ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UN VIDEO PORTERO BASADO EN RASPBERRY PI

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

JOSÉ GABRIEL SANDOVAL PÉREZ

DIRECTOR: M.Sc. CARLOS EGAS ACOSTA

Quito, enero 2021

AVAL

Certifico que el presente trabajo fue desarrollado por José Gabriel Sandoval Pérez, bajo mi supervisión.

_____

M.Sc. CARLOS EGAS ACOSTA
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo José Gabriel Sandoval Pérez, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

_____

JOSÉ GABRIEL SANDOVAL PÉREZ

AGRADECIMIENTO

Agradezco a todas las personas que día a día me han apoyado y deseado siempre lo mejor en mi vida profesional.

ÍNDICE DE CONTENIDO

RESUMEN

El prototipo de Video Portero diseñado en este proyecto de titulación, tiene la finalidad de facilitar la comunicación entre los usuarios residentes de un domicilio (sea una casa, condominio o edificio residencial) y la persona que se encuentre fuera de la residencia, esto dando uso a la capacidad de procesamiento y facilidad de uso de la Raspberry Pi a la que se le conectan los componentes necesarios para poder digitar el código del domicilio deseado, así como reproducir y capturar audio y video digitales. De igual forma haciendo uso de protocolos estandarizados se logra una comunicación inalámbrica (WLAN) en una red LAN entre el Video Portero y un dispositivo con SO Android en el que deberá estar instalada la aplicación (desarrollada en este proyecto de titulación). Ya que el dispositivo Android será el receptor de las llamadas, permite reducir el costo de hardware, que caso contrario aumentaría dependiendo del número de residencias en el condominio o edificio, permitiendo tener un gran número de residencias ocupando el mismo Video Portero, así como un gran número de usuarios por residencia.

PALABRAS CLAVE: video portero, Raspberry Pi, LAN, WLAN, SO Android, IU (Interfaz de Usuario)

ABSTRACT

The Video Door phone prototype designed in this titling project, has the purpose of facilitating communication between the resident users at home (be it at its own house, condominium or residential building) and the person outside the residence, this giving use to the processing capacity and the ease use of the Raspberry Pi, to which the necessary components are connected to be able to type the code of the desired address, as well as reproducing and capturing digital audio and video. In the same way, using standardized protocols a wireless communication (WLAN) is achieved in a LAN network between the Video Door phone and a device with Android OS in which the application must be installed (developed in this titling project). Since the Android device will be the receiver of the calls, it allows to reduce the cost of hardware, which otherwise would increase, depending on the number of residences in the condominium or building, allowing to have a large number of residences occupying the same Video Door phone, and also a large number of users per residence.

KEYWORDS: Video Door phone, Raspberry Pi, LAN, WLAN, O.S. Android, UI (User Interface)

1. INTRODUCCIÓN

En la actualidad el uso de los celulares inteligentes ha crecido notoriamente en Ecuador y el mundo. Gracias al avance tecnológico que tienen estos dispositivos y su alta capacidad de procesamiento, es posible desarrollar Aplicaciones que permiten comunicarnos, distraernos y ayudarnos en nuestras tareas cotidianas. Por esta razón el desarrollo de este proyecto de titulación tiene como finalidad darle un uso más a los celulares inteligentes, específicamente a dispositivos con S.O. Android, ya que el desarrollo de Aplicaciones para este software libre, es menos complicado y las herramientas (Android Studio) para el desarrollo de estas no tienen costo para ocupar todas sus funciones.

Por otro lado, el desarrollo de micro-computadoras como Raspberry Pi ha permitido el acceso al poder computacional a personas de diferentes rangos económicos, esto gracias a la fundación “Raspberry Pi Foundation” que han diseñado estos dispositivos sin finalidad de lucrar de estos, si no, de dar acceso a esta tecnología a cualquier persona que lo desee. Estas micro-computadoras permiten tener acceso al software e interacción con el hardware, lo que hace de esta una buena herramienta para realizar proyectos y experimentar con estos.

Gracias a estos dos dispositivos se pueden desarrollar nuevos proyectos en diferentes áreas, en este caso el desarrollo de un Video Portero que permitirá a los residentes de un edificio tener acceso a este mediante un dispositivo Android registrándolo como usuario autorizado con nombre de usuario y contraseñas correspondientes, y de este modo poder recibir notificaciones realizadas desde el Video Portero al respectivo grupo de usuarios de un hogar. Esto facilita a los residentes la comunicación e identificación de la persona que se encuentra en la puerta de calle y de este modo saber si es necesario abrirle el seguro electromecánico de la puerta de entrada o simplemente terminar la comunicación cuando le parezca pertinente. De este modo aumenta la seguridad de la residencia, ya que no necesita acercarse ni abrir la puerta para identificar quién se encuentra del otro lado.

1.1 OBJETIVOS

Diseñar e implementar un Video Portero, el cual podrá ser contestado o monitoreado mediante cualquier dispositivo que opere con un S.O. Android con la Aplicación desarrollada y tenga los datos de acceso correspondientes.

Los objetivos específicos de este Proyecto Técnico son:

- Describir las características requeridas por parte de la Raspberry Pi y el dispositivo Android para la comunicación entre estos dispositivos.

- Diseñar e implementar el hardware que mantendrá el video-portero.

- Diseñar el software que se implementará en la Raspberry Pi y el de la aplicación a instalarse en el dispositivo Android.

- Comprobar el funcionamiento individual de los dispositivos y la interacción de los mismos después de realizar las configuraciones iniciales en la Raspberry Pi y en el dispositivo Android.

1.2 ALCANCE

En la culminación de este trabajo, se presentará un producto final demostrable, es decir que se podrá observar un producto listo para la instalación en una vivienda. Con la única necesidad de realizar configuraciones iniciales como son: la conectividad a la red WLAN del Video Portero, configuración de una dirección IP fija en el mismo, instalación de la aplicación (desarrollada en este proyecto) en un dispositivo Android, y conexión correcta al seguro electromecánico, que permitirá que se abra la puerta cuando el usuario con acceso lo desee. Configuraciones que para la demostración del funcionamiento del prototipo ya estarán realizadas.

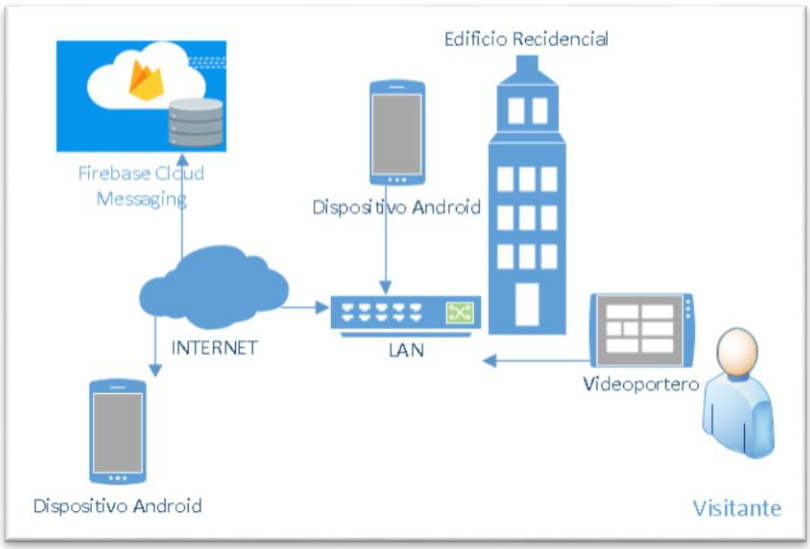

Figura 1.1. Esquema de funcionamiento de Video Portero

El principal escenario de operación de este prototipo es la red LAN del hogar, esto debido a que se ocupan direcciones IP privadas en estos y la mayoría de ISP otorgan direcciones

IP públicas dinámicas en sus planes de internet para el hogar, lo que no haría visible al Video Portero fuera de la red LAN doméstica. Debido a esto, la transmisión de audio y video se mantendrán únicamente en la red LAN. Por otro lado, para poder realizar las notificaciones desde el Video Portero al dispositivo Android, se ocupará un servicio gratuito multiplataforma que otorga la empresa Google llamado 'Firebase Cloud Messaging'. A través de este, es posible enviar mensajes de notificación desde un servidor a un dispositivo celular sin importar la red en la que se encuentre [1]. Por esta razón será necesario mantener el Video Portero conectado a una red LAN con acceso a internet, de igual forma los dispositivos Android deberán tener acceso a internet para que puedan recibir las notificaciones.

Vale la pena recalcar que únicamente los dispositivos Android que se encuentren conectados en la red LAN donde se encuentre instalado el Video Portero podrán tener acceso a la comunicación de audio y video. De la misma forma tendrá posibilidad de abrir el seguro electromecánico. Pero cualquier dispositivo con la aplicación, registro correspondiente y que tenga acceso a internet podrá recibir las notificaciones cuando alguien timbre a su departamento.

1.3 MARCO TEÓRICO

1.3.1 RASPBERRY PI

Es una micro-computadora económica (gracias a la fundación Raspberry Pi Foundation que ha trabajado para dar accesibilidad al poder computacional a personas alrededor de todo el mundo [2]), la cual trabaja con el sistema operativo Raspberry Pi OS (que es un software basado en el software libre Debian por lo que también lo convierte en software libre), u otros sistemas operativos que han sido desarrollados exclusivamente para estos dispositivos. Este dispositivo es por lo general usado para aprendizaje de programación y proyectos de investigación e innovación gracias a su económico precio y fácil conectividad a puertos y pines de comunicación. Permite el uso de lenguajes de programación como Python, C/C++, Java entre otros.

Para poder ocupar esta micro-computadora es necesario tener los siguientes elementos [3]:

- Tarjeta memoria Micro SD con el SO a ejecutar.
- Fuente de energía de 5 [V] y 2-2.5 [Amp] con conexión micro-USB.

- Pantalla con puerto de entrada HDMI.
- Cable HDMI.
- Teclado y mouse con conexiones USB.

Los últimos tres elementos son necesarios únicamente para las configuraciones iniciales de la Raspberry Pi ya que después de conectarla a la red es posible manipular al dispositivo mediante protocolo SSH [4] o un escritorio remoto como VNC que es ocupado en Raspberry Pi [5] (que de igual manera deberán ser configurados con anterioridad para su funcionamiento).

1.3.1.1 Modelos de Raspberry Pi y sus características

Debido a que la fundación Raspberry Pi se mantiene innovando en sus productos (principalmente en el nivel de procesamiento y en la conectividad que estos tienen), se puede escoger diferentes modelos de acuerdo a las características que se requieran en el proyecto. Así mismo las diferentes versiones mantienen varias características comunes que permiten la compatibilidad de accesorios útiles en el momento de realizar proyectos con estos dispositivos.

1.3.1.1.1 Características comunes de los diferentes modelos de Raspberry Pi

En la actualidad los modelos que se encuentran en el mercado son Raspberry Pi 1 Modelo A+, B+, Raspberry Pi 2 Modelo B, Raspberry Pi 3 Modelo A+, B, B+, Raspberry Pi Zero y Raspberry Pi Zero W, por lo que las siguientes características serán válidas para estos modelos (la última versión que la fundación sacó al mercado es la Raspberry Pi 4B, el cual no es considerado en este proyecto por el tiempo que se requiere para conseguir una). En el ANEXO A se podrán observar las características individuales de cada uno de estos modelos.

- **Pines GPIO:** una de las principales características que ha mantenido en los diferentes modelos que se encuentran en el mercado, es la cantidad de pines GPIO (General Purpose Input/Output, Entrada/Salida de Propósitos General), en los primeros dos modelos de Raspberry Pi, los Modelos A y B fueron fabricados con 26 pines GPIO pero a partir de los Modelos A+ y B+, que, aparte de tener estos 26 pines GPIO (que se mantienen en el mismo orden para mantener compatibilidad con accesorios ya existentes para modelos previos) se incluyeron 14 pines GPIO dando un total de 40 pines GPIO, esta configuración de pines GPIO que se observa en la Figura 1.2. se

mantiene hasta los últimos modelos en la actualidad que salieron al mercado en el 2018 los Modelos 3B+ y 3A+, al igual que su último Modelo 4B que salió al mercado en el 2020.

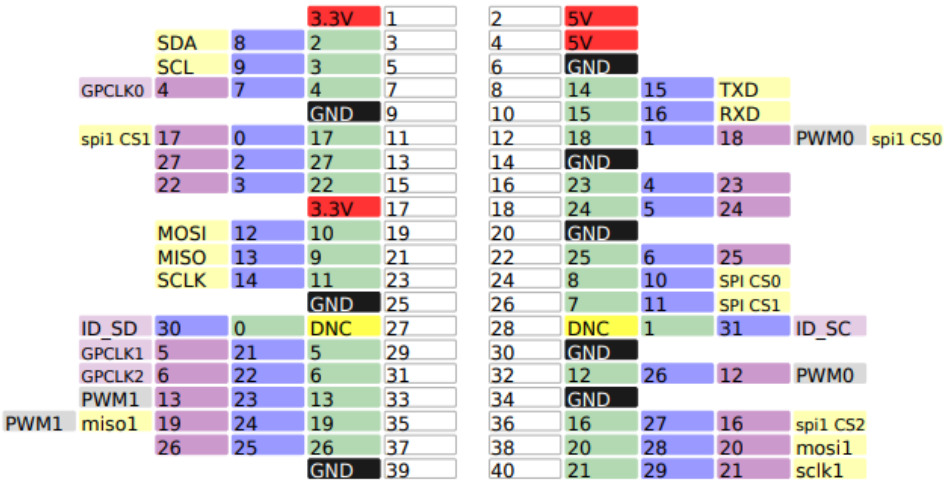

Figura 1.2. Disposición de Pines GPIO en Raspberry Pi [6]

- **Memoria ROM:** Otra característica principal es el uso de tarjetas Micro SD la misma que contendrá el Sistema Operativo de la Raspberry, a diferencia de los primeros modelos que usaban tarjetas SD.

- **Unidad de Procesamiento Grafico (GPU):** Todos estos modelos han mantenido la tarjeta de video Dual Core VideoCore IV® Multimedia Co-Processor.

- **Puerto USB:** El principal conector que se ha mantenido en los diferentes modelos de la Raspberry Pi es el puerto USB 2.0, permitiendo la conexión de periféricos como teclado, mouse o tarjetas de memoria USB. Los distintos modelos tienen diferente cantidad de puertos USB, y otra diferencia es con los modelos Raspberry Pi Zero y Zero W, que tienen un puerto Micro USB OTG (On-The-Go) para reducir el tamaño de la tarjeta considerablemente.

- **Puerto HDMI:** Otro conector que contienen los dispositivos es un puerto HDMI que permite observar el Shell determinado por el S.O., sea este CLI (Command-Line Interface) o GUI (Graphical User Interface). De igual forma para poder reducir el tamaño en las Raspberry Pi Zero y Zero W se ocupó un puerto Mini-HDMI para estos modelos, así como la Raspberry Pi 4B que posee 2 puertos Mini-HDMI, permitiendo al usuario conectar 2 pantallas a este. A estos se puede conectar un adaptador y de esta manera conectar un cable HDMI ordinario.

- **Fuente de energía:** Todos los modelos de Raspberry se alimenta de un voltaje de 5[V], con alguna variación en la corriente que se encuentran entre 2 y 2.5[A], cuya fuente de energía puede ser conectada mediante un puerto Micro-USB que sirve únicamente para este propósito.

1.3.1.1.2 Modelos de Raspberry Pi considerados para el proyecto

Para este proyecto de titulación se consideraron dos opciones para el desarrollo del Video Portero, de acuerdo a sus características que se especificarán a continuación.

❖ **Raspberry Pi Zero W** [7]

La Raspberry Pi Zero W (cuya imagen podemos observar en la Figura 1.3) es la micro-computadora de menor tamaño creada por la fundación Raspberry Pi. Sus características son las siguientes:

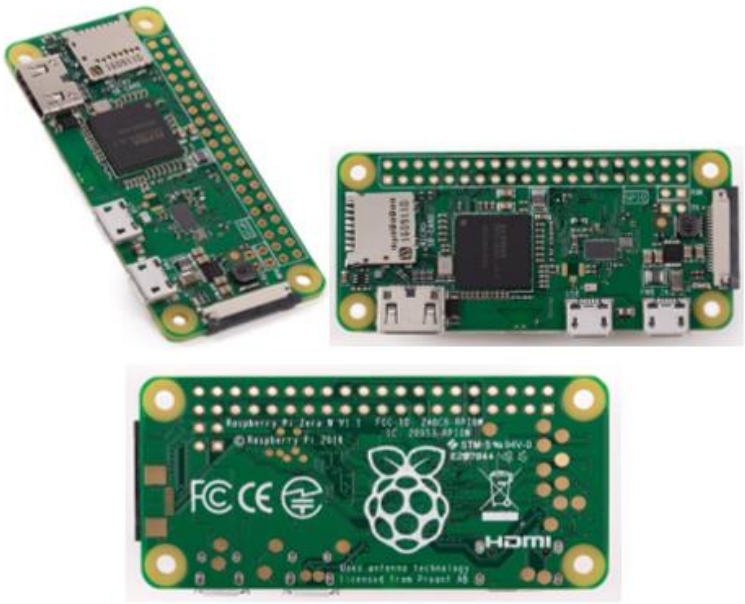

Figura 1.3. Imagen de Raspberry Pi Zero W [7]

- Procesador Broadcom BCM2835 que mantiene una frecuencia de procesamiento de 1 GHz con bajo consumo de energía.

- Memoria RAM de 512 MB para almacenar los programas en ejecución mientras el Video Portero se encuentre encendido.

- 1 puerto Micro-USB OTG que permite conectar dispositivos USB mediante un adaptador y así poder inicializar la micro-computadora ocupando teclado o mouse.

- Este modelo tiene una tarjeta WLAN con una antena que permite el uso de las frecuencias 2.4 GHz del protocolo IEEE 802.11 b/g/n y Bluetooth 4.1 así como BLE (Bluetooth Low Energy).

- El conector de cámara que contiene es de 22 pines MIPI CSI-2 (Camera Serial Interface) compatible con cámaras con conector de 15 pines mediante un cable flexible adecuado.

- Para conectar una interfaz visual a parte del puerto HDMI también tiene una salida analógica de video compuesto (PAL y NTSC) en un par de pines aparte de los pines GPIO como se observa en la Figura 1.4 donde se puede apreciar de igual forma pines de encendido o reset de la Raspberry Pi.

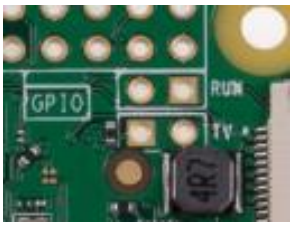

Figura 1.4. Pines adicionales para encendido o reset y para transmisión de video compuesto [7].

- Dimensiones 65x30x5 mm.

Tomando en cuenta estas características, las principales razones por la que se escogió esta micro-computadora para desarrollar las pruebas del Video Portero en este proyecto, es su económico precio, así como su conectividad inalámbrica y reducido tamaño que permite disminuir el tamaño del chasis en el que se encontrarán los diferentes circuitos acoplados.

❖ **Raspberry Pi 3 Model B+** [8]

La imagen de esta micro computadora se la puede observar en la Figura 1.5., esta es una de las últimas versiones desarrolladas por la fundación Raspberry Pi con mayor velocidad de procesamiento y conectividad inalámbrica. Las características de esta microcomputadora son las siguientes:

- Procesador Broadcom BCM2837 que mantiene una frecuencia de procesamiento de 1.4 GHz con bajo consumo de energía.

- Memoria RAM de 1 GB LPDDR2 SDRAM

- Puerto Ethernet

- 4 puerto USB
- Este modelo tiene una tarjeta WLAN y antena que permite el uso del protocolo IEEE 802.11 b/g/n/ac y Bluetooth 4.2 así como BLE (Bluetooth Low Energy).
- Conector de display de 15 pines DSI (Display Serial Interface).
- El conector de cámara que contiene es de 15 pines MIPI CSI-2 (Camera Serial Interface)
- A parte del puerto HDMI también tiene una salida analógica de audio y video con un Jack de 3.5 mm, con audio estéreo y video compuesto (PAL y NTSC).
- Dimensiones 85 x 56 x 17 mm

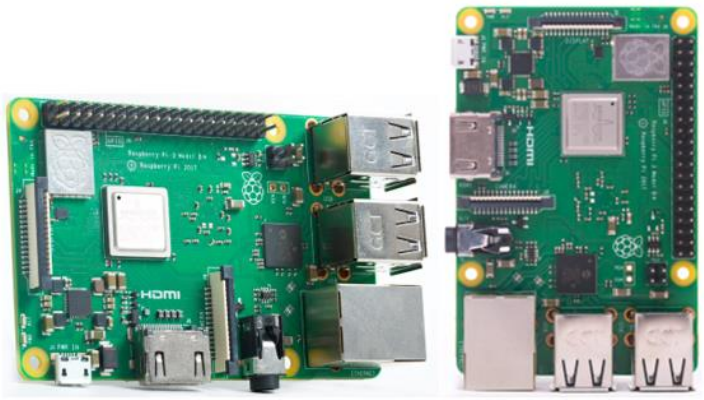

Figura 1.5. Imagen de Raspberry Pi 3B+ [8]

A diferencia del modelo descrito anteriormente, la Raspberry Pi 3B+ es la versión con mayor capacidad de procesamiento y mayor memoria RAM lo que permite la ejecución rápida de los programas necesarios para el funcionamiento del Video Portero. Las desventajas que tiene este modelo es su alto costo y mayor tamaño en comparación a la Raspberry Pi Zero W.

1.3.2 ACCESORIOS DE RASPBERRY PI

Gracias a distintos desarrolladores que desean aportar al avance de Raspberry Pi dándole mayor funcionalidad con accesorios conectados a diferentes puertos como los pines GPIO, USB u otros, junto con el software que controlan estos dispositivos, es posible desarrollar proyectos de forma más fáciles sin la necesidad de diseñar el hardware, permitiendo

ocupar las librerías requeridas para desarrollar el software, controlando el dispositivo a gusto.

1.3.2.1 Cámara

El diseño de la Raspberry Pi desde el principio contemplo el uso de cámaras y displays en sus dispositivos, por esta razón todos sus modelos tienen una entrada exclusiva para el uso de una cámara, así como la inclusión de librerías para el uso de esta, mediante líneas de instrucciones en un terminal de comandos, o mediante el desarrollo de software que permita el uso de la cámara del modo el programador lo desee. De esta forma para este proyecto se consigue dos tipos de cámaras para poder probar y usar la mejor opción, teniendo como única diferencia el ángulo de captura de la lente, por lo que a continuación se observarán las características similares de estos dos modelos [9].

Características

Resolución estática	5Megapixeles
Modos de Video	1080 pixeles a 30 fps, 720 pixeles a 60 fps y 640×480 pixeles a 60/90 fps
Resolución del sensor	2592 × 1944 pixels
Área de sensor	3.76 × 2.74 mm
Tamaño de Pixel	1.4 µm × 1.4 µm

1.3.2.1.1 Modelos de cámaras

- **jt-zero-v2.0**

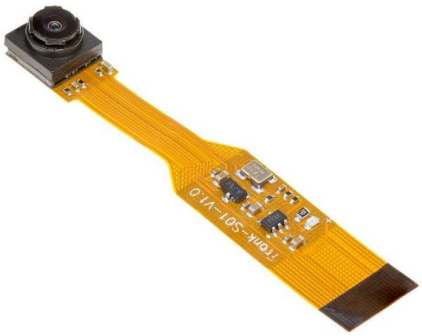

Figura 1.6. Cámara para Raspberry Pi Zero modelo Jt-zero-v2.0 [10]**.**

Este modelo a parte de las características especificadas anteriormente se debe mencionar que tiene un ángulo de captura pequeño, alrededor de 90 grados, es una cámara pequeña en comparación con el otro modelo escogido y es únicamente para usarla con la Raspberry Pi Zero W ya que no permite cambiar de cable flexible para adaptarlo a otros modelos.

❖ **Cámara angular**

Figura 1.7. Cámara angular para Raspberry Pi [11].

Por otro lado, la cámara angular adquirida, tiene una lente que permite una vista angular de 160 grados, es de mayor tamaño, pero gracias al ángulo de visión de esta permite observar un área más grande.

1.3.2.2 Módulo de audio

El módulo de audio obtenido tiene como nombre “ReSpeaker 2-Mics Pi HAT” (de la cual se puede observar un diagrama de este en la Figura 1.8), compatible con todos los modelos de Raspberry Pi. Tiene dos interfaces de entrada de audio (Micrófonos) y la posibilidad de conectar interfaces de salida como un parlante de 8 ohms en un conector de dos pines o audífonos en el conector Jack de audio 3.5 mm. A parte de estas características tenemos las siguientes [12]:

- Un botón programable.
- Un máximo de 48 KHz de frecuencia de muestreo.
- LEDs de colores rojo, verde y azul.
- Puerto mini USB-B únicamente para alimentación de energía.
- Conector para ocupar el I2C de la Raspberry Pi (que forma parte de los pines GPIO).

- Conector para ocupar pin GPIO 12.

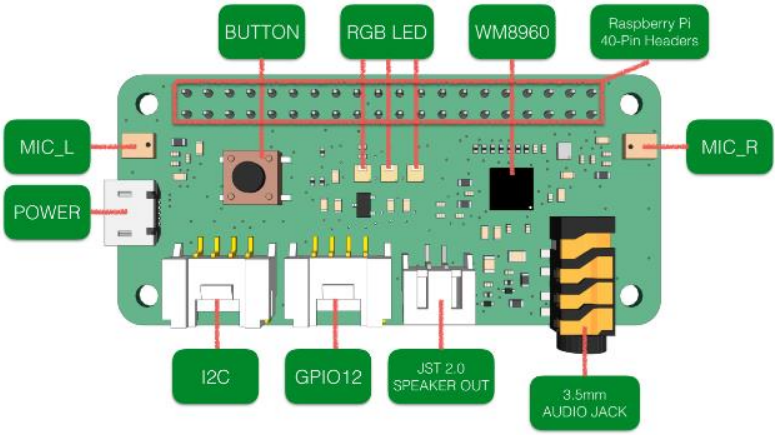

Figura 1.8. Módulo de audio para Raspberry Pi [12]

1.3.3 PYTHON

Python es un lenguaje de programación con licencia de código abierto denominada Python Software Foundation License [13], lo que permite el uso de este software para diversos propósitos sin necesidad de pagar regalías a la fundación que administra Python (Python Software Foundation), debido a esto y las grandes ventajas que tiene este lenguaje de programación se optó para usarlo en la programación de la Raspberry Pi, dando las funcionalidades requeridas para que el Video Portero actué de la manera deseada.

1.3.3.1 Características

A continuación, se describirán las características por la que se escogió a Python como el lenguaje de programación primario en la Raspberry Pi [14]:

- Eficiencia: Python es un lenguaje de programación eficiente ya que permite realizar un programa en menos líneas de código si se compara con el mismo programa realizado con otro lenguaje de programación.

- Sintaxis: La sintaxis que usa Python se asemeja mucho al Lenguaje Inglés lo que permite aprender este lenguaje de programación con mayor facilidad. Es una de las mayores diferencias con otros lenguajes de programación que permite tener menos errores de sintaxis al momento de ejecutar el programa.

- Comunidad Python: Gracias a la amplia comunidad colaborativa que tiene Python es posible solucionar con mayor rapidez algún problema, sea este insignificante o prioritario para el programa, la posibilidad que alguien en la comunidad haya tenido un

problema similar y haya logrado solucionarlo permite ahorrar tiempo y esfuerzo para poder continuar con el desarrollo del programa.

- Librerías: De igual forma gracias a la colaboración de la comunidad se ha extendido ampliamente la librería de Python permitiendo usar código abierto escrito para un propósito específico y poder obtener los resultados requeridos para desarrollar un programa más amplio, simplificando el desarrollo del mismo. Tomando esto en cuenta existen librerías de Python exclusivamente para el uso en la Raspberry Pi y el uso de diferentes accesorios como la cámara, así como el uso de módulos que se conectan a la Raspberry Pi mediante los pines GPIO y de esta forma poder ocupar estos productos que se encuentran en el mercado y mediante las librerías respectivas controlarlos de acuerdo a las necesidades.

- Multiplataforma: Python es un lenguaje multiplataforma, esto quiere decir que está desarrollada para funcionar en equipos con distintos sistemas operativos.

1.3.3.2 Librerías de Python ocupadas

Una parte importante de la programación en Python es el uso de las librerías que desarrollan y comparten programadores de todo el mundo, permitiendo el re-uso de código en forma de módulos y de este modo reducir el tiempo de programación para realizar ciertas actividades. Entre tantas librerías a continuación se listarán las ocupadas en este proyecto de titulación.

- ✓ time
- ✓ subprocess
- ✓ sqlite3
- ✓ socket
- ✓ Most.voip.api
- ✓ RPi.GPIO
- ✓ pyfcm

time: permite varias funciones relacionadas al tiempo, es posible obtener la hora de la región determinada, tener un calendario, ejecutar un contador, conversión de unidades de tiempo, entre otras funciones [15].

subprocess: permite el desarrollo de nuevos procesos, es decir ejecutar un programa en segundo plano, permitiendo la compilación de un programa a la par del programa original,

de igual forma se puede tener acceso a las entradas, salidas y errores de este nuevo programa en ejecución [16].

sqlite3: permite la creación de una base de datos, el acceso a esta para poder realizar modificaciones de la misma. Más detalles de SQLITE se verá más adelante ya que al ser el motor fundamental de la base de datos usada requiere mayor atención [17].

socket [18]**:** da acceso a la interfaz BSD socket [19], este permite el desarrollo de aplicaciones requieran compartir datos entre programas, sean estos ejecutados en el mismo dispositivo o un dispositivo diferentes en la misma red, esto sin la necesidad de acceder o tener un conocimiento completo del resto de capas que trabajan en los protocolos de red. Gracias a este módulo es posible enviar, y recibir datos a través de protocolos como RTP y UTP.

most.voip.api [20]: es una librería rápida y ligera que permite la creación y manejo de sesiones VOIP, es decir que permite la creación u registro de una cuenta SIP en un servidor remoto SIP como lo es Asterisk. Esta se basa en otra librería llamada PJSIP 2.9, que es una librería gratuita de comunicaciones multimedia escrita en el lenguaje de programación C. Implementa protocolos basados en estándares como SIP [21] razón por la cual es necesario instalar para el uso de Most.voip.api.

RPi.GPIO [22]: mediante esta librería es posible el uso de los pines GPIO de las Raspberry Pi, permitiendo colocar a los puertos como entrada o salida para la interacción con dispositivos externos.

Pyfcm [23]: esta librería sirve para desarrollar la interacción con Firebase Cloud Messaging que permite él envió de notificaciones, mensajes y otro tipo de interacción con un dispositivo que contenga una aplicación desarrollada con este servicio. Más adelante se especificarán más detalles de esta plataforma que permitirá enviar notificaciones.

1.3.4 SQLITE

SQLite es una librería realizada en lenguaje C que implementa un motor de base de datos SQL (Structured Query Language- Lenguaje de consulta estructurada) de dominio público por lo que puede ser usada de cualquier forma sea comercial o privada [24]. Es usada para crear y administrar bases de datos, permitiendo leer y almacenar la información directamente en archivos de disco ordinarios. En el proyecto de titulación actual SQLite es usado mediante una librería API (Application Programming Interface- Interfaz de programación de aplicaciones) SQLite3 en Python que permite el uso de esta herramienta

(desarrollada en lenguaje C) mediante el lenguaje Python. Tomando esto en cuenta, las características de SQLite son las mismas, aunque se use el API y son las siguientes:

- Es un sistema autónomo [25]: quiere decir que requiere muy pocas dependencias para su funcionamiento. La librería completa de SQLite se encuentra encapsulada en una sola fuente de archivo que no necesita herramientas especiales para su ejecución.

- No requiere servidor [26]: a diferencia de otras bases de datos SQL que trabajan con el modelo Cliente/Servidor, la base de datos de SQLite opera directamente en el mismo equipo en el que es ejecutada la aplicación.

- No requiere configuración [27]: para el uso de la base de datos SQLite no requiere configuración de ningún tipo para que pueda ser usada, ni requiere ninguna acción si por alguna razón el sistema falla o exista una suspensión de energía.

- El formato de la base de datos es multi-plataforma [24]: esto permite copiar y usar el archivo de base de datos entre sistemas de 32-bit y 64-bit de ser necesario, sin tener ningún tipo de inconveniente.

1.3.5 ASTERISK

Asterisk es un framework de software libre, desarrollado principalmente para computadoras con S.O. LINUX, es un motor para el desarrollo de aplicaciones de comunicación más usado en el mundo, con el que se puede implementar video llamadas, VOIP, correo, entre otras aplicaciones [28]. Este será implementado en la Raspberry Pi y permitirá el establecimiento de la comunicación de voz entre el celular Android y la Raspberry Pi colocada como Video Portero.

1.3.6 FIREBASE CLOUD MESSAGING

Firebase Cloud Messaging es una de los servicios que puede dar la plataforma Firebase, creada por Google para ayudar al desarrollo de aplicaciones, dando soporte de servidores con variedad de opciones para el manejo de datos, almacenamiento, monitoreo, entre otras funciones [1]. Este ofrece un servicio gratuito para un número limitado de aplicaciones que registre el desarrollador. Mediante el uso de Firebase Cloud Messaging es posible enviar mensajes y notificaciones a dispositivos individuales o a un grupo de dispositivos. Para poder realizar el envío de estos mensajes, en primer lugar es necesario registrar a los usuarios con la aplicación desarrollada, después desarrollar en un servidor el código que divida a los dispositivos de acuerdo a la identificación, el grupo o un tema y que indique a la plataforma de Firebase Cloud Messaging que realice el envío de los mensajes [1]. La

gran ventaja es que se puede realizar el envío a una gran cantidad de dispositivos de ser necesario y puede ser escalable sin la necesidad de añadir servidores al momento de tener más usuarios. De igual forma, la plataforma de Firebase ayuda a tener un registro de usuarios que se encuentran ocupando la aplicación y es posible tener acceso a las acciones de esta plataforma mediante la programación en un servidor con el lenguaje de programación deseado. Otro dato importante de esta plataforma es que no es exclusivamente para Android, esta también puede ser usada en otros sistemas operativos de celulares como IOS o Microsoft.

1.3.7 ANDROID

Android es un S.O. desarrollado en la actualidad por Google, para dispositivos celulares, Tablets y otros dispositivos como relojes y televisores, permitiendo el uso de estos dispositivos denominados inteligentes por su poder de procesamiento, conectividad y uso de aplicaciones nativas que permite dar diferente funcionalidad a los teléfonos móviles y dispositivos Android. En la actualidad existen varias versiones de este sistema operativo, facilitando cada vez más el desarrollo de diferentes aplicaciones ya que permite usar nuevas funciones desarrolladas en cada actualización.

1.3.7.1 Arquitectura de Android [29]

Para poder desarrollar aplicaciones para dispositivos Android, es necesario conocer la arquitectura básica de este y así poder conocer el funcionamiento básico del dispositivo. Por lo que a continuación se observará brevemente cada una de las capas de la pila de software de Android que se puede observar en la Figura 1.9.

- ✓ Kernel Linux [30]

La base primaria que ocupa el S.O. Android es el Kernel de Linux que permite el manejo de los distintos recursos de Hardware como la memoria, puertos de entrada y salida, conectividad a la Red, entre otros, como se puede observar en la Figura 1.9.

- ✓ Capa de Abstracción de Hardware

HAL (por sus siglas en inglés Hardware Abstraction Layer), este consiste de varios módulos de librerías. Esta capa crea interfaces estándar entre el software y el hardware, por lo que al usar esta capa el software no interactúa directamente con el hardware si no que lo hace con la capa HAL [30].

- ✓ Tiempo de ejecución de Android

En inglés conocido como Android Runtime ART, permite la ejecución de varias máquinas virtuales mediante la ejecución de archivos DEX que es un formato de código de Bytes diseñado para Android y fue optimizado para el uso en dispositivos de memoria baja. Compila el código de Java en códigos de bytes DEX, permitiendo su ejecución en dispositivos Android. Previo al desarrollo de ART es decir antes de Android 5.0, se ocupaba Dalvik que al igual que el ART desarrollada para dispositivos de memoria baja [30].

