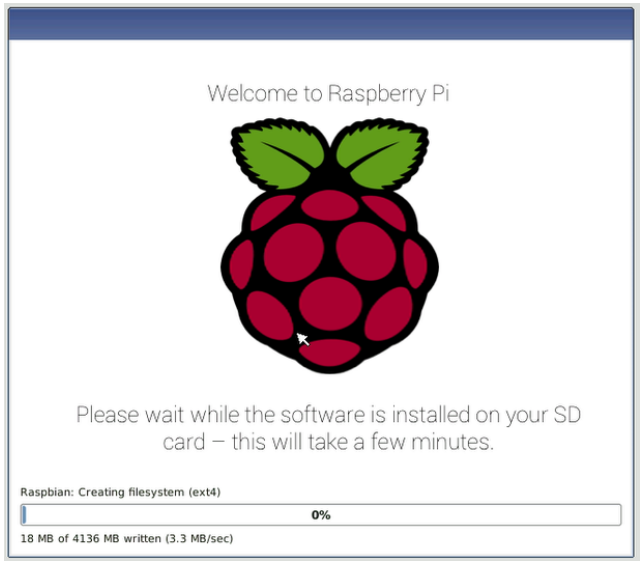

Figura 4. Instalación Raspbian (2).

Una vez termina el proceso de instalación, la Raspberry PI se reinicia y se ejecuta el sistema operativo instalado. Aquí concluye el proceso de instalación.

Si han pasado algunos días desde que descargamos la versión de NOOBS hasta que realizamos el proceso de instalación, puede ser que no estemos instalando la última versión del sistema operativo en nuestra Raspberry PI, por lo que, una vez instalado el sistema operativo, nos vamos al terminal y ejecutamos los siguientes comandos:

sudo apt-get update

sudo apt-get upgrade

El primero de los comandos, actualiza el repositorio de paquetes disponibles y sus versiones, pero no lleva a cabo ningún tipo de instalación ni actualización.

El segundo comando sí que se encarga de llevar a cabo la actualización de los paquetes descargados con el primer comando. Instala las nuevas versiones cuando sea posible.

Con la ejecución de los dos comandos, ya tendríamos nuestro sistema operativo actualizado a la última versión.

4.2.2 Asignar IP fija local a la Raspberry PI

Este paso se podría llevar a cabo desde la consola, como se ha hecho siempre. Pero al tener disponible la versión de escritorio, resulta más visual e intuitivo realizarlo así.

Pinchamos en el símbolo de WiFi, situado arriba a la derecha, y nos vamos a la opción *"Wireless & Wired Network Settings"*. Nos aparecerá la siguiente ventana:

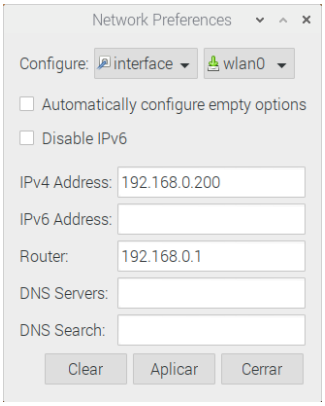

Figura 5. IP fija para la Raspberry PI.

En la casilla de IP Adress, introduciríamos la dirección IP fija que tendrá nuestra Raspberry PI. Esta IP debe estar fuera del rango DHCP de nuestro router, con el fin de evitar conflicto en caso de que la Raspberry PI esté apagada, ya que el router puede darle esta IP a otro dispositivo.

Para evitar esto, elegimos una IP fuera del rango como hemos comentado.

También incluiremos la dirección del router, en su casilla correspondiente, así como los servidores DNS.

Siguiendo estos sencillos pasos, ya tendríamos nuestra Raspberry configurada con una dirección IP local.

Antes de pasar al siguiente paso, por seguridad vamos a cambiar el password de nuestra Raspberry, ya que todas tienen el mismo usuario (pi) con el mismo password (raspberry). Para ello, vamos al terminal y ejecutamos el siguiente comando:

```
passwd
```

Para empezar, nos pide introducir la contraseña actual, que ya hemos dicho que es *raspberry*. Después, nos pide introducir la nueva contraseña. Seguidamente la volvemos a introducir para comprobar, y listo, ya tendríamos nuestra contraseña cambiada.

4.2.3 Configurar un servicio de DNS dinámico

4.2.3.1 Introducción DDNS

En primer lugar, comenzaremos con una pequeña explicación de por qué necesitamos un servidor DNS dinámico (DDNS).

El DNS dinámico es un servicio que permite la actualización en tiempo real de la información sobre nombres de dominio ubicada en un servidor de nombres. El uso principal que se le da es permitir la asignación de un nombre de dominio de Internet a un dispositivo con dirección IP pública variable. Esto permite conectarse con la máquina sin necesidad de conocer la dirección IP de ésta en el momento de la conexión.

El esquema que se muestra en la siguiente imagen trata de esquematizar lo que podría ser nuestra red local.

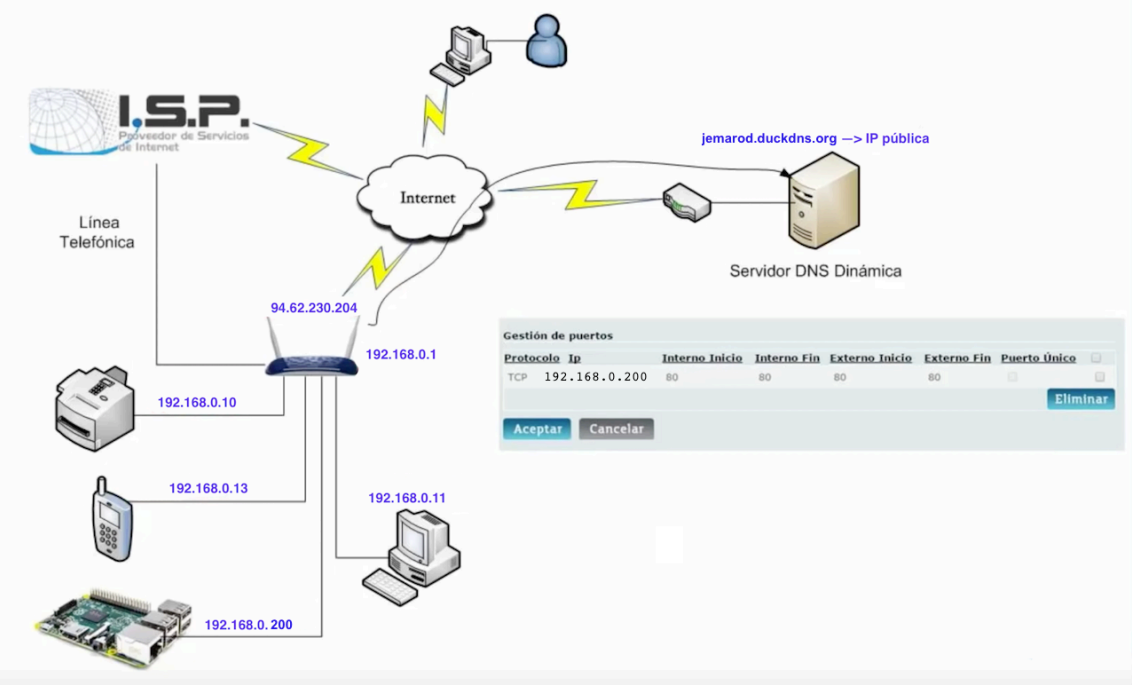

Figura 6. Red Local.

En nuestra red, nos conectamos a través de un Proveedor de Servicios de Internet (ISP), normalmente por cable coaxial o fibra óptica, mediante un router. Nuestro proveedor de servicios nos facilita el acceso a Internet a través de una IP pública.

Esta IP pública, debido a que no hay suficientes IPs fijas para todo el mundo, por así decirlo, es dinámica, lo que quiere decir que cuando apagamos o reiniciamos nuestro router, al encenderlo de nuevo, esta IP habrá cambiado. Esto supone un problema a la hora de conectar servicios en nuestra red local.

También necesitamos una IP fija para nuestra Raspberry PI, ya que cada vez que el router se inicie, asignará a los distintos aparatos conectados una IP que no tiene por qué coincidir con la que tuviesen antes del reinicio del router, pero este problema ya lo hemos solucionado en el paso anterior asignándole una IP fija local fuera del rango de DHCP a nuestra Raspberry PI.

El servidor de DNS dinámica se pondría en contacto con nuestro router, cada vez que este último arrancase, actualizando su IP. De esta forma, no tendríamos que configurar nada en ningún equipo de nuestra red. Con ello, cuando un usuario quiera acceder a la red local, solo tiene que poner el dominio de DDNS, por lo que no será necesario conocer la dirección IP que en ese momento tenga dicha red. El nombre de dominio estará actualizado con la IP pública que en ese momento tenga nuestro router.

4.2.3.2 Candidatos para DDNS

Para llevar a cabo el proceso de configuración de un DNS dinámico, tenemos varias opciones, ya que disponemos de varios proveedores de DNS dinámicos.

Se mostrará a continuación una breve descripción de los más conocidos:

- NO-IP

En primer lugar, se planteó la opción de realizar la configuración mediante este proveedor de DDNS.

NO-IP nos ofrece tanto servicios de pago como gratuitos, pero como es de esperar, estos servicios gratuitos tendrán unas ciertas limitaciones. La principal de estas limitaciones es la caducidad del servicio, que es de 30 días. Más adelante se verá esto al crear el dominio.

Si queremos solucionar esta limitación, no queda otra que pagar.

El servicio tiene un precio de unos 25$ al año, pero tan solo veremos cómo crear una cuenta y cómo configurar un nombre de dominio.

Comenzamos creando una cuenta en NO-IP. Una vez nos hemos registrado, podemos comenzar con la asignación.

Pulsamos sobre la opción de añadir un host nuevo, y veremos lo siguiente:

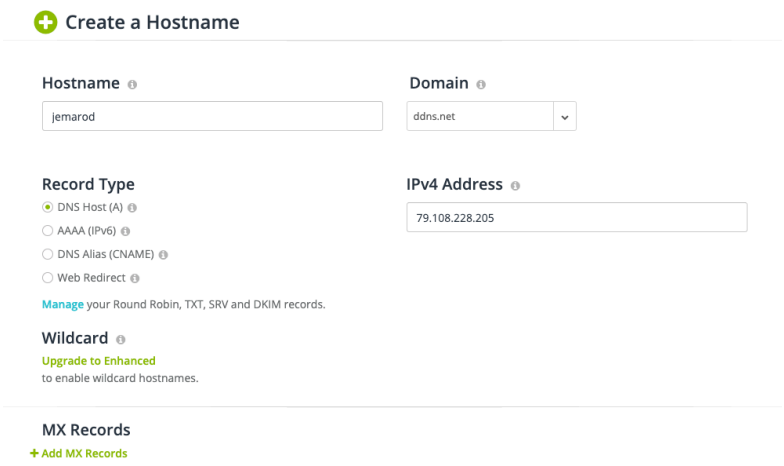

Figura 7. DDNS con NO-IP (1).

Debemos introducir la IP pública que tenemos en ese momento, así como el nombre que queramos tener de dominio, y añadimos el host. Con este paso tan sencillo ya tendríamos el nombre de dominio creado. En la siguiente imagen se muestra el proceso ya finalizado.

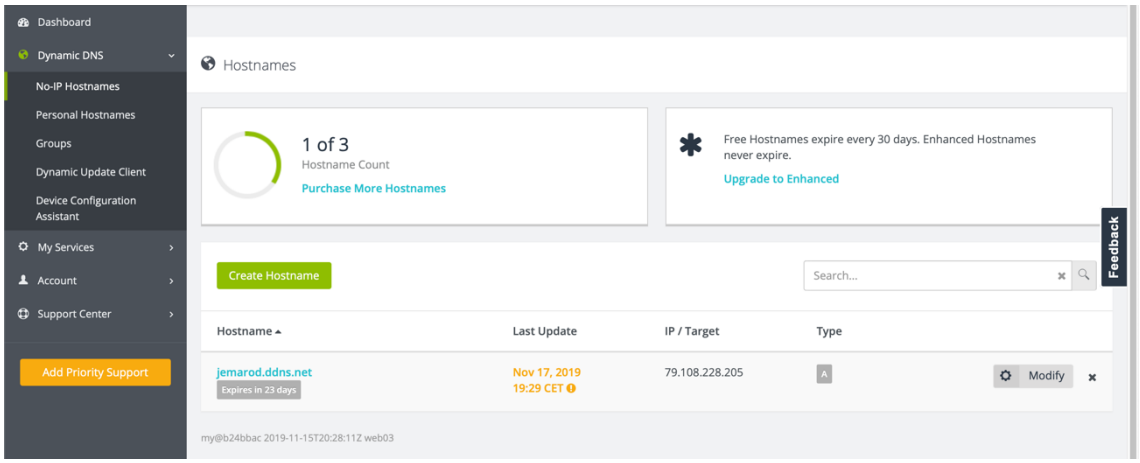

Figura 8. DDNS con NO-IP (2).

Se puede ver debajo del nombre de dominio, que el servicio “expira en 23 días”. Esto es debido a que, al ser un servicio gratuito, tiene una validez de 30 días, tal y como hemos visto antes. Esta es una de las razones por las que finalmente se decidió optar por otro proveedor.

Al pasar este tiempo, repetiríamos el proceso y volveríamos a asignar la IP al nombre de dominio. Esto no es muy práctico, por lo que veremos otros proveedores que nos ofrezcan un servicio gratuito sin fecha de caducidad, como es este caso.

- DynDNS

La siguiente opción que se planteó fue DynDNS, que es una compañía estadounidense que se dedica a soluciones de DNS en direcciones IP dinámicas. Hace unos años ofrecía servicios gratuitos, dando la oportunidad a particulares de crear un servidor en Internet de manera gratuita con una dirección con la forma *midominio.dyndns.com*.

Por estas razones, fue muy popular unos años atrás.

Fue entonces cuando decidieron convertir este servicio en un servicio de pago.

Actualmente no ofrece el servicio gratuito para nuevos clientes, por lo tanto, descartamos este proveedor también.

- Duck DNS

Duck DNS es un servidor DNS totalmente gratuito que nos permite hacer uso de este tipo de servicios sin tener que pagar por él y, sobre todo, sin molestos avisos cada poco tiempo para renovarlo manualmente, como era el caso de NO-IP.

Por lo tanto, el proceso para la configuración de un DNS dinámico lo llevaremos a cabo con el servicio gratuito que nos aporta Duck DNS.

Vamos a la página web del servicio (https://www.duckdns.org) y accedemos con nuestra cuenta de Facebook o Twitter.

A continuación, debemos escoger un nombre de dominio que se encuentre disponible, es decir, que ningún usuario anteriormente haya dado de alta, y lo introducimos aquí:

Figura 9. DDNS con Duck DNS (1).

No será necesario introducir la dirección IP pública, ya que la coge automáticamente del dispositivo desde el que se está llevando a cabo la acción. Con este paso tan sencillo ya tendríamos el nombre de dominio creado. En la siguiente imagen se muestra el proceso ya finalizado.

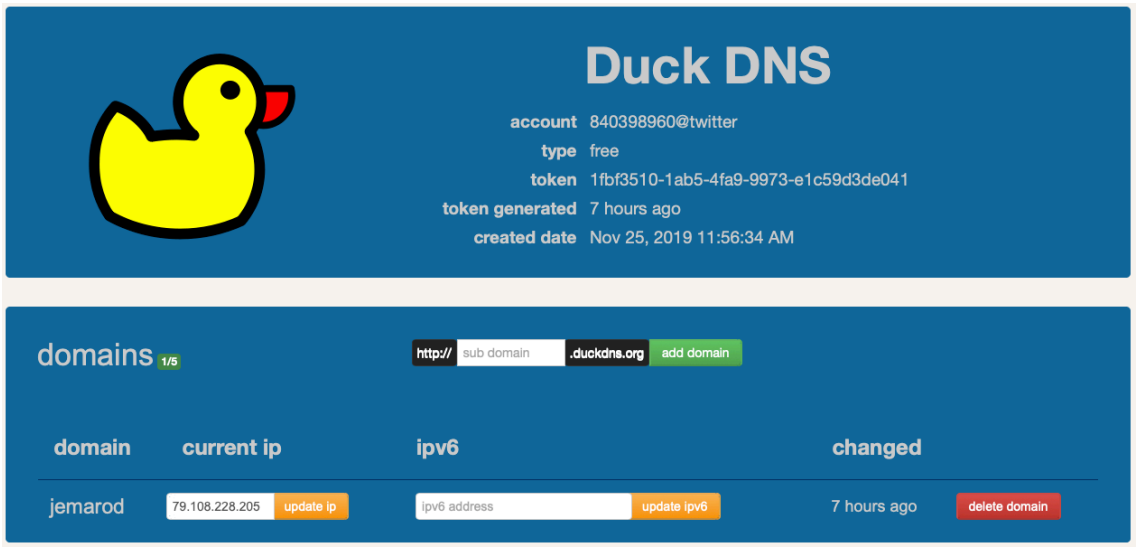

Figura 10. DDNS con Duck DNS (2).

Se puede ver en la parte inferior de la imagen el nombre de dominio, junto a la dirección IP que teníamos en el momento de la realización del proceso.

Podríamos añadir hasta 5 dominios diferentes de forma gratuita con este proveedor.

4.2.4 Apertura de puertos en el router para la conexión VPN

Para este trabajo, se dispone de un router de Vodafone. Para entrar a la configuración del router, introducimos en la barra de direcciones la dirección IP de nuestro router, en mi caso 192.168.0.1. Nos identificamos rellenando usuario y contraseña, que suele estar escrita en una pegatina en la parte posterior del router, y ya estaríamos dentro de nuestro router.

Al router de Vodafone se puede entrar en *"Basic Mode"* y en *"Expert Mode"*. Para las modificaciones que se van a realizar, es necesario estar en *"Expert Mode"*, ya que la otra opción nos limita mucho más en cuanto a posibilidad de configuraciones.

Nos vamos a la pestaña *"Internet"*, y luego a la opción *"Redirección de Puertos"*.

Se nos abre una ventana como la que se muestra en la siguiente imagen:

Figura 11. Apertura de puertos en el Router (1).

Le damos un nombre al servicio y elegimos el dispositivo (la Raspberry debe estar encendida y conectada a la red). La LAN IP, que es la dirección IP fija de nuestra Raspberry PI, la rellenará automáticamente al seleccionar el dispositivo. El protocolo en este caso es UDP, y el tipo PUERTO, ya que solo queremos abrir un puerto, no un rango de puertos. Por último, introducimos el número de puerto que asignaremos a nuestro dispositivo.

El puerto que se suele abrir por defecto es el 1194, pero yo he abierto el 5194 por seguridad. Conviene cambiar los valores por defecto para aumentar la seguridad.

Figura 12. Apertura de puertos en el Router (2).

Una vez hecho esto, vemos en la imagen anterior la redirección de puertos completada, con el dispositivo, su IP y el puerto asignado.

4.2.5 Convertir la Raspberry PI en un servidor VPN

Al tratarse de una aplicación cliente – servidor, necesitamos instalar en nuestra Raspberry PI (servidor VPN) un software, que no es otro que piVPN.

En la parte del cliente, (pc, terminal móvil, …, desde el que vayamos a conectarnos al servidor VPN) también es necesario instalar el correspondiente software para establecer correctamente la conexión. Al implementar una aplicación para terminales móviles, existen en "Play Store" varias aplicaciones para el sistema operativo Android como es el caso de "OpenVPN Connect" o "OpenVPN para Android". Personalmente, he probado las dos aplicaciones tras crear y configurar el servidor VPN y he de decir que funcionan a la perfección ambas, son muy intuitivas y sencillas de emplear, por lo que es indiferente la elección de una u otra.

Pero en este apartado estamos describiendo el trabajo sobre la Raspberry PI.

Por lo tanto, para comenzar con la instalación del software piVPN en nuestro servidor Raspberry PI, tan solo debemos abrir el terminal e introducir el siguiente comando:

curl -L https://install.pivpn.io | bash

Al ejecutar el comando anterior, comienza la descarga de paquetes necesarios para llevar a cabo la instalación de nuestro servidor VPN.

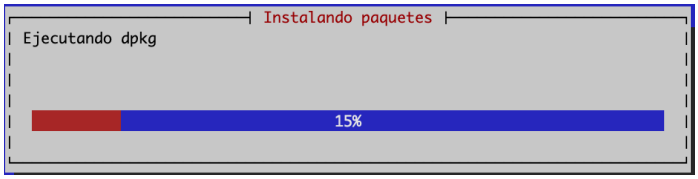

Figura 13. Convertir Raspberry en servidor VPN (1).

Una vez se descarga, nos van apareciendo una serie de pantallas según avanza el proceso de instalación para configurar el servidor VPN tal y como deseemos.

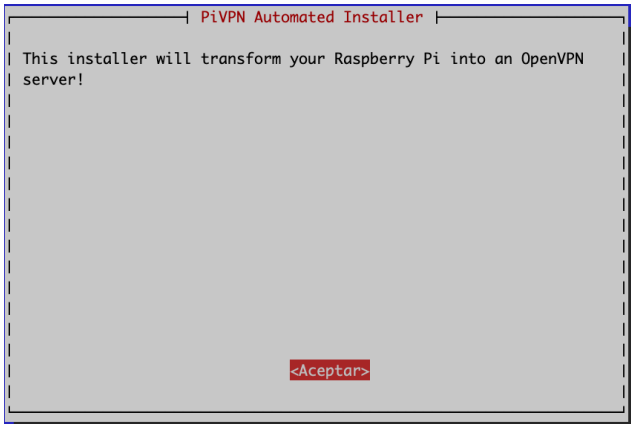

Figura 14. Convertir Raspberry en servidor VPN (2).