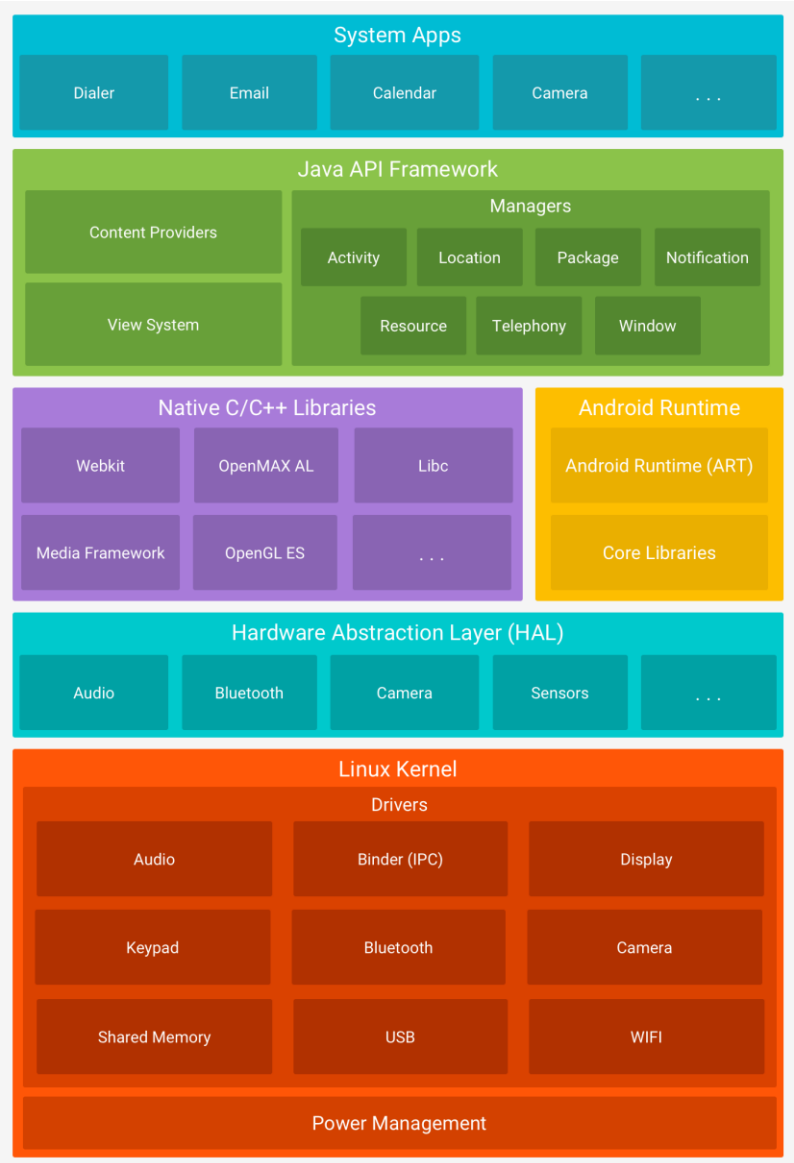

Figura 1.9. Pila de software de Android [30]**.**

✓ Bibliotecas C/C++ Nativas

Android también se basa en código nativo que requieren bibliotecas nativas desarrolladas en C/C++. Gracias a la API del framework de Java que proporciona la plataforma Android

es posible el uso de algunas de estas bibliotecas nativas en las Aplicaciones a desarrollar [30].

✓ Framework de la Java API

El conjunto de funciones que se encuentran en un teléfono con S.O. Android, se encuentran disponibles para reutilizarse mediante API escritas en lenguaje Java. Por esta razón el uso del lenguaje Java es fundamental para el desarrollo de aplicaciones para celulares Android [30].

✓ APPs del sistema

En celulares con S.O. Android, existen aplicaciones básicas como correo, mensajería SMS, calendario, contactos entre otras. A estas APPs se pude tener acceso desde otras aplicaciones debido a que no son aplicaciones con algún estado especial, gracias a esto los desarrolladores pueden crear APPs que ocupen las funciones de estas aplicaciones del sistema como por ejemplo realizar una llamada sin la necesidad de escribir todo el código correspondiente para hacerlo, por lo contrario, únicamente se requiere llamar a la aplicación correspondiente con los datos que requiera. Existen excepciones, como la aplicación de configuración del sistema a la que no se puede reutilizar del mismo modo, pero en su gran mayoría es posible tener este acceso [30].

1.3.7.2 Características

Las principales características de Android que serán útiles para este proyecto de titulación son:

- **Notificaciones [31]:** permite que el usuario de un celular tenga conocimiento de una actividad específica de la aplicación que produce la notificación mediante una alerta, de este modo es posible informar al usuario y este podrá tomar la acción adecuada para el caso.

- **Permisos del sistema [32]:** los permisos de Android dan acceso a elementos privados del celular del usuario como a la cámara, el micrófono, contactos, ubicación, acceso a la red etc. Gracias a esto para realizar una aplicación que requiera ocupar un elemento de estos, es necesario solicitar el permiso y de este modo el usuario sabrá que la aplicación instalada no ocupará más información de la debida.

- **Actividades [33]:** una actividad en Android es el componente de una aplicación que tiene una pantalla con la cual el usuario puede interactuar, en esta se encontrarán botones, imágenes, texto y otros componentes dependiendo de la aplicación a

desarrollar, estas actividades tienen ciclos de vida, por lo que es necesario conocer cómo crear, pausar y cerrar una actividad dentro de la aplicación.

1.3.8 PROTOCOLOS ESTANDAR UTILIZADOS

El uso de protocolos estandar son importantes ya que permite comunicar a dispositivos desarrollados por distintos fabricantes y aun así mantener una comunicación clara. Por lo que a continuación se realizará un pequeño resumen de los protocolos usados.

- **Comunicación TCP:** protocolo de control de transmisión (o más conocido en inglés como Transmission Control Protocol) es un protocolo de capa de transporte (según el protocolo de arquitectura de red TCP/IP) orientado a la conexión, este asegura que los datos enviados lleguen sin errores y en el orden correcto al receptor.

- **HTTP:** Hipertext transfer Protocol es un protocolo de la capa aplicación y es el protocolo principal de la World Wide Web, la cual puede ser usada en cualquier aplicación de cliente/servidor, al contrario de lo que podría dar la impresión el nombre, este protocolo puede transferir más que únicamente hipertexto, dando paso así a texto plano, audio, imágenes o cualquier tipo de información accesible a la red [34].

- **SIP:** son las siglas de Session Initiation Protocol, este es un protocolo de control de la capa aplicación para establecer, modificar y terminar sesiones de comunicación de tiempo real entre los participantes sobre una red de datos IP [35]. Gracias a este protocolo es posible la comunicación de voz sobre la red IP o más conocido como VOIP (Voice Over IP). Ocupando este protocolo en el celular Android y en la Raspberry Pi es posible la comunicación, teniendo como intermedio un servidor de VOIP como lo es Asterisk, que establece la comunicación entre estos dos dispositivos.

2. METODOLOGÍA

Este proyecto se desarrolló principalmente mediante las metodologías de investigación Exploratorio y Aplicativo. Mediante un análisis exploratorio se pudo tener una perspectiva amplia del funcionamiento de varios productos similares al que se desarrolló en este proyecto y así mismo el aprender la programación básica en Java para poder desarrollar una aplicación con la ayuda del entorno de desarrollo ‘Android Studio’, al igual que permitió la obtención del conocimiento necesario para el desarrollo de código en lenguaje de programación Python y la posibilidad de interacción entre dos dispositivos con lenguajes de programación distintos. Mientras que el análisis aplicativo permitió el diseño y la

implementación del prototipo gracias a los conocimientos adquiridos en clases y autoeducación.

Se realizó un análisis de las posibles características que deberá tener un Video Portero para que pueda ser usado fácilmente por cualquier persona, tenga o no un conocimiento tecnológico, con el único requisito de tener un dispositivo celular con S.O. Android. De este modo se obtuvieron las características que se va a intentar dar al Video Portero usando una microcomputadora Raspberry Pi.

Una vez teniendo en claro las características a querer integrar en este Video Portero se analizó las distintas versiones de las microcomputadoras Raspberry Pi y de este modo poder conocer cuál sería la microcomputadora más adecuada y económica para el desarrollo de este prototipo y del mismo modo se investigó que tipos de módulos existen en el mercado, que sean necesarios para el desarrollo de este prototipo. Para continuar con la implementación de hardware, una vez conociendo los módulos que serán conectados al microcomputador, se procede a diseñar un módulo adicional que cumplirá con el resto de funciones necesarias para la implementación del Video Portero.

Ya con el hardware diseñado y ensamblado, se procede a diseñar el software que permitirá el control adecuado de la Raspberry Pi para que tenga la funcionalidad deseada, instalando a la par las librerías necesarias para la correcta compilación y ejecución de los programas.

Del mismo modo se trabaja a la par con el diseño de la aplicación que será instalada en el dispositivo Android, para determinar el tipo de interacción que tendrán estos dispositivos. Una vez teniendo el diseño del software de los dispositivos claros, se procede a la creación del código de estos, implementando el conocimiento adquirido y consultando nuevas formas de poder obtener los resultados deseados de los programas.

Ya desarrollados los programas de los dos dispositivos, se proceden a realizar las pruebas respectivas y se corrigen los errores encontrados evitando modificar el funcionamiento de estos dos dispositivos y evitar que la ejecución del programa colapse por alguna incongruencia en el programa. Para proceder con las pruebas finales y observar el comportamiento del Video Portero y su interacción con el dispositivo Android.

2.1. CARACTERÍSTICAS PRINCIPALES Y SECUNDARIAS DEL VIDEO PORTERO

Para el desarrollo de este proyecto en primer lugar se especifican las características que tendrá este Video Portero, mediante la observación de distintos porteros electrónicos y

video porteros que existen en el mercado, por lo que las principales características que se considerarán para el diseño del prototipo realizado en este proyecto de titulación son las siguientes:

Transmisión de audio: la transmisión de audio es principal para el uso de un portero electrónico, ya que mediante la comunicación verbal se conoce la razón por la que la persona en la entrada se encuentra timbrando. De esta forma el receptor pueda decidir si abrir la puerta de ser necesario o caso contrario culminar con la comunicación. En la mayoría de los casos tienen sistemas de transmisión cableados sean digitales o analógicos. En el prototipo diseñado se realiza una transmisión de audio digital mediante una red LAN, red a la que se encuentra conectado el prototipo mediante una red WLAN manteniendo al Video Portero libre del cableado e instalaciones adicionales con la excepción de la conexión a la fuente de energía y la conexión al seguro electromecánico. El audio a ser transmitido será full-duplex, por lo que no será necesario el presionar algún botón para alternar entre transmisión y recepción, como suelen usarse dispositivos half-duplex. Uno de los principales retos de esto será mantener los retardos de transmisión al mínimo y que, de este modo, la comunicación de voz fluya con normalidad.

Transmisión de Video: del mismo modo que el audio, el video será digital y trasmitido a través de la red LAN, pero esta vez con una transmisión simplex, enviando los paquetes de imagen desde el Video Portero al dispositivo Android que lo solicite. Considerando que la visualización del video debe tener una resolución suficiente para poder identificar a una persona que se encuentre cerca de la cámara, pero sin consumir demasiado procesamiento y ancho de banda en la red, por lo que se considera reducir el consumo de procesamiento en el dispositivo activando la transmisión de video únicamente cuando el usuario lo considere pertinente.

Gestión de usuarios: la mayoría de porteros electrónicos o video porteros tiene un número limitado de receptores, usualmente uno por departamento en el caso de un edificio residencial y se encuentra ubicado en una zona específica de la vivienda (lo más común es en la sala, comedor o cocina). Para no poner una limitante en el número de usuarios y que todos los residentes que deseen puedan contestar independientemente de donde se encuentren en el hogar, el receptor será un dispositivo con S.O. Android en el que se encuentre instalada la aplicación desarrollada, permitiendo dar un número casi ilimitado de departamentos y de usuarios, teniendo únicamente como limitación la memoria del Video Portero para el almacenamiento de los datos de usuario en la base de datos.

Teclado para marcado: existen diferentes tipos de teclados que ocupan en los porteros electrónicos, usualmente un botón por cada departamento del inmueble, pero para no poner límite al número de residencias que estarán ocupando este dispositivo de acuerdo al número de pulsadores, se ocupa un teclado matricial donde se digita el número que se desee y presione LLAMAR para enviar la notificación a los usuarios de ese departamento. De este modo únicamente quedará en la configuración limitar el número de departamentos y no será necesario variar el número de botones del teclado.

2.2. IDENTIFICACIÓN DE LOS SISTEMAS

Para el desarrollo de este proyecto es necesario tener claro el funcionamiento que va a tener cada uno de los sistemas a desarrollar u ocupar, los mismos que se pueden observar en la Figura 2.1.

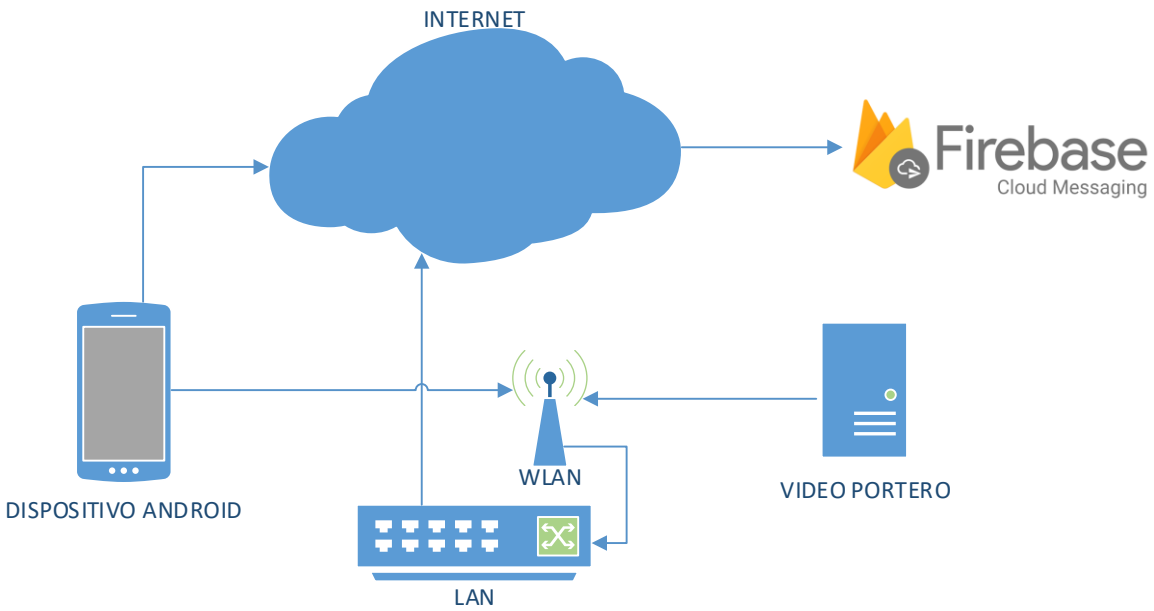

Figura 2.1. Gráfica de sistema en funcionamiento

Para esto es posible identificar 5 sistemas principales:

- Video Portero
- Dispositivo Android
- Red LAN
- Internet
- Firebase Cloud Messaging

2.2.1. VIDEO PORTERO

Está compuesto de hardware y software que permitirán la conectividad a la red LAN, para el envió de notificaciones a los dispositivos Android, asociados con el departamento requerido mediante un teclado en el Video Portero. Así mismo, permitirá el broadcast HTTP de Video digital capturado por una cámara instalada en este, adicional al audio digital de entrada y salida, teniendo una comunicación de audio full-dúplex.

El software que correrá en este dispositivo, ademas de controlar todos los sistemas del Video Portero, permitirá la autentificación de usuarios para poder evitar que alguien no autorizado intente controlar el video portero, que, al tener acceso al seguro electromecánico del portón, se convierte en un riesgo de seguridad de no realizar esto.

2.2.2. DISPOSITIVO ANDROID

El dispositivo Android que se ocupará, deberá tener instalada la aplicación diseñada en este proyecto de titulación, la misma que permitirá al usuario registrarse o autentificarse con el Video Portero a través de mensajes enviados mediante el protocolo TCP/IP en el momento que el usuario llene los datos requeridos.

Una vez realizado el proceso de autentificación, el usuario tendrá acceso al Video Portero, es decir que se le permitirá observar el video que este transmitirá, así como tener comunicación de audio y poder abrir el seguro eléctrico que tenga el portón.

Esto lo podrá hacer el usuario siempre y cuando se encuentre en la misma red LAN en la que se encuentra el video portero. En cambio, si se encuentra fuera de esta red LAN, pero con acceso a internet, únicamente recibirá una notificación si alguien ingreso el código que corresponde a su domicilio.

2.2.3. RED LAN

La red LAN permite la comunicación entre el Video Portero y el dispositivo Android, que estarán conectadas a este mediante una red WLAN, dando movilidad al dispositivo Android y evitando la necesidad de instalar un cable de red para conectar el video portero a la misma.

De igual forma permitirá el acceso al Internet, el cual se identificará su propósito a continuación.

Vale recalcar que el diseño de la red LAN no se encuentra entre las actividades de este proyecto de titulación.

2.2.4. INTERNET

Al tener conectividad al internet, el Video Portero tendrá acceso al servidor de Google, donde tiene los servicios de Firebase, entre los cuales se ocupará el servicio de Firebase Cloud Messaging que será mencionado a continuación.

2.2.5. FIREBASE CLOUD MESSAGING

Permite él envió de notificaciones a los celulares que se encuentren con la aplicación asociada a este sistema. Para ello, en la aplicación del dispositivo móvil, se debe programar con los elementos necesarios para esta asociación con lo que obtendrá un numero único denominado TOCKEN. Con este TOCKEN se podrán enviar las notificaciones al dispositivo móvil desde el video portero, así no se encuentre encendida la aplicación, con el único requerimiento que tenga acceso al internet.

2.3. FUNCIONAMIENTO ESPECÍFICO DE LOS SISTEMAS

En la Figura 2.2. se puede observar los sistemas principales ocupados para el desarrollo de este proyecto de titulación. La red LAN e internet son considerados como medio de comunicación ocupados únicamente para él envió de los mensajes correspondientes.

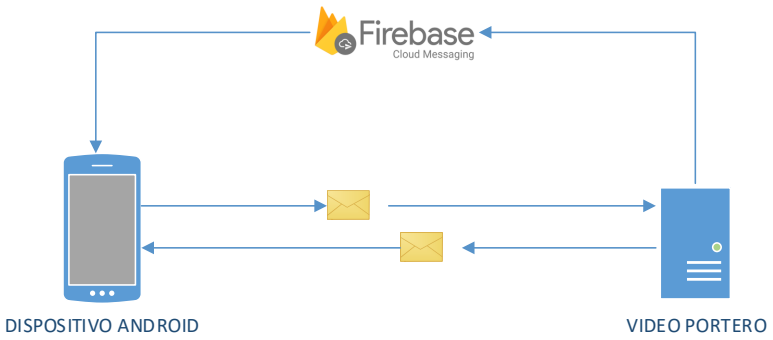

Figura 2.2. Diagrama de sistemas en funcionamiento.

2.3.1. VIDEO PORTERO

El sistema de video portero tiene varias funciones para poder cumplir su propósito general, entre estos pueden destacar;

Servidor de mensajes TCP/IP: Permite la recepción de mensajes enviados mediante el protocolo TCP/IP y responde a estos dependiendo del tipo de requerimiento enviado en el mensaje, principalmente es usado para el registro, autentificación y configuración de usuarios, así como la apertura del seguro electrónico del portón, activación y cancelación del broadcast de video HTTP, e inicio y final de la comunicación de audio. Los mensajes recibidos son provenientes del dispositivo Android donde se encuentra la aplicación desarrollada, por lo que esto dependerá de las acciones que el usuario quiera tomar desde su dispositivo.

Servidor HTTP: realiza una transmisión broadcast del video capturado por la cámara del video portero, esta se activa cuando un usuario de la aplicación móvil enciende la aplicación y del mismo modo puede ser desactivada cuando este lo desee o al cumplir un tiempo límite.

Servidor de VoIP: permite la comunicación de audio entre el video portero y el dispositivo Android, encargándose del envío de mensajes de control SIP entre estos dispositivos para iniciar, mantener y terminar una comunicación de audio.

Comunicación de voz: mediante el protocolo SIP (que permite enviar mensajes de control al servidor VoIP) es posible la comunicación de voz con otros dispositivos registrados a este dispositivo. Una vez que el video portero recibe el mensaje de requerimiento de comunicación de voz por parte del dispositivo móvil, se envía un mensaje SIP al servidor VoIP para iniciar la comunicación únicamente con el dispositivo móvil que envío el mensaje.

Envío de notificaciones: a través de las funciones que tiene Firebase Cloud Messaging, el video portero puede realizar el envío de notificaciones a los dispositivos Android que se encuentren registrados en el video portero. Empezando por digitar correctamente el código de uno de los departamentos, para poder identificar a que usuarios será enviada esta notificación.

2.3.2. DISPOSITIVO ANDROID

El dispositivo Android permite la instalación de la aplicación desarrollada en Android Studio, y gracias a esta, es posible el uso de los distintos sensores y protocolos que permiten la comunicación con el video portero, por lo que es la interfaz que ayudará al usuario interactuar con el video portero.

Gracias a esto, desde el dispositivo Android va a ser posible realizar las siguientes actividades:

Registro: cuando es la primera vez que el usuario desea usar el video portero, este tendrá que ingresar los datos del departamento en el que reside, y una vez verificado que estos sean correctos se procede a ingresar el nombre de usuario y contraseña para enviar al Video portero y proceder al registro en este.

Autentificación: permite verificar que el usuario que intenta ingresar a través de la aplicación al video portero sea un usuario previamente registrado, de este modo no permitir que personas no autorizadas tengan acceso a este.

Recepción de notificaciones: gracias a la facilidad que da el servicio de Firebase Cloud Messaging es posible la recepción de notificaciones y el despliegue de estas en el dispositivo móvil sin la necesidad de mantener la aplicación activa, y con el único requerimiento de tener acceso a internet.

Comunicación de voz: debido al uso del protocolo SIP, es posible la comunicación de voz haciendo uso de un servidor de VoIP, que en este caso se encuentra en el mismo video portero. Para que la comunicación de voz inicie se envía un mensaje TCP/IP con este requerimiento al video portero, y este iniciará la comunicación de voz.

Reproducción de video: para poder observar las imágenes de video que son transmitidas desde el video portero, la aplicación instalada en el dispositivo Android permite la apertura de una página HTTP en la pantalla de la aplicación, en la que se podrá observar las imágenes con un tiempo mínimo de diferencia. Desde la misma aplicación se podrá apagar o encender el broadcast de imágenes HTTP cuando sea necesario. Todo esto enviando mensajes de control a través del protocolo TCP/IP.

Control del seguro electrónico: al igual que el resto interacciones con el video portero, para poder abrir el portón electrónico se envía un mensaje al video portero mediante el protocolo TCP/IP con los datos requeridos para esto, y se recibirá un mensaje una vez que esta acción se cumpla, mensaje que se observará momentáneamente en el dispositivo móvil.

2.3.3. FIREBASE CLOUD MESSAGING

Para acceder a este servicio ofrecido por Google únicamente es necesario registrarse en su plataforma y obtener la clave del servicio. Con esta se desarrolla el software en el video portero para realizar el envío de notificaciones a través del servidor de esta plataforma que permite localizar a los dispositivos móviles específicos, con la única condición de que el dispositivo se encuentre con acceso a internet.

Se ocupa este servicio ya que los dispositivos móviles no siempre se encuentran con la misma dirección IP o incluso no se encuentra en la misma red, adicional a esto, en los dispositivos móviles, la recepción de notificaciones se la realiza así la aplicación no se encuentre activa, solo requiere tener la aplicación con los datos necesarios registrados en el video portero.

2.4. REQUERIMIENTOS

2.4.1. REQUERIMIENTOS DE VIDEO PORTERO

2.4.1.1. REQUERIMIENTOS DE HARDWARE

Por parte del video portero, las funciones que realiza este dispositivo, hace necesario que disponga de algunos elementos que trabajen en conjunto, como son los siguientes:

Interfaz de audio: esta interfaz hace referencia a un micrófono y parlante, necesarios para la comunicación de voz. Adicional a estos elementos, es necesario tener un circuito que haga la transformación analógico-digital y viceversa para poder enviar y recibir audio digital. Elementos que deben ser compatibles con la Raspberry Pi.

Interfaz de imagen: permite la captura de imágenes a ser transmitidas. Esta cámara debe ser compatible con la Raspberry Pi, ya que esta micro computadora tiene un conector exclusivo para esta.

Teclado: el teclado es una interfaz de entrada necesaria para que el usuario ingrese datos al video portero, y así este tome la acción solicitada.

Micro computadora: permite la integración y el procesamiento de todos los elementos, así como la ejecución de software que controlará todo el sistema. Este de preferencia debe tener integrado un módulo de red WLAN que cumpla por lo menos con el protocolo 802.11n para no tener que añadirlo y hacer más robusto el sistema. Así como deberá tener suficiente memoria para la instalación del sistema operativo y el software.

Así mismo estos elementos deben ser pequeños para que el producto final no tenga un gran tamaño.

2.4.1.2. REQUERIMIENTOS DE SOFWARE

El software por su parte, deberá tener control de todos los elementos de hardware, de este modo deberá tomar la acción necesaria cuando se presione un botón del teclado, encender y apagar la cámara cuando sea requerido, inicializar la comunicación de audio y procesar los mensajes de control entrantes por la red WLAN.

2.4.2. REQUERIMIENTOS DE DISPOSITIVO ANDROID

2.4.2.1. REQUERIMIENTOS DE HARDWARE

El uso de un dispositivo móvil inteligente garantiza que tenga los elementos necesarios para que la aplicación a ser instalada sea completamente funcional. Estos elementos serán, micrófono, parlante, pantalla táctil, sensor de proximidad, tarjeta de red WLAN y memoria disponible de al menos 115 MB. Los dispositivos que cumplen con estos requerimientos, y será posible instalar la aplicación son celulares y Tablets que tengan el sistema operativo Android 5 hasta Android 8, esto debido a que en las actualizaciones que realiza Android en sus distintos sistemas operativos algunos elementos de programación no son compatibles con otras versiones.

2.4.2.2. REQUERIMIENTOS DE SOFWARE

La aplicación deberá ser intuitiva para el usuario, es decir que deberá ser fácil de usar, dando al usuario las indicaciones necesarias para que la utilice y que realice las acciones que este desea, sin que sea necesario conocimientos adicionales. Este deberá tener la posibilidad de enviar mensajes de control a través del protocolo TCP/IP por lo que debe tener permitido el acceso a la red WLAN y requerirá de igual forma activar el uso del micrófono y parlante cuando sea requerido.

2.5. ANÁLISIS

2.5.1. VIDEO PORTERO

De acuerdo a lo visto anteriormente, el video portero permitirá la comunicación con dispositivos móviles como servidor de mensajes TCP/IP, lo que únicamente le permite responder a los mensajes recibidos, tomando acción dependiendo del mensaje recibido. Este podrá recibir únicamente los mensajes de los dispositivos que se encuentren en la misma red LAN, ya que está pensado para que trabaje en la red alquilada de un hogar, y no tiene una dirección IP publica fija para poder direccionar los mensajes a través de esta. Para que el video portero pueda enviar notificaciones a los usuarios, se hace uso de la herramienta Firebase Cloud Messaging, proporcionada por Google para el desarrollo de aplicaciones, y gracias a esta los dispositivos móviles registrados en el video portero pueden ser localizados, se encuentren en una red LAN con acceso a internet o en una red celular, ya que esta herramienta localiza al dispositivo móvil con la única condición que se encuentre conectado a internet. Por esta razón, el envío de las notificaciones se podrá realizar a todos los usuarios registrados y que tengan acceso a internet, mas no podrán interactuar con el video portero si no se encuentran en la misma red LAN que este.

2.5.2. DISPOSITIVO ANDROID

Se eligió usar dispositivos inteligentes con S.O. Android para aprovechar la capacidad de procesamiento que tienen hoy en día estos dispositivos y la facilidad que dan Android para realizar aplicaciones con Android Studio, un entorno de desarrollo completamente gratis. Gracias a esto es posible el uso de los sensores del dispositivo Android a la conveniencia y requerimiento de la aplicación a desarrollar. En este caso específico, micrófono, parlante, sensor de proximidad y pantalla táctil, así como la posibilidad de enviar y recibir mensajes TCP/IP.

2.6. DISEÑO E IMPLEMENTACIÓN DE HARDWARE Y SOFTWARE EN VIDEO PORTERO

Una vez se tiene claro cómo se desea que funcione el prototipo, se procede a realizar el diseño del hardware y software para que cumplan de la forma más adecuada estas características.

En la parte de hardware se podrá observar todos los dispositivos adicionales a la Raspberry Pi que estarán conectados a este, así como la inicialización del S.O. de esta, ya que se ejecuta en la Raspberry Pi un S.O. que debe ser inicializado y configurado antes de poder ejecutar el software realizado.

Por la parte del diseño de software se podrán observar los programas en el lenguaje de programación Python que serán ejecutados en la micro-computadora. Y posteriormente en la sección 2.7. se expondrá la aplicación desarrollada para los dispositivos Android.

2.6.1. DISEÑO DE HARDWARE

Gracias al desarrollo de los diferentes accesorios que tiene la Raspberry Pi es posible el uso de estos, facilitando la implementación y programación de este, ya que junto a estos vienen instrucciones y librerías de Python ya probadas y listas para la implementación. Por esta razón el único diseño que se realizó en hardware es el del teclado y el circuito de activación del relé. A continuación, en la Figura 2.3., se puede observar un esquema de cómo van conectados estos dispositivos.

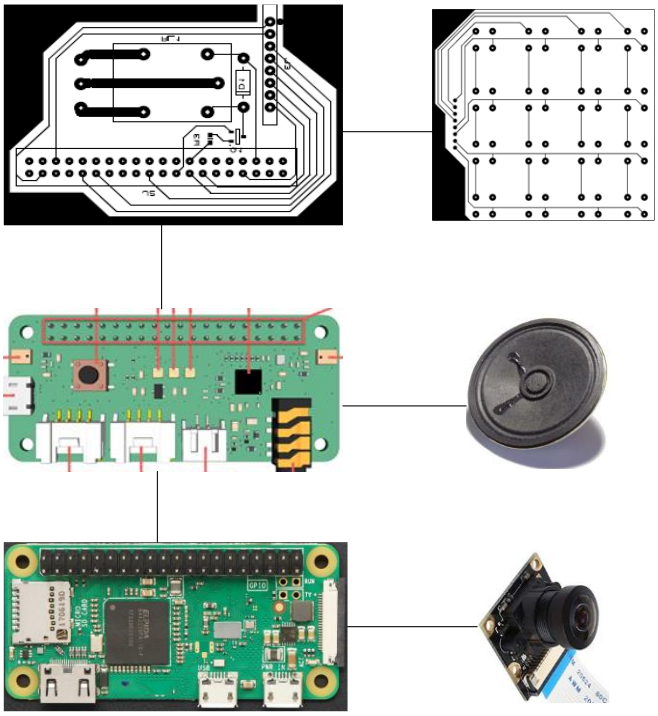

Figura 2.3. Esquema de conexión de hardware**.**

2.6.1.1. Cámara

Gracias a que la Raspberry Pi tiene ya un puerto para poder conectar una cámara como se mencionó con anterioridad en el capítulo 1.3.2.1.1., únicamente es necesario tener una cámara compatible y conectarlo del modo adecuado en la Raspberry, para posterior mente proceder a realizar las configuraciones de software necesarias.

2.6.1.2. Tarjeta de Audio

Para poder tener audio de entrada y salida en el dispositivo, es necesario conectar a la Raspberry Pi un módulo de audio denominado 'ReSpeaker 2-Mics Pi HAT', cuyas características se encuentran detalladas en el capítulo 1.3.2.2. Este se conectará mediante los pines GPIO, pero, aunque se encuentren conectados todos los puertos únicamente se ocupará 14 pines como se puede ver en la Figura 2.4., de estos incluyendo los pines de energía (5 V y tierra), lo que nos da la posibilidad de ocupar el resto de pines requeridos.

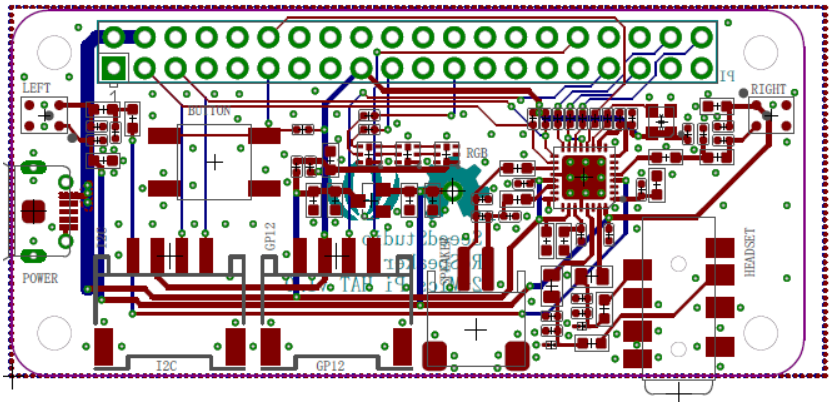

Figura 2.4. PSV de módulo de audio ReSpeaker 2-Mics Pi HAT [12]**.**

Adicional al módulo de audio, es necesario conectar a este un pártale de 8 ohmios para que sea posible escuchar el audio transmitido desde el dispositivo Android. Este se conectará en el conector "JST 2.0 SPEAKER OUT" exclusivo para este propósito y que se podrá observar en la Figura 2.5.

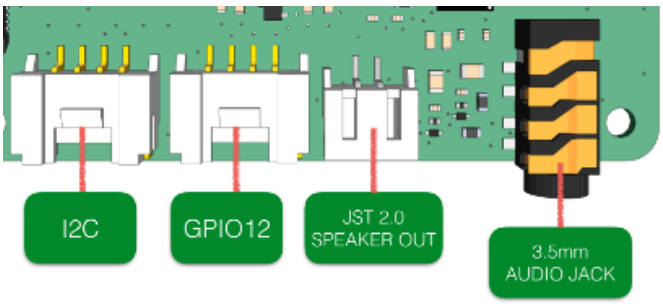

Figura 2.5. Conectores de modulo "JST 2.0 SPEAKER OUT" [12]

Los pines de la Raspberry Pi que se ocuparán para esta tarjeta de audio son:

Pines 2, 3, 5,11, 12, 13, 17, 19, 32, 33, 35, 38, 40.

De los cuales solo se podrán reusar los pines que alimentan de energía al circuito.

2.6.1.3. Módulo de control de apertura de seguro electromecánico y teclado matricial.

El desarrollo del circuito que permite la apertura del seguro eléctrico y para poder conectar un teclado al sistema del Video Portero, se desarrolló en una sola placa para poder ahorrar espacio y aprovechar los pines que no se ocupan con la tarjeta de audio antes mencionada. La mayor cantidad de pines a ocuparse será para el teclado, esto debido a que requiere 8 para poder realizar un barrido matricial de un teclado 4x4 que permita el funcionamiento de 16 botones. Estos 8 pines serán configurados 4 como salida y 4 como entrada, y su funcionamiento está controlada mediante el programa Python con el archivo denominado 'Teclado_y_Notificaciones.py'

Pin 7: out4

Pin 13: out3

Pin 15: out2

Pin 21: out1

Pin 29: In 4

Pin 31: In 3

Pin 36: In 2

Pin 37: In1

El teclado matricial que se conecta a estos pines consta de 16 pulsadores acomodados en una matriz 4x4 como se observa en la Figura 2.6.

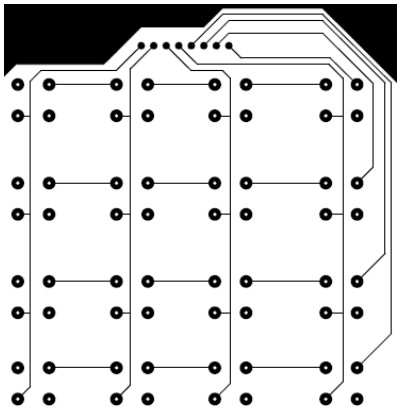

Figura 2.6. Esquema de placa para teclado matricial

Esta funciona gracias a un programa que se ejecuta en un bucle infinito y únicamente se interrumpe al presionar uno de los botones, permitiendo realizar la acción requerida por este botón y volviendo al bucle predeterminado, el funcionamiento detallado del programa que habilita este teclado se podrá observar más adelante.

Adicional a estos, se requiere para la apertura del seguro eléctrico, el uso de un pin de salida controlado por la programación en Python, un pin de voltaje y pin de GNR, en este circuito se ocupa un relé que permitirá el paso de energía necesario para la apertura de este seguro eléctrico (usualmente 12 V-4 A) en el momento que el programa lo determine cumpliendo la misma función de un pulsador manual ordinario. Para esto se ocuparán los pines:

Pin 6: conexión a tierra del circuito.
Pin 16: pin de salida controlada mediante programación.
Pin 17: pin DC de 3.3 [V].
Para realizar esta parte del circuito se ocuparon los siguientes elementos:
Transistor PNP BC857B
Zener 1N4733
Resistencia de 100 ohmnios
Relé con bobina de 5 [V] y soporta un paso de 250 VAC - 10 A o 30 VDC - 10 A.

El diagrama del diseño se puede observar a continuación en la Figura 2.7.:

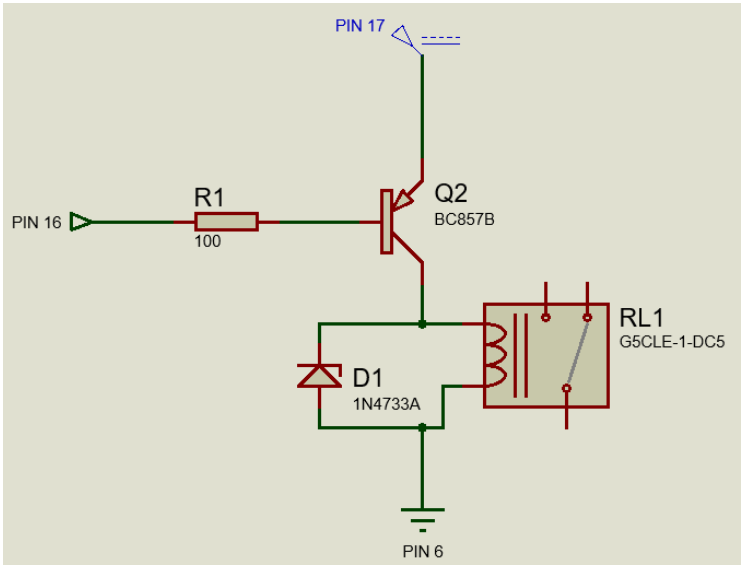

Figura 2.7. Diseño de control de apertura de seguro eléctrico.

En un estado normal el Pin 16 se mantendrá con un voltaje alto de 3.3 V lo que mantiene al relé en la posición de reposo, ya que no está recibiendo energía para activarlo. En el momento en el que la programación cambia de estado el Pin 16, este pasará a ser 0 [V], lo que permitirá que el transistor PNP fluya corriente hacia el relé y active este para que el seguro electromecánico se abra.

Este Pin 16 se activará únicamente cuando un usuario autorizado presione el botón respectivo en la aplicación, y mantendrá una duración en el estado bajo de 1 segundo, suficiente para que el seguro electromecánico se habrá.

2.6.2. IMPLEMENTACIÓN DE HARDWARE DE VIDEO PORTERO

Antes de poder realizar las pruebas se realiza el ensamblaje del Video Portero en la carcasa diseñada y construida en una impresora 3D. Esta fue diseñada para que pueda calzar una Raspberry Pi Zero W que es el modelo más pequeño de estas microcomputadoras en la actualidad, pero también podrá usarse una Raspberry Pi 3B+ acomodando el cable de energía correctamente. En la imagen de la Figura 2.8. se puede observar varias imágenes de esta carcasa.

Figura 2.8. Carcasa construida para el ensamblaje del Video Portero.

Se procede primero a colocar la Raspberry Pi Zero W, ajustándola mediante 4 tornillos, como se muestra en la Figura 2.9.

Figura 2.9. Colocación y ajuste de Raspberry Pi en carcasa.

Una vez en posición se podrá colocar el modulo diseñado para la conexión del teclado y que contiene el circuito de apertura del seguro electromecánico del modo en que se puede observar en la Figura 2.10.

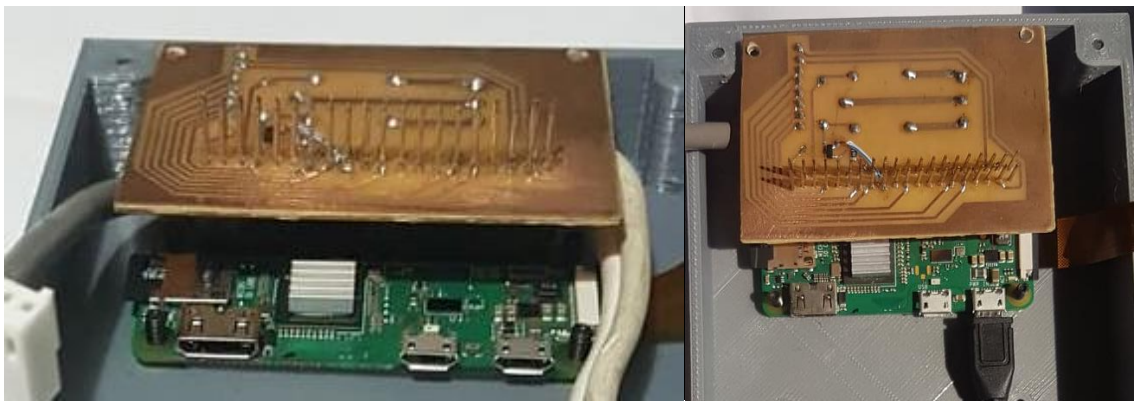

Figura 2.10. Ensamblaje de módulo de teclado y circuito de apertura de puerta.

Antes de colocar el modulo anterior se conectará un cable de 8 hilos con conector de 8 pines en línea al módulo, este permitirá la conexión al teclado matricial y de igual forma el cable que se tendrá que conectar al seguro eléctrico.

Para después de este colocar el módulo de audio como se ve en la Figura 2.11.

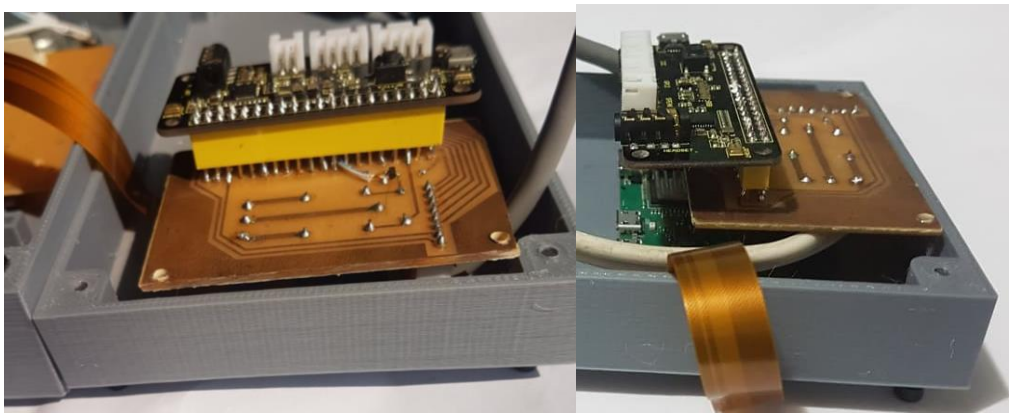

Figura 2.11. Ensamblaje de módulo de audio

En la parte frontal de la carcasa se colocarán los periféricos, es decir el parlante, el teclado y la cámara, los micrófonos se encuentran integrados a la tarjeta de audio por lo que no fueron mencionados. Por lo que quedará como se observa en la Figura 2.12.

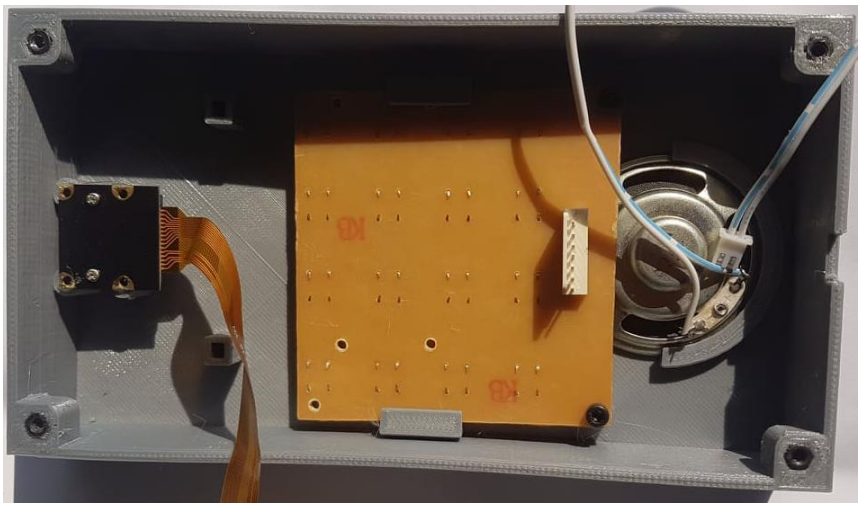

Figura 2.12. Ensamblaje de periféricos en carcasa frontal.

El parlante se mantendrá en su posición bajo presión, mientras que se tendrá que ajustar el teclado con dos tornillos para evitar que se mueva. Y la cámara por la falta de tornillos adecuados se la pego con silicona en su lugar.

Ya con todo en su lugar se proceden a conectar a los periféricos con los respectivos cables, por lo que antes de sellar quedará como se puede observar en la Figura 2.13.

Figura 2.13. Conexión de periféricos con sus respectivos cables.

Con la excepción del teclado que se deberá conectar antes de cerrar la carcasa debido al cable corto que se colocó.

Para facilitar la configuración de una nueva red inalámbrica y dirección IP fija (configuración que se puede observar en el manual de usuario del ANEXO DD), se encuentran instalados un puerto USB para la conexión de un teclado y una salida de video compuesto. Este ultimo se le podrá conectar a cualquier televisor con entrada de video compuesto, caso contrario, será necesario conectarlo al puerto HDMI antes de cerrar la carcasa del video porteo, ya que si no se encuentra conectado a la red no se podrá realizar ninguna manipulación al no poder ocupar el protocolo SSH.

Una vez configurada la red y sellada la carcasa se procederá a conectarla la fuente de energía para poder realizar las pruebas de funcionamiento finales.

2.6.3. DISEÑO E IMPLEMENTACIÓN DE SOFTWARE

Una vez armado el Hardware e instalados los programas y librerías requeridas como se podrá observar en el ANEXO B, se puede continuar con el diseño implementación y ejecución del software que permitirá el funcionamiento de la Raspberry Pi como Video Portero y de la Aplicación que funcionará en conjunto.

2.6.3.1. Programación de Raspberry Pi en Python

Como se mencionó previamente en la Raspberry Pi se ocupó un S.O. ligero que se lo puede encontrar en la página oficial de Raspberry [36] denominado Raspberry Pi OS Lite (Raspbian Buster Lite) para evitar que exista un exceso de procesamiento en interfaces gráficas que no serán ocupadas, por lo que únicamente trabaja con un terminal de comandos desde donde se desarrollarán y prueban los programas de este proyecto de titulación.

El programa inicial, junto con los programas secundarios y funciones se encuentran dentro de la carpeta cuyo directorio es "/home/pi/VideoPorteroPython/". En este directorio se encuentran los siguientes archivos Python y directorios (el diagrama de directorios completo puede observarse en el ANEXO C).

BaseDatosSQLite.py
BD
FuncionesDatos.py
HttpVideo
Paquete
RecepcionDeMensajes.py
Teclado_y_Notificaciones.py
Tones
VideoPortero.py
VideoPorteroLogs
Voip.py

Todos necesarios para el correcto funcionamiento del Video Portero.

En general los diferentes archivos '*.py' se ejecutan dependiendo de las acciones tomadas en cada paso, pero en la Figura 2.14. se observa que programa puede ejecutar otro programa '*.py' o únicamente una función de este.

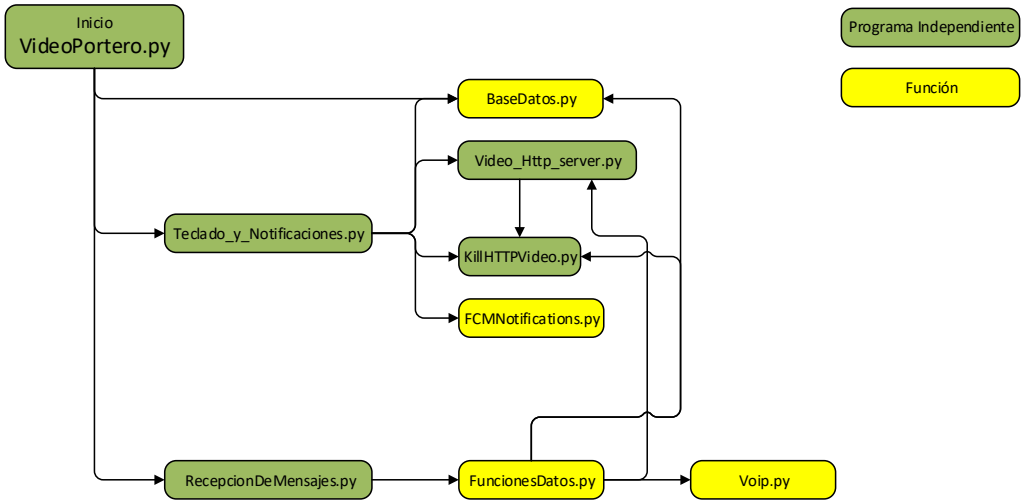

Figura 2.14. Diagrama de flujo general de programas y funciones.

A continuación, se detallará el funcionamiento del código escrito en los archivos antes mencionados.

2.6.3.1.1. Ejecución del programa al encender la Raspberry

Para que el programa se ejecute en el momento de encender la Raspberry es necesario ocupar una aplicación existente en algunos sistemas operativos basados en Linux, entre estos se encuentra Raspbian, y a la aplicación se ingresa mediante el comando [45]:

crontab –e

Mediante esta aplicación se puede escribir comandos que se quieran ejecutar periódicamente o inmediatamente después de una acción, en este caso se requiere ejecutar un programa Python después de que la Raspberry se encienda. Vale recalcar que al momento de ejecutar el comando previamente mencionado se ingresará a un editor de texto donde explica lo que se puede hacer, todo lo escrito en este archivo está comentado por lo que no se ejecutará al menos que se quite el símbolo que los caracterizan como comentario que es el numeral “#”.

Para ejecutar el programa Python deseado, se escribirá la siguiente línea de comando:

@reboot Sleep 50; sudo /usr/bin/python2.7 /home/pi/VideoPorteroPython/VideoPortero.py

Resumen del comando

- ✓ @reboot -Identifica la acción que debe ocurrir para que se ejecute el comando deseado.

- ✓ Sleep 50 -Indica que después de que ocurra la acción que accionará la ejecución del comando se esperará un tiempo de 50 segundos para su ejecución, esto se lo realiza para que encienda correctamente la Raspberry y que se conecte a la red antes de comenzar.
- ✓ sudo -para que ejecute en modo administrador
- ✓ /usr/bin/python2.7 -Se ingresa la ubicación de la aplicación Python para ejecutar el programa
- ✓ /home/pi/VideoPorteroPython/VideoPortero.py -Se ingresa el archivo *.py que será ejecutado con su directorio completo antes de este.

De este modo el programa se ejecutará sin necesidad de hacerlo manualmente cada vez que se resetee o se encienda la Raspberry, pero adicionalmente es posible mandar todos los LOGS a un archivo para poder conocer si existe algo que no permita el buen funcionamiento del programa, esto se lo realiza añadiendo a la misma línea el siguiente complemento:

>> /home/pi/VideoPorteroPython/VideoPorteroLogs/VideoPorteroLogs.txt

- ✓ >> añadiendo este símbolo enviará todos los LOGs al archivo de texto deseado.
- ✓ /home/pi/VideoPorteroPython/VideoPorteroLogs/VideoPorteroLogs.txt - es la dirección exacta del archivo en el que se quiere almacenar los LOGs.

2.6.3.1.2. Inicio del programa base

El programa base que será ejecutado al momento que se inicie la Raspberry tiene el diagrama de flujo observado en la Figura 2.15. En la Raspberry, este archivo de programa tiene el nombre de “VideoPortero.py” y se encuentra dentro del directorio “/home/pi/VideoPorteroPython/” como se mencionó anteriormente, mientras que el código escrito en el archivo se lo puede observar en el Anexo D.

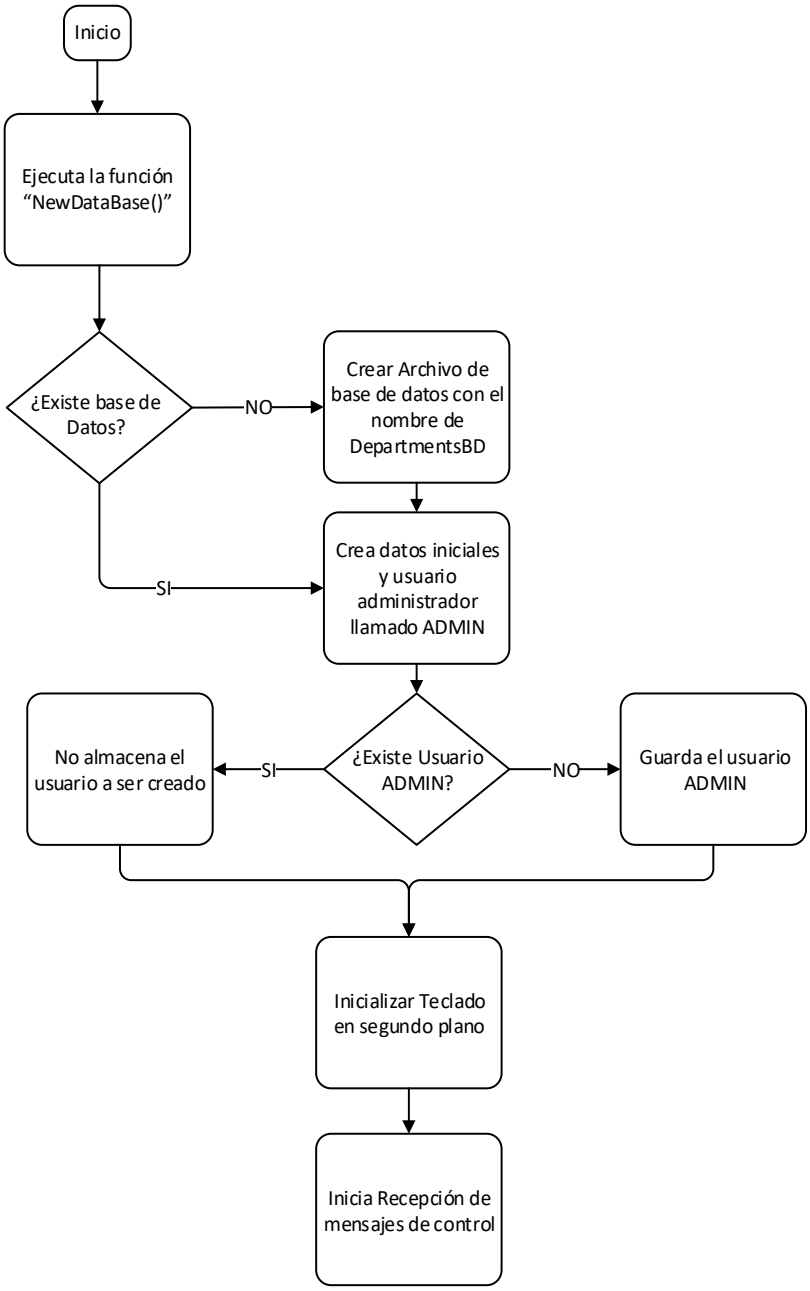

Figura 2.15. Diagrama de flujo de programa inicial ejecutado en Raspberry

Al iniciar este programa se intenta crear el archivo de base de datos mediante la función "NewDataBase ()" que se encuentra dentro del archivo de funciones "BaseDatosSQLite.py" (en este archivo se encuentran varias funciones para el manejo de la base de datos, que serán explicadas de acuerdo sea requerido), este creará un archivo de base de datos denominado "DepartmentsDB" dentro del directorio "/home/pi/VideoPorteroPython/BD". Esta base de datos constará de los siguientes elementos:

- ✓ ID_DEP -es el identificador del departamento con un máximo de 5 caracteres y con la peculiaridad que no puede ser nulo este valor.

- ✓ CLAVE_DEP -es la clave del departamento, no importa cuántos usuarios se encuentren registrados en el departamento, siempre tendrán esta misma clave. Puede tener un máximo de 10 caracteres, pero no puede ser un valor NULO.

- ✓ USER_ADMIN -Identifica a un único usuario como administrador del departamento, permite que este usuario cambie la calve del departamento, o elimine a usuarios que se encuentren registrados en dicho departamento. Únicamente el primer usuario registrado en cada departamento podrá tener estos permisos de administrador y tendrá que ingresar con el nombre deseado seguido de la palabra ADMIN, como, por ejemplo; Kathy ADMIN.

- ✓ USER_NAME -es el nombre de usuario que cada residente del departamento y usuario de la aplicación debe tener, este es un nombre único de un máximo de 20 caracteres.

- ✓ CLAVE_USER -es la clave única de cada usuario, tendrá como máximo 5 caracteres.

- ✓ TOCKEN - es un código único que permite identificar los celulares Android a los que se enviarán las notificaciones, esto gracias a la aplicación Firebase Cloud Messaging que ha implementado Google para el desarrollo de aplicaciones.

- ✓ IP_ADD -es la dirección IP del celular Android, registrado para el envío de información de control.

En el caso de ya existir esta base de datos con el nombre de ¨DepartmentsDB¨ se ignorará este paso y procederá a crear los datos requeridos para que un usuario tenga acceso como Administrador de los departamentos, es decir que este usuario podrá añadir o eliminar los identificadores de los departamentos, así como cambiar la clave de los mismos, eso con el fin de evitar que antiguos residentes tengan acceso a las mismas claves y por lo tanto acceso al edificio. Para esto se almacenan los siguientes datos predeterminados:

ID_DEP="DEP_ADMIN"
CLLAVE_DEP="password"
USER_NAME="ADMIN"
CLAVE_USER="password"

Datos que se almacenarán en la misma base de datos antes mencionada. Todos los usuarios que se registren dentro del “ID_DEP” denominado “DEP_ADMIN” tendrán los

permisos de administrador de todos los departamentos, por lo que está reservado para los dueños o administradores del edificio y se recomienda cambiar la clave del departamento en el momento en se inicie a usar. Si ya existe este usuario “ADMIN” entonces el programa procederá sin almacenar este usuario de nuevo y procederá a realizar el barrido del teclado.

Se ejecuta el barrido de Teclado en segundo plano, esto para que se mantenga ejecutando en un bucle infinito, aunque otros usuarios se estén registrando u ocupando el Video Portero de alguna forma. Este programa ejecutado en segundo plano se llama “Teclado_y_Notificaciones.py” y se ejecuta en segundo plano mediante el comando:

```
subprocess.Popen(["python2.7","/home/pi/VideoPorteroPython/Teclado_y_Notificaciones.py"])
```

donde el comando “subprocess.Popen()” acepta cualquier comando que se puede ejecutar desde el terminal de comandos, de este modo ejecuta un nuevo programa Python como lo es “Teclado_y_Notificaciones.py”, mientras que este programa de barrido de teclado y envió de notificaciones se lo observara a profundidad más adelante con su propio diagrama de flujo.

Después de ejecutar el barrido de teclado se procede a ejecutar la función “RecepcionDeMensajes()” que se encuentra dentro del archivo “RecepcionDeMensajes.py”, este permite que se active el socket en la espera de la llegada de un mensaje desde un celular Android sea para registrarse como usuario, editar datos de usuario, contestar una llamada pendiente, entre otras funciones que se detallarán más adelante, de igual forma este proceso tiene su propio diagrama de flujo con el cual se explicará su funcionamiento.

2.6.3.1.3. Barrido de teclado y envió de notificaciones

El barrido del teclado matricial 4X4 se lo realiza en un segundo plano tomando en cuenta el programa inicial. En el momento que inicia el barrido de teclado se mantiene en un bucle infinito dando energía columna por columna con un periodo de pocos milisegundos, bucle únicamente interrumpido al momento de presionar una tecla. El archivo de este programa se encuentra en el directorio de la Raspberry "/home/pi/VideoPorteroPython/” y se llama “Teclado_y_Notificaciones.py". El código de este archivo se puede observar en el ANEXO E, mientras que el diagrama de flujo de este programa se lo puede observar a continuación en la Figura 2.16.

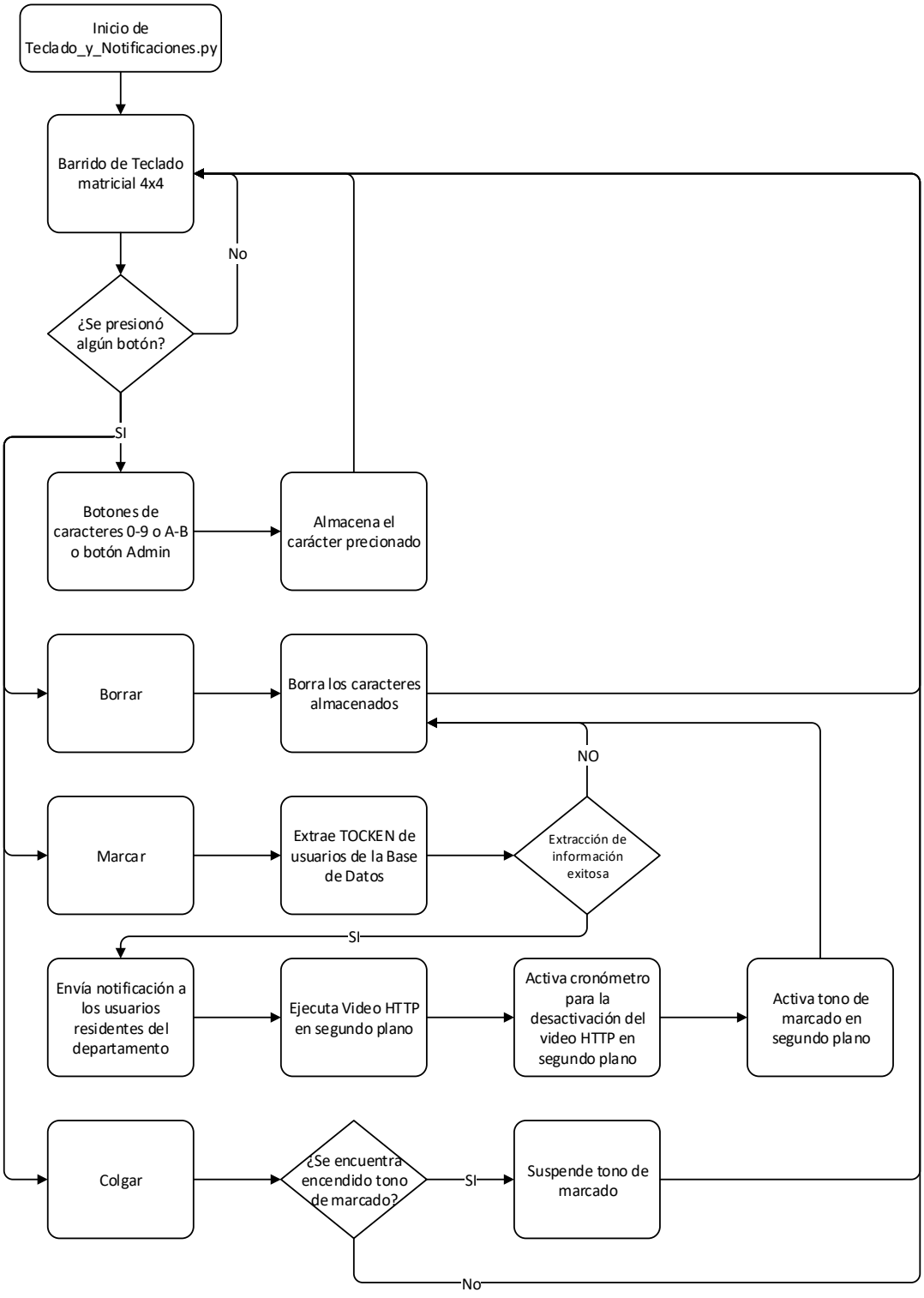

Figura 2.16. Diagrama de flujo de barrido de teclado

Cuando el usuario presione un número o un carácter de identificación (en este caso 0-9, A-B o ADMIN) este carácter o caracteres serán almacenados temporalmente.

Por otro lado, después que el usuario presione los caracteres del departamento deseado tendrá la opción de borrar, que en cuyo caso eliminará los datos antes almacenados permitiendo iniciar de nuevo, pero en el caso de presionar marcar, el programa determinará si el número digitado corresponde a algún número de departamento almacenado en la base de datos, esto lo realiza intentando extraer datos de este determinado departamento con el comando:

BD=BaseDatosSQLite.ReadInformationColumnOfDep("TOCKEN",DepID)

Este comando llama a una función del archivo BaseDatosSQLite.py (que está desarrollada principalmente con la librería 'sqlite3' para el manejo de información de la base de datos) para extraer información de acuerdo al identificador del departamento (expresado con la variable DepID), en este caso se requiere extraer los TOCKENS del mismo. Después de esto existirán 2 opciones que las veremos a continuación:

a. En el caso de no existir este identificador del departamento o de existir el identificador del departamento, pero no existe ningún usuario registrado aun, la variable BD quedaría nula por lo que no podrá continuar, se eliminará lo digitado y regresará al bucle del barrido del teclado.

b. En una segunda opción al estar correcto el numero identificador del departamento y tenga al menos un usuario registrado, este extraerá los TOCKENs en la variable "BD" en forma de una matriz 1xN donde N es el número de usuarios que se encuentran registrados en el departamento determinado.

Al tener el o los TOCKENS en la variable "BD" se procede a extraer estos datos en una lista, para posteriormente de acuerdo a la cantidad de usuarios registrados en el departamento enviar las notificaciones de la siguiente forma:

a. En el caso de existir únicamente un usuario se envía la notificación con la función:

   FCMNotifications.SendNotifucation(DIR)

   Que es una función escrita en el archivo FCMNotifications.py (basada en la librería pyfcm) que envía la notificación a un solo usuario donde "DIR" es el TOCKEN escrito en texto plano ya no como una lista.

b. En el caso de existir más de un usuario registrado se enviará la notificación mediante la función:

FCMNotifications.SendNotifucations(BD)

Que es una segunda función del mismo archivo mencionado anteriormente, pero en este caso, en vez de recibir texto plano recibe como dato “BD” que es la lista de TOCKENS a las que se desea mandar las notificaciones.

Después de realizar el envío de la notificación se activa el streaming de video en segundo plano realizado con el protocolo HTTP (gracias a Miguel Grinberg que publico en un blog un tutorial [46] de donde se extrajo el código base para modificarlo a la necesidad), este será ejecutado con el siguiente comando :

subprocess.Popen(["python3.5","/home/pi/VideoPorteroPython/HttpVideo/Video_Http_server.py"])

Seguido de un contador para desactivar ese streaming de video en un tiempo determinado (de igual forma ejecutado en un segundo plano para poder continuar el proceso sin la interrupción del contador) y de este modo no se mantenga activo cuando no es necesario. El comando en Python es el siguiente:

subprocess.Popen(["python3.5","/home/pi/VideoPorteroPython/HttpVideo/KillHTTPVideo.py"])

Enseguida a esto se activará el tono de marcado por unos segundos para que el usuario sepa que está en la espera de ser atendido, nuevamente en segundo plano para regresar enseguida al barrido de teclado para que tenga la opción de cancelar el llamado o digitar nuevamente el número de departamento si es lo requerido. El comando ocupado es el siguiente:

subprocess.Popen(["aplay","/home/pi/VideoPorteroPython/tones/tono1.wav"])

2.6.3.1.4. Recepción de mensajes de control

La recepción de mensajes de control está programada de modo usuario-servidor, esto quiere decir que la Raspberry Pi como servidor se mantendrá en la espera de un mensaje y tomará la acción respectiva dependiendo de su contenido. Este permitirá la recepción de mensajes de control que servirán para autenticar a los usuarios, modificar la base de datos, actualizarla, o eliminar datos de esta, encender o apagar el Streaming de video, activar el interruptor que abrirá la puerta, entre otras acciones que serán comentadas cuando sea necesario. El diagrama de flujo se encuentra presentado en la Figura 2.17, mientras que el

código redactado en el archivo “Recepcion_de_Mensajes.py” se lo podrá observar en el ANEXO F.

Al iniciar este programa se inicializan los datos requeridos para poder activar un socket que recibirá y enviará los mensajes, es decir la dirección IP que tiene el dispositivo y el puerto elegido para esta aplicación. Este socket se activará con el protocolo TCP para poder recibir y enviar los mensajes sin errores y en el orden correspondiente.

Después de inicializar el Socket con la dirección IP y puerto correspondiente, se realiza la apertura de este puerto, si la micro-computadora no se encuentra conectada a la red WLAN este no podrá proceder y se mantendrá en un bucle hasta que se logre conectar correctamente.

Seguido a esto se activará el socket respectivo para que comience a escuchar y se mantendrá en este modo hasta que reciba un mensaje TCP al puerto respectivo.

Al recibir un mensaje se procederá a convertir este mensaje de texto plano en una lista, donde se dividirán los diferentes campos, para poder identificar que tipo de mensaje se recibió.

A continuación, se ejecutará la función “FuncionesDatos.DataType1 (datos)”, que tiene como datos de entrada la lista obtenida del mensaje entrante y dará como resultado una lista con datos de aceptación o rechazo del mensaje dependiendo si la autenticación de los datos son los correctos o no, así mismo en la lista se incluye el o los datos necesarios para enviar de regreso al usuario (detalles de esta función se las observará más adelante).

Para culminar se envía un mensaje con la respuesta del requerimiento y nuevamente se mantiene en modo de escucha en la espera de un nuevo mensaje.

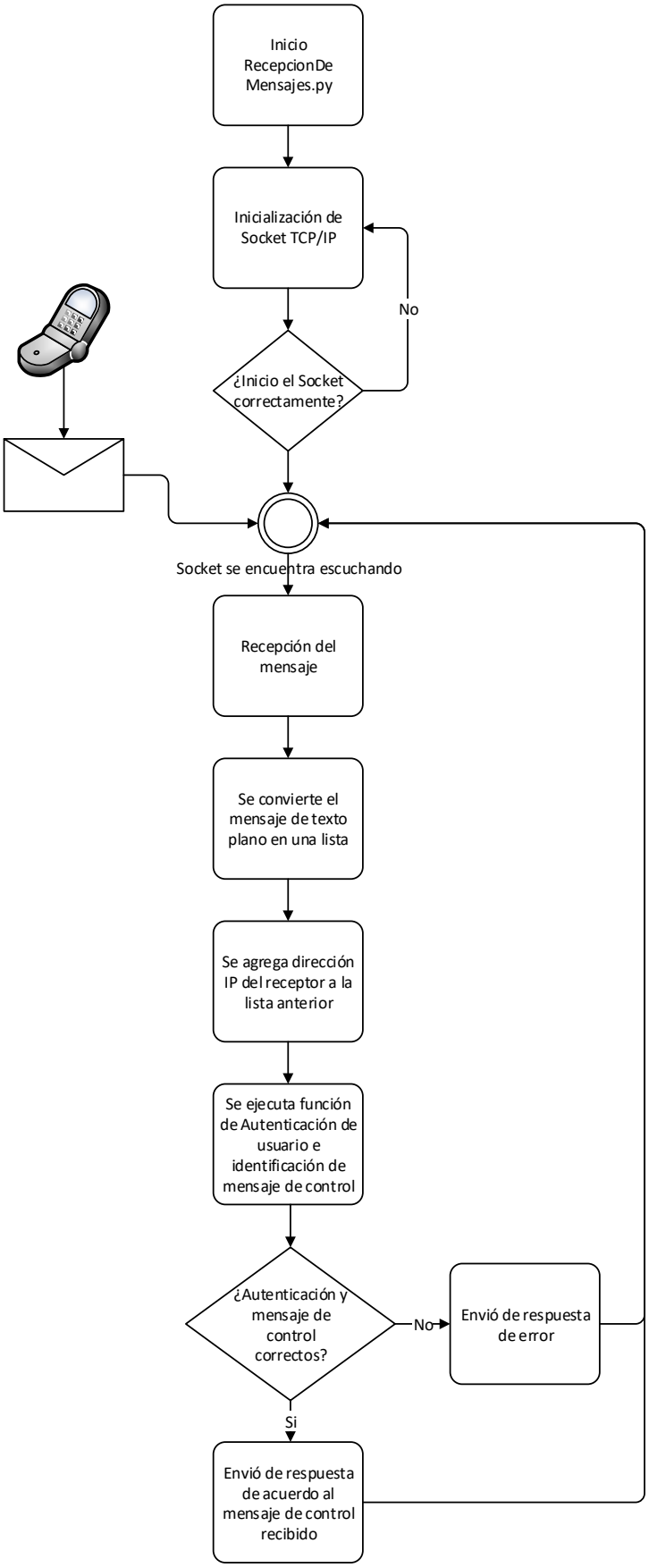

Figura 2.17. Diagrama de flujo de recepción de mensajes de control

2.6.3.1.5. Identificación de usuario y tipo de mensaje de control

Después de la recepción correcta y un pequeño tratamiento que se realiza a los datos recibidos para colocarlo en el formato adecuado, se ejecuta la función que se nombra en el archivo como "FuncionesDatos.py", esta permite autentificar al usuario que envió el mensaje, así como distinguir la razón por la cual el mensaje de control fue enviado. En la Figura 2.18, se encuentra el diagrama de flujo correspondiente a la función mencionada, mientras que en el ANEXO G se puede observar el código completo que se encuentra en el archivo respectivo.

La variable que ingresa a esta función es una lista (un conjunto de elementos almacenados en una variable) donde siempre se encontrarán los siguientes valores:

'ACCION'- identifica que a continuación se encuentra la acción requerida por el usuario en ese momento, como por ejemple: 'RECUSERACCESS'

'ID_DEP'- indica que a continuación se encuentra la identificación del departamento, como, por ejemplo: 'DEP_ADMIN'

'CLAVE_DEP'- indica que a continuación se encuentra contraseña asociada al ID de departamento antes mencionado, como, por ejemplo: 'password2019'

'USER_NAME'- indica que a continuación se encuentra la identificación del usuario, como, por ejemplo: 'Gabriel'

'CLAVE_USER'- indica que a continuación se encuentra la contraseña asociada el nombre del usuario, como, por ejemplo: 'password2000'

De esta forma es posible acceder a la información requerida en el momento deseado, es decir que al necesitar obtener los nombres del usuario se requiere ubicar la posición de esta variable y extraerla de la lista.