La primera de ellas tan solo nos indica que esta instalación convertirá nuestra Raspberry PI en un servidor OpenVPN, por lo que aceptamos y pasamos a la siguiente, que nos dice que PiVPN es un servidor y necesita una dirección IP estática. Esto no debe preocuparnos ya que, en pasos previos, hemos configurado nuestra Raspberry PI con una dirección IP estática (192.168.0.200).

Figura 15. Convertir Raspberry en servidor VPN (3).

Por lo tanto, aceptamos. En la siguiente imagen, nos muestra la dirección IP estática, que coincide exactamente con la que nosotros le habíamos asignado anteriormente.

El Gateway, o puerta de enlace, es la dirección IP de nuestro router.

Figura 16. Convertir Raspberry en servidor VPN (4).

Como todo está correcto, podemos avanzar con el proceso, por lo que continuamos con “Sí”.

La siguiente imagen nos muestra un cuadro informativo, una advertencia sobre que nuestro router pueda asignar esta dirección IP a otro dispositivo y crear un conflicto.

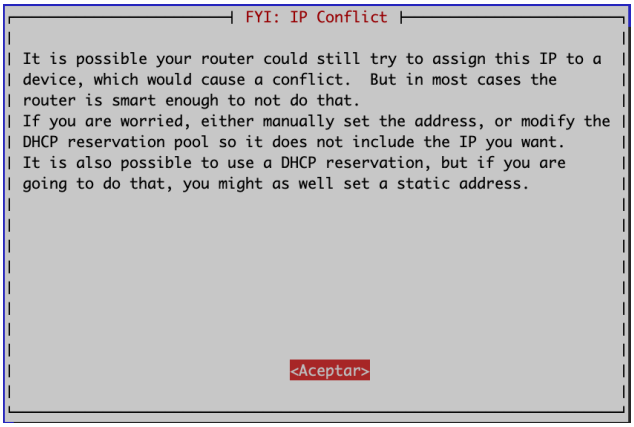

Figura 17. Convertir Raspberry en servidor VPN (5).

Sabemos que esto no va a pasar, ya que nos hemos asegurado de que esta dirección esté fuera del rango DHCP de nuestro router.

Por lo tanto, continuamos con el proceso.

La siguiente captura nos informa de que necesita un usuario local para establecer la configuración.

Figura 18. Convertir Raspberry en servidor VPN (6).

Continuamos con “Aceptar”, ya que el siguiente paso será la creación de dicho usuario.

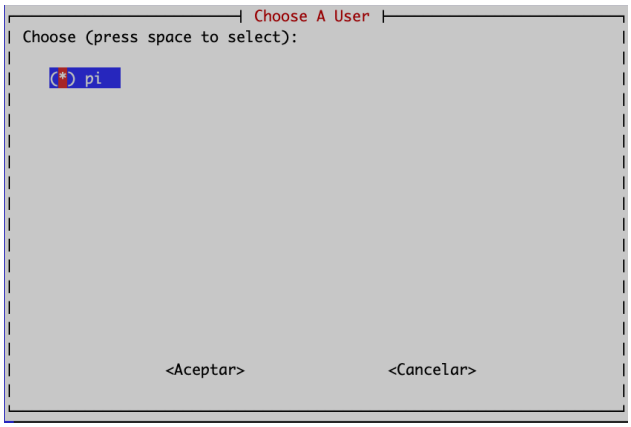

Figura 19. Convertir Raspberry en servidor VPN (7).

Actualmente, el único usuario que hay creado en nuestra Raspberry es el usuario “pi”. Si tuviésemos más usuarios, podríamos elegir con cuál de ellos continuar. Por lo tanto, continuamos con el usuario “pi”.

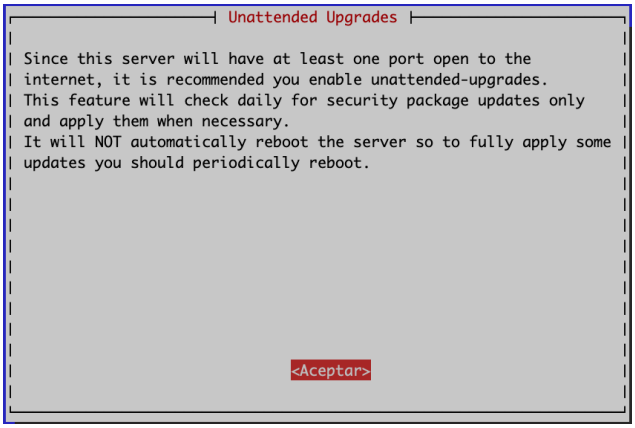

Figura 20. Convertir Raspberry en servidor VPN (8).

En la imagen anterior podemos ver que nuestro servidor VPN se va a conectar a Internet, y nos recomienda que configuremos las actualizaciones desatendidas para que el sistema se nos actualice periódicamente. Aceptamos.

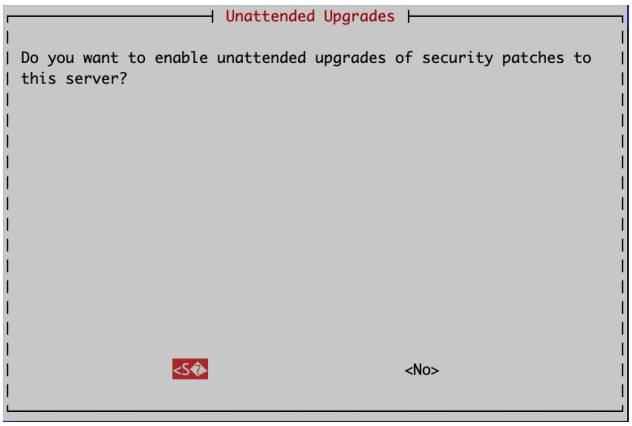

Figura 21. Convertir Raspberry en servidor VPN (9).

Como podemos ver, nos pregunta si queremos habilitar las actualizaciones que anteriormente hemos comentado. Decimos que sí y continuamos con la configuración.

Tras la realización de este proceso, que suele tardar unos 25 – 30 segundos, nos encontramos con la siguiente imagen, donde nos pregunta qué protocolo vamos a utilizar, UDP o TCP.

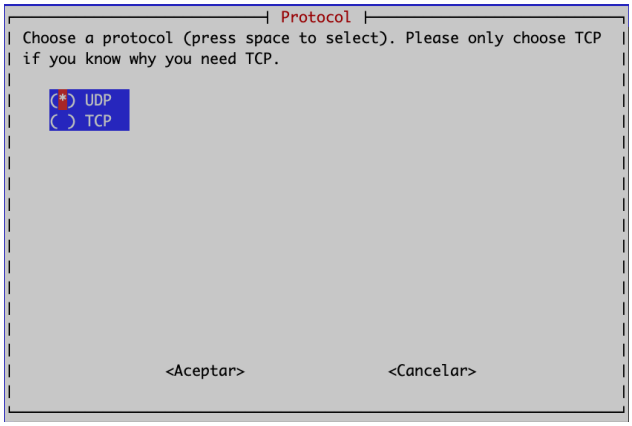

Figura 22. Convertir Raspberry en servidor VPN (10).

Aquí, vamos a recordar que, en la configuración de los puertos de nuestro router, nos encontramos con la misma pregunta. En aquella ocasión, seleccionamos el protocolo UDP. La razón de esta elección se comenta a continuación:

La diferencia está en que UDP es mucho más rápido. A cambio de esta gran ventaja, no garantiza la entrega de paquetes. Esto no supone ningún problema para nuestra VPN. Lo que sí nos interesa es que nuestra VPN sea lo más rápida posible, por lo que volvemos a elegir UDP aquí también y aceptamos para continuar.

A continuación, nos encontramos con una ventana que nos pregunta qué puerto vamos a utilizar para la conexión.

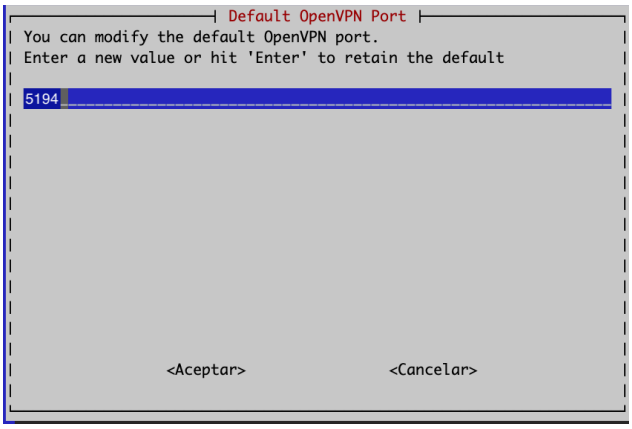

Figura 23. Convertir Raspberry en servidor VPN (11).

Recordando de nuevo la configuración de los puertos del router, vimos que por defecto era el puerto 1194 el que se empleaba, pero por seguridad, con el fin de dificultar un poco las cosas en caso de sufrir algún ataque, lo cambiamos al 5194.

Por lo tanto, volvemos a introducir el puerto 5194 aquí también y aceptamos.

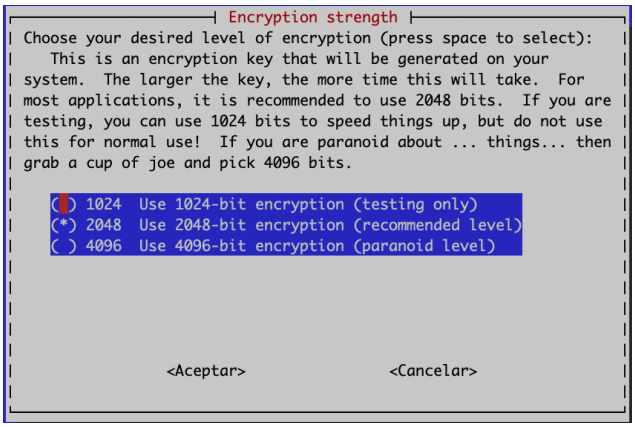

Figura 24. Convertir Raspberry en servidor VPN (12).

En la imagen anterior, nos pregunta la cantidad de bits para la encriptación del túnel.

Por defecto, nos recomienda emplear 2048 bits. Esta cantidad es suficiente para la aplicación que nosotros vamos a implementar. Emplear 4096 bits aumentaría bastante la seguridad, pero también ralentizaría mucho el proceso. Por lo tanto, continuamos con 2048 bits.

Figura 25. Convertir Raspberry en servidor VPN (13).

Seguidamente comienza un proceso que tardará un poco, ya que se tienen que generar los certificados y claves en el servidor. Puede alargarse hasta unos minutos, por lo que paciencia.

Cuando el proceso anterior acaba, nos encontramos con la imagen que se muestra a continuación.

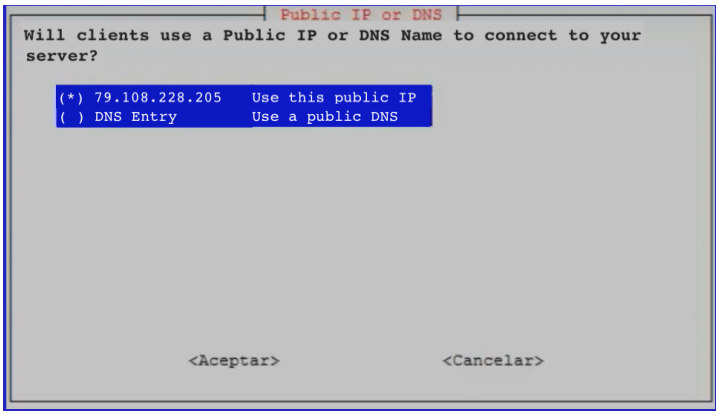

Figura 26. Convertir Raspberry en servidor VPN (14).

Aquí debemos decidir entre seleccionar nuestra dirección IP pública o introducir un nombre de dominio de un servidor DNS. Ya que mediante el proveedor de DNS dinámico Duck DNS creamos el nombre de dominio, pues lo introducimos. Seleccionamos "DNS Entry", y aceptamos.

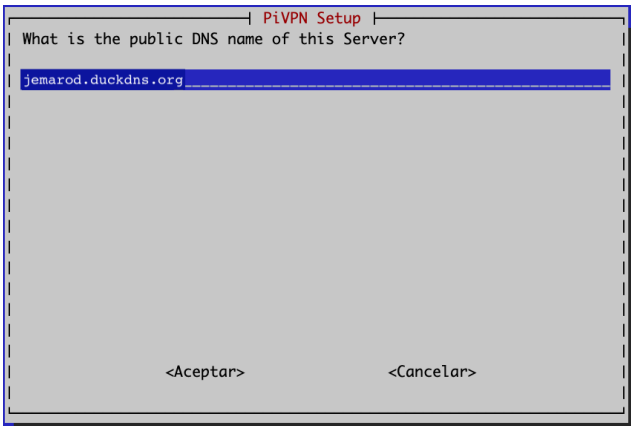

Figura 27. Convertir Raspberry en servidor VPN (15).

Ahora debemos introducir el nombre de dominio que creamos en Duck DNS, que si recordamos era *jemarod.duckdns.org*. Continuamos pulsando en "Aceptar".

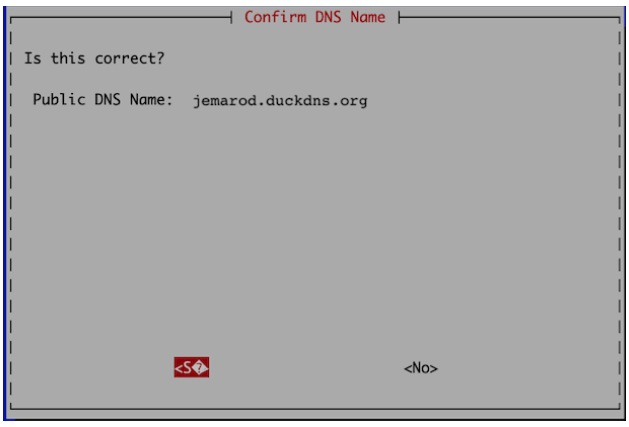

Figura 28. Convertir Raspberry en servidor VPN (16).

Confirmamos que el nombre de dominio es el correcto.

En la siguiente imagen nos pide seleccionar un proveedor DNS para, por ejemplo, cuando navegamos por Internet, asociar los nombres de dominio a una IP. En este caso, dejaremos el que viene por defecto, que es el de Google. Aceptamos y continuamos.

Figura 29. Convertir Raspberry en servidor VPN (17).

En la imagen anterior, tenemos una ventana informativa que nos indica que ejecutemos el comando "pivpn add" para crear los archivos .ovpn de los clientes.

Continuamos.

El proceso está llegando a su fin. La siguiente imagen nos recomienda reiniciar nuestra Raspberry una vez completada la instalación. Aceptamos y continuamos.

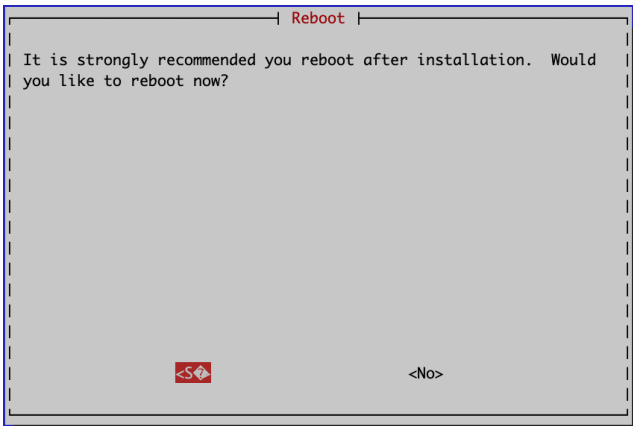

Figura 30. Convertir Raspberry en servidor VPN (18).

Por lo tanto, nos encontramos ante la última de las pantallas, que nos indica que el sistema será reiniciado a continuación.

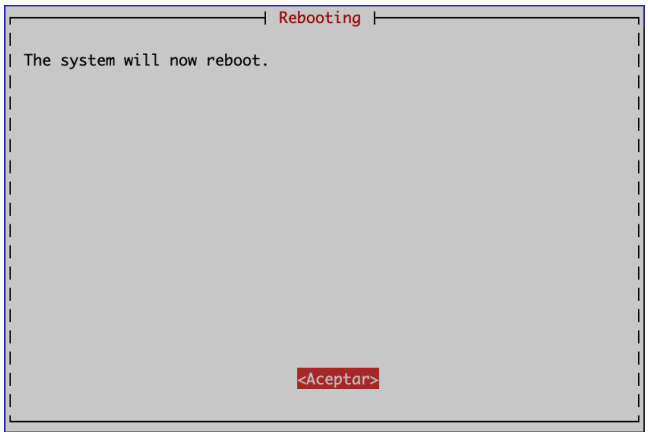

Figura 31. Convertir Raspberry en servidor VPN (19).

Al aceptar, el sistema se reinicia y el proceso ha terminado.

Haciendo un breve repaso de lo que llevamos hasta ahora, comentar que acabamos de instalar el software necesario en la parte del servidor. Ahora tenemos que crear los usuarios que utilizaremos para conectarnos desde los clientes. Al crear estos usuarios, se nos generará un archivo .ovpn, que es el que tendremos que copiar a los clientes desde los que queramos utilizar el servicio VPN.

Tal y como nos decía una de las últimas capturas (Figura 29), ejecutaremos el comando “sudo pivpn add” para crear los perfiles que vamos a utilizar en nuestro servidor VPN.

Seguidamente, debemos introducir el nombre que queramos darle al cliente VPN. También nos pide una contraseña, por lo que la introducimos. Comenzará entonces el proceso de generación de la clave privada para el cliente. Una vez acabado el proceso, nos informa de que el proceso de creación de nuestro cliente1.ovpn ha finalizado con éxito.

Nos indica también la ubicación de dicho fichero cliente1.ovpn, que suele ser en el directorio /home/pi/ovpns.

El proceso de creación del cliente que acabamos de describir se muestra en la siguiente imagen.

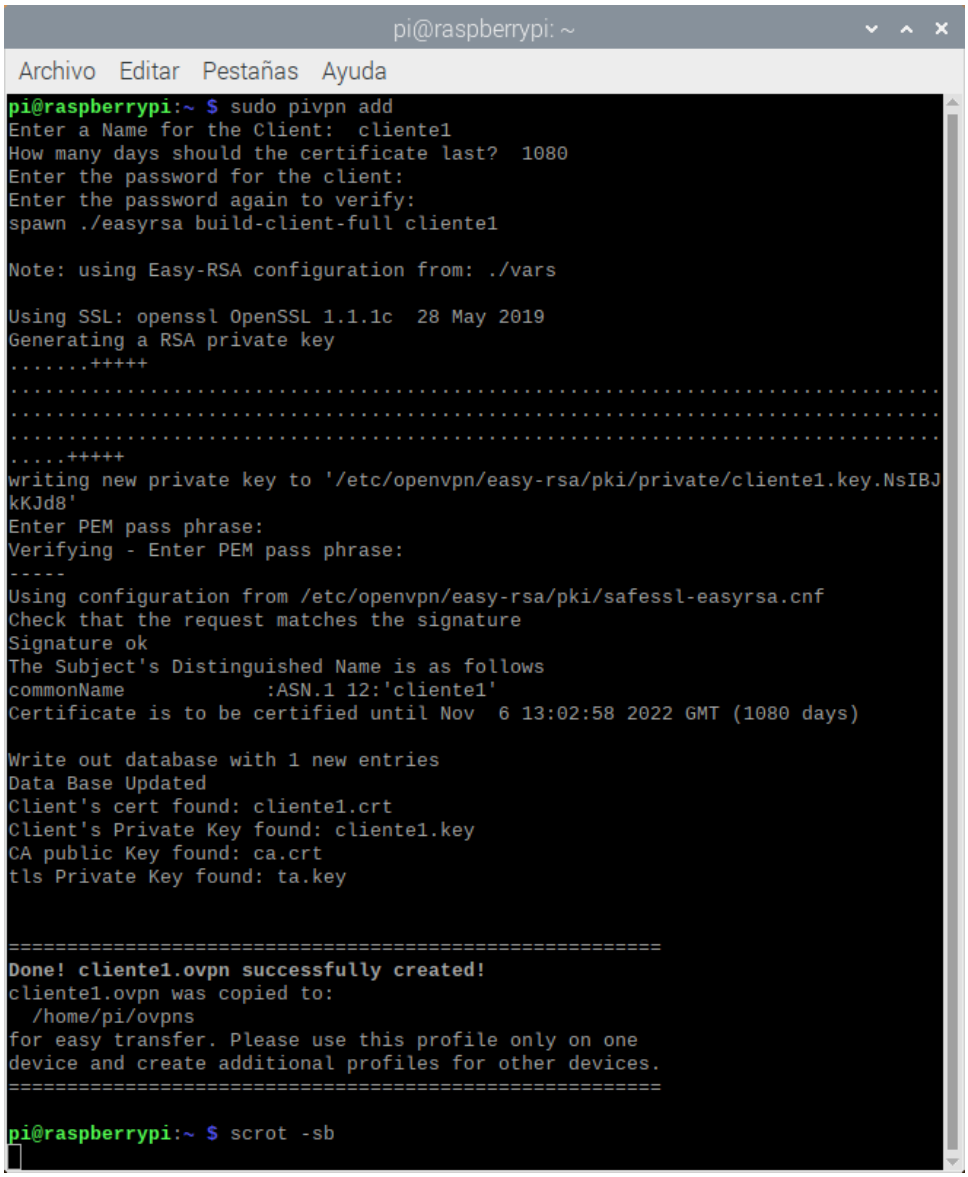

Figura 32. Creación de cliente VPN.

Una vez instalado el software de OpenVPN en nuestro cliente, este archivo es el que tenemos que importar en dicho cliente para poder establecer la conexión con el servidor VPN. Nos pedirá también la contraseña que hemos creado anteriormente.

Ya tendríamos nuestro servidor VPN completamente configurado, con el perfil para el cliente creado. Por lo tanto, ahora tan sólo queda comprobar que el trabajo realizado hasta ahora está correctamente ejecutado. Para llevar a cabo esta comprobación, procederemos a intentar conectarnos al servidor VPN.

4.3 Conexión al servidor VPN desde un terminal Android

El objetivo es conectarnos a nuestro servidor VPN desde un terminal Android.

Para llevar a cabo esta conexión, disponemos de dos buenas opciones en cuanto al software a instalar en nuestro terminal, como se comentó al principio.

Estas dos aplicaciones son “OpenVPN Connect” y “OpenVPN para Android”.

Las dos imágenes que se muestran a continuación corresponden a las dos aplicaciones mencionadas.

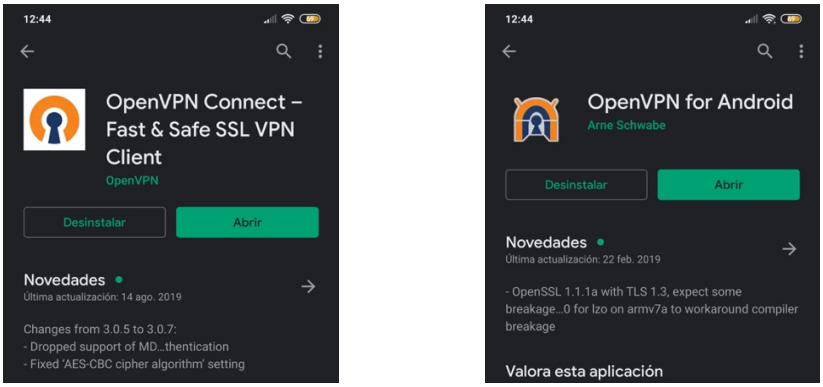

Figura 33. Aplicaciones Android para conexión a servidor VPN.

Para continuar con el proyecto, me he decantado por la primera opción, por lo que las siguientes imágenes y explicaciones corresponden a la aplicación "OpenVPN Connect".

El primer paso es instalar la aplicación, y una vez instalada, la abrimos.

Nos encontramos con la siguiente pantalla:

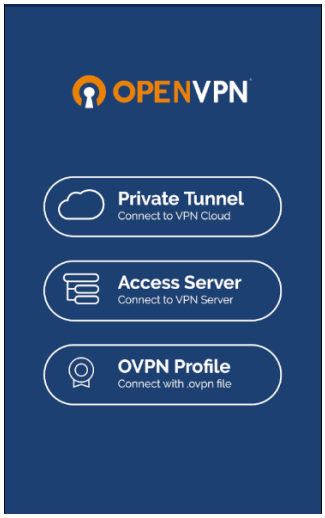

Figura 34. Ventana de inicio Open VPN Connect.

Como lo que queremos es importar el perfil clienteprueba.ovpn que hemos creado anteriormente, hacemos click en *"OVPN Profile"*, y nos aparece la siguiente imagen:

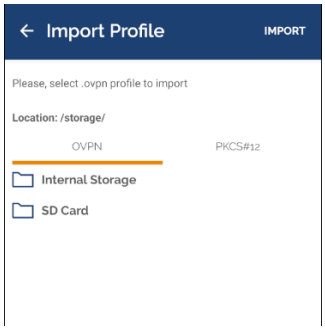

Figura 35. Importar perfil VPN en Open VPN Connect (1).

Aquí debemos seleccionar el fichero clienteprueba.ovpn para importar el perfil. Este fichero, hemos debido copiarlo previamente en el terminal Android, bien en la tarjeta SD, o en el propio terminal. Por lo tanto, lo que debemos hacer es buscarlo en la ubicación que se encuentre.

En mi caso, se encuentra en la carpeta *Download* dentro del almacenamiento del teléfono, por lo que iré a esa carpeta.

En la siguiente imagen, se muestra el perfil .ovpn en la ubicación descrita.

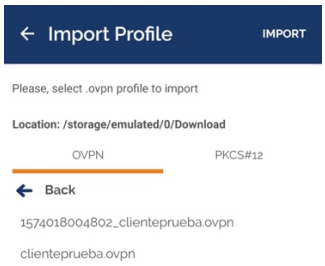

Figura 36. Importar perfil VPN en Open VPN Connect (2).

Una vez aquí, solo debemos hacer click encima del perfil clienteprueba.ovpn para importarlo.

Inmediatamente nos pedirá la contraseña que introducimos en el servidor VPN a la hora de crear el perfil. La introducimos y comenzará el proceso de conexión.

Si todo ha ido bien, debemos ver algo parecido a la siguiente imagen.

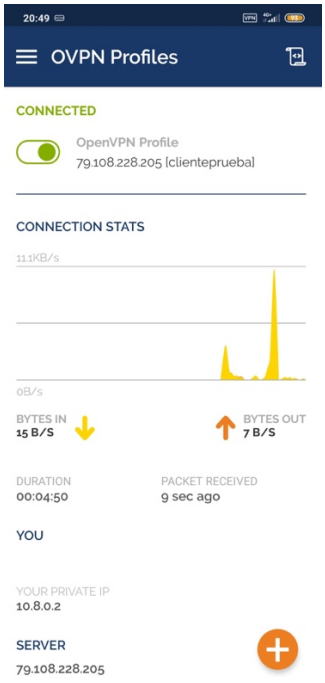

Figura 37. Conexión a servidor VPN en Open VPN Connect.

4.4 Instalación y configuración de Motion

En este punto, ya tenemos nuestro servidor VPN funcionando correctamente, así como la configuración de los clientes desde los que nos conectaremos a dicho servidor.

Siguiendo con el hilo del proyecto, ahora nos falta que cuando las cámaras conectadas a nuestra Raspberry PI detecten movimiento, se genere algún tipo de aviso, ya sea fotos, alerta, correo, etc.

La cámara con la que se harán las comprobaciones sobre esta parte del proyecto es el módulo de cámara de Raspberry, una cámara capaz de tomar fotos y vídeos Full HD 1080p, y puede controlarse mediante programación.

Posteriormente se ampliará el proyecto con alguna cámara IP conectada por Wi-Fi de manera inalámbrica a la Raspberry.

Para llevar a cabo esta parte, haremos uso del software *MOTION*.

4.4.1 Introducción a Motion

Con el fin de dar una breve explicación de en qué consiste el software *MOTION*, incluiré algunas de las características más significantes que nos proporciona la Wikipedia:

- Motion es un detector de movimiento por software.
- Está desarrollado para el sistema operativo Linux.
- Monitorea señales de vídeo de muchos tipos de cámaras.
- Permite detectar si una parte de la imagen ha cambiado.
- Permite guardar imágenes y el vídeo cuando detecta movimiento.
- Herramienta basada en la línea de comandos.
- Tiene como salida .jpeg (imágenes), .mpeg (vídeo).

Teniendo en cuenta todas las facilidades que nos ofrece este software, sería incomprensible no hacer uso de él, por lo que procedemos con el proceso de instalación y configuración.

Como siempre, antes de realizar cualquier instalación en nuestra Raspberry PI, procedemos a actualizarla por si hubiese alguna versión más reciente a la nuestra.

Tan solo son dos comandos fáciles de recordar que nunca está de más ejecutar previamente a cualquier instalación.

```
sudo apt-get update
sudo apt-get ugrade
```

Sobre estos comandos ya he comentado al principio cuál es la función de cada uno, por lo que no la repetiremos aquí.

4.4.2 Instalación de Motion

Una vez actualizada nuestra Raspberry PI, procedemos con la instalación del software *MOTION*. El comando con el que se instalará es el siguiente:

```
sudo apt-get install motion
```

La ubicación donde se instalará motion, es en el directorio /etc/motion.

Por seguridad, conviene desinstalar algunas librerías que podrían causar conflicto con motion, aunque este paso no es obligatorio. Para realizarlo, se ejecuta el siguiente comando:

```
sudo apt-get remove libavcodec-extra-56 libavformat56 libavresample2 libavutil54
```

Algo similar ocurre con el siguiente paso, que consiste en la instalación de algunas librerías empleadas por motion, que probablemente para el uso que vamos a hacer de él no afecte, pero se pueden instalar por si acaso. El comando es el siguiente:

```
sudo apt-get install libmariadbclient18 libpq5 libavcodec57 libavformat57 libavutil55 libswscale4
```

Una vez tenemos el software instalado, antes de arrancarlo y configurarlo, conviene comprobar si las cámaras que hemos conectado han sido detectadas correctamente.

Posteriormente, cuando queramos añadir una cámara USB a nuestro proyecto, ejecutaremos el comando siguiente:

```
lsusb
```

Este comando nos mostrará información acerca de todos los dispositivos USB conectados a nuestra Raspberry PI, ya sean cámaras, o el propio teclado y ratón (en caso de que los tuviésemos conectados). En la siguiente imagen muestro una captura de pantalla del terminal tras la ejecución del comando anterior:

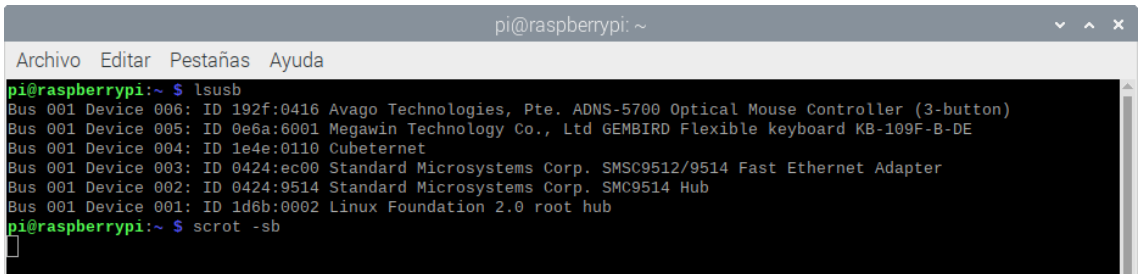

Figura 38. Dispositivos USB conectados a la Raspberry PI.

El primer dispositivo USB encontrado se trata del ratón (Mouse Controller), el segundo dispositivo se trata del teclado (Gembird Flexible Keyboard), y el tercer dispositivo es la cámara USB conectada.

Para la comprobación de esta parte, se hará uso del módulo de la cámara de Raspberry, no usaré de momento la cámara USB para esta parte como ya he comentado antes. Este módulo no va conectado mediante USB, sino en el puerto situado entre los puertos Ethernet y HDMI.

En la imagen que muestro a continuación, puede apreciarse dicha conexión del módulo de cámara con la Raspberry PI.

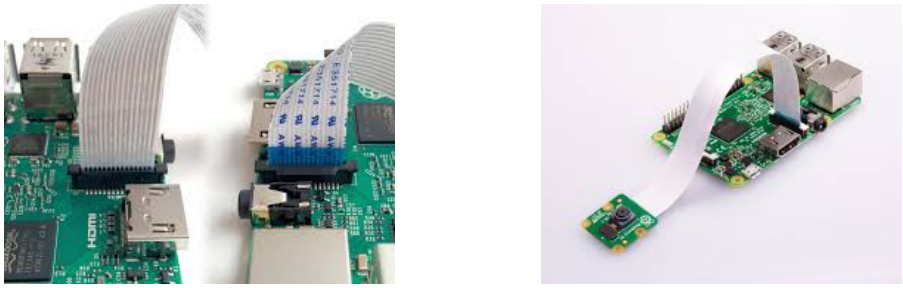

Figura 39. Módulo de cámara Raspberry PI.

Esta cámara suele estar deshabilitada por defecto, por lo que la habilitamos manualmente de una manera muy sencilla, tan solo debemos ejecutar el comando:

sudo raspi-config

Ahora seleccionamos *"Enable camera"* y presionamos "*enter"*. Por último, vamos a "*Finish*" y reiniciamos. Con este paso tan sencillo, ya tendríamos nuestro módulo de cámara habilitado.

Hay otra forma de habilitar la cámara, y es desde el menú de inicio de la Raspberry. Vamos a la opción de *"Preferencias"* y a continuación pinchamos en *"Configuración de Raspberry PI"*. Se nos abre una ventana como la que muestro en la siguiente imagen, vamos al apartado de *"Interfaces"* y tan solo hay que pinchar en *"Activo"*. Nos pedirá reiniciar la Raspberry para que se modifique la configuración. Por tanto, reiniciamos y continuamos con el proceso.

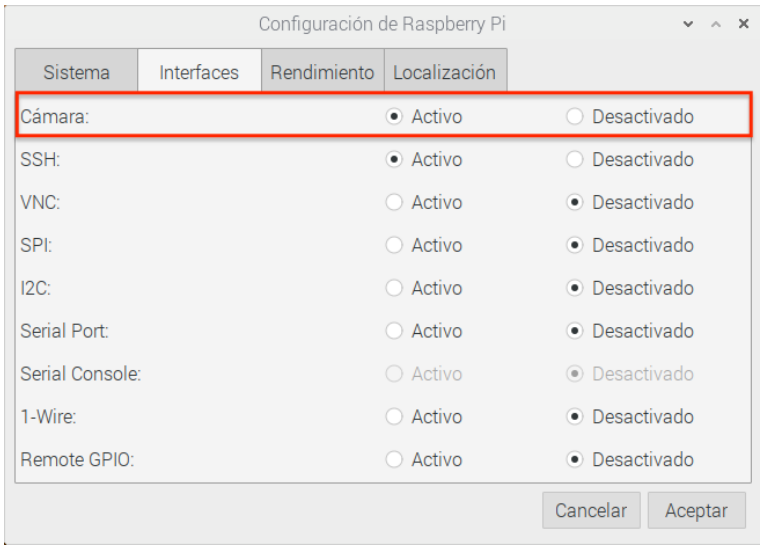

Figura 40. Habilitar la cámara en Raspberry PI.

4.4.3 Configurar los archivos de Motion

Ahora entramos en la parte de configurar los archivos de MOTION. El principal es el que se encuentra en el directorio "/etc/motion/motion.conf", y es el que servirá para llamar a otros archivos de configuración en el caso de que dispongamos de varias cámaras (posteriormente cuando se añada la cámara USB tendremos dos cámaras, el módulo de cámara de la Raspberry y la propia cámara USB). Por lo tanto, existe la posibilidad de crear un archivo de configuración específico con unas configuraciones diferentes para cada cámara.

En la siguiente imagen podemos ver el archivo principal *motion.conf* recuadrado en color rojo junto con los distintos archivos para cada una de las cámaras que se añadan (*camera1-dist.conf* y *camera2-dist.conf* recuadrados en verde), todos ellos ubicados en el directorio "/etc/motion" anteriormente mencionado.

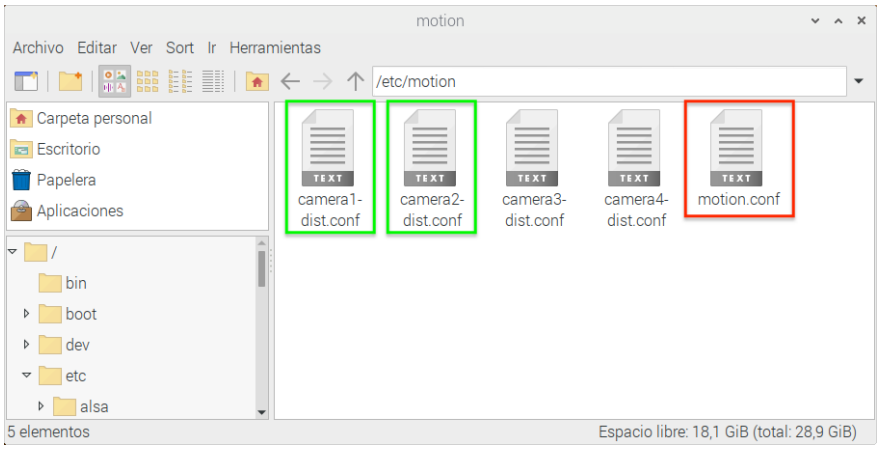

Figura 41. Archivos de Motion.

En este archivo (*motion.conf*) es donde configuramos, entre otros muchos parámetros, la rotación que queremos aplicar a las imágenes capturadas (rotate), la anchura y altura de las imágenes (width / height), las imágenes por segundo que capturará la cámara (framerate), el nivel de brillo (brightness), contraste (contrast), saturación (saturation), calidad de las imágenes capturadas (quality, valor de 0 a 100), el tipo de imagen (jpeg, ppm, webp), tasa de muestreo del vídeo creado, el puerto donde podremos ver el vídeo en streaming y muchos parámetros más.

Comentaré a continuación las configuraciones en los parámetros más significativos que he llevado al cabo, con la explicación correspondiente. Hay muchos parámetros más como ya he comentado, pero éstos son los que más afectan al proyecto:

- daemon on : al cambiar el estado esta opción a *'on'*, hacemos que motion se ejecute nada más encender nuestro servidor VPN (Raspberry PI).
- threshold 4000 : aquí indicamos el número de píxeles que deben cambiar en la imagen para ser considerado como movimiento. El valor ha sido elegido tras realizar numerosas pruebas y ver cuál es el valor que más se ajusta a la presencia de una persona en la imagen.
- event_gap 20 : este valor indica los segundos que deben de pasar después de haberse producido movimiento para que se considere un nueve evento.
- output_pictures on : con este valor en *'on'*, indicamos que queremos que se capturen imágenes del movimiento.
- picture_type jpeg : aquí indicamos el formato de las imágenes de salida.
- ffmpeg_output_movies on : ponemos este valor en *'on'* para codificar y capturar el movimiento en forma de vídeo.
- ffmpeg_bps 400000 : indicamos el bitrate (tasa de bits o datos que son procesados por unidad de tiempo) que se usará en el codificador de vídeo, en este caso 400 kbits por segundo.
- ffmpeg_video_codec mkv : aquí indicamos de qué tipo va a ser el contenedor del vídeo de salida, que en este caso será mkv.
- target_dir /home/pi/movimiento : aquí introducimos la ruta donde queremos que se guarden las imágenes capturadas dentro de nuestro servidor.
- stream_port 8081 : este valor indica el puerto al que conectarnos para ver el vídeo que está captando nuestra cámara en directo.
- stream_quality 100 : valor entre 0 y 100 que indicará la calidad del vídeo en directo, por lo que al elegir 100, elegimos máxima calidad.
- stream_localhost off : con el valor en *'off'*, podemos acceder al vídeo en directo desde cualquier máquina, no solo desde la que se ejecuta motion.

- webcontrol_port 8080 : este valor indica el puerto TCP/IP del que el servidor http escucha.
- on_event_start /usr/bin/python /home/pi/newmailingimage2.py : activamos la ejecución del script de python indicado siempre que se produzca un nuevo evento (on event start). Este script es el encargado de que se envíe la alerta por email al usuario con el fin de avisarle de que se ha detectado movimiento.

Hay otro archivo a tener en cuenta a parte del anterior, se trata del archivo 'motion' que se encuentra en la ubicación '/etc/default'. Aquí tan solo tenemos un parámetro, que es el que debemos modificar y cambiar su estado a *'yes'* con el fin de configurar la ejecución como servicio. El parámetro es el siguiente:

- start_motion_daemon=yes

Algunos comandos para el control del servicio motion, se muestran a continuación:

- sudo service motion reload
- sudo service motion restart
- sudo service motion start
- sudo service motion stop
- sudo service motion status

Con la opción *reload*, recargamos el servicio en caso de que queramos aplicar algún cambio en los archivos de configuración.

Con la opción *restart*, reiniciamos el servicio en caso de que se nos quede colgado o queramos reiniciar.

Con la opción *start*, iniciamos el servicio.

Con la opción *stop*, detenemos el servicio.

Con la opción *status*, detectamos errores en el servicio y podemos visualizar el estado del servicio en tiempo real.

4.4.4 Añadir cámara USB al proyecto

Una vez hemos asegurado el perfecto funcionamiento de nuestro detector de movimiento con la cámara de la Raspberry PI, podemos añadir la cámara USB a nuestro proyecto. La cámara que se usará es la cámara OMEGA C12SB, una sencilla cámara USB que muestro en la siguiente imagen:

Figura 42. Cámara USB.

Como he comentado antes, al tener más de una cámara, debemos modificar la arquitectura del proyecto, ya que no nos bastará sólo con el archivo *motion.conf*, sino que deberemos tener un

archivo más para cada una de las cámaras, por lo que, al tener dos cámaras, necesitaremos *camera1-dist.conf* y *camera2-dist.conf*.