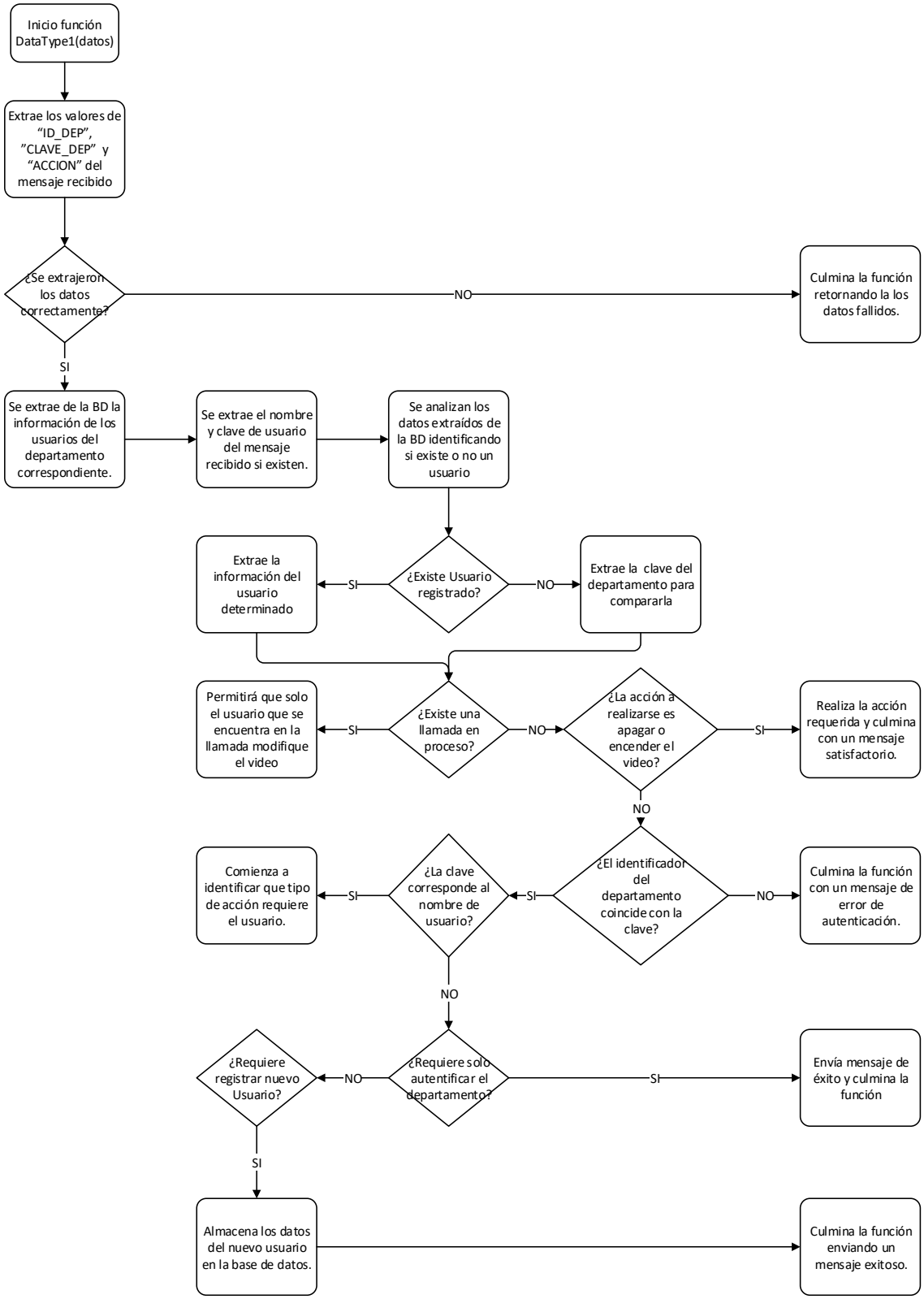

Figura 2.18. Diagrama de flujo de la función **Funciones_Datos.py**

Teniendo esto claro se procederá a redactar el proceso que ejecuta la función 'FuncionDatos.py':

Después de inicializar las librerías requeridos por la función se procede a extraer los valores de “ID_DEP”, “CLAVE_DEP”, y acción de la lista que ingresa a la función, misma lista que es enviada desde el teléfono móvil del usuario, estos datos son requeridos en primer lugar para conocer si se recibió el mensaje correctamente y con los valores necesarios para comenzar la autenticación.

De no lograr extraer estos datos correctamente se enviará un mensaje de error al móvil del usuario, por el que se le dará a conocer al móvil que no se pudo ejecutar la acción requerida, caso contrario avanza la función y extrae los datos requeridos de la base de datos, es decir que usando el “ID_DEP” (obtenida en el paso anterior), se extrae la información de todos los usuarios que se encuentren registrados en este departamento.

Una vez extraído correctamente de la base de datos los datos mencionados, se colocan en otras variables de texto el nombre y la clave de usuario obtenidas de la lista recibida desde el dispositivo móvil. Para proceder a identificar si en la información extraída de la base de datos se encuentra el usuario correspondiente. Es decir que se compara el nombre de usuario “USER_ID” con los datos de usuarios del “ID_DEP” correspondiente.

Si existe este usuario se procederá a extraer únicamente los datos de este, o siendo más específico se extraerá el DEP_ID, CLAVE_DEP, USER_ID, CLAVE_USER, TOCKEN e IP_ADD, caso contrario se extrae de cualquier usuario únicamente la clave del departamento que fue correctamente identificado.

Después de cualquiera de las dos acciones anteriores se procede a verificar si existe una llamada en proceso, esto para no permitir que otro usuario pueda interrumpir la comunicación o pueda variar el estado de la cámara, por lo que sí existe una llamada en proceso se registra el nombre del usuario en la llamada y permitirá únicamente que este tenga algún tipo de acción con respecto al estado de esta.

Si no existe ninguna llamada en proceso, se procede a revisar si la acción en efecto es el apagar o encender el Streaming de Video, cabe recalcar que si el usuario llega a enviar el mensaje con esta acción es porque ya fue autentificado con anterioridad por lo que en la aplicación móvil llego a la actividad donde permite realizar estas acciones, caso contrario (de no ser autentificado) no llegará a realizar ninguna de las acciones siguientes.

Si la acción extraída de la lista es “START_HTTP_VIDEO” Entonces se procederá a ejecutar esta acción mediante el comando;

subprocess.Popen(["python3.5","/home/pi/VideoPorteroPython/HttpVideo/Video_Http_server.py"])

permitiendo que mediante un subproceso se ejecute el streaming de video, mientras que al ser la acción “STOP_HTTP_VIDEO” se ejecutará el siguiente comando;

subprocess.Popen(["killall","python3.5"])

que elimina todos los subprocesos ejecutados mediante el programa Python3.5, pero manteniendo en ejecución los que se realizaron mediante Python 2.7 que es desde donde se ejecuta la programación principal.

Y seguido de cualquiera de estos se enviará como resultado de la función un mensaje de éxito para que sea enviado al móvil del usuario, después del cual se mantendrá en la espera de un nuevo mensaje.

Si ninguna de estas dos se encuentra en el mensaje de control se procede a la comparación de la clave del departamento obtenida de la base de datos con la extraída del mensaje recibido. En este caso, si es incorrecta la autenticación, se procede a enviar un mensaje de error como valor de la función. Pero de ser autentificado correctamente la clave de departamento procede a realizar lo mismo con la clave de Usuario.

Al realizar la comparación de la clave del usuario si es incorrecta se procederá a verificar si el tipo de acción fue “RECDEPACCESS” que es un a solicitud para la autenticación del departamento con su correspondiente clave, de ser así enviará una respuesta positiva al resultado de la función culminando esta y en la espera de un nuevo mensaje de entrada, pero caso contrario la última opción por la que no pudo haber autentificado al usuario sería por que recién se está creando a un usuario de dicho departamento, esto se lo comprueba verificando si la acción es “ADDUSER”, de ser así se creará un nuevo usuario en dicho departamento, con el nombre y la calve ingresada por el usuario, culminando la función con un resultado exitoso que será enviado al teléfono móvil para poder quedar en la espera de un nuevo mensaje. Aunque si al final no es “ADDUSER” la acción requerida en ese caso enviará un mensaje de error al resultado de la función, el mismo que será enviado al móvil que realizo el envío del mensaje.

Regresando a la autenticación del usuario, si esta es correcta se continuará con la identificación de la acción solicitada por el dispositivo móvil, teniendo las siguientes opciones no mencionadas en el diagrama:

"RECUSERACCESS"- acción con la que el teléfono móvil solicita autenticación del usuario, y que de este modo pueda acceder a la actividad principal de la aplicación móvil.

"SPECTING_CALL"- envía esta acción cuando el usuario desea contestar una llamada o simplemente desee comunicarse con el Video Portero para escuchar lo que se encuentra alrededor de este. Al verificar que es esta acción la requerida por el móvil después de ser autentificado el usuario, el video portero verificará que no se encuentra ninguna llamada en proceso mediante el estado de una variable booleana denominada “call_Free”, donde si esta tiene un valor verdadero determina que se encuentra libre de llamadas y al ser Falso se encuentra ya en progreso una llamada. Una vez determinado que no existe otro usuario con una comunicación en progreso se procede colocar la variable “call_Free” en falso y almacena en una segunda variable (denominada “call_name”) el nombre del usuario que se encuentra realizando esta solicitud. Para que otros usuarios no puedan interrumpir esta comunicación. Seguido de esto cancela el tono de marcado (sí estuvo en ejecución), y realiza la llamada atreves de un programa denominado en el directorio como “Voip.py” ejecutándolo en segundo plano. Programa que será detallado más adelante. Y para finalizar, sale como resultado de la función un mensaje de éxito para después enviarlo al dispositivo móvil y mantener en la espera de un nuevo mensaje.

“CLOSE_CALL”- esto permitirá que únicamente el usuario que mantiene una comunicación de voz con el video portero pueda terminar esta comunicación en el momento que desee, apagando de igual forma el streaming de video si este se encuentra encendido.

“OPEN_DOOR"- como lo menciona el nombre de esta acción, esta permitirá que el usuario abra el seguro electromecánico. Cualquiera de los usuarios registrados podrá realizar esta acción, aunque se encuentre alguien más en una comunicación de voz con un usuario en el video portero.

“CONFIG”- Permite ingresar a la actividad de la aplicación móvil donde podrá realizar la configuración en los usuarios y departamento dependiendo el tipo de usuario que ingrese, si el usuario es un usuario registrado en el departamento “DEP_ADMIN” se le permitirá aumentar o eliminar departamentos, así como observar y cambiar la clave de los mismos. Por otro lado, si el usuario pertenece a un departamento diferente se le permitirá configurar su propia clave y nombre de usuario, salvo que el usuario sea administrador del departamento, en este caso podrá eliminar usuarios del departamento correspondiente.

“CONFIG1”- Permite que un usuario ya registrado cambie su nombre y contraseña, datos que serán recibidos adjunto en el mismo mensaje.

“ASKDEPIDDATA”- esto permitirá al administrador principal observar la lista de departamentos numerados en el Video Portero, esta lista se solicitará siempre que el administrador ingrese a configuración.

“ASKDEPPASSWORD”- se envía la solicitud con esta acción para poder recibir la contraseña del departamento seleccionado.

2.6.3.1.6. VOIP

En el momento que el Video Portero recibe un mensaje de control con la acción "SPECTING_CALL" mencionada anteriormente, se ejecuta el programa ‘Voip.py’ en segundo plano, permitiendo que continúe la recepción de mensajes de control. Este se ejecutará mediante el siguiente comando de Python:

subprocess.Popen(["python2.7","/home/pi/VideoPorteroPython/Voip.py"])

Ya que es un programa individual tiene su propio diagrama de flujo que se lo puede observar en la Figura 2.19. que se muestra más adelante y el código que se encuentra escrito dentro del archivo ‘Voip.py’ se lo puede observar en el ANEXO H, código que se encuentra basado en un tutorial realizado por Most Voip API [47] y modificado para que cumpla con los requerimientos del proyecto.

Como se puede observar claramente en el diagrama de flujo, este programa empieza almacenando una única extensión después de inicializar las librerías requeridas en este programa. Esta extensión es la que tendrá el dispositivo Android y que se registrará en Asterisk en el momento que inicia una comunicación de voz, y será el único dispositivo que va a tener una comunicación de voz hasta que cuelgue la llamada.

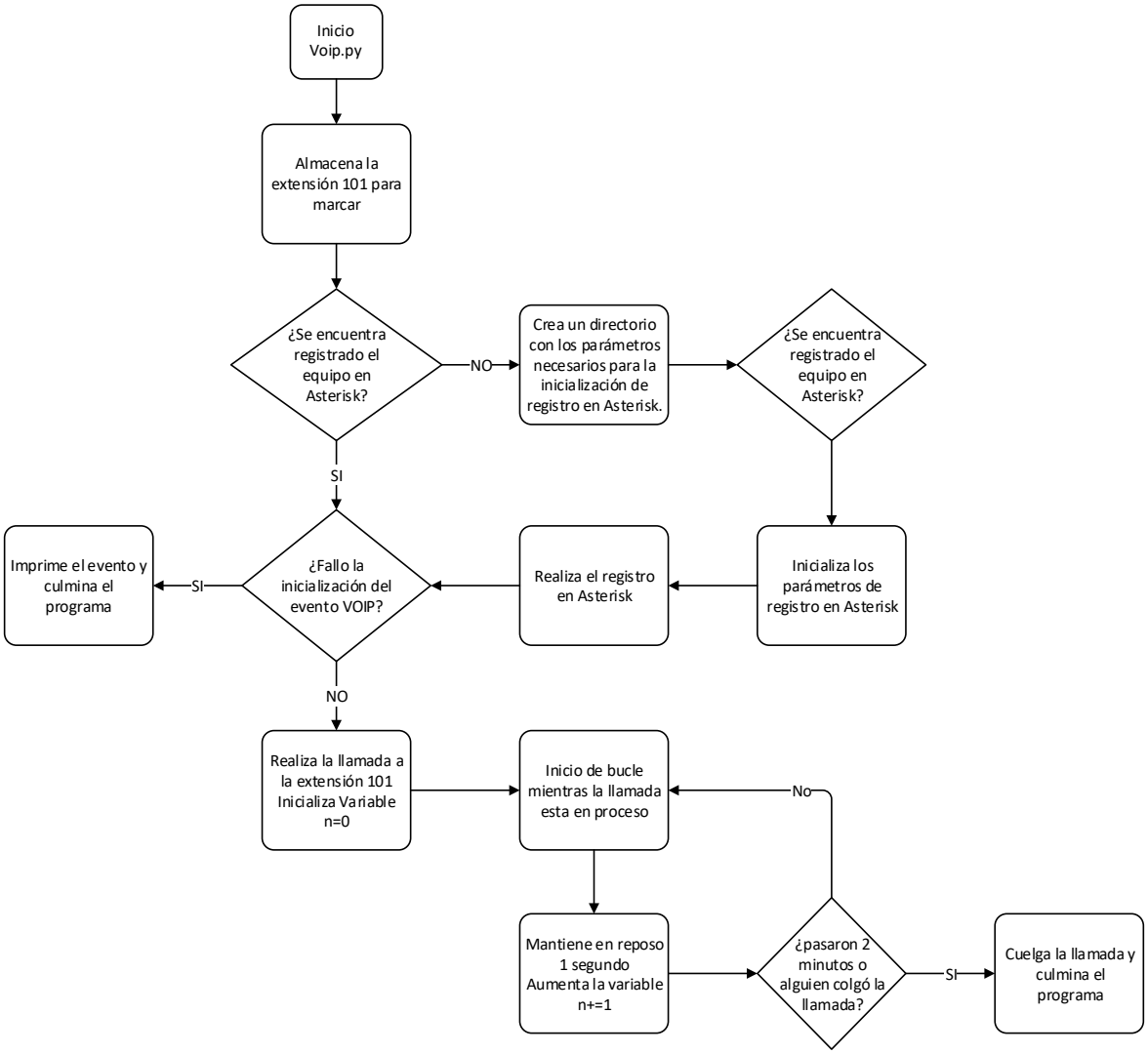

Figura 2.19. Diagrama de flujo de llamadas de VOIP

Una vez almacenada la extensión a la que se va a realizar la llamada se procede a averiguar si ya se registró el video portero en Asterisk para realizar la llamada, si no es así, se creará una variable con los datos requeridos para que sea posible su registro, los datos son los siguientes:

'username' = 'Rasp', es el nombre que describe al usuario

‘sip_server_address' = '192.168.1.150', es la dirección IP del servidor de Asterisk

'sip_server_user' = 'Rasp', es el nombre de usuario registrado en la cuenta SIP

'sip_server_pwd' = 'password', es la contraseña registrada en la cuanta SIP

'sip_server_transport' = 'udp', es el protocolo a ocupar en la capa de transporte, por defecto este se encuentra como ‘tcp’

'log_level' = 0, es el nivel de importancia de los LOGs a proyectar, a mayor número más información será desplazada a los LOGs.

'debug' = False, activa la depuración de este usuario.

Ya con estos datos almacenados en la variable ‘voip_params’ se continúa con el envió de estos parámetros a Asterisk para proseguir con realizar el registro del usuario en el servidor de Asterisk.

Una vez realizado el registro, o si ya se encontraba registrado previamente el equipo en Asterisk procede a confirmar el correcto registro de este. Ocupando la variable ‘event’ donde se almacena el último mensaje enviado por Asterisk para el conocimiento del estado de la comunicación. Si esta variable es “VOIP_EVENT__LIB_INITIALIZATION_FAILED” significa que el registro e inicialización del equipo en Asterisk fue fallida y el programa antes de culminar imprimirá este mensaje.

Si el mensaje es diferente al mencionado anteriormente, en este caso se procederá a conectar la comunicación de voz mediante el siguiente comando:

z=my_voip.make_call(my_extension)

con el cual realizará la llamada a la extensión ‘101’ perteneciente al dispositivo Android que realizó la solicitud de llamada.

Para mantener esta comunicación de voz se procede a entrar a un bucle que realizará un conteo mediante una variable igualada a 0 y coloca a este programa en pausa durante 1segundo por cada vuelta del bucle y aumentando el valor de la variable. De este modo ingresa a una condicional que permitirá culminar la comunicación a los 120 segundos o si el usuario del dispositivo Android cierra la llamada, en este último caso esta acción se verá reflejada en la variable ‘event’ como “VOIP_EVENT__CALL_REMOTE_HANGUP". En cualquiera de las dos opciones se cierra la comunicación de voz, se imprimirá en consola el aviso de esta acción y culminada con el programa.

2.6.3.1.7. Funciones BaseDatosSQLite.py

Estas funciones permiten manipular y obtener información de la base de datos creada exclusivamente para este proyecto, con la ayuda de estas funciones se podrá:

Crear la base de datos:

“NewDataBase()”

Añadir usuarios nuevos:

"AddInformation(ID_Dep1,Clave_Dep2,User_Admin,Usuario3,Clave_User4,TOCKEN5,IP_ADD6)"

Leer toda la información de la base de datos:

"ReadInformation()"

Leer la información únicamente de un departamento específico:

"ReadInformationOfDep(ID_Dep1)"

Obtener los nombres de todos los departamentos:

"ReadInformationColum(Column)"

Obtener los nombres usuarios de un departamento específico:

"ReadInformationColumnOfDep(Column,ID_Dep1)"

Cambiar nombre de usuario, clave y número de TOCKEN de un usuario específico:

"ActualizarUser_Pass(Usuario3_1,Usuario3,Clave_User4,TOCKEN5,IP_ADD6)"

Cambiar clave de un departamento específico:

"ActualizarDep_Pass(DEP_ID1,DEP_PASSWORD)"

Borrar todos los datos de un departamento específico:

"BorrarDepBD(dep_id)"

Borrar a un usuario específico:

"BorrarUser(User)"

Estas funciones se basan en la librería 'sqlite3' y se puede observar el código completo en el ANEXO I.

2.6.3.1.8. Streaming de video mediante HTTP

Para que el video sea transmitido es necesario ejecutar el programa "Video_Http_server.py" código que se encuentra basado en el artículo de un blog denominado "Video Streaming with Flask" [38] y se puede observar el código entero junto con las funciones que ocupa en el ANEXO J.

El principio de este código es transmitir una imagen mediante el protocolo HTTP cada cierto tiempo, permitiendo que al visualizar estas imágenes se observen objetos en movimiento. Ocupa la librería "Flask" para el diseño de la página HTTP y la transmisión de la misma.

Por otro lado, para cancelar el streaming de imágenes mediante HTTP se ejecuta el programa "KillHTTPVideo.py" cuyo código se lo puede observar en el ANEXO K.

2.6.3.1.9. Envío de notificaciones

Para realizar en envió de notificaciones a los dispositivos Android se ejecuta una de las funciones de "FCMNotifications.py" que se puede observar su código completo en el ANEXO L, en este se encuentran dos funciones:

SendNotification(registrarion_id): que permitirá enviar a una sola persona la notificación.

SendNotifications(registrarion_id): que permitirá enviar a varios usuarios la misma notificación al mismo tiempo.

Este necesitará una clave denominada “api_key” que le direccionará al servicio registrado en Firebase Cloud Messaging.

2.7. DISEÑO E IMPLEMENTACIÓN DE APLICACIÓN PARA DISPOSITIVO ANDROID DESARROLLADA EN ANDROID STUDIO

2.7.1. FUNCIONAMIENTO BÁSICO DE LA APLICACIÓN

La aplicación desarrollada para este proyecto de titilación está basada en el siguiente diagrama de flujo que se observa en la Figura 2.20. Cuyo avance entre actividades únicamente se realiza de acuerdo a las acciones que desee el usuario, es decir dependen del botón que el usuario presione en una respectiva actividad, por lo que más adelante se podrá observar detalladamente los procesos de cada actividad.

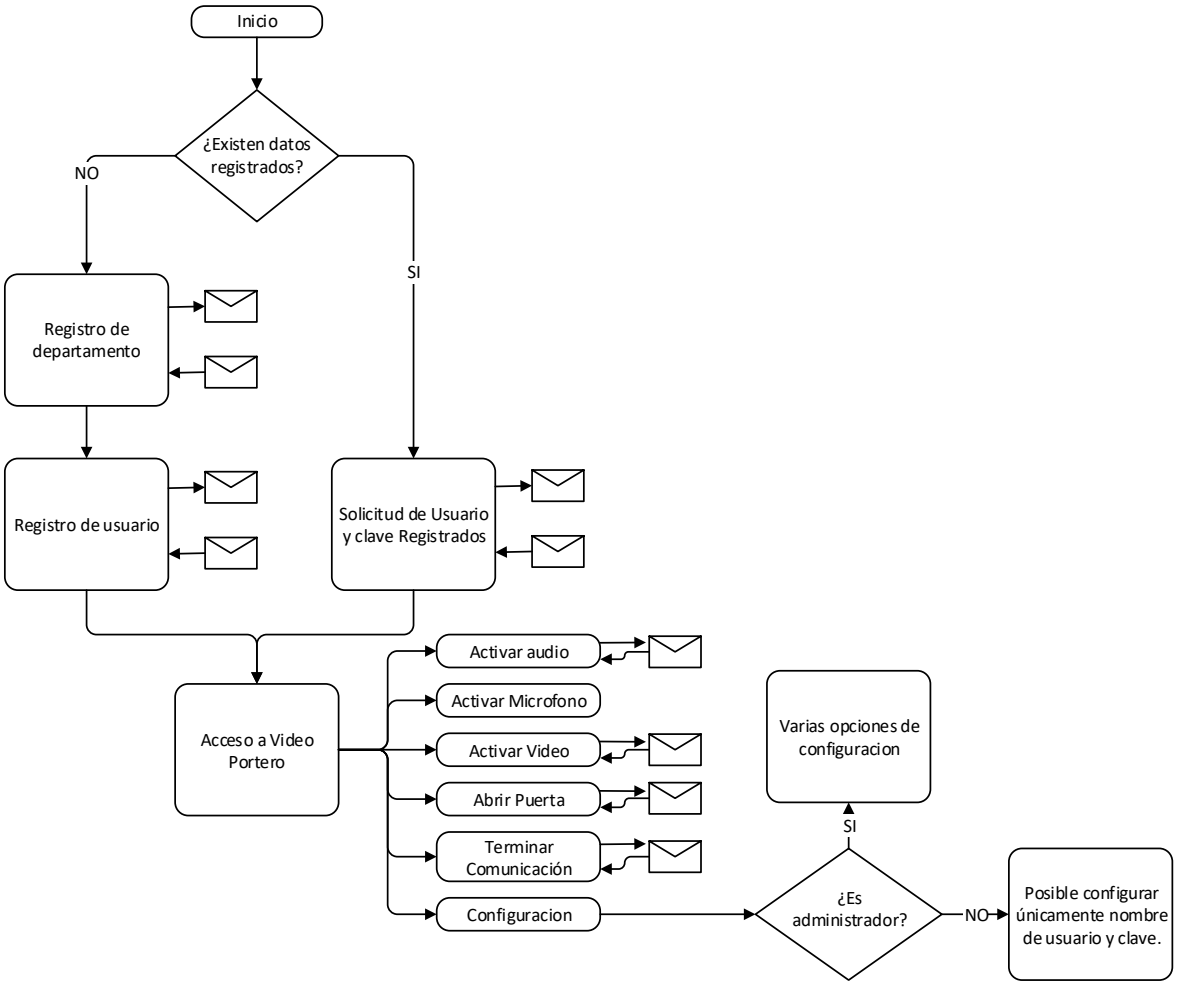

Figura 2.20. Diagrama de flujo general del funcionamiento de Aplicación

Al iniciar la aplicación se ingresa a la actividad 'MainActivity.java', esta, dependiendo si es la primera vez que sea usada la aplicación o no, tendrá uno o dos botones respectivamente, que permitirán al usuario ingresar nuevos datos de departamento o ir directamente a autentificar los datos de usuario, en cualquiera de los dos casos se enviarán mensajes

TCP/IP a la microcomputadora Raspberry Pi para que esta compare y registre la información de ser necesaria.

Una vez pasado la autentificación de departamento y registro o autentificación de usuario en sus respectivas actividades, se abre la actividad 'VideoAccess.java', esta permitirá el uso de 7 botones con los cuales podrá:

Iniciar la comunicación de voz y silenciarla.

Activar y desactivar el micrófono.

Activar y desactivar la transmisión de video

Colgar la llamada

Activar el seguro electromecánico para la apertura de la puerta.

Configurar la base de datos de usuarios.

En el caso de la configuración de la base de datos (que registra el código de departamento, con su respectiva clave, así como los nombres de usuarios, clave de usuario, TOCKEN de cada usuario y la dirección IP de este), el tipo acción que podrá tener el usuario dependerá de la clasificación de este que se dividen en tres:

'ADMIN' Administrador general: tiene acceso a la configuración de toda la base de datos, en términos generales, es decir que puede crear o eliminar departamentos, eliminar usuarios de un respectivo departamento, así como editar su propio nombre de usuario y contraseña.

'ADMIN_USER' Administrador de departamento: este por otro lado podrá eliminar a usuarios únicamente de su respectivo departamento, y de igual forma editar su propio nombre de usuario y clave.

'USER' Usuario de departamento: en este caso lo único que podrá configurar este tipo de usuario será su propio nombre de usuario y clave.

2.7.2. DESCRIPCIÓN DE PROGRAMACIÓN DE LA APLICACIÓN

Para poder realizar una aplicación para Android desde Android Studio en primer lugar es necesario conocer cómo se divide este ambiente de desarrollo, los códigos principales se los va a encontrar escritos con lenguaje de programación JAVA que son denominados

como actividades, pero aparte de estos es necesario crear un ambiente gráfico para que el usuario pueda interactuar con la aplicación, es decir una ventana gráfica que contenga imágenes, texto, botones, entre otros elementos, a esta ventana gráfica se la denomina un 'LAYOUT' y tiene un formato " *.xml", y a cada una de estos layouts se asocia a una actividad (en formato " *.java") que controla las acciones de cada elemento de estas ventanas. Tomando en cuenta esto a continuación se procederá a explicar el funcionamiento de las actividades que controlan a cada uno de los LAYOUTs, cuyos elementos serán mencionados con los nombres asignados, pero no serán detallados a menos de ser necesario.

2.7.2.1. MainActivity

La actividad inicial permite al usuario observar un botón que dará acceso a la actividad de autentificación de un departamento determinado, o por otro lado permite que el usuario ingrese directamente a la actividad que le permitirá la recepción de video y comunicación de audio con la Raspberry Pi, si ya existían los registros de usuario adecuados. El código del archivo 'MainActivity.java' de esta actividad se encuentra en el ANEXO M, mientras que el código del layout 'activity_main.xml' se lo puede observar en el ANEXO N. En la Figura 2.21., se puede observar el diagrama de flujo que permite a esta actividad tomar las acciones necesarias de acuerdo a la información que tiene disponible.

La actividad empieza cargando las librerías requeridas para que las funciones de esta se ejecuten con normalidad. Una vez cargadas estas librerías se procede a definir las variables para a continuación ejecutar la función 'onCreate', que se ejecuta únicamente al iniciar la actividad permitiendo dar forma a la ventana que será visible para el usuario de acuerdo a la información almacenada, por lo que dentro de esta función 'onCreate' se ejecutarán lo que resta del diagrama de flujo.

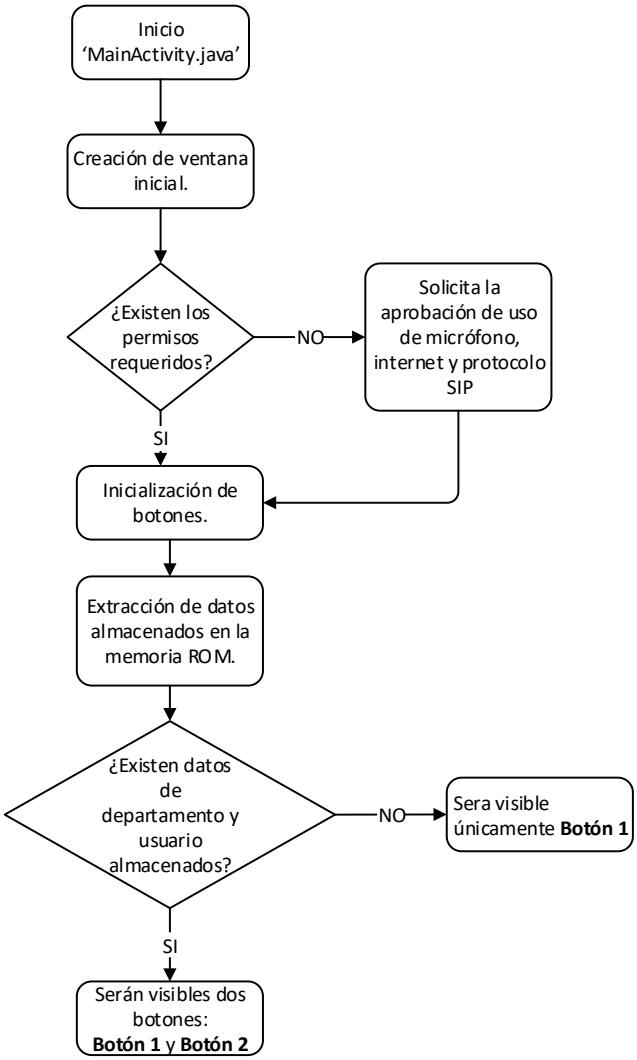

Figura 2.21. Diagrama de flujo de la actividad 'MainActivity.java'

Se empieza asociando el LAYOUT que se mostrará en esta actividad al usuario, para a continuación conocer si existen los permisos para poder usar el micrófono, internet y protocolo SIP. Si es la primera vez que se ejecuta la aplicación en el celular estos permisos no estarán establecidos por lo que se procederá a ingresar a una ventana auxiliar como se puede observar en la Figura 2.22., donde se solicita al usuario dar estos permisos, que lo puede realizar únicamente presionando aceptar en esta ventana para continuar.

Figura 2.22. Ventana auxiliar para solicitar permisos requeridos.

Una vez dados estos permisos o si los permisos ya se encontraban activos previamente, se procede a inicializar los botones de la actividad como objetos del siguiente modo:

```
FloatingActionButton buttonA1 = findViewById(R.id.ButtonA1);
ImageButton buttonA2 = findViewById(R.id.ButtonA2);
```

En primer lugar, identificando que tipo de botón es, en el primer caso 'FloatingActionButton' que es un botón que se caracteriza por ser circulares y encontrarse por encima de la interfaz de usuario, así como tener su propio comportamiento de movilidad en comparación al resto del layout [49]. Mientras que el segundo botón es un 'ImageButton' que permite colocar una imagen como botón en vez de solo texto como se podrá observar en otras actividades más adelante. A continuación, se procede a ingresar un nombre para identificar al botón, seguido de la función que permitirá asociar el ID del botón colocado en el archivo 'activity_main.xml'. Estos dos botones se pueden observar en la Figura 2.23.

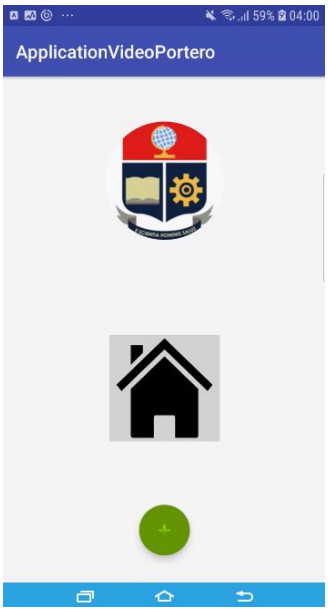

Figura 2.23. Interfaz de usuario de la actividad MainActivity.

Al tener las variables asociadas a los botones respectivos se procede a obtener las variables esenciales desde la memoria ROM si es que estas se encuentran almacenadas, para determinar si un usuario ya se había registrado en un uso previo de la aplicación. Estas variables son:

'DepartID'- identificación del departamento al que se encuentra registrado el usuario.

'DepartPassword'- es la contraseña asociada al identificador de departamento.

‘IpAddressRBpi’- la dirección IP de la Raspberry Pi para poder tener comunicación directa con esta.

‘Port1RBpi’- número de puerto TCP en el cual se mantiene escuchando la Raspberry Pi por mensajes entrantes.

‘UserName’- el nombre de usuario que se encuentra registrado.

Estos datos únicamente podrán ser almacenados en la memoria ROM si ya fueron correctamente procesados y autentificado en un previo uso de la aplicación.

Con estos datos se podrá determinar si es necesario desplegar el botón ‘buttonA2’ (como se puedo observar con anterioridad en la Figura 2.23) con el cual dará acceso directo a la actividad que permite autentificar al usuario. Por lo que mediante una condicional ‘IF’ (como se puede observar a continuación) se determina si alguna de estas variables se encuentra vacía, en cuyo caso no se permitirá la aparición del botón mencionado como se puede observar en la Figura 2.24., caso contrario si todas las variables se encuentran con datos almacenados, el botón ‘buttonA2’ será visible.

```
if (DepartID!= null && DepartPassword!=null &&  IpAddressRBpi!=null
&& Port1RBpi!=null && UserName!=null)

{

  buttonA2.setVisibility(View.VISIBLE);
}
```

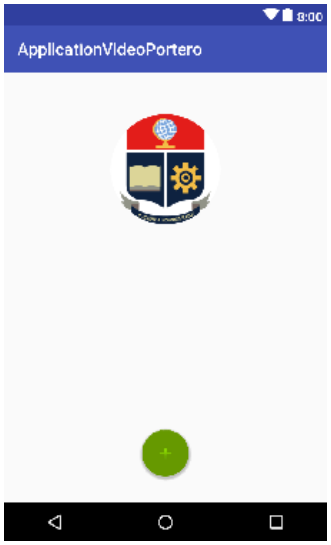

Figura 2.24. Interfaz de usuario MainActivity sin tener un usuario registrado

Una vez determinada la visibilidad de este botón se procede a determinar la acción que se procederá a realizar una vez que el usuario presione un botón determinado. Como por ejemplo el código a continuación.

```
buttonA1.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View v) {
          NextWinConfigData();
          OpenDepRegistrationLayout();
          }
});
```

se puede ver que al objeto 'buttonA1' se le invoca el método 'setOnClickListener()' con el parámetro 'View.OnClickListener()', esto permitirá que el código escrito dentro de la función 'onClick()' se ejecute en el momento que el botón es presionado, que en este caso ingresará a una función que permite determinar el valor de la variable 'VideoOrConfig' almacenándola en la memoria ROM, para proceder a la segunda función que dará paso a la actividad 'DepartmentDataRegistration.java', esto mediante le código que se puede observar a continuación.

```
Intent intent =new Intent(this,DepartmentDataRegistration.class);
startActivity(intent);
```

Para poder conocer claramente la acción que realizará cada botón el momento de ser presionados, se puede observar a continuación el diagrama de flujo de estos en la Figura 2.25.

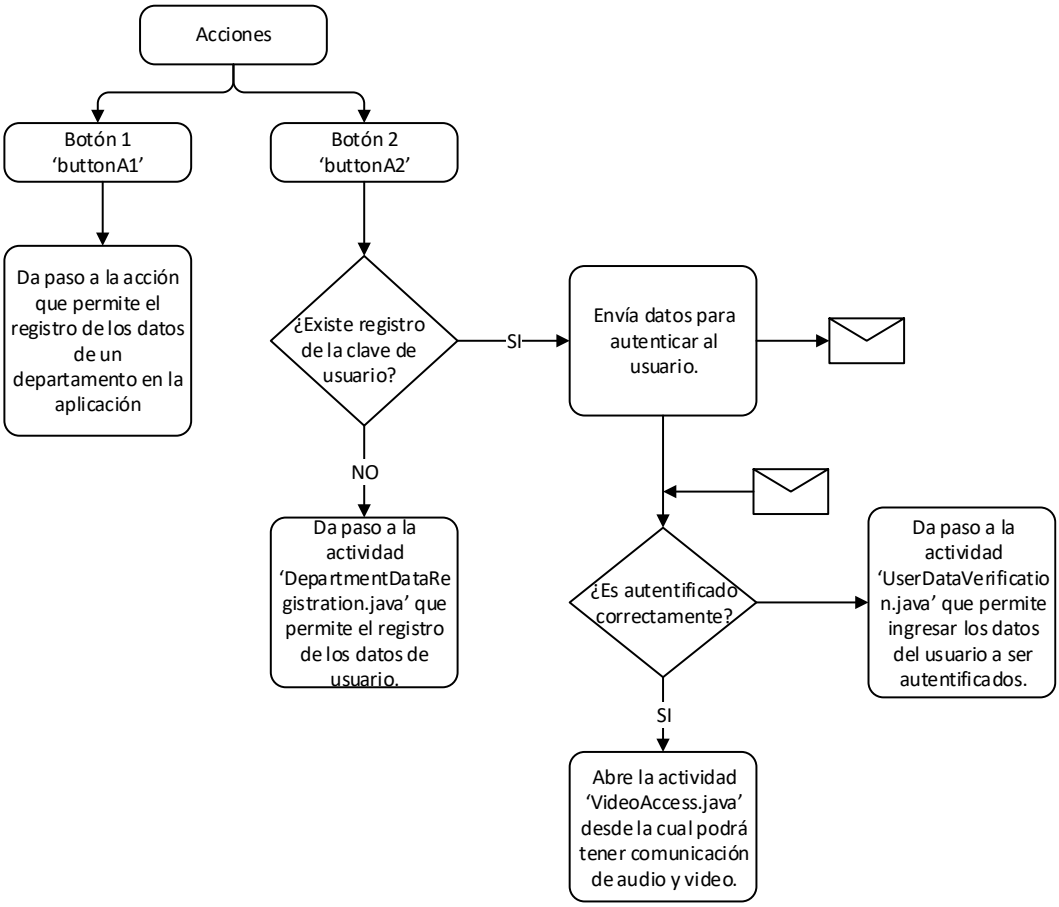

Figura 2.25. Diagrama de flujo de botones de la actividad 'MainActivity'.