La configuración más importante ya la hemos hecho en el archivo principal (*motion.conf*), por lo que ahora tan solo debemos descomentar de este archivo las últimas líneas donde se tiene en cuenta los dos archivos que necesitaremos a partir de ahora (*camera1-dist.conf* y *camera2-dist.conf*). Esto es tan sencillo como eliminar el punto y coma que hay delante de estas dos líneas recuadradas en color rojo:

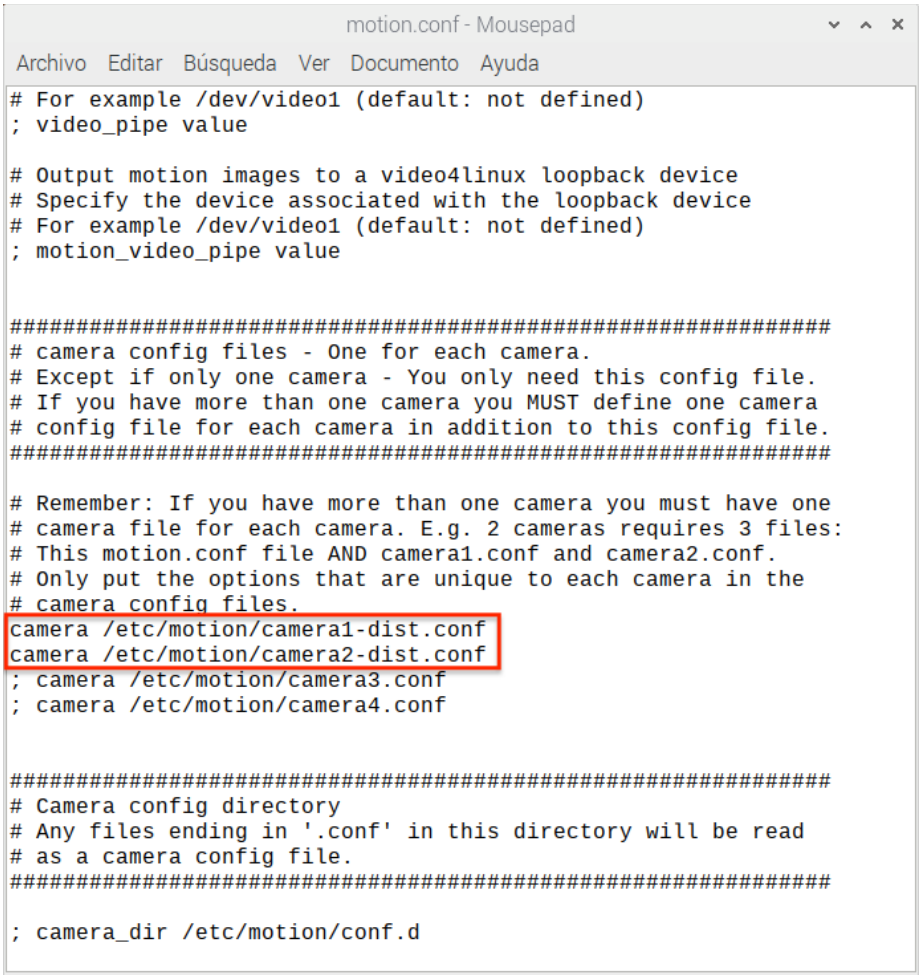

Figura 43. Archivo motion.conf.

Con eso tendríamos listo el archivo, ya que acabamos de habilitar los archivos para las dos cámaras que emplearemos.

Ahora debemos ir a esos archivos y en ellos tan sólo debemos asociar la cámara física que se corresponde al archivo, definir el puerto para visualizar el vídeo en streaming y el directorio donde se guardarán las imágenes capturadas cuando se detecte el movimiento.

- videodevice /dev/video0 : dispositivo de vídeo (cámara física) asociado al archivo de esta cámara.
- text_left CAMERA 1 : texto que aparecerá a la izquierda de la imagen capturada y el vídeo en directo visualizado.
- target_dir /home/pi/movimiento : directorio donde queremos que se guarden las imágenes capturadas dentro de nuestro servidor.
- stream_port 8081 : puerto al que conectarnos para ver el vídeo que está captando nuestra cámara en directo.

Con estos cuatro parámetros tendríamos correctamente configurado nuestro archivo de cámara.

De igual manera hay que configurar el archivo para la segunda cámara, eligiendo un puerto distinto y cambiando el dispositivo de vídeo ya que se trata de una cámara distinta, claramente.

4.4.5 Aviso por correo electrónico cuando se produzca movimiento

Para terminar con esta parte nos quedaría elaborar el script de Python que debe ejecutarse para enviar un correo electrónico adjuntando las imágenes tomadas cuando se detecte movimiento.

Cuando configuramos el archivo motion.conf ya indicamos en un parámetro que debe ejecutarse este script on_event_start, es decir, cuando se detecte el movimiento. Lo recuerdo en la siguiente línea:

- on_event_start /usr/bin/python /home/pi/newmailingimage2.py

El script vemos que se encuentra en la ubicación "/usr/bin/python /home/pi", y el nombre de dicho script es *newmailingimage2.py*.

Su función no es otra que cuando sea ejecutado, envíe un correo electrónico a la cuenta que haya sido elegida adjuntando un número determinado de imágenes, por ejemplo, las 5 últimas imágenes que hayan sido tomadas.

Capítulo 5. Desarrollo de la aplicación para terminal móvil

La segunda parte del proyecto consiste en el desarrollo de una aplicación para terminales con sistema operativo Android con el fin de plasmar sobre la pantalla de un terminal móvil el trabajo realizado hasta ahora. Dicho de otra forma, una aplicación donde ver lo que nuestras cámaras están retransmitiendo en directo, así como la configuración y descripción de estas.

5.1 Descripción general de la aplicación

A continuación, realizaré una breve descripción de lo que va a ser la aplicación desarrollada.

La aplicación constará de una primera actividad, que como es de esperar será una actividad de *login*, es decir, donde nos registraremos con usuario y contraseña para el acceso a nuestra aplicación.

Una vez dentro, tendremos acceso a las distintas cámaras que tengamos registradas, con posibilidad de ver el contenido de cada una si así lo decidimos.

Desde esta segunda actividad, tendremos la opción de modificar, añadir o borrar alguna cámara. También tendremos la posibilidad de ir directamente a la aplicación de Gmail con el fin de comprobar si hemos recibido algún aviso de movimiento y ver las imágenes de ese evento. Por último, también podremos dirigirnos al Teléfono para avisar a la Policía.

Si hacemos click en la opción de modificar cámaras, iremos a una tercera actividad para realizar las opciones anteriormente descritas (añadir una nueva cámara, actualizar los parámetros de una cámara ya existente o borrar una cámara). Esta actividad quizá sea la más compleja de implementar, ya que será la que trabajará con la base de datos donde se almacena la información de cada cámara añadida.

5.2 Entornos de desarrollo posibles

Tras este breve resumen de las partes de las que va a constar la aplicación, toca ponerse manos a la obra y comenzar con el desarrollo de esta.

El primer paso es decidir con qué entorno de desarrollo vamos a trabajar. Aquí se nos abre un gran abanico de posibilidades.

A continuación, muestro algunos de los entornos de desarrollo más destacados, con una breve descripción de cada uno de ellos:

- Basic 4 Android: una de las grandes rivales de Android Studio que utiliza el lenguaje VisualBasic para su programación. Se puede decir que es un entorno más gráfico y menos abstracto.
- Mono para Android: nos ofrece la posibilidad de desarrollar apps en Android desde nuestro entorno Microsoft en otros lenguajes a parte de Java, como C#, .NET o VisualBasic.
- App Inventor: entorno visual donde podemos programar sin escribir ni una sola línea de código. Es un entorno simple y rápido, aunque esto pueda ser también un punto débil.
- LiveCode: plataforma donde podemos desarrollar apps para Android, pero también para iOS, Windows y Linux. Este era uno de los entornos con mayor posibilidad para la realización de la App, pero finalmente se decidió emplear otro.
- HTML5: es el entorno ideal para realizar apps multiplataforma. También ofrece la posibilidad de realizar actualizaciones de forma instantánea. No hace falta comentar que el lenguaje de programación empleado es HTML.
- Appcelerator Titanium: esta es una de las plataformas más activas para el desarrollo de aplicaciones móviles Android del momento. Como ejemplo, he de comentar que eBay o PayPal implementan sus apps desde este entorno de desarrollo.

- Otros de los entornos de desarrollo que podríamos usar son InDesign CS6, Ruboto, Rhomobile Rodes, Adobe Air, …

5.3 Introducción al entorno de desarrollo Android Studio

Finalmente he optado por emplear Android Studio, que es un IDE (Integrated Development Environment o Entorno de Desarrollo Integrado) oficial para la implementación de aplicaciones para Android basado en IntelliJ IDEA. Consta de un potente editor de códigos, un emulador bastante rápido, así como de las múltiples herramientas para desarrolladores de IntelliJ.

Figura 44. Logo Android Studio.

Una de las funciones más relevantes que tiene es la integración que posee con GitHub (plataforma que permite alojar proyectos utilizando el sistema de control de versiones Git), donde podemos ir alojando nuestro proyecto según las distintas versiones que vayamos realizando, para no perder el trabajo realizado en caso de realizar alguna modificación que nos estropee la App ofreciéndonos la posibilidad de acceder a la versión anterior.

También nos permite importar código de ejemplo de una manera muy sencilla.

Otra de las funciones interesantes es que su sistema de compilación es flexible y se basa en Gradle, cosa que es un gran alivio, ya que, por ejemplo, en Eclipse había que empaquetar el proyecto, cosa que causaba un gran dolor de cabeza.

Mediante Android Studio podemos desarrollar aplicaciones para todos los dispositivos Android, ya que podemos elegir la versión sobre la que implementar después.

A la hora de crear un proyecto desde cero, nos da la posibilidad de comenzar con distintas plantillas de proyectos comunes de Android, como por ejemplo Basic Activity, Empty Activity, Bottom Navigation Activity, Fullscreen Activity, Google Maps Activity, etc.

Nos ofrece la posibilidad también de tener un dispositivo Android virtual (emulador) donde ir simulando nuestra aplicación y ver el progreso en lo que sería un terminal real.

En la imagen que se muestra a continuación hay un ejemplo del emulador que acabo de comentar. En ese terminal es donde iremos probando la aplicación desarrollada en Android Studio cada vez que simulemos nuestro proyecto.

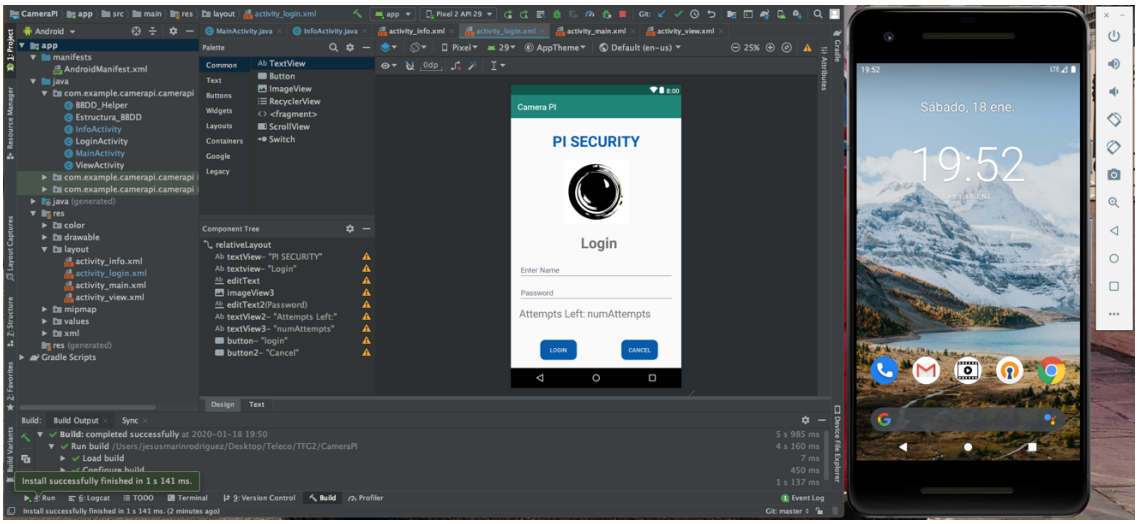

Figura 45. Interfaz Android Studio.

En cuanto a la estructura que tienen los proyectos realizados en Android Studio, tenemos distintos tipos de módulos como es el caso de los módulos de apps para Android, módulos de bilbioteca y módulos de Google App Engine.

Dentro de nuestro proyecto, tenemos tres carpetas destacadas, que es donde se encuentran los ficheros que componen el proyecto:

- manifest: donde se encuentra el archivo AndroidManifest.xml, archivo que debe estar presente en todos los proyectos. Su función es describir la información esencial de la aplicación. Contiene el nombre del paquete de la app, los componentes de la app, los permisos que necesita la app para acceder a las partes protegidas del sistema, funciones de software y hardware, …
- java: aquí se encuentran todos los archivos de código fuente.
- res: contiene los diseños XML, strings, imágenes, …, es decir, los recursos sin código.

Un ejemplo de estos archivos y módulos se muestra en la siguiente imagen, que es un fragmento recortado de la imagen anterior.

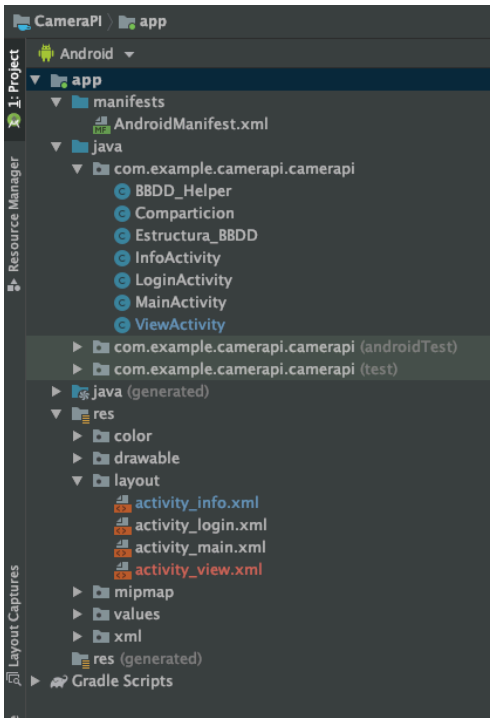

Figura 46. Ficheros App.

Podemos ver dentro de la carpeta *manifest* el archivo *AndroidManifest.xml* comentado.

Dentro de la carpeta *java* se ubican todos los archivos de código fuente programados por el desarrollador. Estos se encargan de la funcionalidad de la aplicación.

Por último, en la carpeta *res* vemos las subcarpetas *color, drawable, layout, mipmap, values,* y *xml*, que como he comentado antes, son los recursos sin código. Aquí se diseña el aspecto físico de las distintas pantallas y actividades de nuestra aplicación. También es donde se deben incluir las imágenes que añadimos en las actividades, o la imagen que elegiremos como logo de nuestra app.

Tras la explicación acerca del entorno de desarrollo Android Studio y ver algunas de sus principales y más destacadas características, así como tras hacer un repaso de cómo se organizarán los archivos y ficheros que compondrán nuestra aplicación, comenzaré a mostrar la estructura de la aplicación implementada con sus distintas actividades.

5.4 Actividades de la aplicación

5.4.1 LOGIN ACTIVITY

Tal y como comenté al principio, la primera actividad será una actividad de *login*. Muestro una imagen del aspecto que tendría la actividad en un terminal móvil Android (archivo *activity_login.xml*) en la imagen siguiente y posteriormente realizo la descripción y explicación de la actividad.

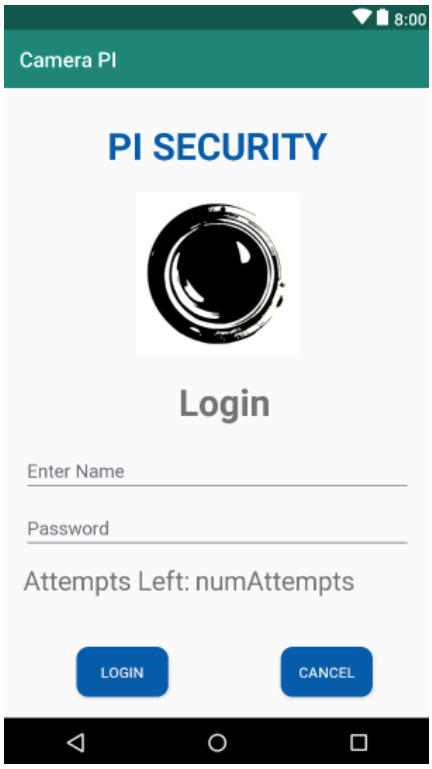

Figura 47. Layout Login Activity.

Como se puede apreciar, se trata de una actividad simple ya que su función no es otra que la de la identificación del usuario.

Consta de dos *textView*, el que contiene el texto de "PI SECURITY" y el texto de "Login". Este texto no puede ser modificado por el usuario ni nada por el estilo, tan solo se trata de texto informativo.

Entre estos dos elementos tenemos un *imageView*, donde podemos visualizar lo que sería el objetivo de una cámara, ya que el proyecto va de cámaras y seguridad. La inclusión de este *imageView* ha sido simplemente por un motivo estético.

Debajo del *textView* de "Login" nos encontramos con dos *editText*, que son cuadros de entrada de texto por parte del usuario. Lógicamente ahí es donde el usuario debe introducir tanto su nombre de usuario como la contraseña para, posteriormente comprobar si estos valores corresponden a los de la persona que posee el sistema de seguridad.

No son exactamente iguales ambos *editText*, ya que el que a la contraseña se refiere, tiene una propiedad con el fin de que al introducir los caracteres que conforman dicha contraseña, estos caracteres aparecen de la forma "*****", como medida de seguridad que debe de tomarse en el caso de las contraseñas. Esta propiedad se llama *inputType* y le he dado el valor *textPassword*.

Otra de las propiedades de los *editText* es la que tiene la función de mostrar texto en el *editText* cuando aún el usuario no ha introducido nada. En este caso, se trata del texto "Enter Name" y "Password", que es una manera de indicarle al usuario la utilidad de cada cuadro de entrada de texto. Esta propiedad se llama *hint*, y es tan sencilla de usar como poner en la propiedad el texto que queremos que se muestre cuando aún no se ha escrito nada en el cuadro de entrada texto.

Tras estos dos *editText*, tenemos dos nuevos *textView*. El primero de ellos nos muestra "Attempts Left", es decir, intentos para probar usuario y contraseña. El segundo, nos indica el número de

intentos que quedan. Este segundo *textView* va cambiando de valor a medida que se introducen valores erróneos de usuario y/o contraseña.

Al ver estos dos cuadros de texto, podemos deducir que esta actividad tiene un mecanismo de seguridad que no es otro que, si se supera el número de intentos, no nos deja probar más. Debemos abandonar la aplicación. El número de intentos ha sido fijado por el programador en 5.

Cada vez que alguna de las credenciales sea incorrecta, el contador inicialmente fijado en 5 (el número de intentos posible) irá decrementandose en una unidad. Si el contador llega a 0 quiere decir que hemos consumido el total de intentos, por lo que se deshabilitará el botón "LOGIN" de la parte inferior, y esto hará que sea imposible entrar a la aplicación, y por lo tanto, a las cámaras.

Ambas credenciales, nombre de usuario y contraseña, también están definidas en el código por el programador.

En la parte inferior de nuestra primera actividad tenemos dos *buttons*, uno con el texto "LOGIN" y otro con el texto "CANCEL". Estos dos botones son los encargados de cambiar a otra actividad, bien a la actividad siguiente si introducimos correctamente las credenciales y pulsamos el botón "LOGIN", o bien a salir de nuestra aplicación en el caso de pulsar el botón "CANCEL".

Hasta ahora tan solo hemos comentado el aspecto del archivo *activity_login.xml*, pero cada archivo xml lleva asociado un archivo de código fuente. El archivo de código fuente asociado a este archivo xml se llama *LoginActivity.java*. Aquí es donde se programa todo el argumento de la actividad (clases, variables, lógica, …).

Al final de la memoria, en el apartado de "Anexos" incluiré el código correspondiente a cada una de las múltiples actividades que conforman la aplicación con el fin de evitar llenar estas partes de código y que sea todo más intuitivo y sencillo para el lector.

5.4.2 INFO ACTIVITY

Como se ha indicado en el *Intent* de la actividad anterior, la siguiente actividad en el caso en que las credenciales estén correctamente introducidas, se trata de *InfoActivity.java*.

Siguiendo con el esquema, primero veremos la imagen del aspecto que tendría la actividad en un terminal móvil Android (archivo *activity_info.xml*) en la imagen siguiente y posteriormente realizo la descripción y explicación de la actividad.