Como se mencionó en el ejemplo anterior, el botón 1, nombrado en el código como 'buttonA1', al presionarlo permitirá el ingresar a una nueva actividad denominada 'DepartmentDataRegistration.java' desde la cual se registrarán los datos del departamento en el que se encuentra viviendo el usuario.

En el segundo botón denominado 'buttonA2' al ser presionado, en primer lugar, requiere saber si existe almacenada en la base de datos la calve de usuario, si es así, enviará un mensaje a la Raspberry Pi con la acción 'RECUSERACCESS' y todos los datos requeridos para la autentificación, de este modo en la microcomputadora procesará esta información como se mencionó previamente y enviará un mensaje afirmando o negando el acceso a este usuario, acceso que le permitirá ir directamente a la actividad 'VIdeoAccess.java' sin necesidad de ingresar nuevamente los datos de usuario. En el caso de ser negado el acceso se procederá a abrir la actividad 'UserDataVerification.java' desde donde se podrá ingresar el nombre y la clave del usuario previamente registrados y almacenados en la base de datos para proceder a la autentificación. Todo esto mediante el siguiente código;

```
if (SavePassword && UserPassword != null) {
    String[]
DepartEip={IpAddressRBpi,Port1RBpi,"ACCION","RECUSERACCESS","ID_DEP
",
            DepartID,"CLAVE_DEP",DepartPassword,"USER_NAME",
            UserName,"CLAVE_USER",UserPassword};
    new SendMessage().execute(DepartEip);
} else {
    NextWinConfigData();
    OpenUserDataVerificationLayout();
}
```

en donde la función 'sendMessage' será la encargada de enviar y recibir segundo plano la información requerida, así como discriminar las acciones de acuerdo a las respuestas obtenidas desde la Raspberry Pi.

2.7.2.1.1. Función 'sendMessage()'

Esta función es usada en varias de las actividades de esta aplicación y trabaja del mismo modo en todas estas, con la única diferencia en el tipo de acción tomada de acuerdo a la respuesta obtenida, por lo que será explicada únicamente en esta sección.

Para poder observar las acciones que cumple esta función se puede observar el diagrama de flujo en la Figura 2.26.

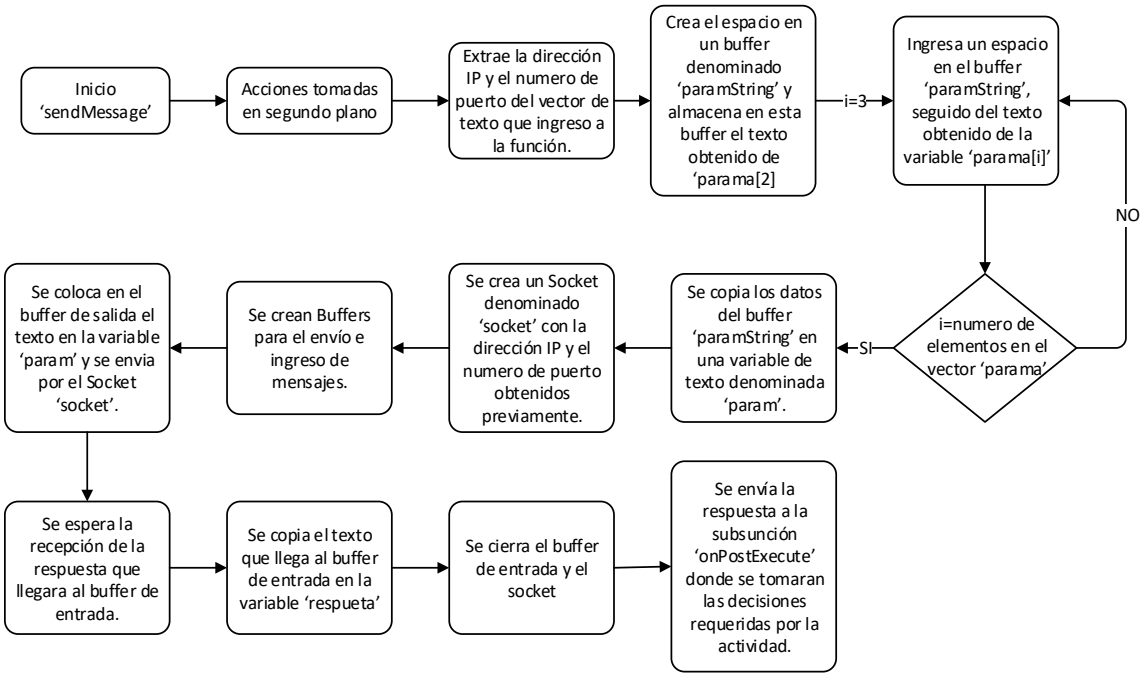

Figura 2.26. Diagrama de flujo de función 'sendMessage()'

La función 'sendMessage()' recibe como parámetro un vector de texto, donde se encuentra la dirección IP, el número de puerto TCP y los datos que serán enviados a la Raspberry Pi para su procesamiento. Todas las acciones tomadas por esta función se encuentran en

segundo plano por lo que la aplicación puede continuar con normalidad mientras se envía y espera una respuesta desde la microcomputadora, aunque esta es casi inmediata. Como se puede observar en el diagrama de flujo esta función empieza determinando que será ejecutada en segundo plano, y procede a extraer la dirección IP y Puerto TCP del vector de texto, ubicados en la variable 'params[0]' y 'params[1]' respectivamente, después de esto crea un espacio en la memoria RAM denominado Buffer, para poder almacenar los datos del vector de texto y después poder convertirlo en una sola variable de texto que será enviada. Para esto una vez creado el buffer se ingresa a un bucle FOR que permitirá tomar cada una de las variables del vector de texto y colocarla en orden en el buffer creado. Finalizado el bucle se procede a pasar los datos del buffer a una variable denominada 'param'. Con esto se culmina el tratamiento de los datos que serán enviados y se procede a alistar la forma que serán enviados.

Para poder tener comunicación mediante el protocolo TCP/IP en un tipo de comunicación cliente servidor, en primer lugar, el servidor debe de mantenerse escuchando para que pueda recibir los mensajes de los clientes y de este modo es como se mantiene la Raspberry Pi, en modo de escucha, ya que será el servidor en este caso. Por parte del cliente únicamente requiere crear y mantener activo un Socket durante la transmisión y en la espera de una respuesta, caso contrario no será necesario mantener activo. Para activar este Socket se lo realiza con el siguiente código.

```
Socket socket = new Socket(ipaddress, PORT);
socket.setSoTimeout(2000);
```

En donde la primera línea de código crea el socket con la dirección IP y el número de puerto TCP del servidor, mientras la segunda coloca un límite de tiempo que se mantendrá activa en la espera de una respuesta (en este caso 2000useg). Después de crear este socket creamos dos buffers para el envío y recepción de mensajes en este caso nombrados como 'out' e 'in' respectivamente y se los expresará del siguiente modo en el código.

```
BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));


PrintWriter out = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()));
```

Teniendo listos estos pasos se puede proceder a enviar el mensaje requerido mediante el protocolo TCP, y para esto solo es necesario una línea de comando:

```
out.print(param);
```

que enviará el texto de la variable ‘param’ a la dirección IP y puerto TCP determinado en el Socket. A continuación, se puede cerrar este buffer de salida y se lo realiza mediante el comando:

```
out.flush();
```

se lee el buffer de entrada una vez este ya tenga la respuesta del servidor, para después poder cerrar este buffer y cerrar el socket, todo esto de la siguiente forma:

```
String respuesta=in.readLine();
in.close();
socket.close();
```

Permitiendo enviar esta variable de respuesta al método que se ejecutará una vez culmine esta parte del programa, teniendo siempre como resultado en la variable ‘respuesta’ un texto plano que deberá ser procesado si es necesario.

2.7.2.2. DepartmentDataRegistration

Esta actividad permite que el usuario escriba la dirección IP y número de puerto que mantiene la Raspberry Pi activo, datos sin los cuales no se podrá comunicar con la Raspberry Pi, y del mismo modo tendrá que ingresar el identificador del departamento y la clave de este que ya se encuentren registrados en la base de datos de la Raspberry Pi, en el caso de ser la primera interacción con la Raspberry Pi, al departamento que se ingresará se lo identifica como ‘DEP_ADMIN’ con la contraseña predeterminada ‘password’ (que por seguridad se sugiere cambiar), mediante el uso de este el usuario que se registre en este departamento tendrá la autorización de aumentar el número de departamentos así como colocar las claves de estos a su propia discreción, proceso que se verá más adelante.

El código JAVA de la programación de esta actividad se encuentra en el ANEXO O, mientras que el código XML del Layout se encuentra en el ANEXO P.

El procedimiento inicial de esta actividad es estrictamente lineal sin la necesidad de aumentar o eliminar botones, por lo que siempre se mantendrá con la misma interfaz de usuario (UI por las siglas en inglés User Interface). Una vez culmine el proceso de importación de librerías, inicialización de variables, inicialización de los botones la UI se la podrá observar cómo se muestra en Figura 2.27.

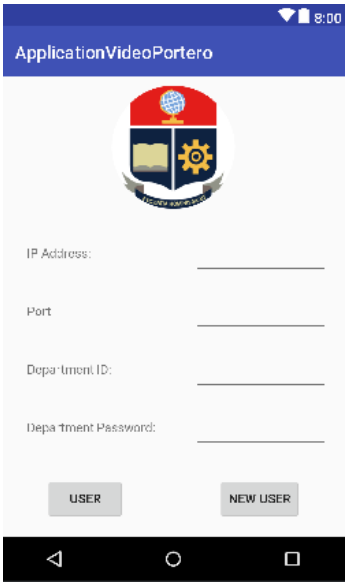

Figura 2.27. UI de la actividad 'DepartmentDataRegistration'

Por otro lado, una vez se presione uno de los botones se tendrá una serie de acciones que determinará cual será la siguiente actividad a ser abierta, por lo que se presenta en la Figura 2.28, el diagrama de flujo de estos botones.

El botón 3, nombrado como 'buttonB1' en el programa JAVA permite que un usuario ya existente en la base de datos pueda acceder directamente a la actividad que le permite autentificar sus datos, mientras que el botón 4 denominado como 'buttonB2' permite a un nuevo ingresar a la actividad de registro para que pueda registrar su nombre de usuario y clave que serán almacenados en la base de datos. Para esto se tendrán que cumplir ciertas condiciones que se podrán ver a continuación.

Al presionar el botón 'buttonB1' coloca la variable booleana 'NewOldUser' como verdadero, esta variable determinará más adelante que botón es el que se presionó, en este caso el botón representa que se registrará a un nuevo usuario. En cambio, sí se presiona el botón 'buttonB2' se colocará esta variable booleana 'NewOldUser' como falso, representando que el usuario ya se encuentra registrado, a continuación de este se almacenará en la memoria ROM una variable booleana 'VideoOrConfig' el valor de verdadero, esta variable determina si después de verificar los datos de la actividad 'UserDataVerification.java', la siguiente actividad de esta sea 'VideoAccess.java' o una de las actividades que servirá para configurar los datos de usuario.

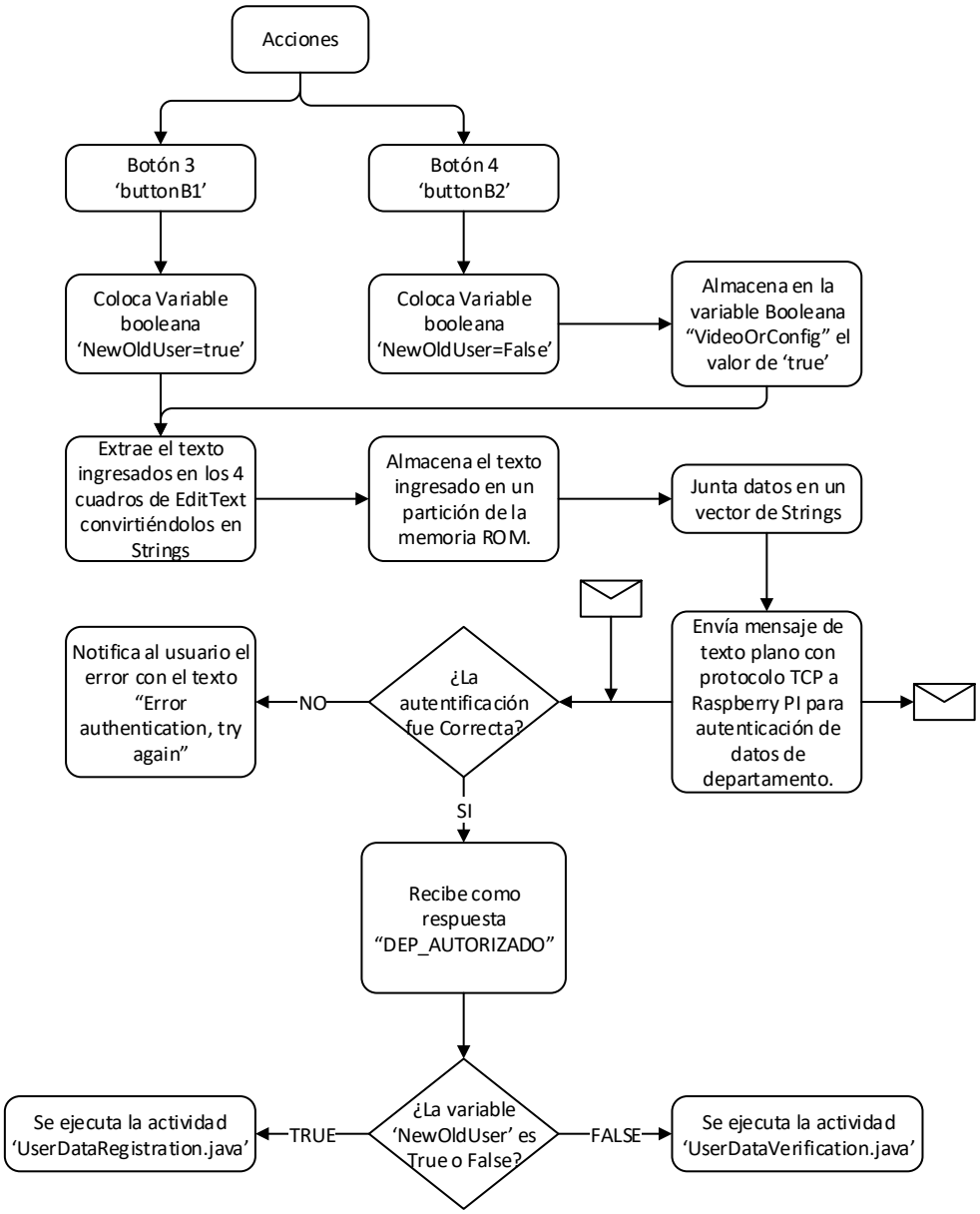

Figura 2.28. Diagrama de flujos de los botones de la actividad 'DepartmentDataRegistration'.

Una vez presionado uno de los botones y colocado el valor de la variable 'NewOldUser' las dos opciones tendrán el mismo procedimiento que se detalla a continuación.

Se extrae el texto de los 4 EditText, donde debería estar colocado la dirección IP y número de Puerto de la Raspberry Pi con la que desea comunicarse, el identificador y la clave del departamento. Después de esto los datos son almacenados en el espacio de memoria ROM correspondiente a cada uno de estos datos, como, por ejemplo:

```
SharedPreferences
IPAddressRBpi=getSharedPreferences(MainData,MODE_PRIVATE);
```

```
SharedPreferences.Editor editor=IPAddressRBpi.edit();
editor.putString("IpAddressRBpi", DepIpAddress);
editor.apply();
```

Que en la primera línea crea o identifica el espacio de memoria para el puntero 'IPAddressRBpi', la segunda línea apunta al espacio correspondiente para empezar la edición de esta, precediendo a ingresar el dato correspondiente para culminar guardando lo editado. Esto sucederá cada vez que se desee almacenar un valor en la memoria ROM, cambiando la línea 3 si es una variable booleana o numérica.

Ya que las variables se encuentren almacenadas se procede a enviar estos valores a la Raspberry Pi, colocando estos en un vector de textos y ejecutando la función 'SendMessage()', ejecutándose como se mencionó anteriormente.

```
String[]
DepartEip={DepIpAddress,Port1RBpi,"ACCION","RECDEPACCESS","ID_DEP",
        DepID,"CLAVE_DEP",DepPassword,"USER_NAME","CLAVE_USER"};


new SendMessage().execute(DepartEip);
```

una vez recibida la respuesta, se compara si es la respuesta esperada en una condicional 'IF', esperando como respuesta 'DEP_AUTORIZADO' si todo salió correctamente, y otro valor como 'FALSO1' si hubo un error en la autentificación. De ser correcta esta autentificación se procede a determinar cuál botón fue el que se presionó, con una condicional 'IF' preguntando si la variable booleana 'NewOldUser' es verdadero o falso, como se mencionó anteriormente si es verdadero significa que el usuario es nuevo y se dirigirá a la actividad 'UserDataRegistration.java' mientras q si este es falso significa que ya existe un usuario registrado por lo que se abrirá la actividad 'UserDataVerification.java', actividades que se podrán observar más adelante.

2.7.2.3. UserDataRegistration

El propósito principal de esta actividad es enviar a la Raspberry Pi el nombre y la clave del usuario para que se registre en la base de datos en el departamento respectivo, pero también puede configurar el nombre de usuario y la clave es este, la acción que tome dependerá de las condiciones que se cumplan o no. El inicio de la actividad en cualquiera de los dos casos es similar y variará una vez se presione el único botón para continuar. Para esto se puede observar el código JAVA de esta actividad en el ANEXO Q mientras que en el ANEXO R se puede observar el código XML que plantea la interfaz de usuario. Nuevamente el inicio de esta actividad es bastante básico (y se puede observar el diagrama

de flujo de este en la Figura 2.29.), que aparte de inicializar las respectivas librerías, variables y botones, permite recuperar el nombre de usuario que se encuentra en la memoria ROM y colocarlo en el EditText denominado 'NewUserEdit' mediante el siguiente código:

```
if(UserName!=null){
  NewUserNameEdit.setText(UserName);
}
```

de no existir un registro de usuario anterior en este dispositivo celular no realizará ninguna acción y procederá a definir la acción del botón.

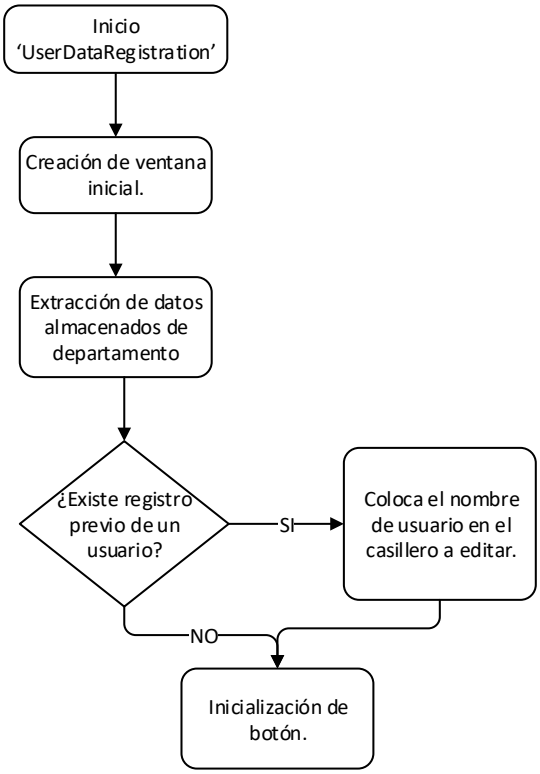

Figura 2.29. Diagrama de flujo inicial de la actividad 'UserDataRegistration.java'

Por otro lado, el proceso que se desarrolla una vez se presiona el botón es un poco más complejo ya que debe distinguir desde que actividad llega para poder saber que acción tomar. Para esto se puede observar el diagrama de flujo en la Figura 2.30.

Al presionar el único botón que tiene esta actividad empieza obteniendo los textos de las tres EditText y colocándolos en variables. Se procede a comparar las variables que contienen la clave que el usuario desea usar y se compara entre sí para verificar que el usuario tenga claro la clave que está colocando, si en la comparación de estas dos claves se determina que no son similares se le anunciará al usuario el error que está cometiendo

para que corrija la falla antes de proceder. Pero si, por otro lado, se encuentra correcto, procede a verificar que tipo de acción es la que va a proceder. Esto con la variable booleana almacenada en la memoria ROM 'VideoOrConfig' que se extrajo en la inicialización del programa, mediante esta variable que es configurada desde otras actividades se determina si la acción a tomar será el añadir los datos de usuario a la base de datos si la variable es verdadera o editar los datos de un usuario cambiándolos por los que se encuentran registrando en esta actividad si la variable es falsa.

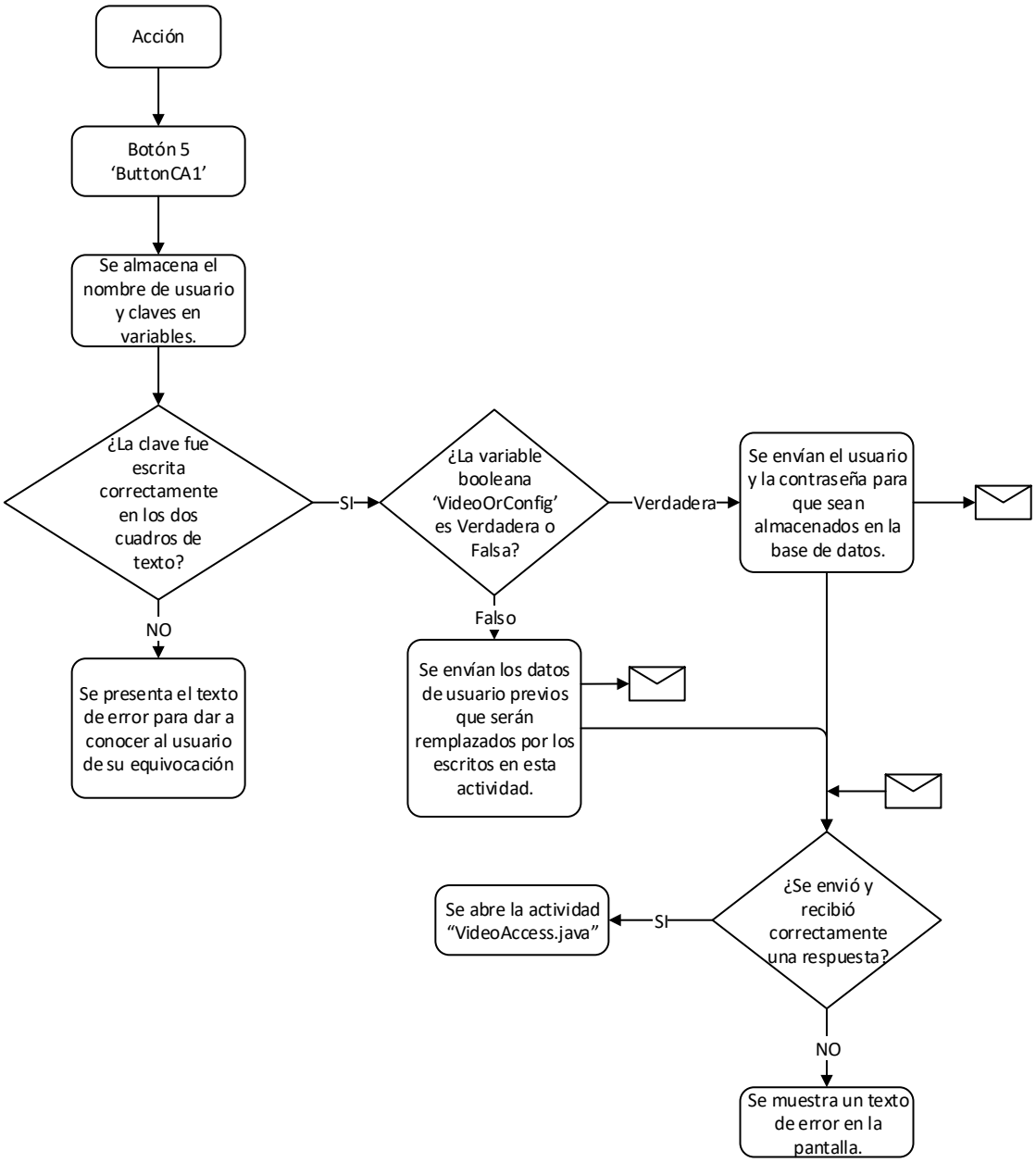

Figura 2.30. Diagrama de flujo de botón 'ButtonCA1' en actividad 'UserDataRegistration.java'.

En cualquier caso, se envía un mensaje a la Raspberry Pi con el nombre de la acción a tomarse y se recibida una respuesta si se realizó correctamente la solicitud que será 'USUARIO_REGISTRADO' que permitirá avanzar a la actividad 'VideoAccess.java', y de no ser esta la respuesta recibida desde la Raspberry Pi o de tener un error de comunicación se presentará en el TextView 'Autoritation' la frase 'Wrong basic information' manteniéndose en la misma actividad en la espera de corregir los datos para enviar nuevamente la información.

2.7.2.4. UserDataVerification

Esta actividad tiene como objetivo principal autentificar al usuario, enviando los datos colocados en esta actividad a la Raspberry Pi y comparándolos con la base de datos. De este modo el usuario podrá avanzar a la siguiente actividad que puede ser 'VideoAccess.java', 'AdminConfiguration.java' o 'UserDataRegistration.java' dependiendo de las distintas variables que pueden afectar a este proceso. Para esto se puede observar el código Java de esta actividad en el ANEXO S, así como el código XML del Layout en el ANEXO T que da la interfaz de usuario sin ser editada por el programa JAVA y se la puede observar en la Figura 2.31.

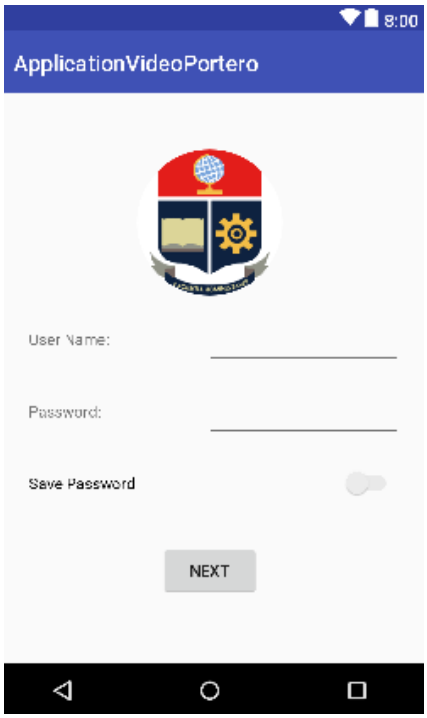

Figura 2.31. UI de la actividad 'UserDataVerification'.

Como se puede observar en el diagrama de flujo de la inicialización de esta actividad en la Figura 2.32, mantiene las mismas características que el inicio de la actividad anterior.

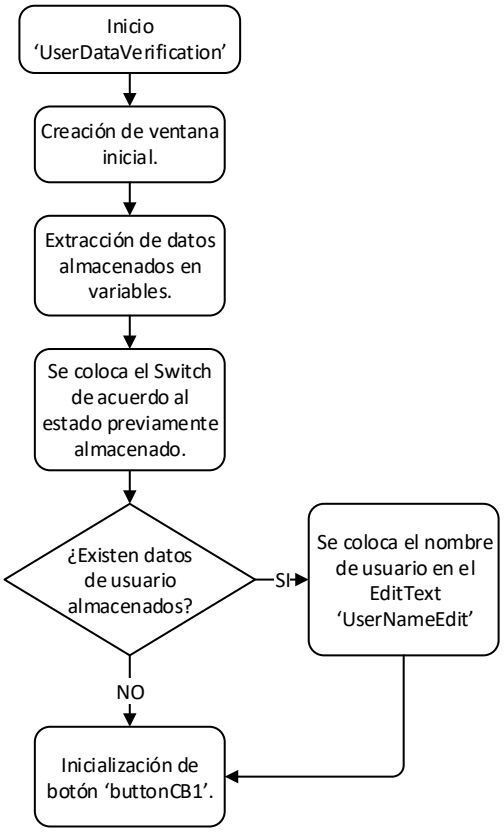

Figura 2.32. Diagrama de flujo de inicio de actividad 'UserDataVerification.java'.

Con la diferencia que el Switch nombrado como 'passwordSwitch' será colocado en la posición requerida, de acuerdo a la variable booleana extraída de la memoria ROM.

La programación del botón por otro lado es muy diferente, ya que tendrá que enviar la información a ser verificada y recibir una respuesta, dependiendo de si requiere únicamente acceso a la actividad 'VideoAccess.java' o si requiere tener acceso a configurar los datos, donde se tendrá como respuesta que tipo de usuario es, para conocer qué actividad de configuración tendrá que abrirse a continuación. Para esto se puede observar el diagrama de flujo en el ANEXO U donde se ve las acciones se pueden tomar cuando se presiona el botón, así como la acción al cambiar de posición el Switch.

2.7.2.4.1. PasswordSwitch

Este Swittch permite al usuario decidir si desea que su clave se almacene en el celular y de este modo pueda acceder directamente a la actividad 'VideoAccess.java' donde se podrá comunicar con el VideoPortero. Cada vez que cambia de posición del switch la variable booleana 'SavePassword' cambiará de acuerdo a la posición del switch. Como se puede ver en la Figura 2.31 en donde este switch se encuentra apagado el valor de la

variable ‘SavePassword’ sería falso mientras que en la Figura 2.33. se puede observar que el Switch se encuentra encendido, el valor de esta misma variable sería Verdadero, una vez presionado el botón ‘buttonCB1’, esta variable será almacenada en la memoria ROM para que el momento de encender la aplicación se conozca cual fue la última decisión tomada por el usuario y así pueda acceder directamente a la actividad principal,

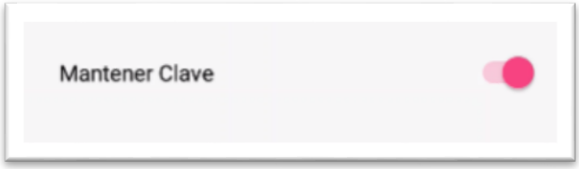

Figura 2.33. Switch en posición de encendido para almacenar la clave de usuario.

2.7.2.4.2. ButtonCB1

La principal acción de este botón es enviar el nombre y clave del usuario a ser autentificados en la Raspberry Pi, pero también tendrá que enviar qué tipo de acción es la que desea realizar el usuario, ya que esta actividad autentifica al usuario para ingresar a la actividad ‘VideoAccess.java’ o a una de las actividades que permitirá al Administrador o usuario configurar información en la base de datos, esta discriminación se la realizará del siguiente modo.

En primer lugar, se almacenan en variables determinadas el nombre y la clave de usuario que se encuentran escritas en los EditText correspondientes,

```
UserName=UserNameEdit.getText().toString();
UserPassword=UserPasswordEdit.getText().toString();
```

después de esto se almacenarán estos datos en la memoria ROM junto con la variable ‘SavePassword’ ocupando dos funciones en las que se encuentra el código para realizar estas acciones.

```
SafeUserData();
PermanentPassword();
```

Ya almacenados estos datos se procede a determinar si la siguiente actividad será para acceder al video portero o para realizar alguna configuración, esta decisión la toma usando una condicional ‘IF’ a con la variable booleana ‘VideoOrConfig’, variable que será editada desde una actividad previa, es decir que si el usuario llegó a esta actividad desde la

‘MainActivity.java’ o desde ‘DepartmentDataRegistration.java’ el valor de esta variable será VERDADERO, pero si se ingresa desde la actividad ‘VideoAccess.java’ esto quiere decir que el usuario presionó el botón de configuración por lo que el valor de esta variable será FALSA .

```
if (VideoOrConfig) {

  String[]
DepartEip={IpAddressRBpi,Port1RBpi,"ACCION","RECUSERACCESS",

"ID_DEP",DepartID,"CLAVE_DEP",DepartPassword,"USER_NAME",UserName,

"CLAVE_USER",UserPassword};



new SendMessage().execute(DepartEip);

}else{

  String[]
DepartEip={IpAddressRBpi,Port1RBpi,"ACCION","CONFIG","ID_DEP",
            DepartID,"CLAVE_DEP",DepartPassword,"USER_NAME",
            UserName,"CLAVE_USER",UserPassword};
    new SendMessage().execute(DepartEip);
}
```

Tomando en cuenta esto al ser el valor de la variable ‘VideoOrConfig’ es VERDADERA, se enviarán los datos correspondientes junto con la acción ‘RECUSERACCESS’, acción que la Raspberry reconoce como verificación de datos para dar acceso al usuario al VideoPortero, dando como respuesta ‘USER_AUTORIZADO’ si el usuario fue autentificado correctamente y permitiendo al usuario avanzar a la actividad ‘VideoAccess’ o ‘FALSE1’ si existió un error en la autentificación de este usuario, indicando al usuario la falla con la frase ‘Access not granted’ y manteniendo en la espera de la corrección de los datos y que el usuario presione nuevamente el botón..

Por el otro lado, si la variable ‘VideoOrConfig’ tiene el valor de FALSO, se enviará la información requerida junto con la acción ‘CONFIG’ donde la programación de la Raspberry Pi determinará qué tipo de usuario es la persona que envió esta solicitud. Teniendo en cuenta los tres tipos de usuarios que se mencionó en el capítulo 2.7.1 de esta tesis. Del mismo modo, si existe un error en la autentificación de los datos enviados se recibirá un mensaje de error y se mostrará al usuario el mensaje ‘Access not granted, chek the user name and password’, pero después de autentificar correctamente e identificar el tipo de usuario se recibirá como respuesta una de estas tres opciones que tendrán sus respectivas acciones.

‘USER’- al recibir esta respuesta se determina al usuario como un residente normal del departamento al que se encuentra registrado por lo que la siguiente actividad que se abrirá mediante la función ‘ConfigUsuario()’ (que se puede observar a continuación), será ‘UserDataRegistration.java’ donde junto con la variable booleana ‘VideoOrConfig’ en FALSO podrá cambiar su nombre y clave de usuario.

```
public void ConfigUsuario(){
    Intent intent = new Intent(this, UserDataRegistration.class);
    startActivity(intent);
}
```

‘ADMIN_USER’- mediante la función ‘ConfigAdmin(DepAdm_Ad)’ que tiene como entrada una variable booleana, en este caso el valor de ‘DepAdm_Ad’ será FALSO y este valor se almacenará en la memoria ROM para ser transferida a la siguiente actividad que será ‘AdminConfig.java’, actividad que ocupará esta variable booleana para realizar las configuraciones correspondientes.

‘ADMIN’- de igual forma al ‘ADMIN_USER’ se ejecutará la función ‘ConfigAdmin(DepAdm_Ad)’ pero en este caso el valor en la variable ‘DepAdm_Ad’ será VERDADERO, y del mismo modo será almacenado, para posterior a esto abrir la actividad ‘AdminConfig.java’. La función ‘ConfigAdmin(DepAdm_Ad)’ se la puede observar a continuación.

```
public void ConfigAdmin(Boolean DepAdm_Ad){
    SharedPreferences
setting=getSharedPreferences(MainData,MODE_PRIVATE);
    SharedPreferences.Editor edit=setting.edit();
    edit.putBoolean("DepAdm_Adm",DepAdm_Ad);
    edit.apply();

    Intent intent = new Intent(this, AdminConfiguration.class);
    startActivity(intent);
}
```

2.7.2.5. AdminConfiguration

Esta actividad permite al usuario categorizado como administrador general o administrador de un departamento obtener información desde la base de datos y editarla, como se mencionó en el capítulo 2.7.1. El código JAVA de esta actividad se encuentra en el ANEXO V, mientras que el código XML del Layout se encuentra en el ANEXO W. Ya que la UI inicial cambiará de acuerdo al tipo de usuario, así como la información que despliegue en la lista que será mencionada más adelante, se detallará el proceso inicial de esta actividad y los

dos resultados posibles al culminar la configuración inicial de la actividad. Esto basado en el diagrama de flujo que se muestra en la Figura 2.34.

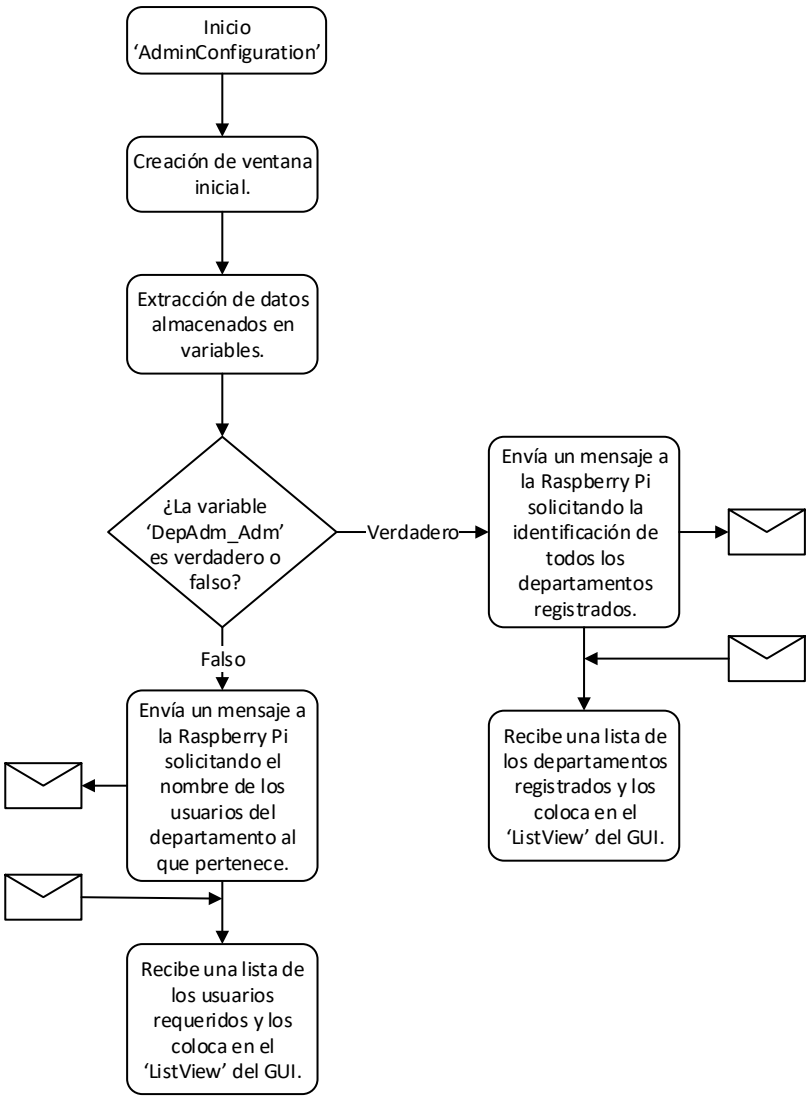

Figura 2.34. Diagrama de flujo de la inicialización de la actividad 'AdminConfigurarion'.

En primer lugar, al igual que las otras actividades se inicializan las librerías necesarias, así como las variables y se extraen los datos almacenados en la memoria ROM. Uno de los datos principales para esta sección del programa será la variable booleana 'DepAdm_Adm', variable que mediante una condicional 'IF' permite el saber qué tipo de información solicitar a la Raspberry Pi.

Debido a esto si la variable 'DepAdm_Adm' es verdadera se enviará a la Raspberry Pi un mensaje solicitando recibir los nombres de los departamentos que se encuentran registrados en la base de datos, esto se lo realiza enviando como acción

‘ASKDEPIDDATA’, y de estar correcto toda la autentificación se obtendrá como respuesta la lista de departamentos que se encuentran registrados en la base de datos, con la palabra ‘LISTA’ encabezando la lista.

Esta lista llega como un solo texto plano y deberá ser dividida en un vector para extraer las variables deseadas. Esto se lo puede hacer con la siguiente línea de código.

```
String[] MSJ1= MSJ.split(",");
```

ya separada la lista en el vector de textos se puede extraer el primer texto que será la respuesta a ser comparada y así el programa podrá tomar la decisión de acuerdo a esta, en este caso al recibir como respuesta la palabra ‘LISTA’ se procede a ejecutar la función ‘List(MSJ1)’

```
public void List(String[] MSJ1){
    List1=findViewById(R.id.List1);
    ArrayAdapter<String> adapter=new
ArrayAdapter<>(this,R.layout.text,MSJ1);
    List1.setAdapter(adapter);
    List1.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view,
int position, long id) {
            String DEPDATA=(String)
List1.getItemAtPosition(position);
            DepDATA.setText(DEPDATA);
        }
    });
}
```

Función con la que se podrá desplegar los departamentos obtenidos en el vector de texto, adicionalmente dentro de esta función se especifica la acción a tomar si se selecciona un artículo en esta lista, que es ingresar el nombre del artículo seleccionado en el EditView ‘DepDATA’. Y podremos observar a la interfaz de usuario como en la Figura 2.35.

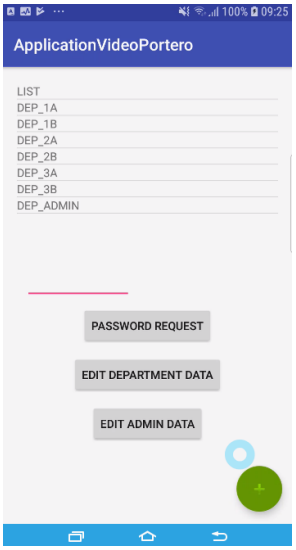

Figura 2.35. UI de la actividad 'AdminConfiguration' para un usuario administrador general.

En cambio, al ser la variable 'DepAdm_Adm' falsa se envía a la Raspberry Pi un mensaje con la acción 'CONFIG' y adicional a este el texto 'USER_OF_DEP', con esto la programación en Python entenderá que se está requiriendo la lista de usuarios del departamento en el que se encuentra el usuario registrado. Por lo que si todo sale correctamente se recibirá una lista de usuarios que pertenecen al mismo departamento que el usuario que realizó la solicitud, y encabezando esta lista el texto 'DEPARTMENT_USER' para que al tener esta lista como un vector de texto se pueda extraer el primer ítem y proceder de acuerdo a esto, por lo que en este caso cambiará toda la interfaz de usuario, del siguiente modo:

Se retira el LinearLayout denominado 'LinearLayout2'.

Se retira el botón flotante nombrado 'NewDepartment'.

Se retira el botón 'DepPassw'.

Se retira el botón 'EditAdmin'.

Se cambia la etiqueta visible por el usuario y el propósito del botón 'AskPassword', de 'PASSWORD REQUEST' a 'Change department password'.

Se cambia la etiqueta visible por el usuario y el propósito del botón 'EditDepartmentData' de 'Delete Department' a 'Edit Department Admin Data'.

Se vuelve visible el Linear Layout denominado 'LinearLayout1.

Se vuelve visible el botón nombrado ‘Delete_user’.

La variable booleana ‘List1OrList2’ se le coloca el valor de ‘FALSO’.

Después de todos estos cambios en la interfaz de usuarios se procede a ejecutar la función ‘List(MMSJ1)’ que coloca visible la lista de usuarios del mismo modo que se realizó anteriormente. Para culminar con un mensaje instantáneo que dirá "The configuration was successful.". y la interfaz de usuario se verá como se muestra en la Figura 2.36.

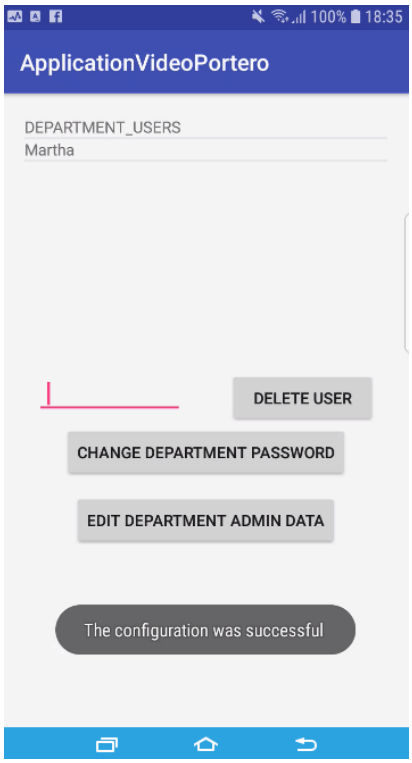

Figura 2.36. UI de la actividad ‘AdminConfiguration’ para un usuario administrador de departamento.

Una vez observando las diferentes interfaces de usuario que se pueden observar en esta actividad de acuerdo al tipo de usuario, se puede dar paso a conocer que acciones van a realizar estos botones dependiendo del estado de las variables que se encuentran en esta actividad. Esto se lo puede ver en los diagramas de flujo de cada uno de los botones que se presentarán individualmente y se explicarán con detalle a continuación.

2.7.2.5.1. AskPassword

Este botón tiene dos funciones dependiendo de las variables booleanas que se encuentren con valor verdadero o falso, para esto se puede observar el diagrama de flujo en la Figura 2.37. que permite observar de mejor forma el uso de este botón.

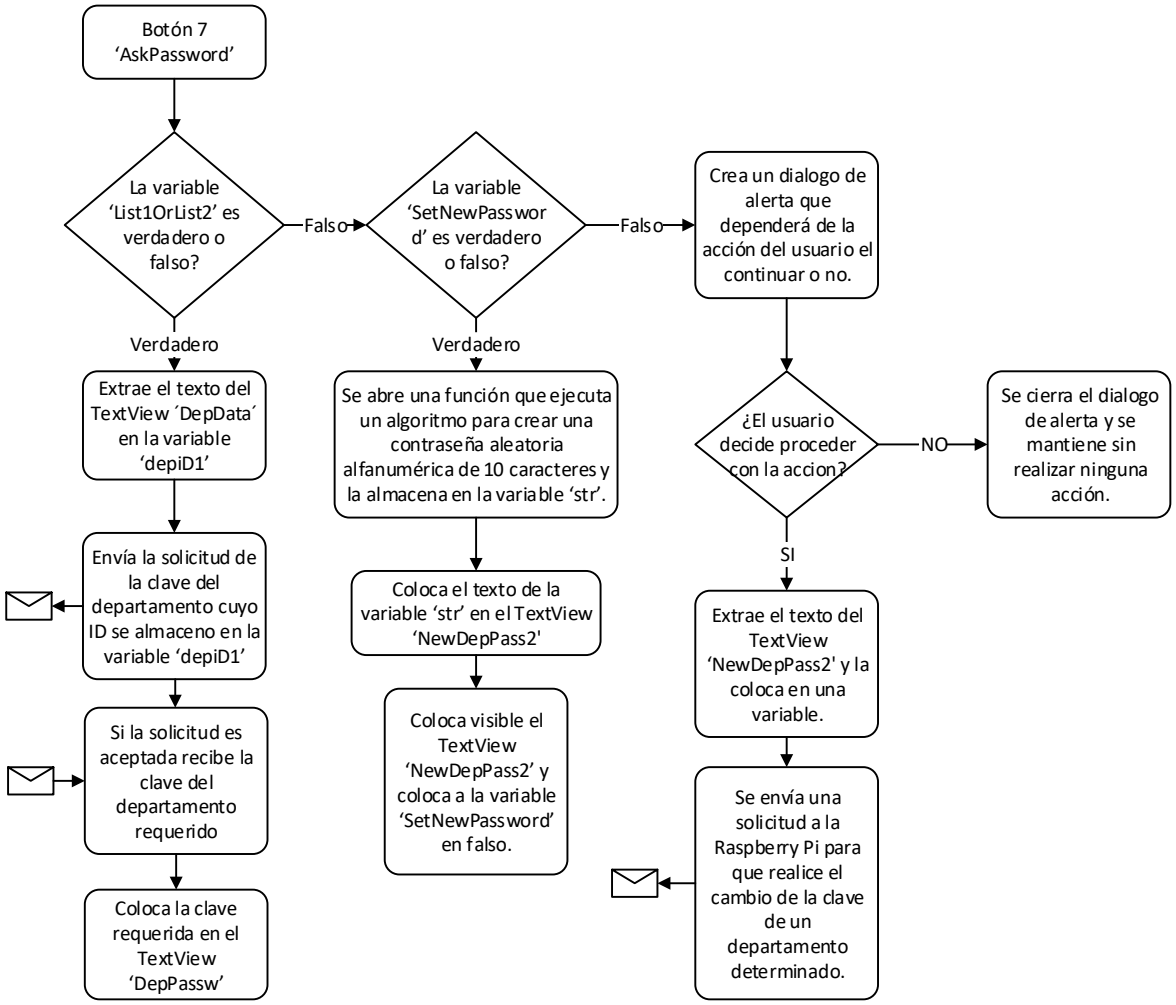

Figura 2.37. Diagrama de flujo del botón 'AskPassword'.

Después de presionar este botón, lo primero que se realiza es averiguar qué tipo de acción va a tomar dependiendo de la variable booleana 'List1OrList2'. Algo básico que se va a poder observar en toda la actividad 'AdminConfituration', es que esta variable será verdadera únicamente para el usuario que ingrese como administrador general.

Si la variable 'List1OrList2' es VERDADERA quiere decir que la interfaz de usuario se encuentra como se puede observar en la Figura 2.35. y en primer lugar se obtendrá el texto que se encuentra en el EditText 'DepDATA' donde debe de estar escrito el nombre de uno de los departamentos ya registrados en la base de datos y se enviará este dato junto con la acción 'ASKDEPPASSWORD', información con la que la Raspberry entiende que debe enviar la contraseña del departamento correspondiente. Al momento de recibir la respuesta que llegará de igual forma una lista en un solo texto que será separado como se mencionó anteriormente, dejando como resultado un vector de texto con dos elementos, el primero que es asignado para conocer la acción que se deberá tomar en este caso es

‘PASSWORDIS’, mientras que la segunda es la clave solicitada, la cual será colocada en el TextView ‘DepPassw’ culminado así las acciones de este botón.

Pero si, al contrario, la variable ‘Lisr1OrList2’ es FALSA la interfaz de usuario cambio y tendrá dos opciones, dependiendo si el usuario es administrador general o administrador de un departamento específico, en el primer caso la interfaz se verá como en la siguiente Figura 2.38 interfaz a la que se llegará escogiendo un departamento para editar, que se verá más adelante.

Figura 2.38. UI del administrador general.

Mientras que la interfaz de usuario del administrador de un departamento se la puede observar como se muestra en la Figura 2.36.

Por lo que este botón actúa con su segunda función, en ambos casos para cambiar de contraseña al departamento correspondiente. Por lo que se presentará una nueva variable booleana a ser evaluada, esta es ‘SetNewPassword’ que determinará cuantas veces se presionó este botón, ya que la primera vez, es decir cuando esta variable es VERDADERA, se crea un texto de 10 dígitos alfanuméricos aleatorios mediante la función ‘randomPassword()’ para después colocar este texto en el EditText llamado ‘NewDepPassw2’, hace visible el RelativeLayout ‘NewDepPasswL’ que contiene este texto e inmediatamente después se colocará esta variable ‘SetNewPassword’ en FALSO. Pasos que se pueden observar en el código siguiente.

```
String str=randomPassword();
NewDepPassw2.setText(str);
NewDepPasswL.setVisibility(View.VISIBLE);
SetNewPassword=false;
```

En cambio, una vez la variable ‘SetNewPassword’ sea FALSO, significa que el botón es presionado por segunda vez, por lo que se procederá a realizar el cambio de la clave del departamento, pero no sin antes tener un mensaje emergente advirtiendo al usuario lo que está a punto de hacer, donde el usuario tendrá la opción de cancelar este proceso presionando NO, lo que anulará la acción y se mantendrá en la espera de que el usuario realice algo más o de continuar con el cambio presionando SI dando permiso al cambio de esta contraseña y se enviará un mensaje a la Raspberry Pi con la acción ‘CONFIG’ adicional del texto ’CHANGE_DEP_PASS’ seguido del nombre del departamento que se desea cambiar la clave y a continuación de este el texto ‘NEW_DEP_PASS1’ seguido de la nueva contraseña, estos datos adicionales a los requeridos usualmente para la autentificación, por lo que los datos que se enviarán se puede ver a continuación.

```
String[] DepartEip = {IpAddressRBpi, "5005", "ACCION", "CONFIG",
"ID_DEP",DepID, "CLAVE_DEP",DepartPassword, "USER_NAME", UserName,
"CLAVE_USER", UserPassword,"CHANGE_DEP_PASS",depiD,"NEW_DEP_PASS1",

NewPass};
```

Con estos datos la programación en Python procesará la información y cambiará la clave del departamento para todos los usuarios que residan en este, y al culminar esta acción la respuesta que recibirá el celular Android dependerá de que tipo de usuario envió el mensaje, si lo envió el administrador general tendrá como respuesta ‘DEPARTMENT_UPDATA’ con lo que culminará regresando la ventana de configuración inicial, observada en la Figura 2.35. Pero si la envió el administrador de un departamento específico la respuesta será ‘DEPARTMENT_USERS’ y la interfaz de usuario se mantendrá en la misma que se puede observar en la Figura 2.36.

2.7.2.5.2. EditDepartData

En el caso de este botón dependiendo de las variables booleanas puede realizar una de tres acciones, para el administrador general actuará para solicitar los nombres de usuario de un departamento específico o para eliminar un departamento seleccionado, en el caso de ser un administrador de un departamento permitirá al usuario avanzar a otra actividad donde podrá cambiar su nombre y clave de usuario, el funcionamiento de este de lo puede observar en el diagrama de flujo que se encuentra en la Figura 2.39.

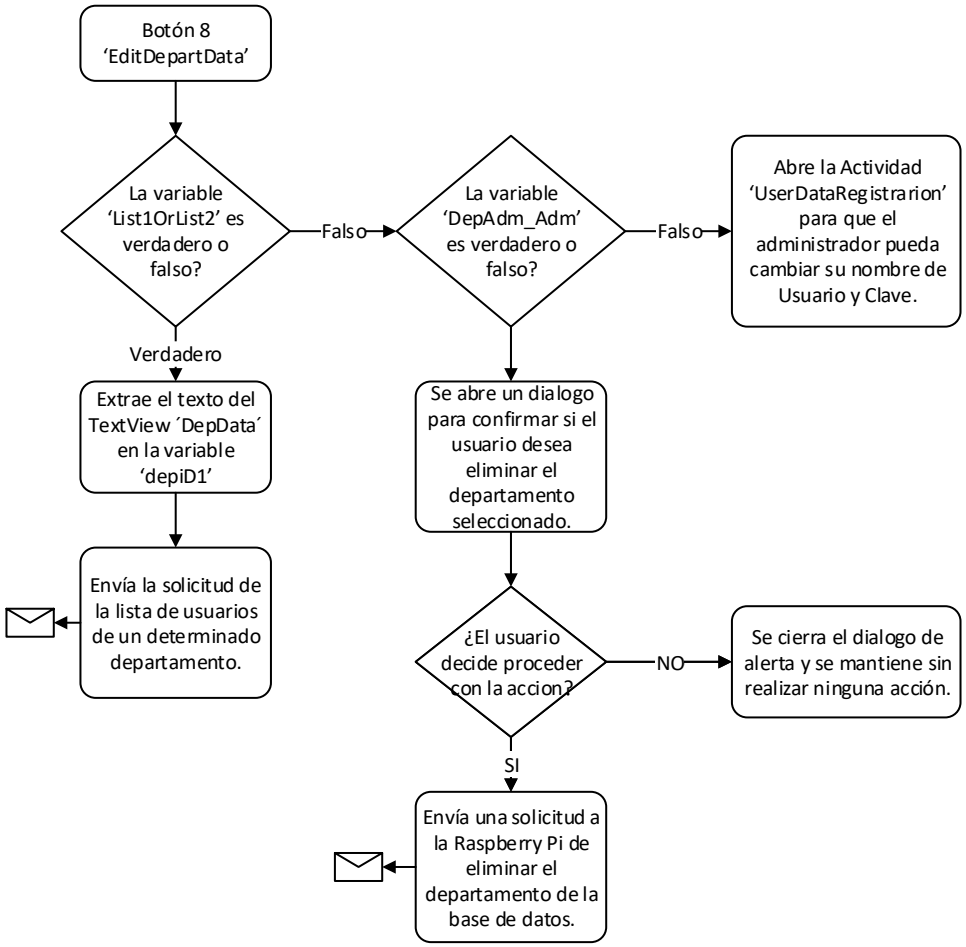

Figura 2.39. Diagrama de flujo del botón 'EditDepartData'.

Al presionar este botón inicia al igual que el capítulo 2.7.2.5.1 procesando la variable 'List1OrList2'. Si esta es VERDADERO se procederá a extraer el texto del EditText 'DepDATA' en el que debe de estar el nombre de un departamento y colocarlo en la variable 'depiD' para enviar este dato en un mensaje con el propósito de solicitar la lista de nombres de usuario que tiene este departamento. Una vez enviado el mensaje y procesada la información en la Raspberry Pi, la respuesta será una lista en una sola variable de texto que será convertida en un vector de texto y empezará con 'DEPARTMENT_USERS' seguido de la lista de nombres de usuarios solicitados. Esto permitirá cambiar la interfaz de usuario del siguiente modo:

Se retira el LinearLayout denominado 'LinearLayout2'.

Se retira el botón flotante nombrado 'NewDepartment'.

Se retira el botón 'DepPassw'.

Se retira el botón 'EditAdmin'.

Se cambia la etiqueta visible por el usuario y el propósito del botón ‘AskPassword’, de ‘PASSWORD REQUEST’ a ‘Change department password’.

Se cambia la etiqueta visible por el usuario y el propósito del botón ‘EditDepartmentData’ de ‘Edit Department Data’ a ‘Delete Department’.

Se vuelve visible el Linear Layout denominado ‘LinearLayout1.

Se vuelve visible el botón nombrado ‘Delete_user’.

La variable booleana ‘List1OrList2’ se le coloca el valor de ‘FALSO’.

Y nuevamente se ejecutará la función ‘List(MSJ1)’ mencionada ya en el inicio de la actividad. Esto dará como resultado tener la interfaz de usuario vista ya en la Figura 2.35.

Pero si la variable ‘List1OrList2’ es FALSO procederá a procesar la variable booleana llamada ‘DepAdm_Adm’, si esta variable es FALSO significa que se encuentra en la UI que se observa en la Figura 2.36. y se procederá a abrir la actividad ‘UserDataRegistration’ donde el usuario podrá editar su propio nombre y clave de usuario. Pero de ser esta variable VERDADERA significa que la UI se encuentra como se observa en la Figura 2.38. lo que significa que el usuario borrará de la base de datos el departamento seleccionado, pero antes de mandar el mensaje con esta solicitud se abrirá una ventana de diálogo para confirmar que el usuario este consiente de la acción que va a tomar. De seleccionar NO en este cuadro de diálogo, se cancelará la acción y se esperará a que el usuario decida realizar alguna actividad más, pero si el usuario selecciona SI se procede a enviar el mensaje con el pedido de eliminar el departamento seleccionado de la base de datos, obteniendo como respuesta nuevamente la lista de departamentos que se encuentran en la base de datos, pero esta vez iniciando con el texto ‘DEPARTAMENT_UPDATA’ indicando que se realizó la actualización.

2.7.2.5.3. Delete_user

Este botón tiene un único propósito sin importar el tipo de usuario, y es el de eliminar a un usuario determinado de la base de datos y su diagrama de flujo se lo puede observar en la Figura 2.40. Se empieza extrayendo el texto del EditText ‘DepData’ donde debe de estar escrito el nombre de usuario que será eliminado. Una vez este este nombre en la variable ‘depiD’ se abre un cuadro de diálogo para confirmar que el usuario quiera realizar esta acción, por lo que si el usuario presiona NO se anulará la acción. Pero si el usuario presiona Si, se procederá a enviar un mensaje a la Raspberry Pi la cual procesará la información y

eliminará al usuario de la base de datos, una vez hecho esto la respuesta será nuevamente la lista de usuario del departamento sin el usuario que fue eliminado.

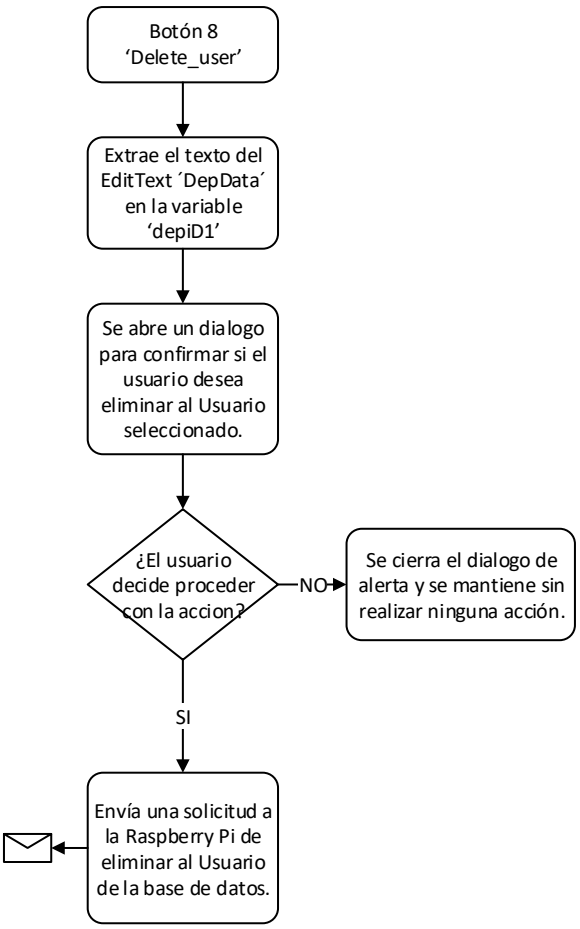

Figura 2.40. Diagrama de flujo de botón 'Delete_User'.

2.7.2.5.4. NewDepartment

Este es un botón flotante, que permite al usuario cambiar la interfaz de usuario de la actividad 'AdminConfiguration' para poder ingresar el nombre de un departamento, así como su clave correspondiente. Esto se va a realizar como se muestra en el diagrama de flujo de la Figura 2.41.

El presionar el botón 'NewDepartment' termina realizando 3 acciones:

Apaga el LinearLayout llamado 'LinearLayout1'.

```
LinearLayout1.setVisibility(View.GONE);
```

Hace visible el LinearLayout llamado 'LinearLayout2'

```
LinearLayout2.setVisibility(View.VISIBLE);
```

Apaga el botón ‘NewDepartment’

```
NewDepartment.setVisibility(View.GONE);
```

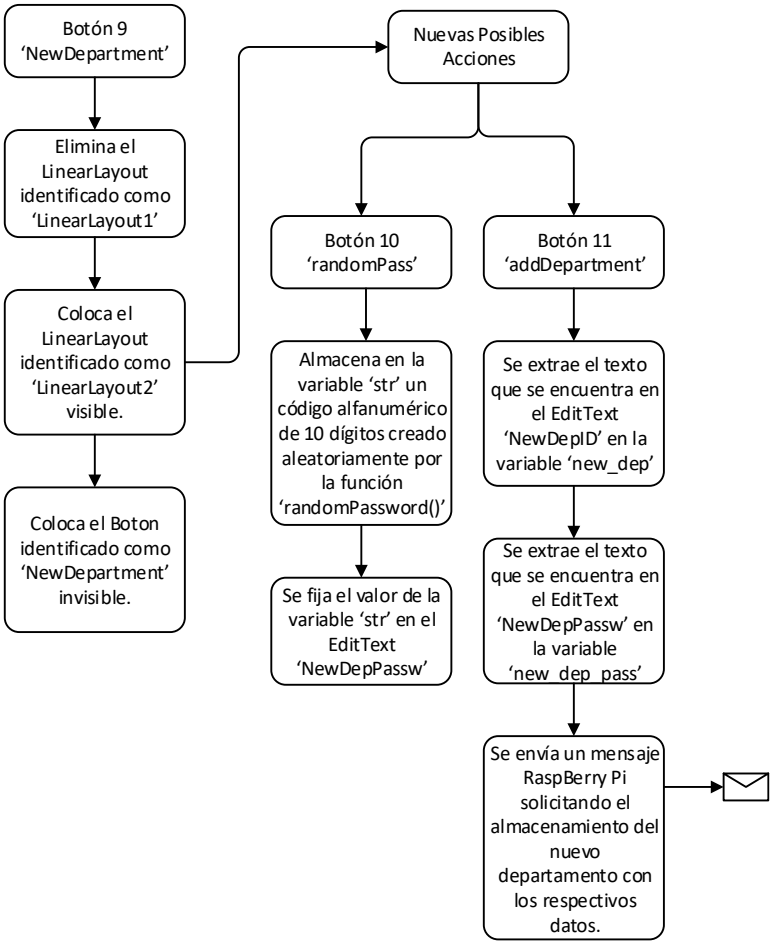

Figura 2.41. Diagrama de flujo de botón ‘NewDepartment’

Estas acciones hacen tener un UI completamente diferente en la misma actividad, como se observa en la Figura 2.42.

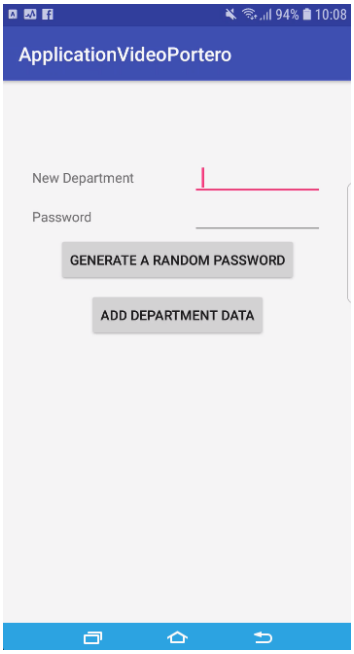

Figura 2.42. UI para añadir un departamento en la base de datos.

Lo que da paso a las acciones que procederán una vez se presione uno de estos botones.

Al presionar el botón ‘randomPass’ permite generar un texto alfanumérico aleatorio que será colocado en el EditText ‘NewDepPassw’ y así el usuario podrá elegir si desea conservar esta como la clave del departamento a crear o si desea cambiarla.

Por otro lado, si se presiona el botón ‘addDepartment’ se empezará colocando en variables los textos escritos en los EditText donde deberían estar escritas el nombre del departamento y su clave, esto será enviado con la acción ‘CONFIG’ y también se incluirá el texto ‘NEW_DEP’ seguido del nombre de departamento que se desea incluir, también el texto ‘NEW_DEP_PASS’ seguido de la clave de este nuevo departamento. Con estos datos incluidos a los datos que son enviados para la autentificación, la programación de la Raspberry Pi incluirá este departamento a la base de datos y responderá con la lista de todos los departamentos incluido el que se guardó recientemente, y en al recibir esto, el celular Android procederá a regresar a la interfaz de usuario inicial de esta actividad, es decir la que se puede observar en la Figura 2.35.

2.7.2.5.5. EditAdmin

Este botón permite al Administrador acceder a la actividad ‘UserDataRegistration.java’ desde donde se le permitirá cambiar su nombre de usuario y su clave, como ya se explicó en el funcionamiento de esta actividad.

2.7.2.6. VideoAccess

Esta actividad permitirá al usuario de la aplicación comunicarse con la persona que se encuentre frente al Video Portero, así como podrá observarla si lo requiere. De igual forma podrá abrir el seguro electromecánico para que la persona pueda acceder a la residencia. Para esto se establecieron 6 botones principales que permitirán realizar las siguientes acciones:

'AudioButton'- Permite al usuario iniciar la comunicación o poner en espera la comunicación.

'MicophoneButton'- Permite activar el micrófono para que el usuario en el lado del Video Portero pueda escuchar a la persona que ocupa la aplicación.

'ChooseVideo6'- Permite al usuario activar o desactivar el video

'openDoorButton'- Permite al usuario activar el seguro electromecánico y así que se pueda abrir la puerta en la que este está colocado.

'closeCallButton'- Permite al usuario de la aplicación culminar la comunicación.

'configButton'- Permite al usuario ingresar a configurar los datos de la base de datos, es decir a las actividades que se mencionó anteriormente.

Pero antes de proceder a explicar el funcionamiento de los botones es necesario conocer la inicialización de la actividad ya que cumple funciones fundamentales para la conectividad de audio y video con la Raspberry Pi. Por lo que se puede observar a continuación el diagrama de flujo de esta actividad en la Figura 2.43. Mientras que el código de la programación en Java se encuentra en el ANEXO X, y el código XML en el ANEXO Y.

Como todas las actividades se empieza inicializando las librerías y variables que serán ocupadas. Después de esto se procederá a extraer los datos necesarios que se encuentran almacenados en la memoria ROM y una vez se tenga esta información se envía un mensaje a la Raspberry Pi solicitando que comience a transmitir el Streaming de Video, esto debido a que la inicialización de este se demora unos segundos antes de poder observar la imagen. Una vez enviado este mensaje en segundo plano el programa seguirá ejecutando el resto del código sin necesidad de esperar una respuesta inmediata, pero la respuesta que se obtendrá si se ejecutó todo correcto en la Raspberry Pi será el texto 'HTTP_RUNING', y al ser así lo que observará el usuario en la pantalla será un mensaje instantáneo con el texto 'Video Online'.

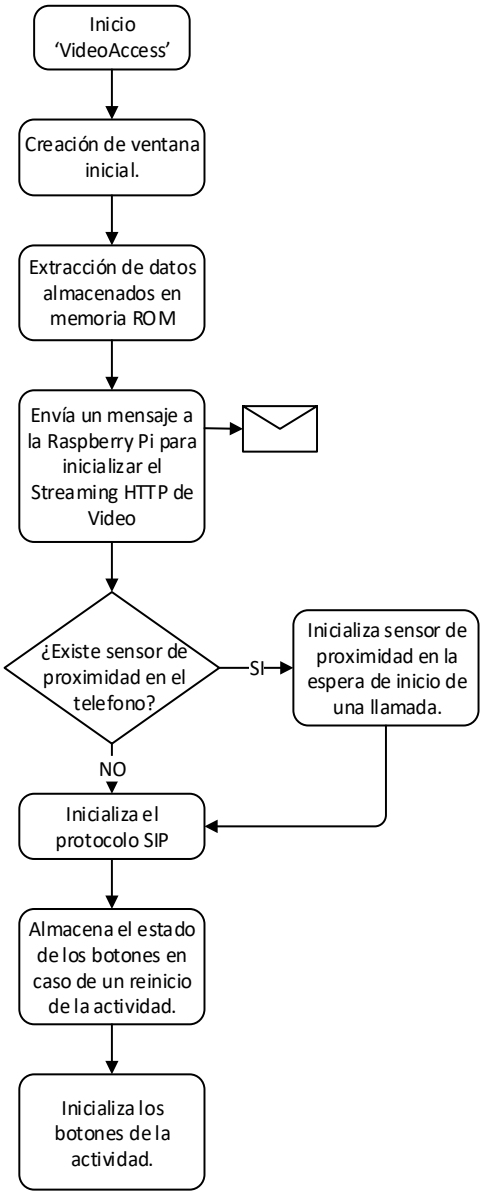

Figura 2.43. Diagrama de flujo de la inicialización de la actividad'VideoAccess.java'.

Después de enviar el mensaje se procede a conocer si el dispositivo móvil tiene un sensor de proximidad para activar o no la posibilidad de encender y apagar la pantalla del celular en caso que el usuario desee hablar sin observar la pantalla y más bien escuchando directo del auricular. Por lo que sí existe sensor de proximidad se procede a inicializar la acción que tomará la aplicación cuando este sensor se active, es decir cuando un objeto este demasiado cerca de este sensor o cuando se aleje de este, información que se verá más adelante.

Ya activado o no el sensor de proximidad se inicializa el protocolo SIP para poder registrar al dispositivo en el servidor ASTERISK y de este modo poder tener una comunicación de

audio cuando sea requerido. Mientras esta conectividad con el servidor SIP transcurre en segundo plano se podrán ver mensajes instantáneos, que confirmarán el estado que tiene este registro, es decir que se al momento que se inicia el registro se mostrará el mensaje 'Registering with SIP server…', el momento que ya se registró se observará el mensaje 'Ready', y si existe alguna falla en el registro se podrá observar el texto 'Registration faild.'

Mientras el registro transcurre en segundo plano el programa almacenará los estados de tres variables booleanas que permitirán mantener el estado del audio, micrófono y video si la actividad se reinicia, situación que suele pasar cuando se voltea el celular.

Después de esto se colocarán los estados de estos botones con los datos almacenados si es que existen o con los valores predeterminados. Culminando esto se procede a inicializar los botones y las acciones que tomarán cada vez que sean presionados, por lo que la interfaz de usuario inicial se verá cómo se puede observar en la Figura 2.44.

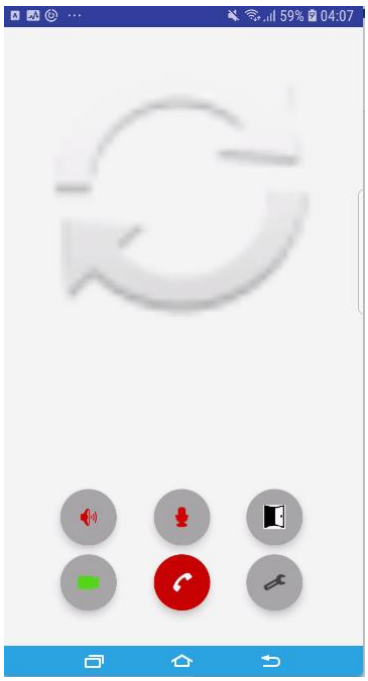

Figura 2.44. UI de actividad 'AccessVideo.java'

2.7.2.6.1. Cambio de estado en sensor de proximidad

El aproximarse o alejarse del auricular accionará esta acción que permite al usuario escuchar realizar la comunicación de voz ocupando este auricular o el altavoz, para esto el dispositivo Android debe estar activo y funcional el sensor de proximidad que en aplicaciones de comunicaciones lo ocupan del mismo modo, al confirmar es posible ocupar este sensor será posible realizar las siguientes acciones que se muestren en el diagrama de flujo que se muestra a continuación en la Figura 2.45.

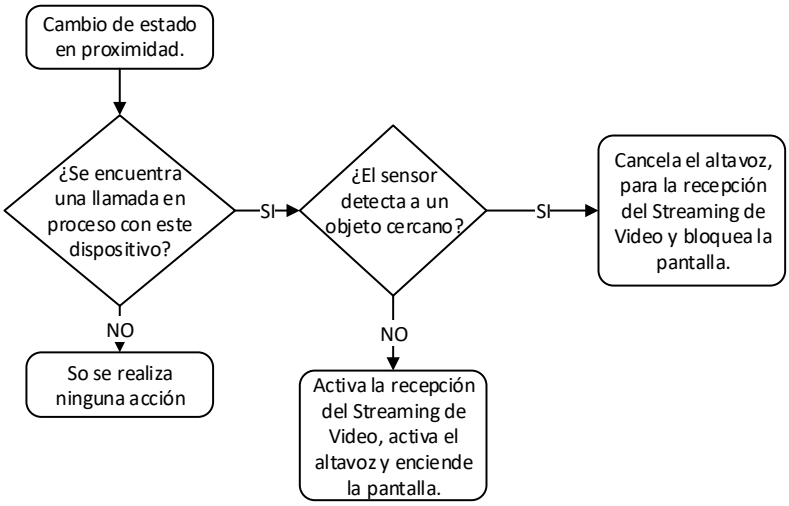

Figura 2.45. Diagrama de flujo del cambio de estado en el sensor de proximidad.