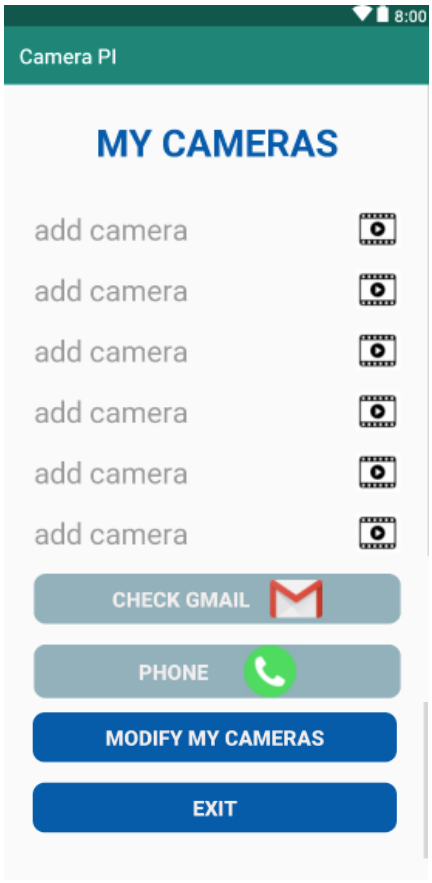

Figura 48. Layout Info Activity.

En este caso tenemos una actividad algo más compleja que la anterior. Mediante esta actividad, tendremos acceso a la lista de cámaras añadidas, así como la posibilidad de ver el vídeo que cada una de ellas está captando en directo. También podremos modificar, añadir o eliminar las cámaras de nuestro proyecto.

Una última acción que podemos llevar a cabo desde esta actividad es ir rápidamente a comprobar el correo electrónico para ver las imágenes capturadas por nuestras cámaras cuando se detecta movimiento. Para ello solo debemos pulsar sobre el botón *"Check gmail"*.

Repasando las partes de las que consta el archivo *activity_info.xml* de esta actividad, vemos que en primer lugar nos encontramos con un *textView* que nombra a nuestra actividad, "MY CAMERAS". Nos quiere decir que a continuación tenemos la lista de las cámaras que posee nuestro proyecto. En la imagen anterior, esta lista está vacía ya que aún no ha sido añadida ninguna cámara, pero cuando esto suceda, ahí aparecerá el nombre de las cámaras añadidas.

Seguidamente observamos seis *textView*, que serán los contenedores de las cámaras añadidas. No quiere decir que porque haya seis *textView* solamente estaremos limitados a añadir seis cámaras, sino que una vez se completen estas seis cámaras y sigamos añadiendo, se añadirán tantos *textView* como sea necesario.

Como he comentado antes, estos cuadros de texto poseen la característica de cuando no se ha escrito nada aún, mostrar el texto que el programador elija en la propiedad *hint*. En este caso el texto elegido es "add camera", indicando que aún se puede añadir una cámara en el cuadro de texto que contenga ese mensaje.

Otra propiedad nueva que en este caso tenemos en los *textView* usados en esta actividad es *drawableRight*, que como su nombre indica, se trata de la imagen añadida a la derecha del cuadro

de texto. Esta imagen, que coincide con el icono de la aplicación, es un indicativo de que al pulsar sobre ella accederemos al vídeo en directo de la cámara añadida en ese cuadro de texto.

La acción de acceder al vídeo cuando se pulse sobre el cuadro de texto se ejecuta porque se ha decidido así configurando otra propiedad de estos *textView*. Se trata de la propiedad *onClick*, cuyo nombre nos indica que algo va a suceder cuando se pulse sobre ella. Ese "algo" es nada mas y nada menos que el argumento de la propiedad, en este caso *viewVideo*, que se trata de una función programada en el archivo *InfoActivity.java*, que es el archivo de código fuente asociado a esta actividad del que hablaremos al acabar de comentar el aspecto visual (archivo *activity_info.xml*) de la actividad.

En la parte inferior, volvemos a tener tres *buttons*. El primero de ellos con el texto "CHECK GMAIL", ya que nos abrirá inmediatamente la aplicación de GMAIL de nuestro terminal Android para comprobar el correo. El siguiente botón contiene el texto "MODIFY MY CAMERAS", que nos llevará a otra actividad (*MainActivity.java* que tiene asociado su archivo xml *activity_main.xml*), actividad donde podremos añadir una nueva cámara, modificar y/o eliminar alguna cámara ya existente. Esto se llevará a cabo debido a que en la propiedad *onClick* de este *button*, se ha indicado el nombre del método a ejecutar (*modify*). El último botón sirve para abandonar la aplicación, como su nombre nos indica, "EXIT". También se ha indicado en la propiedad *onClick* de este tercer *button* que se ha de ejecutar el método *exitApp*, que más adelante será descrito.

Un detalle muy importante que no podemos olvidar comentar es que todos estos elementos que conforman el *layout* de la actividad (*textView*, *button*, …), están contenidos dentro de un *ScrollView*, otro elemento que nos permite desplazarnos verticalmente en este caso, ya que el *layout* de la actividad contiene más elementos de los que cogen en el largo de la pantalla del dispositivo.

Una vez añadimos el ScrollView al archivo .xml de nuestro proyecto, automáticamente aparece la típica barra que nos permite hacer "scroll", es decir, subir y bajar en el *layout* de la actividad. En un lenguaje menos técnico podríamos decir que el usuario puede desplazar la información con el dedo de forma vertical en la pantalla de su dispositivo.

Hasta aquí llega la descripción del archivo *activity_info.xml*, pero como en la actividad anterior, cada archivo xml lleva asociado un archivo de código fuente. El archivo de código fuente asociado a este archivo xml se llama *InfoActivity.java*. Aquí es donde se programa todo el argumento de la actividad, que en este caso no es muy complicado.

Antes de pasar a la siguiente actividad, mostraré un ejemplo de cómo quedaría esta actividad con algunas cámaras añadidas.

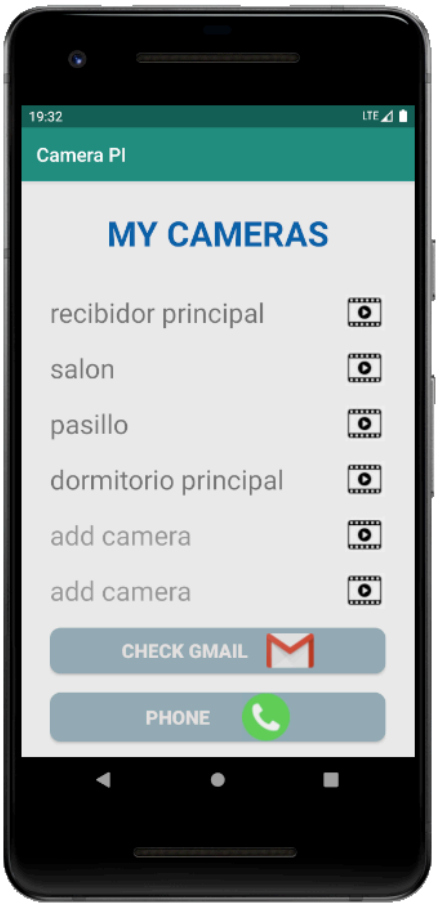

Figura 49. Layout Info Activity (2).

Vemos que hay cuatro cámaras añadidas (recibidor principal, salón, pasillo y dormitorio principal) y tenemos aún la posibilidad de añadir otras dos más, ya que en los dos últimos *textView* encontramos el texto "add camera".

Tras la ejecución de esta actividad recientemente descrita, depende del uso que hagamos de ella iremos a otra aplicación distinta (GMAIL o TELÉFONO) o a una tercera o cuarta actividad, es decir, se nos abren cuatro caminos tras esta actividad.

5.4.3 MAIN ACTIVITY

Como se ha indicado en el *Intent* del método *modify* anteriormente descrito, la siguiente actividad en el caso en que pulsemos sobre el botón "MODIFY MY CAMERAS", se trata de *MainActivity.java*, actividad que como bien hemos comentado varias veces, nos permite añadir y eliminar cámaras a nuestro sistema de seguridad.

Siguiendo con el esquema, primero veremos la imagen del aspecto que tendría la actividad en un terminal móvil Android (archivo *activity_info.xml*) en la imagen siguiente y posteriormente realizo la descripción y explicación de la actividad.

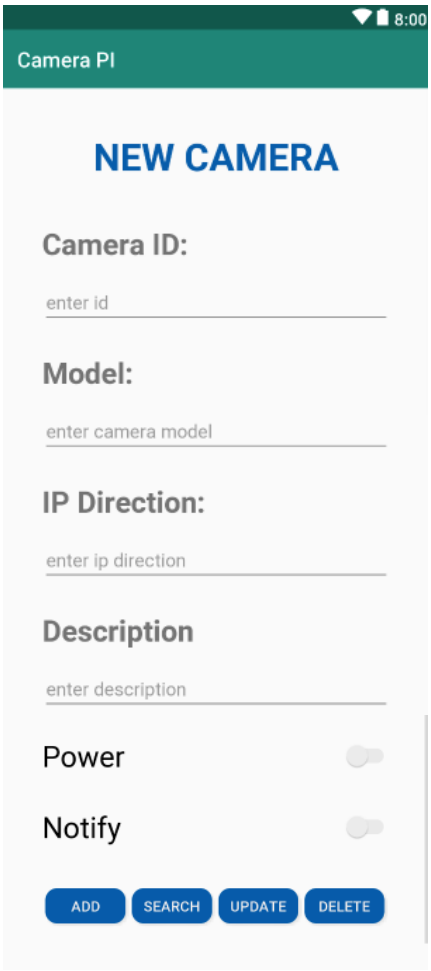

Figura 50. Layout Main Activity.

Mediante esta actividad podremos modificar, añadir o eliminar las cámaras de nuestro proyecto.

Repasando las partes de las que consta el archivo *activity_main.xml* de esta actividad, vemos que en primer lugar nos encontramos con un *textView* que nombra a nuestra actividad, "NEW CAMERA", ya que estamos en una actividad cuya función principal es llevar a cabo la acción de añadir una nueva cámara a nuestro proyecto.

Seguidamente tenemos otro *textView* acompañado de un *editText*. Esto se repite cuatro veces, ya que tenemos cuatro parámetros para rellenar con texto por parte del usuario. Estos parámetros son:

- Camera ID: se trata del número de cámara, es decir, el lugar que ocupará esa cámara añadida en la base de datos y en la lista de la actividad anterior.
  La primera de ellas llevará como Camera ID el número 1, la segunda un 2, … y así sucesivamente.
  Cuando decidamos buscar alguna cámara para ver sus parámetros o modificar alguno, basta con introducir en esta actividad el Camera ID y pulsar en el botón "SEARCH". Inmediatamente nos aparecerán los datos que tengamos rellenados de esa cámara, en caso de que exista la cámara con ese ID.
- Model: aquí el usuario deberá introducir el "modelo" o tipo de cámara del que se trata, como por ejemplo Raspberry Pi Camera, USB Camera, IP Camera, …
  Simplemente es información que se almacenará en la base de datos y que puede ayudar al usuario a identificar cada cámara más tarde.

- IP Direction: en este parámetro deberemos introducir la dirección IP que le hayamos asignado a cada cámara (en caso de que se trate de una cámara IP), la dirección IP de la Raspberry PI (en el caso de que se trate de la Raspberry PI Camera o alguna cámara conectada mediante USB a la Raspberry PI).
  Con este parámetro realizaremos la conexión en la última actividad que nos queda por comentar mediante una conexión web a esta dirección IP introducida.
- Description: con este último parámetro indicaremos el lugar donde tenemos ubicada nuestra cámara, o cualquier otro indicativo que nada más verlo sepamos identificar de qué cámara se trata. Por ejemplo, *cámara entrada, cámara salón, cámara pasillo, cámara porche, …*
  Este también será el parámetro que se colocará en la lista de cámaras de la actividad anterior cuando la cámara aquí editada sea añadida.

Tras el último *textView* (*Description*) acompañado de su *editText* (con su propiedad *hint* con el valor *enter description*), nos encontramos con dos *Switch*. A continuación, comento de qué se trata cada uno de ellos:

- Power: con este *switch* podremos encender y apagar la cámara editada. Lo lógico será que cada vez que añadamos una cámara, activemos este *switch*. Si por alguna razón alguna vez queramos tener una cámara apagada, tan solo debemos modificar el estado de este *switch*.
- Notify: con este *switch* activaremos el aviso por correo electrónico cada vez que la cámara detecte movimiento.

Por último, para terminar con la explicación del fichero *activity_main.xml* nos queda comentar los cuatro botones que observamos en la parte inferior de el *layout*.

Estos botones serán los encargados de ejecutar las acciones de añadir una nueva cámara, buscar los parámetros de una cámara ya añadida, actualizar algunos de los parámetros de una cámara existente y borrar la cámara seleccionada de la base de datos.

- ADD button: como su propio nombre indica, se trata del botón cuya función es añadir la cámara a la base de datos con las características configuradas. Al añadir la cámara, cuando volvamos al *layout* de la actividad *InfoActivity.java*, deberá aparecernos en la lista de "MY CAMERAS".
- SEARCH button: cuando queramos comprobar los detalles de una de las cámaras que tengamos en el proyecto, bien por recordar la configuración o por el hecho de querer modificar alguna propiedad, introducimos el "Camera ID" de dicha cámara y hacemos click en este botón.
  Seguidamente podremos ver la configuración de los parámetros de la cámara buscada.
- UPDATE button: como acabamos de comentar, para cambiar algún parámetro de alguna cámara, previamente debemos buscarla con el botón anterior. Una vez la hemos buscado, modificamos el parámetro que deseemos y pulsamos en el botón "UPDATE" para guardar la cámara con el cambio realizado. Si posteriormente volvemos a buscarla, veremos que ya aparece con el cambio realizado y actualizado.
- DELETE button: de igual forma que al actualizar algún parámetro de la cámara, si queremos eliminar una cámara de nuestra base de datos, y por tanto del proyecto, previamente debemos buscarla con el botón anterior "SEARCH". Una vez la hemos buscado, podemos pulsar sobre este botón "DELETE" para eliminarla. Si posteriormente volvemos a buscarla, veremos que ya no aparece, y nos encontraremos un mensaje que dirá "No se encontró el registro con clave: …" más el ID de la cámara que acabamos de eliminar.

De igual manera que pasaba en la actividad anterior, no podemos olvidar que todos los elementos que conforman este *layout* están dentro de otro *ScrollView*. No repetiré la explicación sobre este elemento porque ya quedó bien explicado en la anterior actividad.

Hasta aquí llega la descripción del archivo *activity_main.xml*, pero como ya hemos dicho en las actividades anteriores, cada archivo xml lleva asociado un archivo de código fuente. El archivo de código fuente asociado a este archivo xml se llama *MainActivity.java*. Aquí, como sabemos, es donde se programa todo el argumento de la actividad, que en este caso es algo más complicado, ya que esta actividad se basa en trabajar con una base de datos también desarrollada por el programador.

Una de las partes más importantes de este archivo, es la creación y manipulación de la Base de Datos empleada para la gestión de los parámetros de nuestras cámaras.

Continuaré con una explicación de cuál es el gestor de Bases de Datos que utiliza Android para almacenar la información, así como la descripción de la elaboración de la Base de Datos desarrollada para nuestra aplicación.

Figura 51. Logo SQLite.

Al hablar del gestor de Bases de Datos de Android, tenemos que hablar de SQLite, que es un sistema de gestión de Bases de Datos relacional.

La principal diferencia de este gestor respecto a otros es que SQLite está escrito en el lenguaje de programación C y es de código abierto.

También tenemos otros gestores importantes de código abierto, como es el caso de MySQL.

Pero al estar escrito en C tenemos varias ventajas, como por ejemplo tener la característica de ser multiplataforma.

Otra diferencia respecto a otros gestores de Bases de Datos es que una Base de Datos SQLite forma parte integral del programa. En otros entornos, cuando nosotros elaboramos una aplicación que conecta con una Base de Datos, esta Base de Datos forma parte de un proceso independiente de nuestro programa, y cada vez que queremos conectar con esa Base de Datos para obtener o modificar información, tenemos que generar procesos que consumen tiempo y recursos.

Con SQLite, como ya hemos dicho, la Base de Datos forma parte del propio programa, lo que provoca que cuando queramos conectar con ella, este proceso se haga de una forma rápida y eficiente.

Una diferencia más con respecto a otros gestores es que la Base de Datos se guarda como un único fichero en el host (en nuestro propio ordenador).

Todas estas características que acabamos de ver provocan una serie de ventajas e inconvenientes:

Ventajas:

- Ocupa muy poco espacio en disco y memoria, por lo que es ideal para aplicaciones móviles que se ejecutan en smartphones o tablets (cuanto menos espacio ocupe, mucho mejor ya que estos dispositivos disponen de menos memoria que, por ejemplo, un ordenador o un servidor).
- Muy rápido y eficiente.
- Multiplataforma.
- No requiere ni administración ni configuración.
- Es de dominio público (código abierto), por lo que no tiene costo.

Inconvenientes:

- No admite clausulas anidadas.
- No existen varios usuarios accediendo de forma simultánea a estas Bases de Datos SQLite, ya que forma parte de nuestra aplicación dentro de nuestro propio ordenador.
- No existe la posibilidad de establecer una clave foránea cuando se crea en modo consola.

Por todas estas características, especialmente porque es muy rápido y eficiente, y porque ocupa muy poco espacio en memoria, es ideal para aplicaciones Android.

Tras esta introducción acerca del gestor de Bases de Datos en Android SQLite, procedo a continuar con la descripción del proceso para elaborar la Base de Datos de nuestra aplicación, que estará formada por dos ficheros de código (*BBDD_Helper.java* y *Estructura_BBDD.java*).

Para la creación de la Base de Datos me ayudaré de la API de Android, una herramienta de gran ayuda para los programadores de Android Studio.

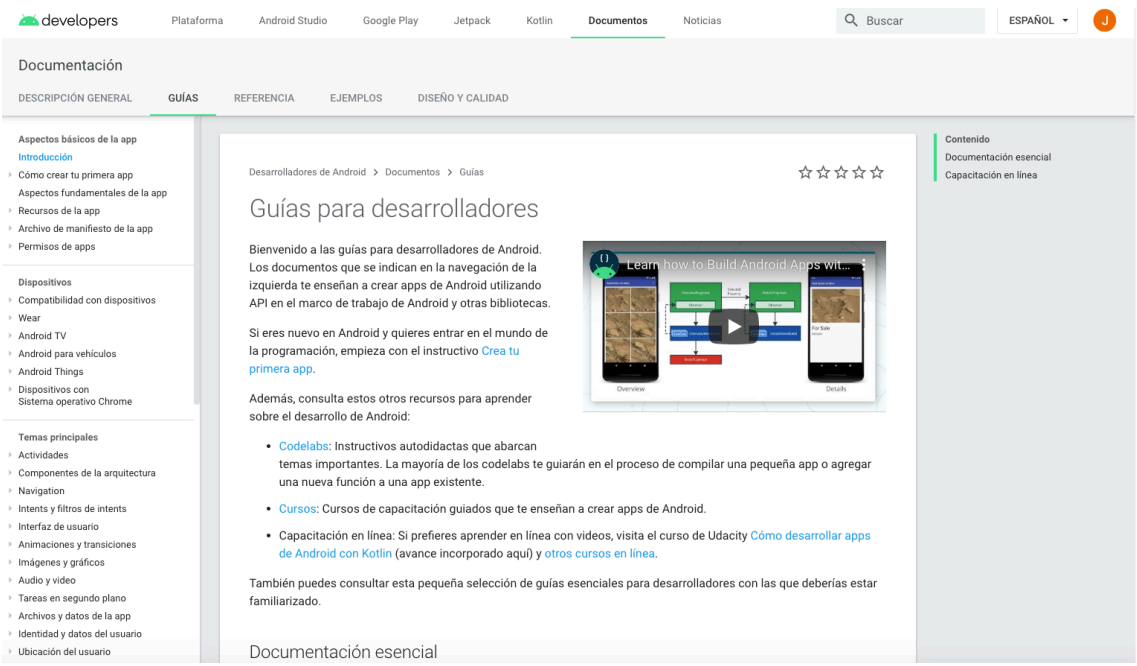

Figura 52. Android Developers Web.

En la imagen que se muestra sobre este párrafo, podemos ver la página principal de la API de Android. Si vamos al menú “Documentos”, que es un menú para desarrolladores, y seguidamente al submenú “Guías”, nos encontramos en una zona donde se explica paso a paso como programar diferentes cosas con Android Studio.

En la parte de la izquierda de la página web podemos observar otra especie de menú vertical donde encontraremos las distintas guías según las funciones que queramos programar.

Si indagamos un poco dentro del menú que acabo de comentar, daremos con una guía llamada "Cómo guardar datos con SQLite", que es justo lo que queremos hacer.

Esta guía nos indica que debemos crear dos clases, aunque hay muchas maneras de hacerlo y no necesariamente hay que crear dos clases, pero el diseño que nos sugiere así lo requiere con el fin de separar la lógica de la Base de Datos de la estructura.

En una de las clases vamos a establecer la estructura de nuestra Base de Datos, es decir, cuántas tablas va a tener y qué campos va a tener cada una de esas tablas. Esta clase se va a llamar *Estructura_BBDD.java*.

Una vez tenemos la clase con la estructura, debemos crear una segunda clase que se va a llamar *BBDD_Helper.java*. Esta clase debe heredar de una clase de la API de Android llamada "*SQLiteOpenHelper*", clase que cuenta con los métodos necesarios para poder realizar diferentes acciones en nuestra Base de Datos:

- onCreate (método que nos permite crear la Base de Datos)
- onUpgrade (método que nos permite actualizar la Base de Datos)
- onDowngrade

Estas dos clases son las que conforman nuestra Base de Datos.