Al tener un cambio de estado en el sensor de proximidad, es decir si se acerca un objeto a cubrir el sensor o si se aleja de este se accionará esta función. Que empezará con una condicional 'IF' para conocer si la variable 'call' es diferente al valor 'null' que se le asigna por default y cambiará si se encuentra en una llamada en proceso. De ser el valor de esta variable 'null' no se realizará ninguna acción ya que sin una llamada en proceso no será necesario realizar ningún cambio. Pero si la variable 'call' es diferente a 'null' se procederá a identificar cual fue el cambio de este sensor de igual forma con una condicional 'IF' como se observa en el siguiente código:

```
if (event.values[0] < sensor.getMaximumRange())
```

Donde se analiza si el valor de la proximidad de un objeto al sensor es menor al máximo posible, de ser así se procede a apagar el altavoz y después de esto se detiene la recepción del streaming de video HTTP.

```
call.setSpeakerMode(false);
WebDisplay.stopLoading();
```

una vez realizadas estas acciones se ejecuta el siguiente código par que la pantalla pueda apagarse sin que se deje de ejecutar la aplicación

```
if(!wakeLock.isHeld()) {
    wakeLock.acquire(10*60*1000L /*10 minutes*/);
}
```

por lo contrario, si la proximidad del objeto es mayor al valor máximo se procede a activar y retomar la recepción del streaming de video HTTP

```
WebDisplay.setWebViewClient(new MyClient());
WebDisplay.loadUrl("http://" + domain + ":" + port);
```

Seguido de la activación del altavoz y el encendido de la pantalla.

```
call.setSpeakerMode(true);
if(wakeLock.isHeld()) {
    wakeLock.release();
```

culminando así con las acciones que se toman con el cambio de estado en el sensor de proximidad.

2.7.2.6.2. AudioButton

Este botón permitirá al usuario iniciar la comunicación de voz, ya que en un inicio estará indicado que se encuentra apagado con el botón pintado en color rojo, pero el momento de presionarlo podrá iniciar la comunicación, otra función que tendrá es que si se lo presiona mientras se encuentra en una comunicación, esta se pondrá en espera, y así mismo recuperar esta llamada en espera. Esto se puede observar en el diagrama de flujo que se encuentra a continuación en la Figura 2.46.

Al presionar este botón en primer lugar se revisa si la variable 'Receiver' es verdadero o falso, de ser FALSO simplemente continuará, pero si es VERDADERO como es su valor predeterminado al iniciar la actividad, se cambia de color el botón que se encuentra en rojo a verde y se envía la acción 'SPECTING_CALL' a la Raspberry Pi indicando que ya se encuentra listo para iniciar la comunicación de voz para después colocar el valor de la variable 'Receiver' en falso.

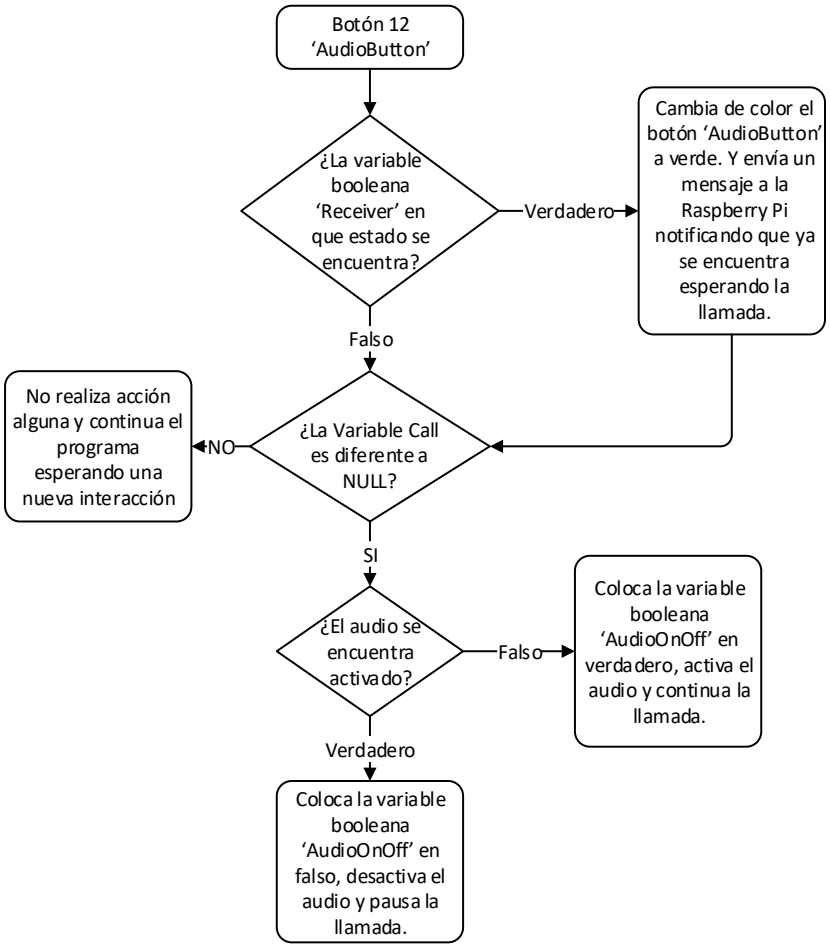

Figura 2.46. Diagrama de flujo del botón 'AudioButton'.

Continuará revisando si existe una comunicación en proceso desde este celular Android al Video Portero, verificando si la variable 'call' es distinta a 'null', de ser esta variable 'null' no realizará ninguna acción más, pero si existe una llamada en proceso, procederá a verificar el estado de la variable booleana 'AudioOnOff', si esta es VEDADERO se colocará esta variable en falso y el botón 'AudioButton' en rojo,

```
AudioButton.setImageTintList(ColorStateList.valueOf(getResources().
getColor(R.color.colorRed)));
```

y procederá a colocar la llamada en espera, esto mediante el protocolo SIP directamente con el servidor ASTERISK que se comunicará ahí mismo con el otro extremo de la llamada indicando esta acción.

```
call.holdCall(30);
```

En el caso de que la variable 'AudioOnOff' sea FALSO, se procede a colocar la variable 'AudioOnOff' en verdadero, se coloca el botón 'AudioButton' en color verde

```
AudioButton.setImageTintList(ColorStateList.valueOf(getResources().
getColor(R.color.colorLightGreen)));
```

y cambia recupera la llamada que se encuentra en espera.

```
if (call.isOnHold()) {
    try {
        call.continueCall(5);
    } catch (SipException e) {
        e.printStackTrace();
    }
}
```

2.7.2.6.3. MicrophoneButton

Al presionar este botón se podrá determinar si se enciende o apaga el micrófono, el apagar el micrófono permite que la comunicación continúe sin que la persona en el lado del Video Portero escuche lo que dice el usuario de la aplicación. La acción de este botón no requiere interacción con la Raspberry Pi y funcionará de acuerdo al diagrama de flujos que se observa en la Figura 2.47.

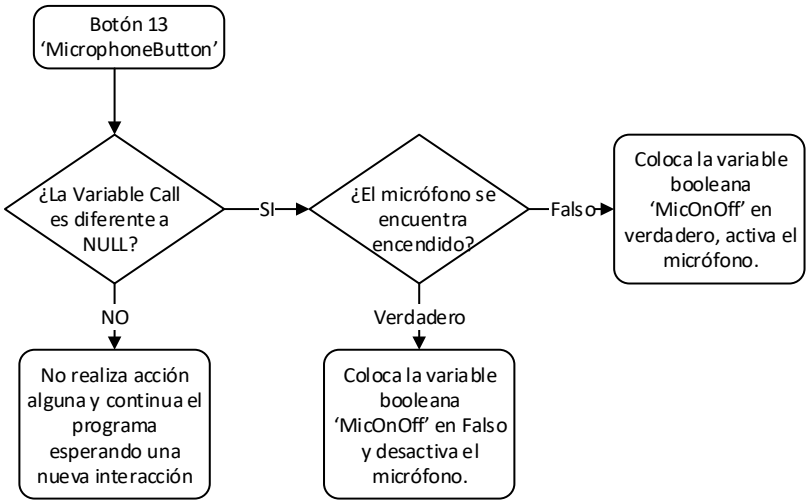

Figura 2.47. Diagrama de flujo del botón 'MicrophoneButton'.

Se empieza averiguando si existe una comunicación en progreso o no con la variable 'call', si esta es 'null' significa que no existe ninguna comunicación por lo que no será necesario tomar ninguna acción, caso contrario si esta variable es diferente a 'null' se procederá a ver el estado actual del micrófono, esto con la variable 'MicOnOff', de ser VERDADERA, se procede a colocar esta misma variable en FALSO, seguido del cambio de color del botón 'MicrophoneButton' a rojo. A continuación, colocó la llamada en silencio y de este modo mantendrá solo una comunicación unilateral.

```
if (!call.isMuted()) {
    call.toggleMute();
}
```

En cambio, si la variable 'MicOnOff' es FALSO se realiza el proceso inverso, es decir colocar esta variable en VEDADERO, cambia el color del botón 'MicrophoneButton' a verde y permite que la comunicación continúe bilateralmente.

```
if (call.isMuted()) {
    call.toggleMute();
}
```

2.7.2.6.4. ChooseVideo6

Este botón servirá para apagar o encender el streaming de video que proviene desde la Raspberry Pi, esto para reducir el procesamiento en la microcomputadora y mejore la calidad de audio de ser necesario. Para conocer el funcionamiento de esta parte del programa se puede ver en la Figura 2.48 el diagrama de flujo de este botón.

Este empezará averiguando en qué estado se encuentra la transmisión de video, esto mediante la variable 'VideoOnOff' que al ser VERDADERO indica que actualmente está activa la transmisión y que al presionar este botón lo que se desea es apagarlo, para esto se envía un mensaje a la Raspberry Pi con la acción 'STOP_HTTP_VIDEO', después del cual se recibirá una respuesta, que si es afirmativa se cambiará de color del botón a rojo y colocará la variable "VideoOnOFF' el valor de FALSO, para culminar mostrando al usuario el texto 'Video Offline' para confirmarle que el streaming de video a detenido su transmisión.

Por el otro lado cuando la variable 'VIdeoOnOff' es FALSO, se procede a enviar el mensaje a la Raspberry Pi con la acción 'START_HTTP_VIDEO' lo que la Raspberry Pi entenderá como una solicitud para empezar a transmitir nuevamente el streaming de video, que si la solicitud es aceptada tendrá como respuesta positiva 'HTTP_RUNING' y se procederá a cambiar de color el botón a verde, de igual forma cambiará la variable 'VideoOnOff' su valor a VERDADERO, para culminar mostrando al usuario el texto 'Video Online' y así el usuario sepa que nuevamente podrá recibir la transmisión de video. También realizará un cambio en el aspecto de la interfaz de usuario ya que hará visible un botón llamado 'ReloadVideo' en sustitución del WebView 'WebDisplay' que es donde se observa el video. Esto debido a que este botón permitirá al WebVier cargar el video una vez se encuentre el streaming activo ya que se demora algunos segundos hacerlo.

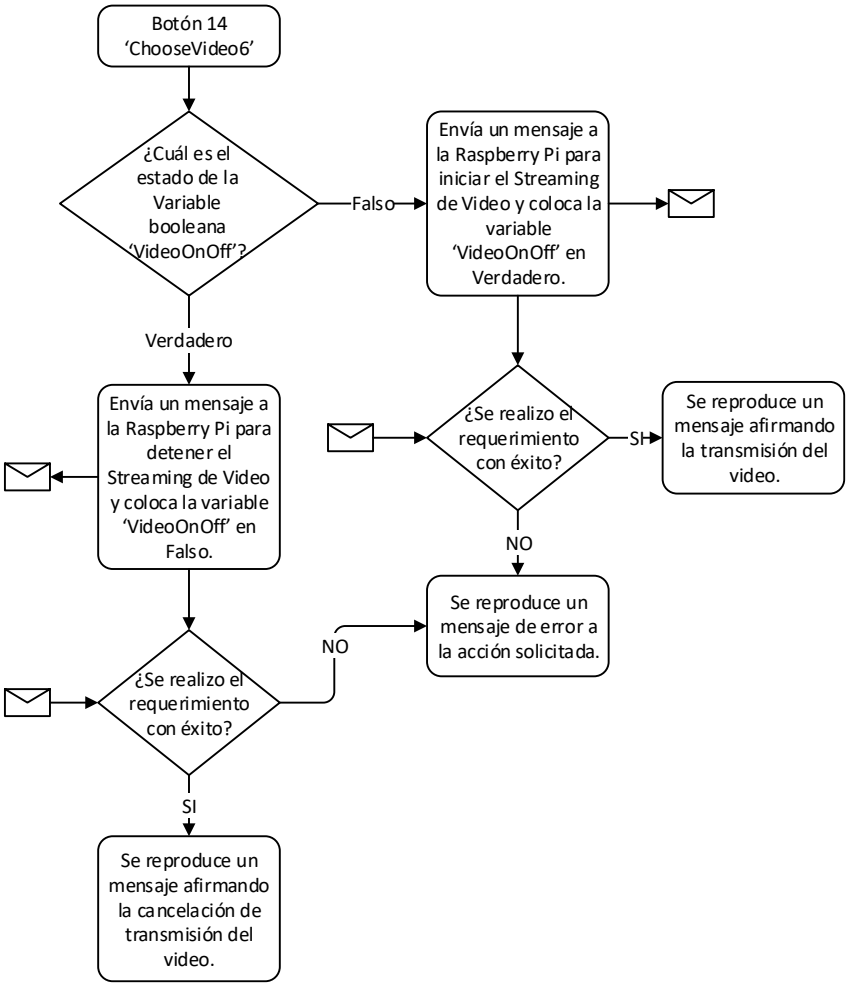

Figura 2.48. Diagrama de flujo del botón 'ChooseVideo6'.

2.7.2.6.5. ReloadVideo

Este botón servirá para recargar el video en el WebView, únicamente con estas dos líneas de código:

```
WebDisplay.setWebViewClient(new MyClient());
WebDisplay.loadUrl("http://" + domain + ":" + port);
```

Donde alista al WebDisplay como cliente y posteriormente carga la página HTTP requerida con la dirección IP y el número de puerto correspondiente.

2.7.2.6.6. openDoorButton

Este botón permitirá al usuario de la aplicación abrir el seguro electromecánico para así dejar pasar a la persona que se encuentra en la entrada de la residencia. Esto se lo realiza enviando directamente un mensaje con la acción 'OPEN_DOOR', con lo que la Raspberry Pi al recibir este mensaje activará el relé al que se encontrará conectado el seguro electromecánico y la fuente de energía que requerirá para abrirse. Después de realizar

esto, se enviará un mensaje de respuesta con el texto 'DOOR_IS_OPEN' lo que lanzará un mensaje instantáneo al usuario con la frase "The door is open!" para que tenga conocimiento que la acción se realizó satisfactoriamente.

2.7.2.6.7. closeCallButton

Para poder culminar la comunicación de voz con el Video Portero primero determina si existe en efecto una llamada analizando la variable 'call' como se ha visto previamente si esta es igual a 'null' procederá ejecutando el resto del código, pero si es diferente a 'null' entonces mediante el protocolo SIP se comunicará al servidor ASTERISK la decisión de culminar la llamada y así mismo se lo comunicará con el otro extremo de la llamada.

Una vez realizado esto se procederá a detener la recepción del streaming de video, se cambiará la variable 'Receiver' a verdadero, cerrará la comunicación SIP que tiene con el servidor ASTERISK, y por último se abrirá la actividad 'MainActivity.java' para que el usuario decida que más desea hacer.

2.7.2.6.8. configButton

Este botón abrirá la actividad 'UserDataVerification.java' para que el usuario pueda enviar sus datos a ser verificados antes de ingresar a las actividades desde las cuales puede editar la base de datos, pero esta actividad no se abrirá sin antes detener la recepción de entrada de video y almacenar la variable 'VideoOrConfig' con el valor de FALSO, variable que determinará que actividad abrirá posteriormente como ya se observó en el capítulo 2.7.2.4.2.

2.7.2.7. Archivos de programas adicionales

Adicional a las actividades mencionadas anteriormente, es necesario 4 archivos adicionales de Java, 3 para la configuración de las notificaciones de entrada, que se las realiza a través de Firebase Cloud Messaging y una para la recepción de las llamadas entrantes desde el Video Portero.

2.7.2.7.1. IncomingCallReceiver.java

Este archivo del programa permitirá la entrada de mensajes transmitidos con el protocolo SIP, dando paso a las llamadas entrantes y permitiendo la conectividad de voz a través del servidor de Asterisk. La realización de esta parte del código fue editada de un programa publicado en Google Developers, que permite la comunicación de dos dispositivos celulares de un modo half-duplex [50], desarrollado como demo para poder conocer el funcionamiento del protocolo SIP. Del cual se obtuvo el código base a ser modificado para

el funcionamiento Full-duplex deseado. Esta función está directamente ligada a la actividad VideoAccess.java, debido a que desde esta actividad se inicializa el protocolo SIP permitiendo registrar e interactuar con el servidor Asterisk. El código modificado de este programa se encuentra en el ANEXO Z.

2.7.2.7.2. ChannelID.java

Esta parte del programa permitirá la configuración de notificaciones, es decir que permitirá configurar la importancia de la notificación, asignándola a un canal de notificaciones, permitiéndole al usuario decidir si mantiene esta importancia o la modifica. El código de este archivo se lo podrá observar en el ANEXO AA.

2.7.2.7.3. MyFarebaseMessagingService.java

Permite la recepción de las notificaciones, así como el desplegar esta notificación en la pantalla para que el usuario la vea lo antes posible, está relacionada directamente con el programa anterior ChannelID.java ya que este determinará si se mantiene la importancia por defecto que tiene la aplicación o si el usuario deseo cambiarla. El código de esta sección fue obtenido de la página oficial de Firebase, donde indica cómo poner en marcha una aplicación con su sistema de notificaciones permitiendo modificar pequeñas partes para que realice la actividad deseada. Este código se podrá encontrar en el ANEXO BB.

2.7.2.7.4. MyFirebaseInstanceIdService.java

Por último, este código permite obtener un código único llamado TOCKEN, el que permitirá la identificación de cada uno de los celulares que tengan esta aplicación, y ayuda al envío de notificaciones a través del servicio que ofrece Firebase Cloud Messaging, esto ayudará a identificar al celular que se encuentre conectado a internet para poder enviar las notificaciones directamente a los usuarios requeridos sin que estos tengan encendidos la aplicación. Este código de igual forma se obtuvo de la página oficial de Firebase [51] y se podrá observar este código en el ANEXO CC.

3. RESULTADOS Y DISCUSIÓN

3.1. PRUEBAS DE FUNCIONAMIENTO

Para realizar las pruebas, se instaló la aplicación en tres celulares Android y se conectó al Video Portero a una fuente de energía. Una vez encendida la microcomputadora deberá ejecutar el programa Python por lo que se deberá esperar un poco menos de un minuto antes de empezar a usarla. Considerando, de igual forma que ya se encuentra configurado el SSID y contraseña correspondiente para que el dispositivo esté conectado a la red y así mismo se encuentra con una dirección IP fija.

Para comprobar esto, desde el CMD de una computadora se realiza un ping a la dirección IP fija de la Raspberry Pi como se observa en la figura 3.1.

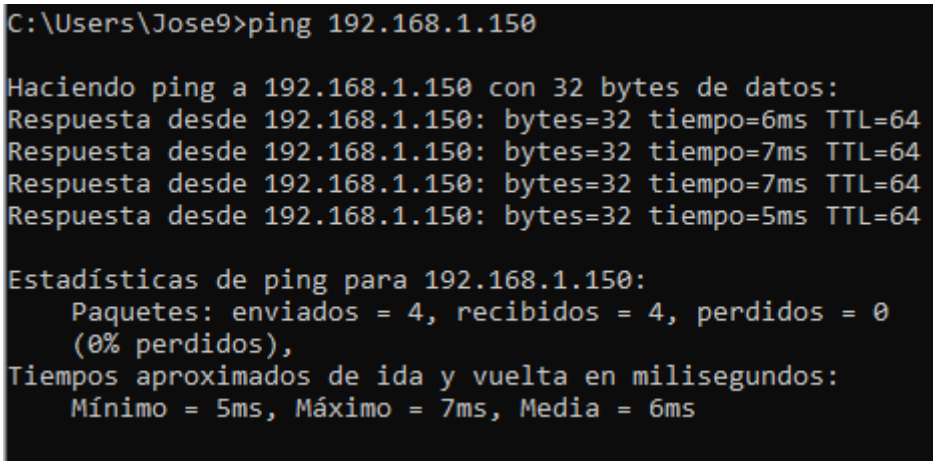

Figura 3.1. Comprobación que la Raspberry Pi se encuentra encendida y funcionando.

Al saber que la Raspberry Pi encendió correctamente, se procede a probar la conectividad entre el Video Portero y el celular con la aplicación celular Android. En el momento que se desea ejecutar el ingreso de los datos de un departamento, si existe un error en la clave o en el nombre del departamento aparecerá en la aplicación el siguiente texto observado en la Figura 3.2.

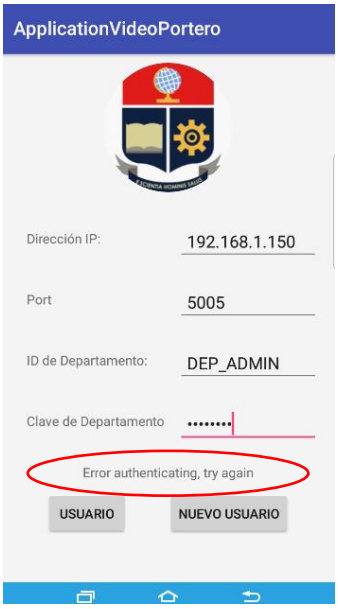

Figura 3.2. Intento de autentificación de departamento fallido.

Y al colocar correcto estos datos procederá a avanzar a la siguiente IU para la autentificación de usuario, en esta primera prueba se demora en procesar esta información y en avanzar a la siguiente ventana aproximadamente 1 segundos, tiempo que variará dependiendo del procesamiento de la microcomputadora.

Al proceder con la autentificación del usuario que se encuentra registrado por defecto en el departamento 'DEP_ADMIN', se marca al switch que mantendrá almacenada la clave y permitirá el acceso directo a la actividad principal, como se observa en la Figura 3.3. y se presiona el botón **SIGUIENTE**.

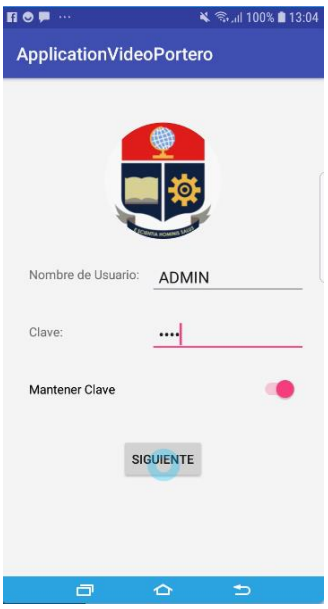

Figura 3.3. Ventana de autentificación de usuario

Una vez se presiona siguiente se accede a la actividad principal que permitirá la comunicación de audio y video, como se puede observar en la Figura 3.4.

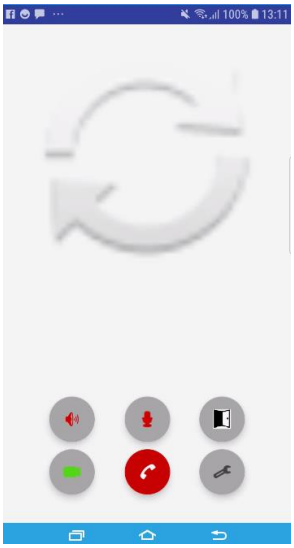

Figura 3.4. Actividad principal de Video Portero.

Hasta este punto se ha podido comprobar el correcto funcionamiento de la autentificación y la capacidad de recepción y transmisión de mensajes entre la microcomputadora y el celular Android.

A continuación, en la Figura 3.5., se podrá observar la recepción de imagen enviado por el Video Portero,

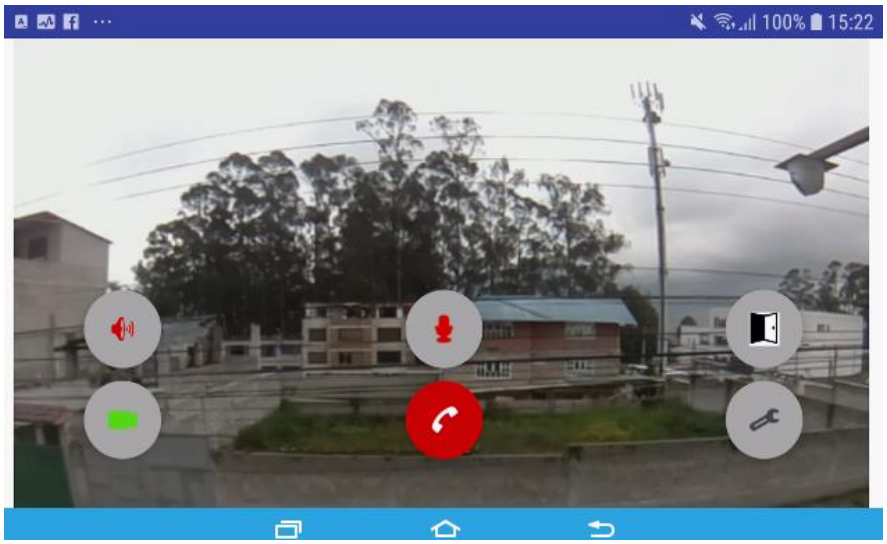

Figura 3.5. Imagen de la actividad VideoAccess.java
reproduciendo el Video recibido.

El video es transmitido y recibido, sin la necesidad de mantener una conexión de audio, lo que permite usar a este video únicamente como cámara de seguridad.

Para poder conocer el retraso del video se coloca un cronometro y alado el celular que va a recibir el video, y la diferencia entre estos dos será el retraso que tiene el video, como se puede observar en la Figura 3.6. que en dos ocasiones su retraso fue de aproximadamente 60 milisegundos, es decir 3/5 partes de un segundo.

Figura 3.6. Pruebas de retraso de Video

Para proceder a empezar una comunicación de Audio se presiona el botón con el que casi inmediatamente se empieza a escuchar el audio proveniente del lado del Video Portero. Este audio tiene un retraso de aproximadamente de un segundo como se puede observar en la Figura 3.7. que es una grabación del audio entrante al Video Portero y el que sale del parlante del celular, estimando el retraso.

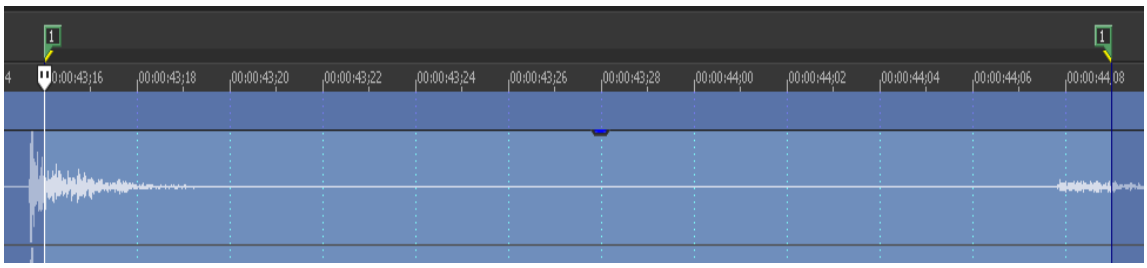

Figura 3.7. Prueba del retraso de la transmisión de audio Video Portero-Celular Android.

Este retraso y el audio distorsionado no permite una comunicación continua y estable por lo que no cumple con las características deseadas para este Video Portero.

Al probar el audio en el sentido opuesto es decir usando el micrófono de entrada en el celular Android y escuchando en el parlante del Video Portero, se tiene un retardo de aproximado de medio segundo como se puede observar en la Figura 3.8.

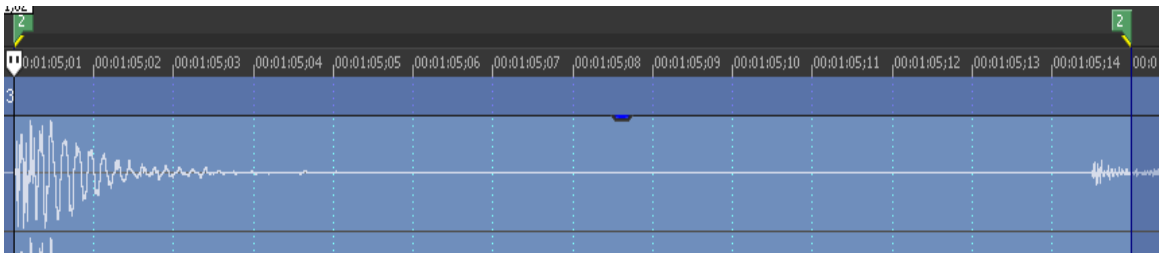

Figura 3.8. Prueba del retraso de la transmisión de audio Celular Android-Video Portero.

Continuando con las pruebas, se afirma el funcionamiento del botón que permitirá abrir la puerta electromecánica remotamente, para que sea visible, se remplazó el seguro electromecánico por un foco como se observa en la Figura 3.9., que se enciende por un segundo, este segundo es el necesario para que el seguro electromecánico se acciones y se quede abierta la puerta.

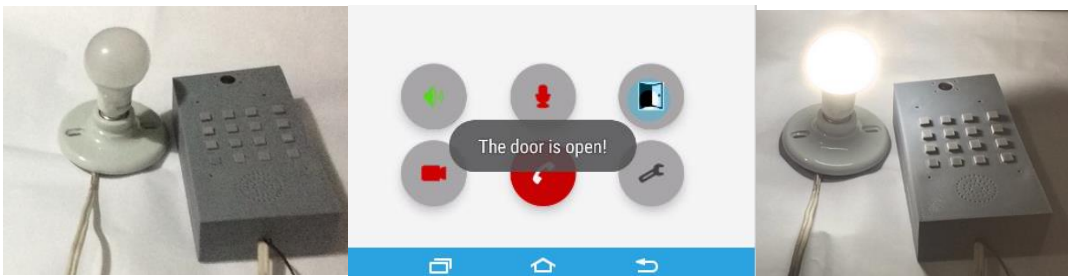

Figura 3.9. Prueba del funcionamiento del accionar del seguro electromecánico.

Después de esto se procede a comprobar la configuración como Administrador general, que permitirá añadir departamentos, que como se ve en la Figura 3.10. aun no existe ninguno, aparte del que viene por defecto para que el administrador general pueda registrarse y manipular la base de datos.

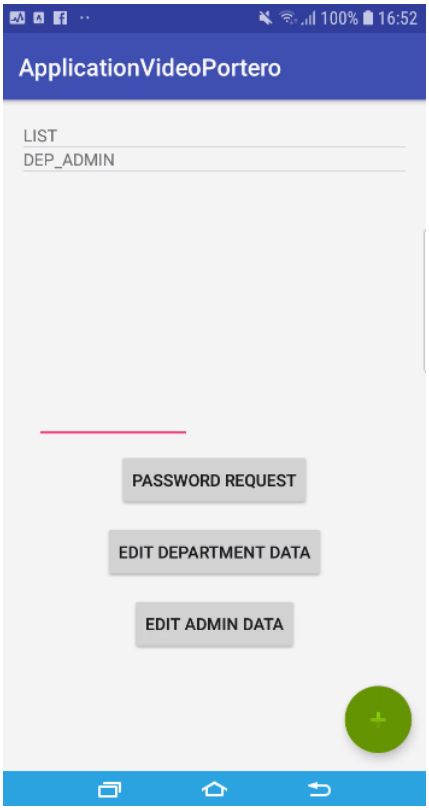

Figura 3.10. Ventana de configuración inicial para Administrador de edificio.

Presionando el botón de color verde con el símbolo + se podrá acceder a un U.I. distinto donde se colocará el nombre de departamento y la clave de este como por ejemplo se puede ver en la Figura 3.11.

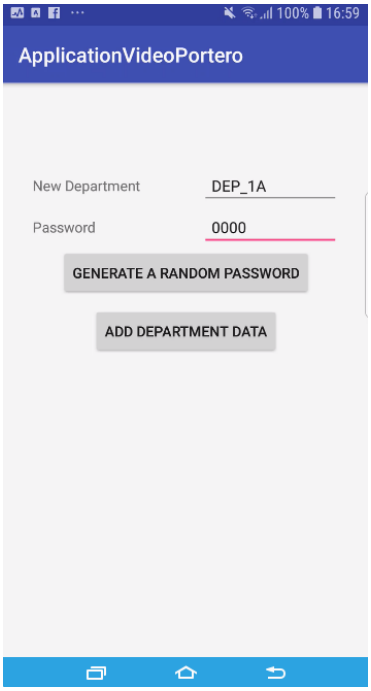

Figura 3.11. Ventana para añadir nuevos departamentos en la base de datos.

Después del cual al presionar **ADD DEPARTMENT DATA**, la actividad regresará a la ventana anterior, presentando en la lista el nuevo departamento como se puede comprobar en la Figura 3.12.

Figura 3.12. Nueva ventana de configuración con departamento añadido.

De esta forma se podrán crear la cantidad de departamentos necesarios de acuerdo a lo requerido en las distintas situaciones y para el propósito de las pruebas, se crearán un total de tres departamentos, sin contar con el de administración.

Se registran a dos nuevos usuarios desde dispositivos diferentes en el Departamento 'DEP_1A'. Para verificar el envío de notificaciones, se realiza el marcado del número de departamento, en este caso 1A, y empieza a sonar el tono de marcado Video Portero por un tiempo máximo de 60 segundos, tiempo en el cual ya llego la notificación a los dos dispositivos celulares como se puede observar en la Figura 3.13.

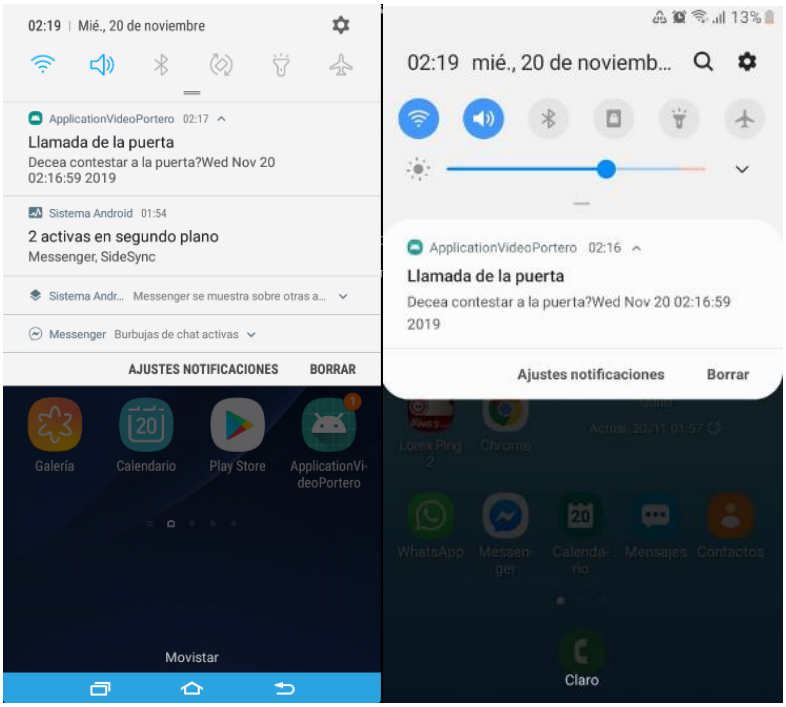

Figura 3.13. Llegada de notificaciones a los celulares del departamento DEP_1A.

Al hacer click en la notificación, se ingresará directamente a la aplicación, desde la cual será necesario acceder a la ventana principal (indicada anteriormente en la Figura 3.4), donde se podrá empezar la comunicación de voz y observar el streaming de video.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- El desarrollo de la aplicación implico aprender el funcionamiento básico del sistema operativo Android, así como los lenguajes de programación Java y XML, y el uso de la herramienta de desarrollo Android Studio que facilitó el desarrollo de la aplicación. Una ayuda fundamental para realizar esta aplicación fue la extensa información que contiene la página Android Developers. Razón por la cual no es necesario conocer todas las funciones y códigos para realizar la aplicación, solo investigar en esta página el cómo se puede realizar lo que se requiere.

- Para la programación de la Raspberry Pi en Python de igual forma existe una amplia comunidad que da su ayuda a nuevos desarrolladores y así mismo la cantidad de información que existe en páginas relacionadas a Python permite el aprender y resolver problemas con mayor facilidad.

- Gracias a protocolos de comunicación estandarizados es posible la comunicación entre dispositivos completamente diferentes, incluso usando lenguajes de programación distintos. Debido a que los desarrolladores de los lenguajes de programación y la gente que continúa realizando aportes a estos, se mantienen al tanto de estos estándares y mantienen actualizado su código.

- La comunicación entre la Raspberry Pi y el celular Android a través del protocolo TCP/IP es fluida y sin inconvenientes, lo que permite conocer que estos pueden ser usados para proyectos que no requieren comunicación en tiempo real sin ningún problema.

- El uso de la Raspberry Pi para aplicaciones de tiempo real no es factible como se pudo observar. Esto debido al nivel de procesamiento que tiene y sobre todo porque este nivel de procesamiento se divide para varias actividades. Para aplicaciones de tiempo real (como la comunicación de audio), usualmente se ocupa hardware dedicado, es decir que únicamente ejecuten el programa diseñado para este y no programas que requiera el dispositivo para mantenerse encendido. Por esta razón se escogió el sistema operativo Raspbian Lite, y así reducir el nivel de procesamiento y consumo de memoria en un GUI, pero no fue suficiente para mantener una comunicación de audio estable.