5.4.4 VIEW ACTIVITY

Tras la ejecución de esta actividad *InfoActivity.java* que describimos algunas páginas atrás, comentamos que según el uso que hiciésemos de ella iríamos a una tercera actividad u otra, es decir, se nos abrían dos caminos: uno era la actividad *MainActivity* que acabamos de explicar, y el otro camino es la última actividad que nos queda por ver en nuestra aplicación, *ViewActivity*.

Como indicamos en el *Intent* del método *viewVideo* presente en la clase *InfoActivity*, la siguiente actividad en ejecutarse en el caso en que pulsemos sobre el icono situado a la derecha del *editText* que contiene el nombre de las cámaras presentes en nuestro proyecto, será *ViewActivity.java*, actividad que nos permitirá visualizar el vídeo que está capturando cada cámara en directo mediante una conexión web.

Siguiendo con el esquema, primero veremos la imagen del aspecto que tendría la actividad en un terminal móvil Android (archivo *activity_view.xml*) en la imagen siguiente y posteriormente realizo la descripción y explicación de la actividad.

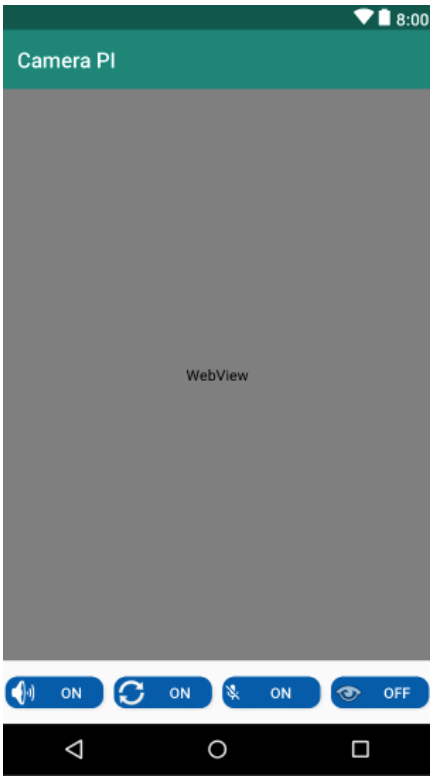

Figura 53. Layout View Activity.

Como podemos intuir tras ver el *layout* de esta actividad, se va a tratar de una actividad algo más simple, ya que tan solo observamos un *WebView* central y cuatro botones en la parte inferior.

En primer lugar, tenemos el WebView que acabamos de comentar. Esta es la parte más importante de la actividad, ya que lo que queremos es ver el vídeo en directo de lo que están captando nuestras cámaras, y esto lo hacemos con una conexión web. Para realizar esta conexión, empleamos un WebView, que no es más que una clase que extiende de la clase View de Android que nos permite mostrar páginas web como parte del diseño de nuestra actividad, justo lo que nos interesa.

En segundo lugar, observamos en la parte inferior del *layout* cuatro botones. No se trata de botones normales y corrientes, sino de cuatro *toggleButtons*, que son botones que nos permiten alternar entre dos estados.

Hasta aquí llega la descripción del archivo *activity_view.xml*, que es algo breve, pero es que esta actividad no tiene otra funcionalidad más que mostrarnos el vídeo mediante la conexión web. Pero como en las actividades anteriores, cada archivo xml lleva asociado un archivo de código fuente. El archivo de código fuente asociado a este archivo xml se llama *ViewActivity.java*. Aquí es donde se programa todo el argumento de la actividad, que en este caso no es muy complicado.

Capítulo 6. Resultados

Para terminar, en este apartado mostraré capturas de cómo se vería en la pantalla de un terminal Android real el vídeo en directo capturado por las tres cámaras de las que dispongo para el proyecto. Cada cámara va conectada al sistema de una manera distinta, con el fin de probar la mayor variedad de posibilidades de conexión (módulo de cámara de Raspberry, cámara USB y cámara IP).

En primer lugar, se muestra cómo se ve el vídeo capturado por el módulo de cámara de la Raspberry PI, que ha sido ubicado junto a nuestro servidor VPN (Raspberry PI) en el recibidor de la vivienda, con el fin de visualizar la puerta de entrada a la casa. Si alguien intenta acceder a la vivienda, en cuanto la puerta se abra mínimamente nuestro sistema nos avisará de ello.

En la vista vertical, abajo a la izquierda se indica el nombre de la cámara, en este caso podemos ver "CÁMARA RECIBIDOR", mientras que abajo a la derecha tenemos la fecha y la hora, para comprobar que no se ha quedado congelada la imagen ni nos han saboteado el sistema. También, a la hora de tomar las fotos cuando se detecte presencia, la instantánea contendrá esta información, información bastante importante.

La conexión de esta cámara, como ya se ha explicado antes, es a un puerto de la Raspberry PI habilitado exclusivamente para esta cámara. Por lo tanto, esta cámara está a escasos centímetros de nuestro servidor (unos 5 cm que tiene de largo la pista de la cámara), y debemos situarla en un lugar estable que no comprometa la seguridad del sistema, ya que de caerse tendríamos un problema.

A la hora de emplear el Punto de Acceso mediante terminal móvil que estudiamos para hacer que nuestro sistema funcione de forma autónoma, no tendríamos ningún problema, ya que la Raspberry PI es quien estaría conectada al Punto de Acceso y la cámara actuaría de igual manera que hasta ahora.

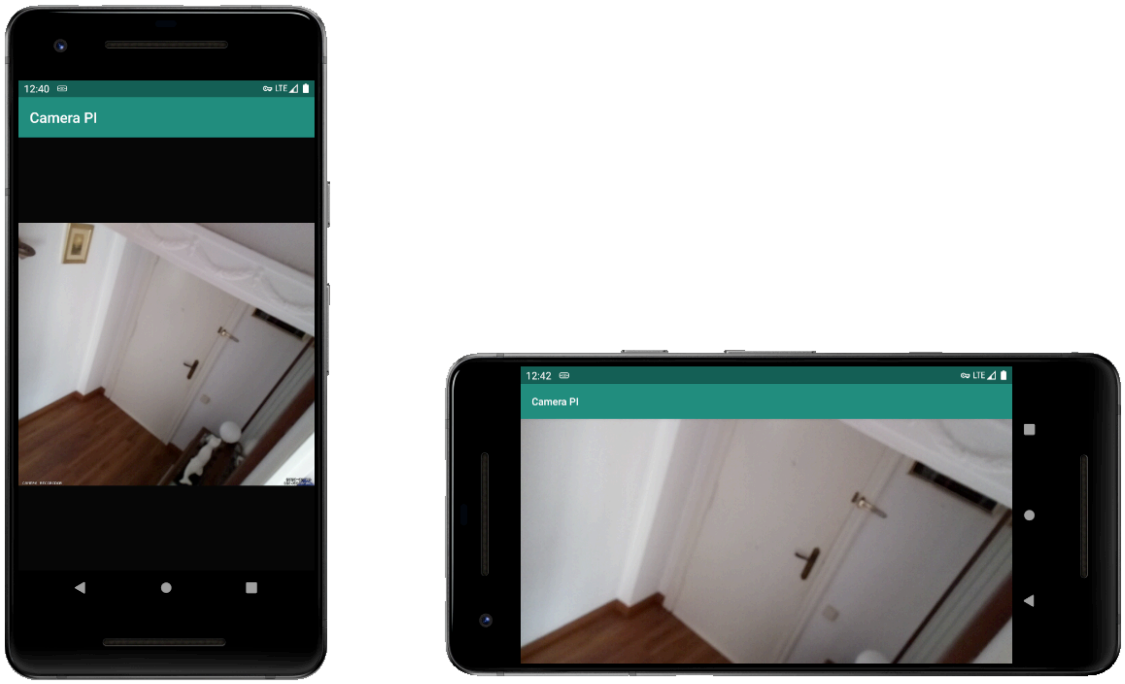

Figura 54. Resultado cámara USB.

A continuación, se muestra cómo se vería el vídeo capturado por la cámara USB, que se ha colocado cerca de la Raspberry PI, pero orientada en dirección opuesta a la cámara anterior con

el fin de abarcar el máximo espacio posible, ya que va conectada mediante un cable USB de poco más de medio metro, cosa que limita un poco ubicación.

Como la puerta la tenemos bien vigilada por la cámara anterior, esta se ha orientado hacia el pasillo, con el fin de tener controladas las puertas a todas las habitaciones de la vivienda.

En la vista vertical, abajo a la izquierda se indica el nombre de la cámara, como vimos en la cámara anterior, pero en este caso se trata de "CÁMARA PASILLO". De nuevo, abajo a la derecha tenemos la fecha y la hora.

La conexión de esta cámara es a un puerto USB de la Raspberry PI. Por lo tanto, esta cámara debe estar situada no muy lejos de nuestro servidor, cosa que no es ningún problema.

A la hora de emplear el Punto de Acceso mediante terminal móvil, que estudiamos para hacer que nuestro sistema funcione de forma autónoma, no tendríamos ningún problema, ya que la Raspberry PI es quien estaría conectada al Punto de Acceso y la cámara actuaría de igual manera que hasta ahora.

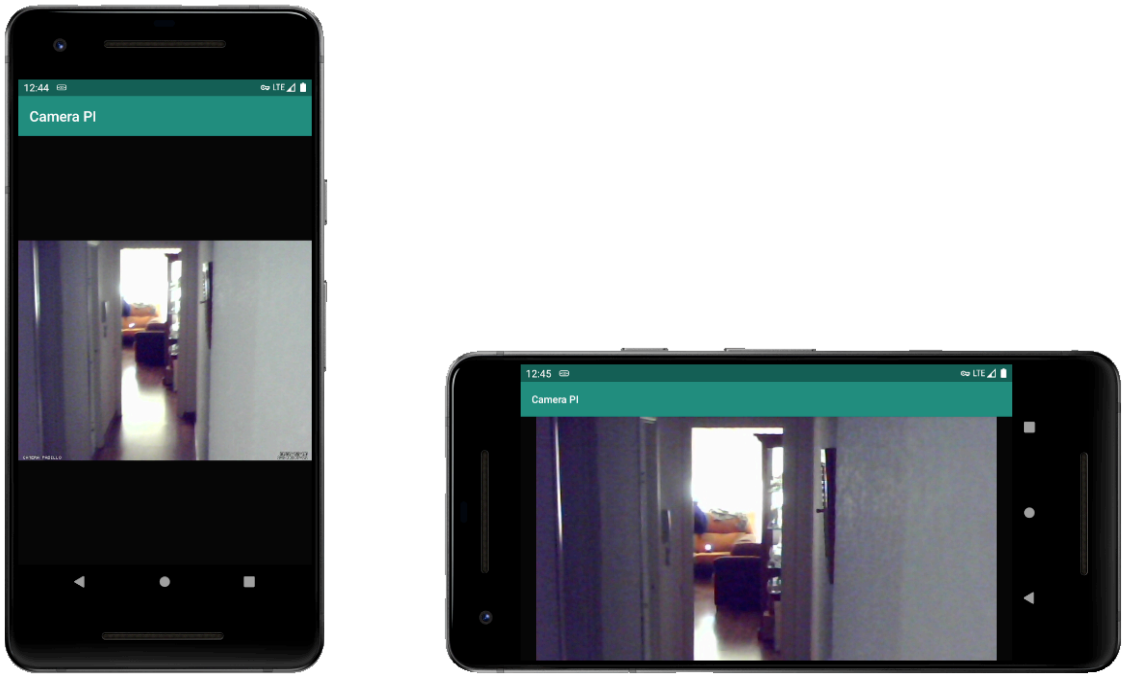

Figura 55. Resultado cámara Raspberry PI.

Por último, se muestra cómo se vería la cámara IP, que ha sido ubicada en el salón, con el fin de tener todo el perímetro de este bien visualizado, desde la otra parte del pasillo que terminaba grabando la segunda cámara (cámara USB).

La conexión de esta cámara se ha establecido mediante conexión Wi-Fi, ya que, al disponer de esta opción, es mucho más cómodo debido a que no tenemos la restricción de la longitud del cable Ethernet, por lo que no estamos obligados a estar a escasos centímetros de nuestro router o Raspberry PI. Podemos situarla en cualquier zona de la casa o superficie a vigilar siempre y cuando llegue la señal con la suficiente potencia para nuestra aplicación.

A la hora de emplear el Punto de Acceso mediante terminal móvil que estudiamos para hacer que nuestro sistema funcione de forma autónoma, no tendríamos ningún problema, ya que la Raspberry PI es quien estaría conectada al Punto de Acceso y la cámara actuaría de forma inalámbrica conectada por Wi-Fi con la Raspberry.

Esta cámara es mucho más completa que las dos anteriores. Como vemos en la parte izquierda de la pantalla del terminal, esta cámara ofrece movimiento 360º.

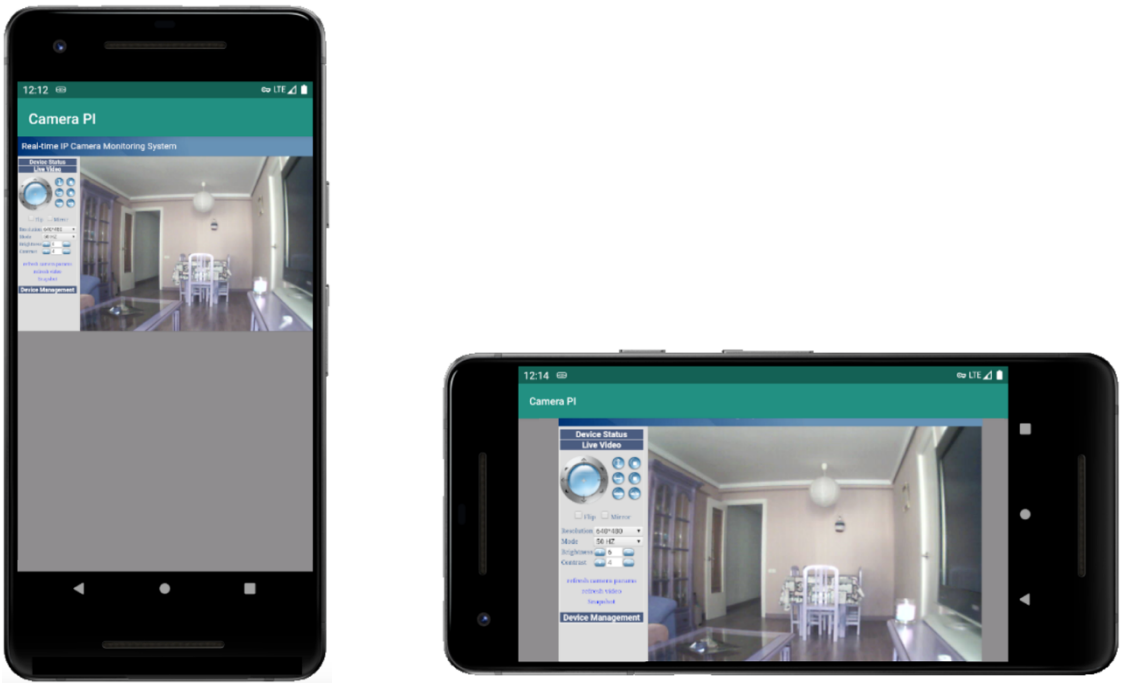

Figura 56. Resultado cámara IP.

Comentando alguna característica más de esta última cámara, que a la vista está que se trata de la más completa y compleja de todas, decir que cuando detecta presencia, envía la alerta por correo electrónico, pero para esta cámara, no hace falta configurar un archivo de Motion (camera.conf), ya que dentro de las propias configuraciones de la cámara, ofrece la opción de introducir la cuenta de correo a la que avisar.

Puede accederse a estas configuraciones en modo administrador.

Capítulo 7. Líneas futuras: sistema autónomo

7.1 Dotar al sistema de conexión red

Con el fin de darle esta propiedad de sistema autónomo, inicialmente se pensó en emplear algún tipo de módulo GSM/GPRS que dotase a nuestro servidor de conexión a Internet insertando en dicho módulo una tarjeta SIM de datos.

Uno de los módulos que se barajó fue el M95 de Quectel, que es un módulo GSM/GPRS cuatribanda compacto que tiene un consumo de energía bajo (unos 2 mA en reposo) y un rango de temperatura (-40 ºC a +85 ºC) que nos permitiría trabajar prácticamente en cualquier situación y zona.

En la imagen que se muestra a continuación tenemos la placa M95, que viene acompañada de su fuente de alimentación, unos auriculares por si quisiésemos emplearla para realizar llamadas telefónicas, un cable convertidor de USB a UART, una antena y un cable RF donde conectaremos la antena.

Figura 57. Módulo GSM/GPRS.

Lo que nos interesaría para nuestro proyecto, es introducirle a este módulo una tarjeta SIM de datos con el fin de poder utilizar nuestro servidor VPN en lugares donde no dispongamos de conexión a Internet por cable. Esto es lo que le daría al sistema esa característica de autonomía.

Como alternativa a este módulo, se ha pensado en emplear un teléfono móvil cualquiera que disponga de la opción de "Compartir Internet", es decir, convertir un terminal móvil en un Punto de Acceso Portátil, opción que el 90% de los teléfonos móviles de los últimos 5 años disponen.

Por lo tanto, se ha pensado en reutilizar un terminal que tengamos por casa y emplearlo como Punto de Acceso para dotar a nuestra Raspberry de conexión red. Del mismo modo que empleando un módulo GSM/GPRS, necesitaremos una SIM para este terminal móvil.

Con esta opción, nos ahorraríamos los 60 euros del módulo y reutilizaríamos un terminal que tuviésemos en casa.

El estado de la batería del terminal es un poco indiferente, ya que nuestro sistema autónomo nos permitirá tenerlo cargando continuamente.

7.2 Dotar al sistema de alimentación inalámbrica

Para dotar a nuestro sistema de la propiedad de sistema autónomo, no basta sólo con crear un punto de acceso mediante tarjeta SIM. Debemos recordar que, como todo sistema electrónico, necesita ser alimentado.

Dotar de energía a nuestra Raspberry PI es una tarea que puede considerarse sencilla debido al consumo de este dispositivo (no es excesivamente elevado como se aprecia en el gráfico de barras de la Figura 58) y también a las mejoras en la eficiencia de las placas solares (cada vez son más eficientes).

No debemos olvidar que necesitaremos una batería, que es la encargada de suministrar la energía a nuestra placa. Las celdas solares son las encargadas de cargar esta batería.

En la siguiente imagen se muestra una comparación entre los consumos de los distintos modelos de Raspberry PI, donde podemos ver que nuestra Raspberry PI 3 consume prácticamente lo mismo que la Raspberry PI 2, siendo capaz de hacer las mismas tareas en la mitad de tiempo. También se observa un menor consumo en estos dos últimos modelos respecto al modelo anterior, la Raspberry PI B:

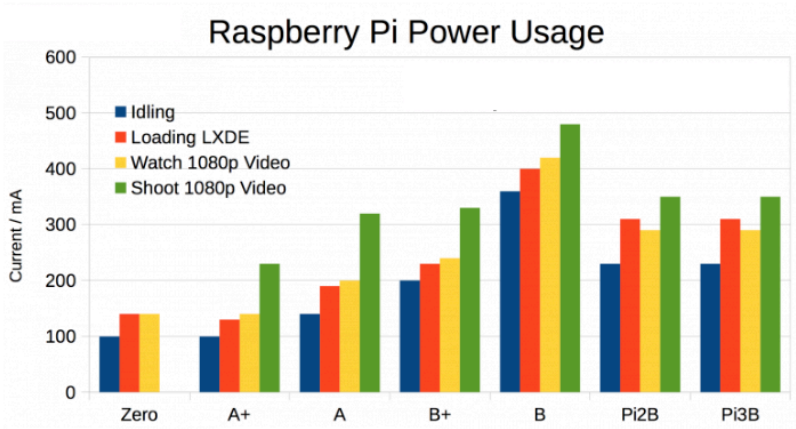

Figura 58. Consumos Raspberry's.

Para realizar el estudio del consumo, no nos sirve con los valores aproximados que se dan en la imagen anterior, ya que son para tareas determinadas.

Por lo tanto, debemos contrastar numerosas fuentes que contengan mediciones reales.

La máxima corriente para la que ha sido diseñada nuestra Raspberry PI es 2.4 A, según nos cuentan las especificaciones del instrumento.

En modo reposo, el consumo ronda los 150 mA, mientras que, soportando una considerable carga de trabajo, este valor puede ascender hasta llegar a los 1000 mA.

El trabajo que nosotros le vamos a exigir a nuestra Raspberry PI no será tan excesivo, sino que estaremos hablando de unos 300 – 400 mA.

Con el fin de no quedarnos cortos ante ninguna situación fuera de lo corriente, nos quedaremos con el valor de los 1000 mA.

Sin embargo, hemos de tener en cuenta que tendremos conectada una cámara USB, sabiendo que los puertos USB requieren un suministro de 2500 mA.

La cámara de la Raspberry PI que nosotros también utilizamos consume 250 mA.

Otros datos que hacen referencia al consumo pero que a nosotros no nos afectan debido a que no usaremos esas funcionalidades son:

- Los pines GPIO que tienen un consumo de 50 mA.
- Los teclados, que requieren de unos 100 mA.
- El puerto HDMI que consume aproximadamente 250 mA.

No debemos olvidarnos de que tendremos que alimentar el teléfono móvil que se comportará como punto de acceso, aunque el consumo de éste no será muy elevado, ya que una de las funcionalidades que más energía consume de un teléfono móvil es su gran pantalla, y en nuestro caso, estará siempre apagada. Aunque debemos tener en cuenta que si disponemos de mala cobertura en la ubicación, el dispositivo consumirá algo más de energía.

Comportándose como punto de acceso, el consumo podría situarse en unos 600 mA.

Todos estos valores han sido contrastados de varias fuentes, y para evitar quedarnos cortos en cuanto a potencia, siempre conviene redondear hacia arriba, y aplicando un porcentaje de seguridad en torno al 30%.

Como fuente de alimentación para nuestro sistema, se ha pensado en utilizar un HAT, mostrado en la siguiente imagen:

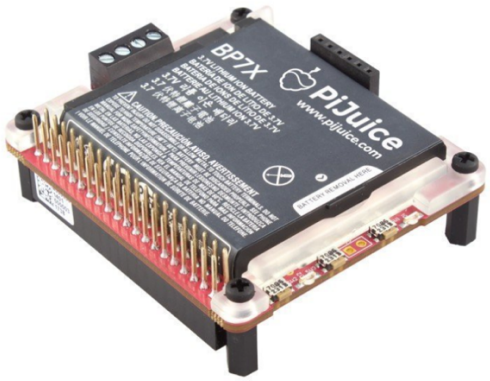

Figura 59. Alimentador para sistema autónomo.

Este HAT nos permitirá tener alimentado nuestro sistema en todo momento. Se trata de un sistema de alimentación ininterrumpida (UPS) configurable para ejecutar un apagado administrado siempre y cuando la potencia no sea la suficiente.

Para garantizar que cubrimos los requerimientos de nuestro proyecto, necesitaremos una batería de unos 4500 mAh (1000 mA de la Raspberry PI, 2750 mA de la cámara USB, 600 mA del terminal móvil) que produzca los 5 V de tensión necesarios.

El HAT consta de una batería de 1820 mAh, que proporcionaría unas 6 horas de funcionamiento. Este tiempo puede que no sea suficiente para mantener nuestro sistema activo durante toda la noche, sin embargo, tiene soporte para añadir una batería de mayor capacidad.

Fácilmente podríamos añadir una batería de 4000 mAh que nos aseguraría hasta 24 horas más de energía.

Si nuestro presupuesto es mayor, en lugar de añadir una batería de 4000 mAh, podemos añadir otra de mayor capacidad, cosa que nos permitiría aguantar nuestro sistema en caso de tener más de dos días sin sol donde no podríamos cargar las baterías.

La mejor opción es añadir una batería que hay específica para el HAT de la Raspberry que nos aportaría 12.000 mAh por tan solo 30 euros. Con esto nos aseguraríamos hasta 4 días de energía sin luz.

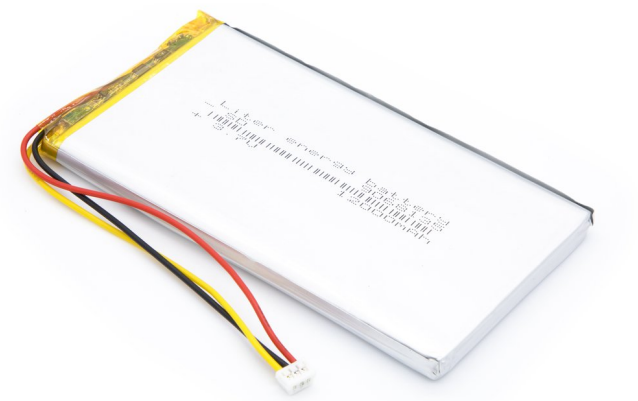

Figura 60. Batería de 12.000 mAh.

Gracias a unos leds de los que dispone, podemos conocer en todo momento el nivel de las baterías.

El HAT anterior se ha convertido en la solución más elegida por los usuarios que quieren abastecer de energía su Raspberry de forma autónoma.

Tiene un precio de 56,90 euros.

Este módulo puede ser alimentado con cualquier tipo de energía renovable.

Para nuestro proyecto, se ha pensado en emplear paneles solares como fuente de energía.

Mientras haya luz solar, no gastaremos energía de nuestra batería, ya que aprovecha la que proporcionan los paneles.

El mismo fabricante del HAT anterior, dispone de un panel solar de 12 Watios de potencia ideal para alimentar nuestra HAT y nuestra Raspberry PI. El ejemplo de panel solar se muestra en la siguiente imagen:

Figura 61. Panel Solar sistema autónomo.

Este panel es resistente al agua, y fácilmente plegable, por lo que es perfecto para ubicarlo al aire libre cubriendo nuestro servidor.

Es compatible con cualquier dispositivo que se alimente a 5 V.

Tiene dos salidas reguladas de 5 V, para que podamos cargar nuestra batería con la que alimentaremos el sistema.

El único inconveniente es su precio, 79 euros.

Con estos dos aparatos podríamos situar nuestro servidor en lugares donde no llegue la corriente eléctrica ni dispongamos de conexión a Internet, tan solo debemos tener cobertura para nuestra SIM y la mayor claridad posible para cargar nuestra batería con el panel solar.

Darle a nuestro sistema la propiedad de sistema autónomo tendría un coste de 135,9 euros (HAT + Panel Solar). No es un precio excesivamente elevado que, en caso de necesidad, por ese precio tendríamos vigilada donde no llegase red ni corriente eléctrica.

Capítulo 8. Bibliografía

[1] Cámara| Desarrolladores de Android | Android Developers. (2020). Retrieved 12 March 2020, from https://developer.android.com/training/camera?hl=es-419

[2] Camera Module - Raspberry Pi Documentation. (2020). Retrieved 12 March 2020, from https://www.raspberrypi.org/documentation/usage/camera/

[3] Carrasco, V. (2020). Configurando Motion con la Cámara de Raspberry Pi – Internet de las Cosas. Retrieved 12 March 2020, from https://www.internetdelascosas.cl/2013/10/13/configurando-motion-con-la-camara-de-raspberry-pi/

[4] Cómo conectarse a un servidor VPN desde Android. (2020). Retrieved 12 March 2020, from https://www.adslzone.net/redes/privacidad/como-conectarse-un-servidor-vpn-desde-android/

[5] Cómo guardar datos con SQLite | Desarrolladores de Android. (2020). Retrieved 12 March 2020, from https://developer.android.com/training/data-storage/sqlite?hl=es-419

[6] Cómo usar las celdas solares para alimentar el Raspberry Pi 3 | DigiKey. (2020). Retrieved 12 March 2020, from https://www.digikey.es/es/articles/techzone/2016/jul/how-to-use-solar-cells-to-power-a-raspberry-pi-3-single-board-computer

[7] How to Send SMTP Email using Raspberry Pi. (2020). Retrieved 12 March 2020, from https://iotdesignpro.com/projects/sending-smtp-email-using-raspberry-pi

[8] Instalar un servidor VPN en Raspberry PI con OpenVPN. (2020). Retrieved 12 March 2020, from https://www.jabenitez.com/2018/10/01/instalar-un-servidor-vpn-en-raspberry-pi-con-openvpn/

[9] Motion (detección de movimiento) con Raspberry Pi. (2020). Retrieved 12 March 2020, from https://eltallerdelbit.com/motion-raspberry/

[10] Pi, C. (2020). Crear un servidor VPN en un Raspberry Pi. Retrieved 12 March 2020, from https://www.ionos.es/digitalguide/servidores/configuracion/crear-un-servidor-vpn-en-un-raspberry-pi/

[11] PI, P. (2020). PIJUICE - PLATAFORMA PORTATIL DE ALIMENTACION PARA RASPBERRY PI - tiendatec.es. Retrieved 12 March 2020, from https://www.tiendatec.es/raspberry-pi/raspberry-pi-alimentacion/805-pijuice-hat-plataforma-portatil-de-alimentacion-para-raspberry-pi-616909467655.html

[12] PiVPN es la opción más fácil y rápida para configurar un servidor OpenVPN en tu Raspberry Pi. (2020). Retrieved 12 March 2020, from https://www.redeszone.net/2017/02/17/pivpn-es-la-opcion-mas-facil-y-rapida-para-configurar-un-servidor-openvpn-en-tu-raspberry-pi/

[13] Revelo, J. (2020). Tutorial De Bases De Datos SQLite En Aplicaciones Android. Retrieved 12 March 2020, from http://www.hermosaprogramacion.com/2014/10/android-sqlite-bases-de-datos/

[14] Uzábal, A. (2020). Servidor de webcam con Motion en una Raspberry Pi. Retrieved 12 March 2020, from https://voragine.net/linux/servidor-de-webcam-con-motion-en-una-raspberry-pi

[15] Vaati, E. (2020). Enviar correos electrónicos en Python con SMTP. Retrieved 12 March 2020, from https://code.tutsplus.com/es/tutorials/sending-emails-in-python-with-smtp--cms-29975

[16] Video Streaming Raspberry Pi Camera | Random Nerd Tutorials. (2020). Retrieved 12 March 2020, from https://randomnerdtutorials.com/video-streaming-with-raspberry-pi-camera/

[17] VPN | Android Developers. (2020). Retrieved 12 March 2020, from https://developer.android.com/guide/topics/connectivity/vpn

[18] Wagle, B., Begum.Nabianwar, A., & Santos, S. (2020). Guide to Raspberry Pi Camera V2 Module | Random Nerd Tutorials. Retrieved 12 March 2020, from https://randomnerdtutorials.com/guide-to-raspberry-pi-camera-v2-module/

[19] 3 maneras alternativas de alimentar una Raspberry Pi - Clon Geek. (2020). Retrieved 12 March 2020, from https://clongeek.com/3-maneras-de-alimentar-raspberry-pi/

[20] 12W, P. (2020). PIJUICE - PANEL SOLAR 12W - tiendatec.es. Retrieved 12 March 2020, from https://www.tiendatec.es/raspberry-pi/raspberry-pi-alimentacion/809-pijuice-panel-solar-12w-616909468041.html

Capítulo 9. Anexos

9.1 Activity Login

La primera parte de estos archivos siempre es la importación de paquetes necesarios en la programación de la actividad. Por ejemplo, si en nuestra actividad vamos a hacer uso de algún botón, editText, Toast, …, debemos importar la clase Button, EditText, Toast, … A continuación, dejo un ejemplo de cómo sería:

```
import android.content.Intent;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
```

El propio Android Studio nos sugiere las clases que necesitamos importar con forme vayamos avanzando en la programación de la actividad. Tan solo debemos pulsar la combinación *Alt + Enter* cuando nos lo sugiera, y se importará automáticamente.

En primer lugar, una vez salvado el detalle anterior, definimos los botones y cuadros de texto que tiene nuestra actividad, y los asociamos con los nombres que tienen en el archivo xml. También definimos un contador y le damos el valor de 3.

```
Button loginButton, cancelButon;
EditText user, pass;
TextView numAttempts;
int counter = 3;                    // Número de intentos fijado en 3
```

Mediante un condicional, comprobamos que lo que se introduce en el cuadro de entrada de texto coincide con las credenciales elegidas en este condicional.

Si son correctas, mediante un *Intent* (objeto que nos proporciona vinculación en tiempo de ejecución entre componentes separados, como puede ser el caso de dos actividades) pasamos a la siguiente actividad, en este caso *InfoActivity.class.* Esto se realiza de la siguiente manera:

```
// Creamos el intent y le indicamos desde donde vamos (this) y a qué actividad vamos
(InfoActivity.class)
Intent intent = new Intent(getApplicationContext(), InfoActivity.class);
startActivity(intent);
```

En caso de que las credenciales no sean correctas, decrementamos el contador y actualizamos el número de intentos como ya he comentado antes. Esto quedaría así:

```
numAttempts.setVisibility(View.VISIBLE);        // Decrementamos en 1 el contador
counter--;              // Mostramos el num de intentos restantes con el contador decrementado
numAttempts.setText(Integer.toString(counter));
```

Si acabamos con el número total de intentos, el contador llegará a 0, y como medida de seguridad, deberemos abandonar la aplicación, ya que no podremos continuar identificándonos porque el botón “LOGIN” estará deshabilitado:

```
if (counter == 0) {          // Si el num de intentos llega a 0, deshabilitamos el botón de login
    loginButton.setEnabled(false);
}
```

Por último, queda comentar la acción del botón “CANCEL”, que simplemente llevará a cabo la acción de abandonar la aplicación al ser pulsado por medio de la función *finish()*:

```
cancelButon.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        finish();
    }
});
```

Con esto, tendríamos completamente visto el código fuente de la primera actividad.

9.2 Activity Info

El archivo *InfoActivity.java* consta de siete métodos:

El primero de ellos es el método *onCreate*, presente en todas las actividades, y es el que se encarga de asociar el código fuente de la actividad con el archivo xml correspondiente. También aquí es donde hacemos referencia a la Base de Datos creada para poder leer de ella desde esta actividad. Por último, también identificamos cada uno de los seis *textView*. Aquí un ejemplo:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_info);


    Comparticion.helper = new BBDD_Helper(this);
    helper = Comparticion.helper;


    camera1 = (TextView)findViewById(R.id.camera1);
}
```

Como se observa, se asocial con el layout de *activity_info.xml*, archivo justamente descrito.

El segundo de los métodos es *modify*, que va a ser el encargado de cambiar a la actividad encargada de llevar a cabo la modificación de las cámaras. La llamada a este método se producirá cuando se pulse sobre el botón que contiene el texto "MODIFY MY CAMERAS" como se ha explicado anteriormente. Este cambio de actividad se lleva a cabo mediante el uso de *Intent*, como he explicado en la actividad anterior. Aquí se puede ver el método *modify*:

```
public void modify (View view){.                    // Vamos a la actividad MainActivity
    Intent i = new Intent(this, MainActivity.class);
    startActivity(i);
}
```

En tercer lugar, tenemos un método sencillo llamado *goToGmail* cuya única función es abrirnos la aplicación GMAIL de nuestro terminal Android con la finalidad de comprobar el correo electrónico, lugar donde son enviadas imágenes capturadas por nuestro sistema cuando se produce movimiento.

El cambio de aplicación también se lleva a cabo mediante el uso de intent, al igual que en el método anterior.

```
public void gmail (View view) {
    Intent g = getPackageManager().getLaunchIntentForPackage("com.google.android.gm");
    startActivity(g);
}
```

Tras este, tenemos el siguiente método llamado *phone* que se encarga de abrirnos la aplicación del TELÉFONO, con el objetivo de realizar de manera rápida una llamada a la policía en caso de que veamos que alguien ha entrado en nuestra vivienda.

```
public void phone (View view) {
    // Vamos a la aplicación PHONE
    Intent p =
getPackageManager().getLaunchIntentForPackage("com.android.contacts");
    startActivity(p);
}
```

El quinto método es *viewVideo*, que va a ser el encargado de acceder al vídeo en directo de las cámaras que tengamos en nuestro proyecto, cambiando a la actividad *ViewActivity.java*. Este método va a ser llamado cuando se pulse sobre cada uno de los *textView* que contengan información sobre una cámara. Este cambio de actividad se lleva a cabo de nuevo mediante el uso de *Intent*. Aquí debemos comprobar cuál es el *textView* que ha sido accionado para elegir correctamente la dirección IP a la que conectarnos para ver la cámara elegida. Esto se consigue mediante el uso del sistema switch – case, un esquema sencillo mediante el cual almacenamos un valor determinado en una variable según que cámara ha sido seleccionada. A continuación, se puede ver el método *viewVideo*:

```
public void viewVideo (View view){
   // Comprobamos cuál de los TextView ha sido pulsado
   // para saber qué dirección IP debemos buscar
   String dirIP;
   int i=-1;
   switch (view.getId()) {
      case R.id.camera1:
         i=0;
         break;
      case R.id.camera2:
         i=1;
         break;
      case R.id.camera3:
         i=2;
         break;
      case R.id.camera4:
         i=3;
         break;
      case R.id.camera5:
         i=4;
         break;
      case R.id.camera6:
         i=5;
         break;
   }

   dirIP = IPdirTot[i];

   // Vamos a la actividad ViewActivity
   Intent o = new Intent(this, ViewActivity.class);
   o.putExtra("laIP", dirIP);            // Pasamos la IP seleccionada a ViewActivity
   startActivity(o);
}
```

En sexto lugar, tenemos el método *exitApp*, que como he dicho antes es llamado al pulsar el botón de "EXIT" y se encarga de abandonar la aplicación. Es un método muy sencillo que se muestra a continuación:

```
public void exitApp (View view){
    // Salimos de la aplicación
    finish();
}
```

Después podemos ver el método *onResume*, método con el que se pretende que cada vez que se modifique alguna cámara, y por lo tanto toquemos la Base de Datos, cuando volvamos a esta actividad, se actualice la lista de cámaras. En este método llamaremos al método siguiente (*leerBBDD*) que será el encargado de interactuar con la Base de Datos en esta actividad.

```
protected void onResume(){
    super.onResume();
    // Leemos de la BBDD para actualizar los TextView
    leerBBDD();
}
```

Por último, en octavo lugar tenemos el método *leerBBDD* que acabamos de comentar. Su función es leer de la Base de Datos para actualizar la lista de cámaras.

Con esto, tendríamos completamente visto el código fuente de la segunda actividad.

9.3 Main Activity

Como los demás archivos de código fuente del proyecto que hemos analizado, este también comienza con la importación de las librerías necesarias para la implementación del código. Omitiendo las librerías básicas que ya comentamos en la primera actividad, por el hecho de trabajar ahora con la base de datos, necesitaremos un par de librerías desconocidas hasta ahora, que son las siguientes:

```
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
```

Como en las actividades anteriores, también declararemos botones, cuadros de texto, switches, y demás elementos asociándolos a los nombres que les hayamos dado en el archivo *activity_main.xml*.

La guía que comentábamos nos proporciona una plantilla de esta clase, por lo que la copiaremos y modificaremos después para que nuestra Base de Datos tenga la estructura que necesitamos.

En primer lugar, nos encontramos con un constructor al que debemos darle el nombre de la clase, es decir, *Estructura_BBDD*:

```
private Estructura_BBDD() {}
```

A continuación, tenemos unas constantes de clase que sirven para crear el nombre de la tabla ("*datosCam*" en mi caso), el nombre de la primera columna, etc. Tenemos que modificar esta plantilla para añadir las columnas que necesitemos para nuestro proyecto. Tras esta modificación, la estructura de mi Base de Datos quedaría así:

```
public static final String TABLE_NAME = "datosCam";
public static final String NOMBRE_COLUMNA1 = "Id";
public static final String NOMBRE_COLUMNA2 = "Model";
public static final String NOMBRE_COLUMNA3 = "Ip";
```

```
public static final String NOMBRE_COLUMNA4 = "Description";
public static final String NOMBRE_COLUMNA5 = "Power";
public static final String NOMBRE_COLUMNA6 = "Notify";
```

Observamos que, a parte del nombre de la tabla, hay 6 columnas más, una para cada parámetro de los que vimos en el *layout* de esta actividad que formarán parte de la cámara añadida al proyecto. Tenemos los 4 *editText* (Id, Model, Ip y Description) y los 2 *Switchs* (Power y Notify), que son los datos que queremos que sean guardados en nuestra Base de Datos.

Tras definir el aspecto de la Base de Datos, tenemos que implementar métodos para crear (ejecutar la estructura definida) y mantener la Base de Datos y la tabla.

En primer lugar, se crean tres constantes de clase, una nos indica el tipo de texto, otra para indicar cuál va a ser el separador de campos y la última contiene la instrucción SQL para crear la tabla.

Estas tres constantes de clase adaptadas al diseño de nuestra Base de Datos quedarían de esta forma:

```
private static final String TEXT_TYPE = " TEXT";
private static final String COMMA_SEP = ",";
public static final String SQL_CREATE_ENTRIES =
    "CREATE TABLE " + Estructura_BBDD.TABLE_NAME + " (" +
        Estructura_BBDD.NOMBRE_COLUMNA1 + " INTEGER PRIMARY KEY," +
        Estructura_BBDD.NOMBRE_COLUMNA2 + TEXT_TYPE + COMMA_SEP +
        Estructura_BBDD.NOMBRE_COLUMNA3 + TEXT_TYPE + COMMA_SEP +
        Estructura_BBDD.NOMBRE_COLUMNA4 + TEXT_TYPE + COMMA_SEP +
        Estructura_BBDD.NOMBRE_COLUMNA5 + TEXT_TYPE + COMMA_SEP +
        Estructura_BBDD.NOMBRE_COLUMNA6 + TEXT_TYPE + " )";
```

El tipo de texto que hemos comentado va a ser “TEXT”, el tipo de separador será una coma “,” y la instrucción para crear la tabla ya está con el nombre de nuestra clase (Estructura_BBDD) y con las demás columnas añadidas separadas por las comas.

Por último, tenemos otra constante de clase denominada *SQL_DELETE_ENTRIES* que tiene como función eliminar la tabla si ya existe. Esta constante de clase se muestra a continuación:

```
public static final String SQL_DELETE_ENTRIES =
    "DROP TABLE IF EXISTS " + Estructura_BBDD.TABLE_NAME;
```

Cada vez que hagamos una operación de actualización o creación en nuestra Base de Datos, se encargará de eliminar la tabla si ya existe, bien para actualizarla o bien para crearla por primera vez.

Con esto ya tendríamos completada la clase *Estructura_BBDD.java*.

Para la creación de esta clase *Helper*, seguiremos la metodología de la clase que conforma la estructura, y es la de coger la plantilla que nos proporciona la API de Android Studio y modificarla con el fin de adaptarla a los requerimientos de nuestra aplicación.

En primer lugar, como hemos dicho, esta clase debe heredar de la clase “*SQLiteOpenHelper*”, por lo tanto, en la declaración de nuestra clase *Helper*, tenemos que añadir este detalle:

```
public class BBDD_Helper extends SQLiteOpenHelper
```

Para adaptar el código de esta plantilla a nuestras necesidades, tan solo debemos añadir el nombre de la clase Estructura_BBDD delante de las constantes de clase *SQL_CREATE_ENTRIES* y

SQL_DELETE_ENTRIES, ya que, al tratarse de constantes de clase estáticas, para poder usarlas necesitamos poner delante el nombre de la clase donde están definidas.

En el fragmento de código siguiente se muestra esta modificación que acabo de comentar delante de las constantes de clase *SQL_CREATE_ENTRIES* y *SQL_DELETE_ENTRIES:*

```
public void onCreate(SQLiteDatabase db) {
    db.execSQL(Estructura_BBDD.SQL_CREATE_ENTRIES);
}


public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL(Estructura_BBDD.SQL_DELETE_ENTRIES);
    onCreate(db);
}
```

Con esto, ya tendríamos este código adaptado a nuestra Base de Datos.

Una última cosa que comentar sobre la clase *BBDD_Helper.java* es estas dos líneas de código, que no pueden pasar desapercibidas, especialmente la segunda de ellas:

```
public static final int DATABASE_VERSION = 1;
public static final String DATABASE_NAME = "Datos_Camaras";
```

La primera de las líneas lo que hace es crear una constante de clase de tipo entero cuya función es hacer referencia a la versión de la Base de Datos. No tiene nada que ver con el número de serialización.

La constante importante es la que se encuentra en la segunda línea, la que describiré a continuación, que se encarga de hacer referencia al nombre de la Base de Datos.

Anteriormente, en la clase *Estructura_BBDD*, lo que hicimos fue dar nombre a la tabla de la Base de Datos y a los campos de esa tabla, pero en ningún momento nombramos a la Base de Datos. Esto es lo que haremos con esta línea, asignándole el nombre de "Datos_Cámaras".

Ahora sí que hemos visto las dos clases que nos sugerían en la API de Android Studio para la implementación de nuestra Base de Datos, la clase *Estructura* y la clase *Helper*.

Continuando con la guía sobre "Cómo guardar datos con SQLite", nos encontramos en el apartado que nos informa de cómo introducir datos en nuestra Base de Datos.

Para ingresar información, primero hemos de crear una instancia en el archivo MainActivity.java, que es donde se desarrolla la actividad que estamos comentado y que hace uso de la Base de Datos. La instancia debe ser de la sublcase "*SQLiteOpenHelper*", y quedaría de la siguiente manera:

```
final BBDD_Helper helper = new BBDD_Helper(this);
```

Antes de seguir con la explicación del código, debemos tener claro cuando vamos a querer que se inserten valores en la Base de Datos, y esto va a suceder cada vez que el usuario pulse sobre el botón "ADD", ya que el usuario rellenará los campos de la cámara, y la añadirá al proyecto con dicho botón. Por lo tanto, debemos poner este botón a la escucha.

No podemos olvidar que, en algún momento, también el usuario querrá actualizar, buscar y/o eliminar datos de la Base de Datos, por lo que debemos poner estos botones también a la escucha. Esto quedaría de la siguiente manera:

```
botonInsertar.setOnClickListener(new View.OnClickListener());
```

```
botonActualizar.setOnClickListener(new View.OnClickListener());
botonBorrar.setOnClickListener(new View.OnClickListener());
botonBuscar.setOnClickListener(new View.OnClickListener());
```

Una vez tenemos los cuatro botones a la escucha, es dentro del método onClick() de cada uno de los cuatro botones donde tendremos que desarrollar el código para que los datos se escriban o borren en la Base de Datos.

Echando un vistazo a nuestra interfaz gráfica, el usuario escribirá la información en los *editText* y dejará en una posición concreta cada uno de los *switches*, y queremos que esta información introducida se guarde en nuestra Base de Datos al pulsar el botón de añadir. Por lo tanto, debemos ir al archivo MainActivity.java y, dentro del método onClick() del botón añadir, programar el código necesario para que esta información introducida sea guardada.

Para ello, lo primero que debemos hacer es crear una instancia de la subclase "*SQLiteOpenHelper"* como ya comentamos unas líneas atrás mostrando la instancia.

Seguimos indicando que pondremos nuestra Base de Datos en modo escritura (en modo *Writable*), ya que queremos escribir en ella. Esto se consigue de la siguiente manera:

```
SQLiteDatabase db = helper.getWritableDatabase();
```

A continuación, debemos utilizar un objeto de tipo *content values* para insertar datos, así como añadir el dato escrito en cada *editText* a su correspondiente columna de la tabla que conforma la Base de Datos. Quedaría así este paso:

```
ContentValues values = new ContentValues();
values.put(Estructura_BBDD.NOMBRE_COLUMNA1, textoId.getText().toString());
values.put(Estructura_BBDD.NOMBRE_COLUMNA2, textoModel.getText().toString());
values.put(Estructura_BBDD.NOMBRE_COLUMNA3, textoIp.getText().toString());
values.put(Estructura_BBDD.NOMBRE_COLUMNA4,
textoDescription.getText().toString());
```

Con la instrucción que hay en el paréntesis, estamos capturando el texto que el usuario ha introducido en la interfaz gráfica (con el *.getText()* y posteriormente convirtiéndolo en un *string* con *.toString()*) y después, con *values.put()* le indicamos que lo vamos a almacenar en la columna 1 de nuestra Base de Datos.

Esto es así con el resto de parámetros de tipo texto introducidos por el usuario.

De igual forma, debemos añadir el estado en que se encuentren los switch modificados por el usuario, pero estos, al tratarse de valores *booleanos* de tipo *true* o *false*, se realizaría de la siguiente manera (tan solo muestro como añadir el estado de un switch ya que los 2 se hacen de la misma forma):

```
if (switchPower.isChecked()){
    values.put(Estructura_BBDD.NOMBRE_COLUMNA5, "true");
} else {
    values.put(Estructura_BBDD.NOMBRE_COLUMNA5, "false");
}
```

Explicando las líneas de código anteriores, el proceso es el siguiente: compruebo si el *switch* "Power" está activado (*true*), y si lo está, guardo en la columna correspondiente el texto "true". Si no lo está, guardo el texto "false". Muy importante, guardo una cadena de texto, no un estado de tipo booleano.

Con esto, ya tendríamos capturados los valores introducidos por el usuario y almacenados en sus correspondientes celdas de la tabla.

Sin embargo, con el fin de tener un poco de *feedback*, completaremos el código añadiendo una funcionalidad que se encargue de mostrar un mensaje cada vez que añadamos datos a nuestra Base de Datos. Esto lo haremos de la siguiente forma:

```
Toast.makeText(getApplicationContext(), "Se guardó la cámara con ID: " + newRowId,
Toast.LENGTH_LONG).show();
```

Ahora si que damos por completado el código del botón de añadir.

Seguiremos el mismo proceso para el botón "SEARCH", cuya función es buscar la información en nuestra Base de Datos y plasmarla en los *editText* y *switches*.

El tutorial que estábamos siguiendo de la API de Android, nos dice que para realizar esta tarea debemos utilizar el método *query()* y pasar los criterios de selección y columnas que deseamos obtener como resultado de esas búsquedas. También nos comenta que el resultado de estas búsquedas que realizamos mediante el método *query()* se devuelve en un objeto de tipo *cursor*.

Ahora la programación del código para este botón la llevaremos a cabo dentro del método onClick() del botón que ejecuta la acción de buscar, para que se ejecute cuando el usuario pulse sobre este botón de buscar (botón "SEARCH" de nuestra interfaz).

En el botón anterior comenzamos haciendo que nuestra Base de Datos se pusiera en modo escritura, ya que lo que queríamos era añadir datos. Ahora sin embargo debemos comenzar poniendo nuestra Base de Datos en modo lectura (en modo *Readable*), ya que lo que queremos es leer datos de ella al realizar una búsqueda. Esto se consigue del siguiente modo:

```
SQLiteDatabase db = helper.getReadableDatabase();
```

Siguiendo con el código, tenemos un *array* de tipo *string* llamado "projection" con el fin de especificar qué columnas de nuestra Base de Datos vamos a usar en nuestra consulta. Para que se lleve a cabo la búsqueda, debemos introducir un criterio, y como ya hemos comentado alguna vez, ese criterio será el "Camera ID", correspondiente a la columna 1 de nuestra Base de Datos. Por lo tanto, las columnas que vamos a consultar serán todas a excepción de la 1, ya que esa la introduce el usuario como criterio. Por lo tanto, este fragmento del código quedaría así:

```
String[] projection = {
      //Estructura_BBDD.NOMBRE_COLUMNA1,
      Estructura_BBDD.NOMBRE_COLUMNA2,
      Estructura_BBDD.NOMBRE_COLUMNA3,
      Estructura_BBDD.NOMBRE_COLUMNA4,
      Estructura_BBDD.NOMBRE_COLUMNA5,
      Estructura_BBDD.NOMBRE_COLUMNA6
};
```

Tenemos la columna 1 comentada ya que esa no queremos que nos la devuelva debido a que es el criterio introducido por el usuario.

A continuación, tenemos el fragmento de código correspondiente al filtrado de registros. En primer lugar, especificamos cual va a ser nuestro criterio, que ya hemos dicho que es la columna 1 que corresponde al "Camera ID":

```
String selection = Estructura_BBDD.NOMBRE_COLUMNA1 + " = ?";
```

Y, en segundo lugar, se crea un *array* en el que nos pide el lugar de donde viene el criterio. Puede ser que una consulta tenga varios criterios, de ahí que se construya el *array*, pero en nuestro caso solo vamos a tener un criterio, que lo indicamos (debemos indicar que el criterio viene del *editText* "textoId") y listo:

```
String[] selectionArgs = {textoId.getText().toString()};
```

Ahora nos toca lidiar con la creación del objeto de tipo cursor que hemos comentado anteriormente, que se va a encargar de recorrer las celdas correspondientes una vez introducido el criterio.

```
Cursor c = db.query(
        Estructura_BBDD.TABLE_NAME, // Referencia a la tabla de la consulta
        projection,              // Array con las columnas a devolver
        selection,               // Columnas que tienen el criterio
        selectionArgs,           // De donde viene el criterio
        null,                    // Agrupar los registros (no nos interesa)
        null,                    // Filtrar por grupos (no nos interesa)
        null                     // Ordenamiento (no lo usamos)
);
```

Ahora ya solo nos queda comenzar a leer valores y mostrarlos en sus correspondientes *editText* y *switches*.

Con el método que se muestra a continuación, situamos el cursor en el primer registro que nos devuelve la consulta para poder hacer una lectura de estos registros devueltos de arriba abajo. Esto lo conseguimos con esta simple instrucción:

```
c.moveToFirst();
```

Una vez aquí, debemos indicar dónde leer esa información del primer registro.

Como sabemos, lo que queremos es que en el campo "Model", aparezca el modelo que introdujo el usuario al añadir la cámara, y así con los demás campos "IP Direction" y "Description". Esto se haría de esta forma tan sencilla:

```
textoModel.setText(c.getString(0));              // columna 0 es la columna modelo
textoIp.setText(c.getString(1));                 // columna 1 es la columna ip
textoDescription.setText(c.getString(2));        // columna 2 es la columna dscripcion
```

Con el método *.setText()* introducimos el texto en el *editText* correspondiente.

Para establecer el estado en que se guardaron los *switches*, el proceso sería similar:

```
if (c.getString(3).equals("true")){
    switchPower.setChecked(true);
} else {
    switchPower.setChecked(false);
}
```

Con esto, damos por completado el código del botón de buscar.

Continuaremos con la programación de los dos botones que aún nos quedan (el botón de actualizar y el de borrar) siguiendo el mismo proceso que para los botones "SEARCH" y "ADD".

En primer lugar, implementaremos el código que hará que el botón "DELETE" elimine los datos de una cámara anteriormente guardada en nuestra Base de Datos.

De nuevo, consultamos el tutorial que estábamos siguiendo de la API de Android, que nos dice que para realizar esta tarea debemos proporcionar algún criterio de selección para identificar las filas a eliminar, que como ya sabemos, se trata del criterio "Camera ID". El método encargado de eliminar las filas correspondientes será el método *delete()*.

Ahora la programación del código para este botón la llevaremos a cabo dentro del método onClick() del botón que ejecuta la acción de eliminar (botón "DELETE" de nuestra interfaz).

En el botón anterior comenzamos haciendo que nuestra Base de Datos se pusiera en modo lectura, ya que lo que queríamos era buscar y recuperar datos. Ahora sin embargo debemos comenzar poniendo nuestra Base de Datos en modo escritura (en modo *Writable*), al igual que cuando configuramos el botón "ADD", ya que la forma en la que borraremos los datos de una cámara es escribiendo cadenas de texto vacías en las celdas que queramos eliminar de nuestra tabla. Esto se consigue de la misma manera que hicimos para el botón de añadir:

```
SQLiteDatabase db = helper.getWritableDatabase();
```

Tras este paso, tenemos que indicar aquí también cuál va a ser nuestro criterio, que como ya sabemos es "Camera ID", dato almacenado en la columna 1 de nuestra tabla.

Por lo tanto, la indicación del criterio sería la siguiente:

```
String selection = Estructura_BBDD.NOMBRE_COLUMNA1 + " LIKE ?";
```

De la misma forma que para el botón anterior, creamos un *array* donde indicamos el lugar de donde viene el criterio. Lo indicamos diciéndole que el criterio viene del *editText* "textoId" de igual manera que en el botón anterior:

```
String[] selectionArgs = {textoId.getText().toString()};
```

Una vez tenemos nuestra Base de Datos en modo escritura, y el criterio que nos dirá lo que debemos borrar, el método *delete()*, al que le pasamos el nombre de la tabla de la que debe borrar y el criterio, se encarga de llevar a cabo este borrado. Esto quedaría de la siguiente forma:

```
db.delete(Estructura_BBDD.TABLE_NAME, selection, selectionArgs);
```

Para completar esta tarea un poco más, incluiré unas líneas de código que se encargarán de resetear los cuadros de texto y *switches* de la cámara borrada una vez pulsemos sobre el botón "DELETE". Es un código sencillo que tan solo escribe una cadena de texto vacía en los *editText* y pone todos los *switches* a *off*. El código mencionado se muestra a continuación:

```
textoId.setText("");                // resetear cuadro de texto
textoModel.setText("");             // resetear cuadro de texto
textoIp.setText("");                // resetear cuadro de texto
textoDescription.setText("");       // resetear cuadro de texto
switchPower.setChecked(false);      // resetear power a off
switchNotify.setChecked(false);     // resetear notify a off
```

Con esto ya tendríamos reseteados los valores de las celdas de la tabla correspondientes al criterio introducido por el usuario.

Sin embargo, con el fin de tener un poco de *feedback*, nuevamente completaremos el código añadiendo una funcionalidad que se encargue de mostrar un mensaje cada vez que eliminemos una cámara de nuestra Base de Datos. Esto lo haremos de la siguiente forma:

```
Toast.makeText(getApplicationContext(), "Se borró la cámara con ID: " +
textoId.getText().toString(), Toast.LENGTH_LONG).show();
```

Con el *Toast* anterior, cada vez que eliminemos datos de la Base de Datos, nos aparecerá un mensaje indicándonos que se ha borrado la cámara con el ID que haya introducido el usuario.

Por último, tan solo nos queda configurar el botón de "UPDATE", que se encargará de actualizar registros que tengamos en la Base de Datos cuando el usuario introduzca el ID de cuya cámara quiera consultar parámetros.

El método que se encargará de llevar a cabo esta acción no es otro que *update()*.

Para este último botón necesitaremos tener nuestra Base de Datos en modo escritura también (en modo *Writable*), cosa que ya tenemos muy clara como se hace:

```
SQLiteDatabase db = helper.getWritableDatabase();
```

Ya que cuando actualicemos los datos escribiremos en la base de datos, haremos uso de la instrucción *values.put()* como ya hicimos al configurar el botón de "ADD".

Por lo tanto, dentro de esta instrucción debemos indicar en qué columnas vamos a introducir información (en todas menos la columna 1 que corresponde al criterio) y adjuntar la información que tendremos en los cuadros de entrada de texto y switches.

Para ello, el proceso es el mismo que a la hora de añadir una nueva cámara que ya vimos:

```
ContentValues values = new ContentValues();
values.put(Estructura_BBDD.NOMBRE_COLUMNA2, textoModel.getText().toString());
values.put(Estructura_BBDD.NOMBRE_COLUMNA3, textoIp.getText().toString());
values.put(Estructura_BBDD.NOMBRE_COLUMNA4, textoDescription.getText().toString());

if (switchPower.isChecked()){
   values.put(Estructura_BBDD.NOMBRE_COLUMNA5, "true");
} else {
   values.put(Estructura_BBDD.NOMBRE_COLUMNA5, "false");
}
```

De nuevo debemos indicar aquí también cuál va a ser nuestro criterio, que como ya sabemos es "Camera ID", dato almacenado en la columna 1 de nuestra tabla.

Por lo tanto, la indicación del criterio sería la siguiente:

```
String selection = Estructura_BBDD.NOMBRE_COLUMNA1 + " LIKE ?";
```

De la misma forma que para los botones anteriores, creamos el *array* donde indicamos el lugar de donde viene el criterio. Lo indicamos diciéndole que el criterio viene del *editText* "textoId" como ya bien sabemos:

```
String[] selectionArgs = {textoId.getText().toString()};
```

Ahora vamos con la instrucción clave, donde se encuentra el método *update()*, encargado de llevar a cabo la actualización. Mostraré primero el código y después lo comentaremos:

```
int count = db.update(
      Estructura_BBDD.TABLE_NAME, // TABLE_NAME está en Estructura_BBDD
      values,
      selection,
      selectionArgs);
```

Los argumentos que necesita son:

- El nombre de la tabla, que como sabemos se encuentra definido en la clase *Estructura_BBDD*.
- Los *values* a insertar, que los definimos anteriormente.
- El parámetro *selection*, que sabemos que es el criterio que ya definimos.
- Y, por último, *selectionArgs*, que es la indicación de dónde procede el criterio, que también está ya definido.

Ahora tan solo nos faltaría introducir el mensaje de *feedback* que hemos incluido en todos los botones para que se nos indique la acción que acabamos de realizar. En este último caso, quedaría de la siguiente forma:

```
Toast.makeText(getApplicationContext(), "Se actualizó la cámara con ID: " +
textoId.getText().toString(), Toast.LENGTH_LONG).show();
```

Con esto tendríamos terminado el último de los botones que conforman esta actividad (*MainActivity.java*). No hace falta decir que esta actividad es la más complicada y extensa de todas, debido principalmente por las dificultado que puede llegar a causarnos el trabajo con las Bases de Datos.

9.4 View Activity

Como ya hemos comentado anteriormente, todos los ficheros de código fuente comienzan con la importación de las librerías que más tarde usaremos en la programación de la actividad. En este caso, debido a que vamos a hacer uso de *WebView*, necesitaremos importar estos tres paquetes:

```
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;
```

A continuación, debemos definir nuestro objeto *webView*, que va a ser de tipo *private*:

```
private WebView webView;
```

También debemos recuperar el valor de la dirección IP correspondiente a la cámara encargada de invocar esta actividad que ha sido seleccionada en la actividad anterior. Esto lo hacemos por medio de los *bundles*.

```
Bundle bundle = getIntent().getExtras();
direccion = bundle.getString("laIP");
```

Nosotros queremos que Android emplee su motor de navegación dentro de la propia aplicación, es decir, sin necesidad de depender de ningún navegador externo. Esto es mucho más profesional y útil que si, por ejemplo, Android emplease el navegador por defecto para cargar la URL especificada. Para hacer esto, debemos incluir estas dos líneas de código:

```
webView = (WebView) findViewById(R.id.webView);
webView.setWebViewClient(new WebViewClient());
```

Tras esto, tan solo nos queda indicar qué web queremos visitar, que en nuestro caso serán las direcciones IP de las cámaras cuyo contenido queramos visualizar. El ejemplo que muestro a continuación nos conectaría a la página principal de Google:

```
webView.loadUrl("https://www.google.es");
```

Igual que antes, si dentro de los paréntesis donde se introduce la dirección a visitar introducimos nuestro parámetro “dirección”, viajaremos a la IP correspondiente a la cámara seleccionada:

```
webView.loadUrl(direccion);
```

Algún extra que podemos añadir a nuestra actividad puede ser que cuando pulsemos en el botón "back" del terminal, se vuelva a la página anterior. Para la realización de esta tarea, bastará con el siguiente código:

```
public void onBackPressed () {
   if(webView.canGoBack()){
      webView.goBack();
   } else {
      super.onBackPressed();
   }
}
```

Tampoco debemos olvidarnos de añadir a nuestro código la función que nos permite habilitar JavaScript para asegurarnos que ninguna web cargue mal. Para ello, tan solo debemos añadir lo siguiente:

```
WebSettings webSettings = webView.getSettings();
webSettings.setJavaScriptEnabled(true);
```

Con esto, tendríamos completamente vista la última actividad, y con ello todo el código fuente de las diferentes actividades que conforman la aplicación.