- El uso de SQLite permite el almacenamiento confiable de los datos de usuarios y gracias a la fácil manipulación que se puede llegar a tener extrayendo y modificando información, es una herramienta fundamental a ser usada en este proyecto.

- La comunicación entre Raspberry Pi y el dispositivo Android es exclusivamente en la red LAN, debido a la necesidad de tener una la dirección IP fija en la Raspberry Pi, cosa que no es posible en la red WAN debido a que los ISP realizan compartición de direcciones IP públicas que se mantienen cambiando cada cierto tiempo.

- Gracias a Google, existe la plataforma Firebase Cloud Messaging que facilita el envío de notificaciones a un dispositivo móvil, sin la necesidad que se encuentre encendida la aplicación, ni tener una dirección IP fija e incluso sin la necesidad que se encuentre el móvil en la misma red LAN que el servidor desde el que se enviará la notificación. Esta plataforma identifica a los dispositivos móviles con un código llamado TOCKEN único para cada dispositivo celular que ocupe la aplicación en la que se encuentra programada con Firebase.

- En el caso que existan cortes de energía la microcomputadora se reiniciará con normalidad al momento que la energía fluya con normalidad, así mismo se ejecutarán los programas para el funcionamiento del Video Portero. Esto gracias a la posibilidad que da la aplicación 'crontab' de ejecutar programas periódicamente o después de una acción específica, como lo es en este caso el encendido de la Raspberry Pi.

4.2. RECOMENDACIONES

- Es necesario mantener un respaldo de la tarjeta microSD para evitar perder la información, principalmente si después de ser instalada se va a registrar una cantidad grande de usuarios, esto evitaría que se tengan que registrar nuevamente en caso de que esta memoria microSD falle, evitando inconvenientes con estos.

- Si existen inconvenientes con la señal Wi-Fi, sea porque la microcomputadora se encuentra lejos o porque existe interferencia, será necesario ocupar un repetidor para que, adicional al retardo de audio y video debido al procesamiento, no exista más retardo debido al bajo ancho de banda que se logre obtener.

- Para mejorar la seguridad de envío de mensajes desde el dispositivo celular y la microcomputadora sería recomendable encriptar estos y no enviarlo como texto plano, a pesar se tenga una red WLAN segura. Es preferible realizar el cifrado desde los

dispositivos de transmisión y recepción de mensajes para aumentar la seguridad siendo esto fundamental para evitar que alguien encuentre la manera de abrir la puerta eléctrica.

- La colocación de un botón de emergencia o programación de una combinación de botones puede ser añadido para que en casos de emergencia un usuario pueda tener una respuesta rápida de los residentes o incluso comunicación con la policía.

- Incluir un pequeño display en el video portero, para que el usuario reconosca lo que esta digitando e incluso, para que pueda escoger a quién desea timbrar, sería una herramienta útil que mejoraría la experiencia del usuario.

- Encontrar la forma de reducir los retrasos que impiden la comunicación de audio eficiente ayudaría enormemente para considerar a este proyecto como algo factible de ser comercializado.

- Permitir que el dispositivo sea visible en el internet (mediante VPNs o un servidor) para que los usuarios no requieran estar conectados en la misma red WLAN que el video portero, de igual forma mejoraría y ampliaría el uso de un dispoditivo como estos.

REFERENCIAS BIBLIOGRÁFICAS

[1] Google Developers, «Firebase Cloud Messaging,» [En línea]. Available: https://firebase.google.com/docs/cloud-messaging/?gclid=EAIaIQobChMI4czf5LWe3wIVBF6GCh03AAo3EAAYASAAEgKXUPD. [Último acceso: 25 Enero 2019].

[2] Raspberry Pi Foundation, «About Us,» [En línea]. Available: https://www.raspberrypi.org/about/. [Último acceso: 12 Septiembre 2019].

[3] Raspberry Pi Foundation, «Setup,» [En línea]. Available: https://www.raspberrypi.org/documentation/setup/. [Último acceso: 15 Diciembre 2018].

[4] Raspberry Pi Foundation, «SSH (Secure Shell),» [En línea]. Available: https://www.raspberrypi.org/documentation/remote-access/ssh/. [Último acceso: 15 Diciembre 2018].

[5] Real VNC, «VNC Connect,» [En línea]. Available: https://www.realvnc.com/es/connect/download/vnc/raspberrypi/. [Último acceso: 2018 Diciembre 15].

[6] SparkFunElectronics, [En línea]. Available: https://static.sparkfun.com/assets/learn_tutorials/6/7/6/PiZero_1.pdf. [Último acceso: 23 junio 2019].

[7] Raspberry Pi Foundation, «Raspberry Pi Zero W,» [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-zero-w/. [Último acceso: 23 Noviembre 2018].

[8] Raspberry Pi Foundation, «Raspberry Pi Model 3B+,» [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. [Último acceso: 23 Noviembre 2018].

[9] Raspberry Pi Foundation, «CAMARA MODULE,» [En línea]. Available: https://www.raspberrypi.org/documentation/hardware/camera/. [Último acceso: 23 julio 2019].

[10] Arducam, «AMAZON,» [En línea]. Available: https://www.amazon.com/Arducam-C%C3%A1mara-esp%C3%ADa-para-Raspberry/dp/B07T948TF3/ref=sr_1_31?__mk_es_US=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=raspberry+pi+camera&qid=1565975588&s=gateway&sr=8-31. [Último acceso: 16 08 2019].

[11] SainSmart, «Amazon,» [En línea]. Available: https://www.amazon.com/SainSmart-Lentes-Angular-Raspberry-Arduino/dp/B00N1YJKFS/ref=sr_1_99?__mk_es_US=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=raspberry+pi+camera&qid=1565974849&s=gateway&sr=8-99. [Último acceso: 16 08 2019].

[12] SEEED WIKI, «ReSpeaker_2_Mics_Pi_HAT,» [En línea]. Available: http://wiki.seeedstudio.com/ReSpeaker_2_Mics_Pi_HAT/. [Último acceso: 16 08 2019].

[13] Python Software Foundation, «History and License,» [En línea]. Available: https://docs.python.org/3/license.html. [Último acceso: 20 Mayo 2019].

[14] E. Matthes, de *Python Crash Couese*, San Francisco, no starch press, 2016, pp. xxxi-xxxii.

[15] Python Software Foundation, «time-Time access and conversions,» [En línea]. Available: https://docs.python.org/3/library/time.html. [Último acceso: 10 Enero 2019].

[16] Python Software Foundation, «subprocess — Subprocess management,» [En línea]. Available: https://docs.python.org/3/library/subprocess.html. [Último acceso: 10 Enero 2019].

[17] Python Software Foundation, «sqlite3 — DB-API 2.0 interface for SQLite databases,» [En línea]. Available: https://docs.python.org/3/library/sqlite3.html.

[18] Python Software Foundation, «socket — Low-level networking interface,» [En línea]. Available: https://docs.python.org/3/library/socket.html.

[19] Hewlett Packard, «BSD Sockets Interface Programmer's Guide Edition 6,» Estados Unidos, Hewlett-Packard Company, p. 17.

[20] Healthcare Flows, «Most Voip Library,» [En línea]. Available: https://most-voip.readthedocs.io/en/latest/. [Último acceso: 15 Enero 2019].

[21] PJSIP, «PJSIP,» [En línea]. Available: https://www.pjsip.org/. [Último acceso: 20 09 2019].

[22] B. Croston, «RPi.GPIO 0.7.0,» [En línea]. Available: https://pypi.org/project/RPi.GPIO/. [Último acceso: 5 Enero 2019].

[23] E. Adegbite, «pyfcm 1.4.7,» [En línea]. Available: https://pypi.org/project/pyfcm/. [Último acceso: 15 Enero 2019].

[24] SQLite, «About SQLite,» [En línea]. Available: https://www.sqlite.org/about.html. [Último acceso: 23 Diciembre 2018].

[25] SQLite, «SQLite is a Self Contained System,» [En línea]. Available: https://www.sqlite.org/selfcontained.html. [Último acceso: 10 Enero 2019].

[26] SQLite, «SQLite Is Serverless,» [En línea]. Available: https://www.sqlite.org/serverless.html. [Último acceso: 10 Enero 2019].

[27] SQLite, «SQLite Is A Zero-Configuration Database,» [En línea]. Available: https://www.sqlite.org/zeroconf.html. [Último acceso: 10 Enero 2019].

[28] Digium, «Getting Started with Asterisk,» [En línea]. Available: https://www.asterisk.org/get-started. [Último acceso: 16 Enero 2019].

[29] J. T. Gironés, «El Gran Libro de Android 2da Edición,» Alfaomega, Marcombo, SA, p. 26.

[30] Google Developers, «Android Developers, Arquitectura de la plataforma,» [En línea]. Available: https://developer.android.com/guide/platform. [Último acceso: 13 Octubre 2018].

[31] Google Developers, «Android Developers, Descripción general de notificaciones,» [En línea]. Available: https://developer.android.com/guide/topics/ui/notifiers/notifications?hl=ES. [Último acceso: 12 Febrero 2019].

[32] Google Developers, «Android Developers, Permisos del sistema,» [En línea]. Available: https://developer.android.com/guide/topics/security/permissions.html?hl=es-419. [Último acceso: 02 Febrero 2019].

[33] Google Developers, «Android Developers, Actividades,» [En línea]. Available: https://developer.android.com/guide/components/activities.html?hl=ES. [Último acceso: 23 Noviembre 2018].

[34] W. Stalling, «HTTP,» de *Data and Computer Comunication 8Edicion*, Pearson Prentice Hall, p. 784.

[35] W. Stallng, «SIP,» de *Data and Computer Comunication 8Edicion* , Pearson Prentice Hall, p. 811.

[36] Raspberry Pi Foundation, «Downloads,» [En línea]. Available: https://www.raspberrypi.org/downloads/. [Último acceso: 19 09 2019].

[37] M. Kerrisk, «crontab,» [En línea]. Available: http://man7.org/linux/man-pages/man5/crontab.5.html. [Último acceso: 21 08 2018].

[38] M. Grinberg, «Video Streaming with Flask,» [En línea]. Available: https://blog.miguelgrinberg.com/post/video-streaming-with-flask. [Último acceso: 10 Febrero 2019].

[39] H. Flows, «Most Voip API, Tutorial 1:Making a Call,» [En línea]. Available: https://most-voip.readthedocs.io/en/latest/python_docs/tutorial/voip_tutorial_1.html. [Último acceso: 10 Enero 2019].

[40] Google Developers, «FloatingActionButton,» [En línea]. Available: https://developer.android.com/reference/android/support/design/widget/FloatingActionButton. [Último acceso: 15 octubre 2019].

[41] Google Developers, «SipDemo,» [En línea]. Available: https://tool.oschina.net/uploads/apidocs/android/resources/samples/SipDemo/index.html. [Último acceso: 19 noviembre 2019].

[42] Firebase, «Configura una app cliente de Firebase Cloud Messaging en Android,» [En línea]. Available: https://firebase.google.com/docs/cloud-messaging/android/client. [Último acceso: 18 noviembre 2019].

[43] Raspberry Pi Foundation, «Raspberry Pi 1 Model A+,» [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-1-model-a-plus/. [Último acceso: 20 Seotiembre 2019].

[44] Raspberry Pi Foundation, «Raspberry Pi 1 Model B+,» [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-1-model-b-plus/. [Último acceso: 20 Septiembre 2019].

[45] Raspberry Pi Foundation, «Raspberry Pi 2 Model B,» [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. [Último acceso: 20 Septiembre 2019].

[46] Raspberry Pi Foundation, «Raspberry Pi 3 Model B,» [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Último acceso: 20 Septiembre 2019].

[47] Raspberry Pi Foundation, «Raspberry Pi 3 Model A+,» [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-a-plus/. [Último acceso: 20 Septiembre 2019].

[48] Raspberry Pi Foundation, «Raspberry Pi Zero,» [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-zero/. [Último acceso: 20 Septiembre 2019].

[49] Raspberry Pi Foundation, «Raspberry Pi 4B,» [En línea]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf. [Último acceso: 18 09 2020].

[50] Raspberry Pi Foundation, «Raspberry Pi 4,» [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. [Último acceso: 18 09 2020].

[51] Raspberry Pi Foundation, «Raspbian,» [En línea]. Available: https://www.raspberrypi.org/downloads/raspbian/. [Último acceso: 19 09 2019].

[52] balenaEtcher, «Flash. Flawless,» [En línea]. Available: https://www.balena.io/etcher/. [Último acceso: 19 09 2019].

[53] I. Holland, «Maker Space- ReSpeaker 2-Mics Pi HAT,» [En línea]. Available: http://www.makerspace-uk.co.uk/respeaker-2-mics-pi-hat/. [Último acceso: 10 Septiembre 2019].

[54] Asterisk, «Index of /pub/telephony/asterisk,» [En línea]. Available: http://downloads.asterisk.org/pub/telephony/asterisk/. [Último acceso: 19 septiembre 2019].

[55] Python Packege Index, «RPi.GPIO 0.7.0,» [En línea]. Available: https://pypi.org/project/RPi.GPIO/. [Último acceso: 17 08 2019].

[56] Python Package Index, «pyfcm 1.4.7,» [En línea]. Available: https://pypi.org/project/pyfcm/. [Último acceso: 18 0 2019].

[57] A. Boudai, «Make a Web App Using Python & Flask!,» [En línea]. Available: https://aryaboudaie.com/python/technical/educational/web/flask/2018/10/17/flask.html#how-to-install-flask.

[58] Picamara, «Installation,» [En línea]. Available: https://picamera.readthedocs.io/en/release-1.12/install.html. [Último acceso: 18 08 2019].

[59] instructable circuits, «GitPi,» [En línea]. Available: https://www.instructables.com/id/GitPi-A-Private-Git-Server-on-Raspberry-Pi/. [Último acceso: 20 09 2019].

ANEXOS

ANEXO A. Características individuales de los distintos modelos de Raspberry Pi.

ANEXO B. Preparación de Raspberry Pi

ANEXO C. Diagrama de directorios y archivos desarrollados.

ANEXO D. Código del archivo “VideoPortero.py” programa inicial del Video Portero.

ANEXO E. Código del archivo “Teclado_y_Notificaciones.py’ programa que controla el teclado y gestiona el envío de Notificaciones al dispositivo Android. Y envío de notificaciones “FCMNotifications.py

ANEXO F. Código del archivo “Recepcion_de_Mensajes.py” que permite la recepción de los mensajes de control

ANEXO G. Código del archivo “FuncionesDatos.py” que gestiona la autenticación e identifica los mensajes de control.

ANEXO H. Código del archivo “Voip.py” que gestiona la comunicación de voz sobre IP.

ANEXO I. Funciones de “BaseDatosSQLite.py”

ANEXO J. Envío de imágenes mediante protocolo HTTP.

ANEXO K. Cancela el streaming de video Http.

ANEXO L. Envío de notificaciones.

ANEXO M. Programación Java de actividad “MainActivity.java”

ANEXO N. Programación XML de actividad “activity_main.xml”

ANEXO O. Programación Java de actividad “DepartmentDataRegistration.java”

ANEXO P. Programación XML de actividad “activity_department_data_registration.xml”

ANEXO Q. Programación Java de actividad “UserDataRegistration.java”

ANEXO R. Programación XML de actividad “activity_user_data_registration.xml”

ANEXO S. Programación Java de actividad “UserDataVerification.java”

ANEXO T. Programación XML de actividad “activity_user_data_verification.xml”

ANEXO U. Diagrama de flujo de botones de actividad “UserDataVerification.java”

ANEXO V. Programación Java de actividad “AdminConfiguration.java”

ANEXO W. Programación XML de actividad “activity_admin_configuration.xml”

ANEXO X. Programación Java de actividad “VideoAccess.java”

ANEXO Y. Programación XML de actividad “activity_video_access.xml”

ANEXO Z. Programación Java de “IncomingCallReceiver.java”

ANEXO AA. Programación Java de “ChannelID.java”

ANEXO BB. Programación Java de “MyFarebaseMessagingService.java”

ANEXO CC. Programación Java de “MyFirebaseInstanceIdService.java”

ANEXO DD. Manual de instalación y usuarios.

ANEXO A

Características individuales de los distintos modelos de Raspberry Pi

Raspberry Pi 1 Modelo A+ [48]

Remplazo en el 2014 a la Raspberry Pi 1 Modelo A

Este dispositivo tiene un procesador Broadcom BCM2835 que mantiene una frecuencia de procesamiento de 700 MHz con bajo consumo de energía.

Este modelo de Raspberry se alimenta de un voltaje de 5V con una corriente de 2A, energía que puede ser conectada mediante un puerto Micro-USB que servirá únicamente para este propósito.

Memoria Ram de 512 MB SDRAM

Puerto Ethernet 10/100 Base T

1 puerto USB

Conector de display de 15 pines DSI (Display Serial Interface).

El conector de cámara que contiene es de 15 pines MIPI CSI-2 (Camera Serial Interface)

A parte del puerto HDMI también tiene una salida analógica de audio y video con un jack de 3.5 mm, con audio estéreo y video compuesto (PAL y NTSC).

Dimensiones 66 x 56 x 14 mm

Figura A.1. Imagen de Raspberry Pi 1 Modelo A+ [48]

Raspberry Pi 1 Modelo B+ [49]

Con un procesador Broadcom BCM2835 SoC que mantiene una frecuencia de procesamiento de 700 MHz con bajo consumo de energía.

Este modelo de Raspberry se alimenta de un voltaje de 5V con una corriente de 2A, con un consumo entre 0.5 y 1W energía que puede ser conectada mediante un puerto Micro-USB que servirá únicamente para este propósito.

Memoria Ram de 512 MB SDRAM

Puerto Ethernet

4 puerto USB

Conector de display de 15 pines DSI (Display Serial Interface).

El conector de cámara que contiene es de 15 pines MIPI CSI-2 (Camera Serial Interface)

A parte del puerto HDMI también tiene una salida analógica de audio y video con un jack de 3.5 mm, con audio estéreo y video compuesto (PAL y NTSC).

Dimensiones 85 x 56 x 17 mm

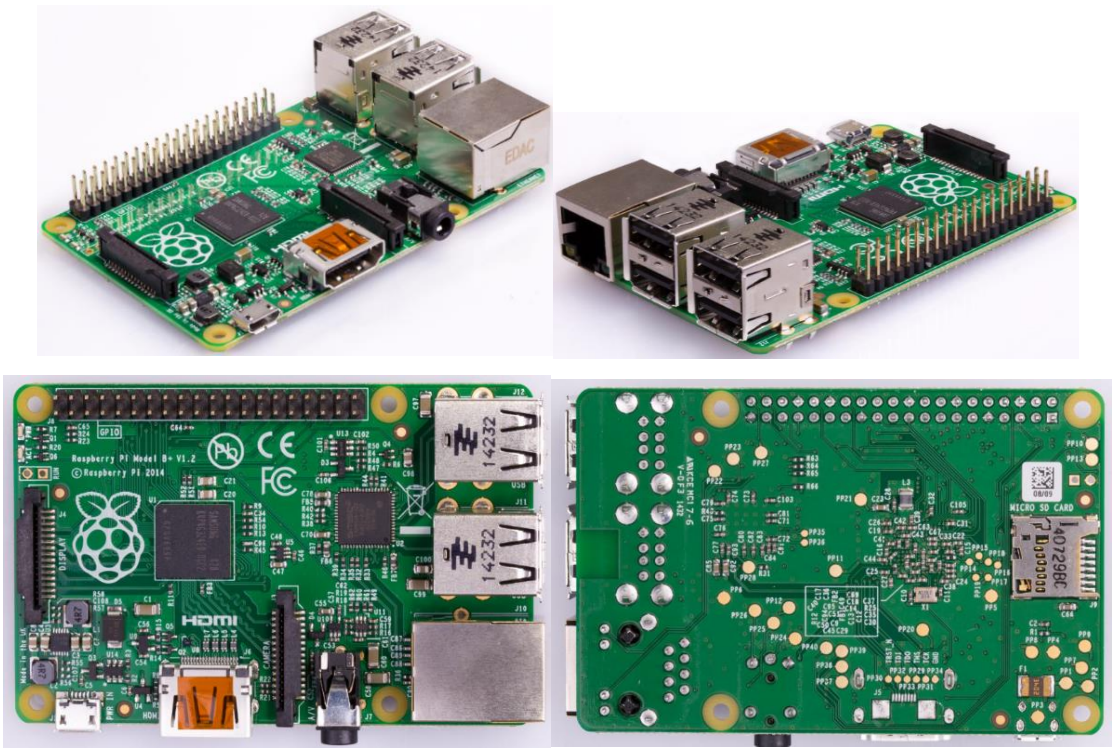

Figura A.2. Imagen de Raspberry Pi 1 Modelo B+ [49]

Raspberry Pi 2 Modelo B [50]

Con un procesador Broadcom BCM2836 SoC que mantiene una frecuencia de procesamiento de 900 MHz con bajo consumo de energía.

Este modelo de Raspberry se alimenta de un voltaje de 5V con una corriente de 2A, cuya fuente de energía que puede ser conectada mediante un puerto Micro-USB que servirá únicamente para este propósito.

Memoria Ram de 1 GB SDRAM

Puerto Ethernet 10/100 Base Y

4 puerto USB

Conector de display de 15 pines DSI (Display Serial Interface).

El conector de cámara que contiene es de 15 pines MIPI CSI-2 (Camera Serial Interface)

A parte del puerto HDMI también tiene una salida analógica de audio y video con un jack de 3.5 mm, con audio estéreo y video compuesto (PAL y NTSC).

Dimensiones 85 x 56 x 17 mm

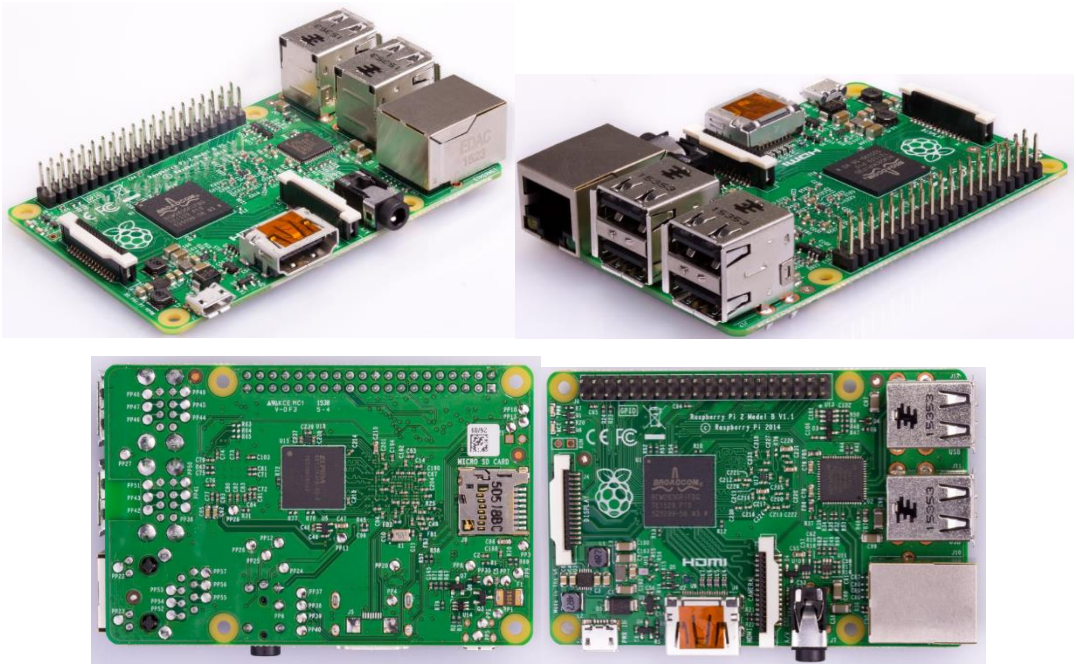

Figura A.3. Imagen de Raspberry Pi 2 Modelo B [50]

Raspberry Pi 3 Modelo B [51]

Con un procesador Broadcom BCM2837 que mantiene una frecuencia de procesamiento de 1250 MHz con bajo consumo de energía.

Este modelo de Raspberry se alimenta de un voltaje de 5V con una corriente de 2.5A, cuya fuente de energía que puede ser conectada mediante un puerto Micro-USB que servirá únicamente para este propósito.

Memoria Ram de 1 GB SDRAM

Puerto Ethernet

4 puerto USB

A diferencia de los modelos anteriores este ya contiene conectividad inalámbrica, en este modelo tiene una tarjeta WLAN y antena que permite el uso del protocolo IEEE 802.11 b/g/n y Bluetooth 4.1 así como BLE (Bluetooth Low Energy).

Conector de display de 15 pines DSI (Display Serial Interface).

El conector de cámara que contiene es de 15 pines MIPI CSI-2 (Camera Serial Interface)

A parte del puerto HDMI también tiene una salida analógica de audio y video con un jack de 3.5 mm, con audio estéreo y video compuesto (PAL y NTSC).

Dimensiones 85 x 56 x 17 mm

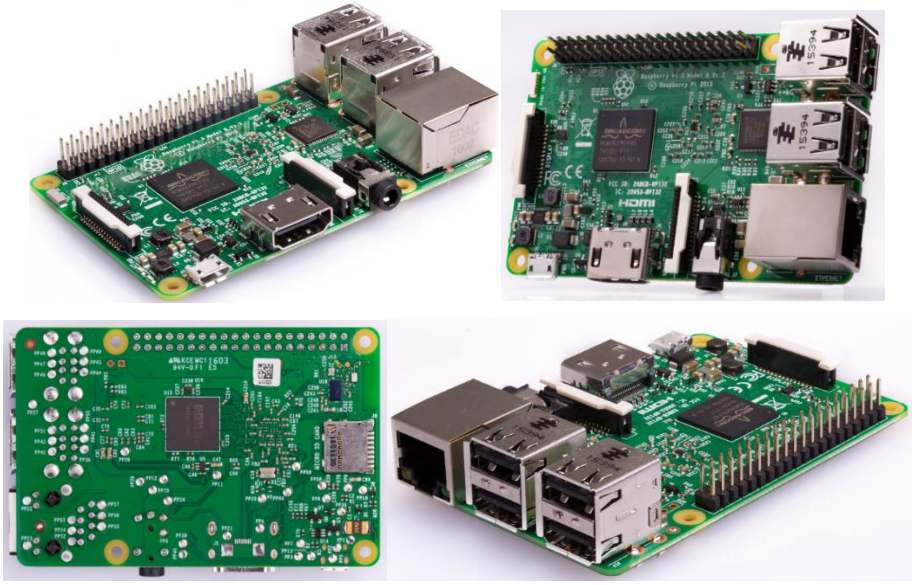

Figura A.4. Imagen de Raspberry Pi 3 Modelo B [51]

Raspberry Pi 3 Modelo B+ [8]

Con un procesador Broadcom BCM2837 que mantiene una frecuencia de procesamiento de 1.4 GHz con bajo consumo de energía.

Este modelo de Raspberry se alimenta de un voltaje de 5V con una corriente de 2.5A, cuya fuente de energía que puede ser conectada mediante un puerto Micro-USB que servirá únicamente para este propósito.

Memoria Ram de 1 GB LPDDR2 SDRAM

Puerto Ethernet

4 puerto USB

Este modelo tiene una tarjeta WLAN y antena que permite el uso del protocolo IEEE 802.11 b/g/n/ac y Bluetooth 4.2 así como BLE (Bluetooth Low Energy).

Conector de display de 15 pines DSI (Display Serial Interface).

El conector de cámara que contiene es de 15 pines MIPI CSI-2 (Camera Serial Interface)

A parte del puerto HDMI también tiene una salida analógica de audio y video con un jack de 3.5 mm, con audio estéreo y video compuesto (PAL y NTSC).

Dimensiones 85 x 56 x 17 mm

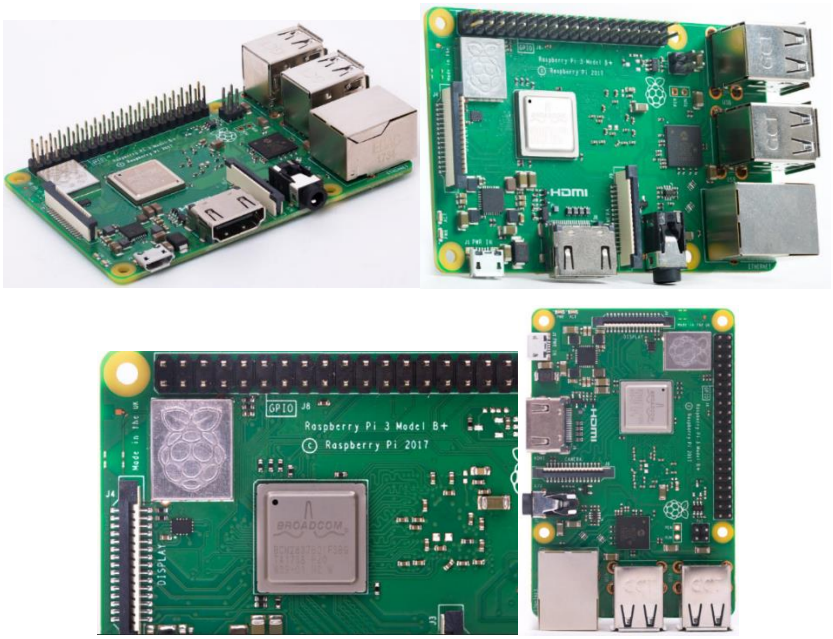

Figura A.5. Imagen de Raspberry Pi 3 Modelo B+ [8]

Raspberry Pi 3 Modelo A+ [52]

Con un procesador Broadcom BCM2837B1 que mantiene una frecuencia de procesamiento de 1.4 GHz con bajo consumo de energía.

Este modelo de Raspberry se alimenta de un voltaje de 5V con una corriente de 2.5A, cuya fuente de energía que puede ser conectada mediante un puerto Micro-USB que servirá únicamente para este propósito.

Memoria Ram de 512 MB LPDDR2 SDRAM

Puerto Ethernet

1 puerto USB

Este modelo tiene una tarjeta WLAN y antena que permite el uso de las frecuencias 2.4 GHz y 5 GHz del protocolo IEEE 802.11 b/g/n/ac y Bluetooth 4.2 así como BLE (Bluetooth Low Energy).

El conector de cámara que contiene es de 15 pines MIPI CSI-2 (Camera Serial Interface)

A parte del puerto HDMI también tiene una salida analógica de audio y video con un jack de 3.5 mm, con audio estéreo y video compuesto (PAL y NTSC).

Dimensiones 65 x 56 x 14 mm

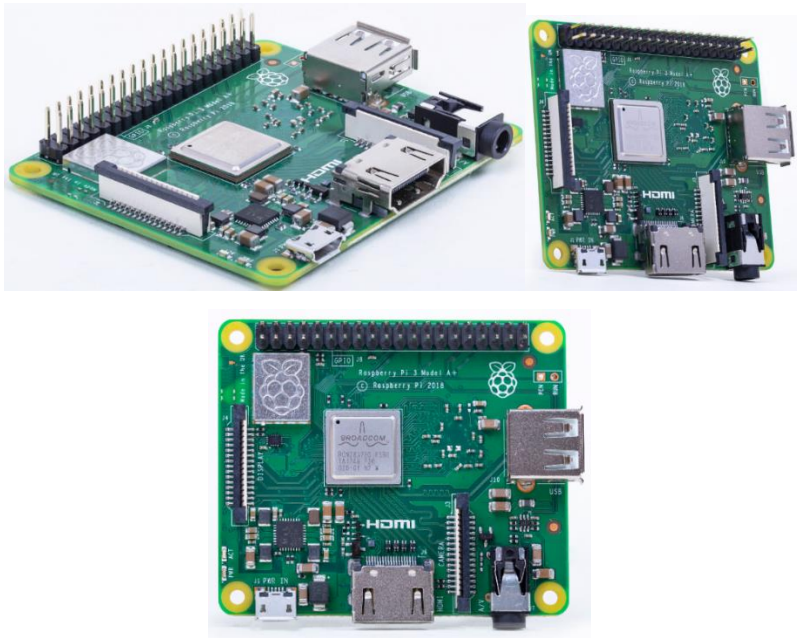

Figura A.6. Imagen de Raspberry Pi 3 Modelo A+ [52]

Raspberry Pi Zero [53]

Con un procesador Broadcom BCM2835 que mantiene una frecuencia de procesamiento de 1 GHz con bajo consumo de energía.

Este modelo de Raspberry se alimenta de un voltaje de 5V con una corriente de 2.5A, cuya fuente de energía que puede ser conectada mediante un puerto Micro-USB que servirá únicamente para este propósito.

Memoria Ram de 512 MB

1 puerto Micro-USB OTG

El conector de cámara que contiene es de 22 pines MIPI CSI-2 (Camera Serial Interface) compatible con cámaras con conector de 15 pines mediante un cable flexible adecuado.

A parte del puerto micro-HDMI también tiene una salida analógica de video con compuesto (PAL y NTSC) en un par de pines adicionales a los pines GPIO.

Dimensiones 65 x 30 x 5 mm

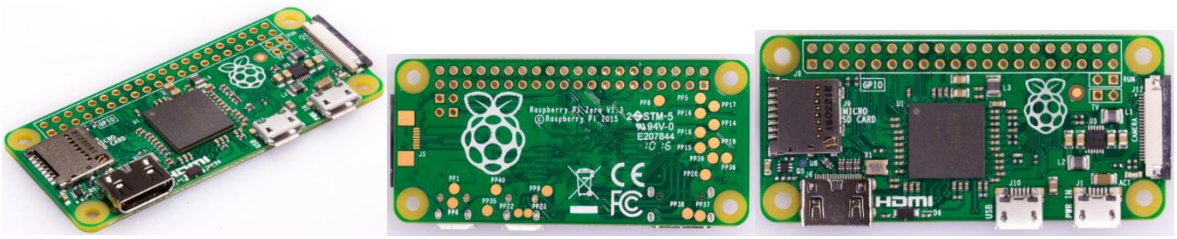

Figura A.7. Imagen de Raspberry Pi Zero [53]

Raspberry Pi Zero W [7]

Con un procesador Broadcom BCM2835 que mantiene una frecuencia de procesamiento de 1 GHz con bajo consumo de energía.

Este modelo de Raspberry se alimenta de un voltaje de 5V con una corriente de 2.5A, cuya fuente de energía que puede ser conectada mediante un puerto Micro-USB que servirá únicamente para este propósito.

Memoria Ram de 512 MB

1 puerto Micro-USB OTG

Este modelo tiene una tarjeta WLAN y antena que permite el uso de las frecuencias 2.4 GHz del protocolo IEEE 802.11 b/g/n y Bluetooth 4.1 así como BLE (Bluetooth Low Energy).

Conector de display de 15 pines DSI (Display Serial Interface).

El conector de cámara que contiene es de 22 pines MIPI CSI-2 (Camera Serial Interface) compatible con cámaras con conector de 15 pines mediante un cable flexible adecuado.

A parte del puerto micro-HDMI también tiene una salida analógica de video con compuesto (PAL y NTSC) en un par de pines adicionales a los pines GPIO.

Dimensiones 65 x 30 x 5 mm

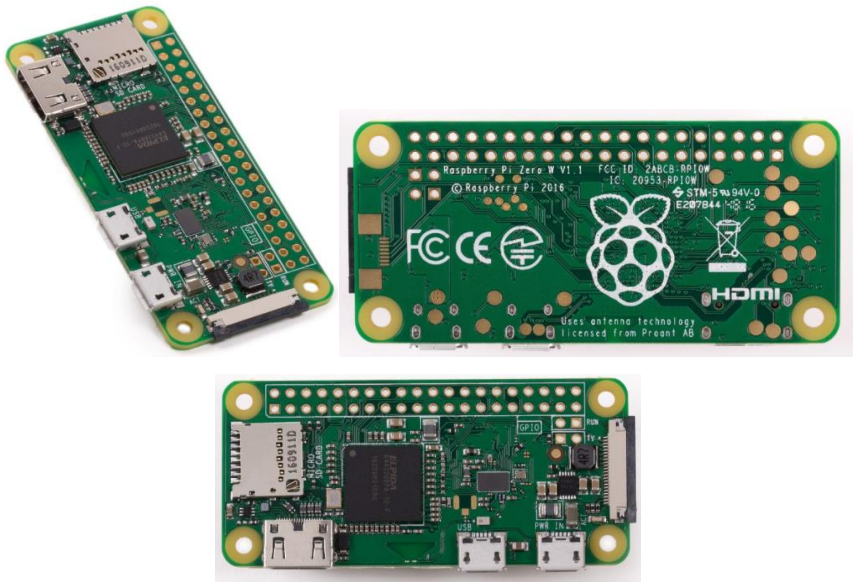

Figura A.6. Imagen de Raspberry Pi Zero W [7]

Raspberry Pi 4B [58]

Con un procesador Quad core 64-bit ARM-Cortex A72 con una velocidad de reloj de 1.5 GHz.

Memoria Ram de 1, 2, o 4 GB LPDDR4 RAM

Tarjeta de video VideoCore VI 3D Graphics.

Este modelo de Raspberry se alimenta de un voltaje de 5V con una corriente de 3A, cuya fuente de energía que puede ser conectada mediante un puerto Micro-USB Tipo C que servirá únicamente para este propósito.

Puerto Ethernet

2 puerto USB2.0 y 2 USB3.0.

Este modelo tiene una tarjeta WLAN y antena que permite el uso de las frecuencias 2.4 GHz y 5 GHz del protocolo IEEE 802.11 b/g/n/ac y Bluetooth 5 con BLE (Bluetooth Low Energy).

El conector de cámara que contiene es de 15 pines MIPI CSI-2 (Camera Serial Interface)

A parte de los 2 puertos micro-HDMI también tiene una salida analógica de audio y video con compuesto (PAL y NTSC) mediante un conector Jack 'A/V' TRS de 4 pines.

Dimensiones 85 x 56 x 14 mm

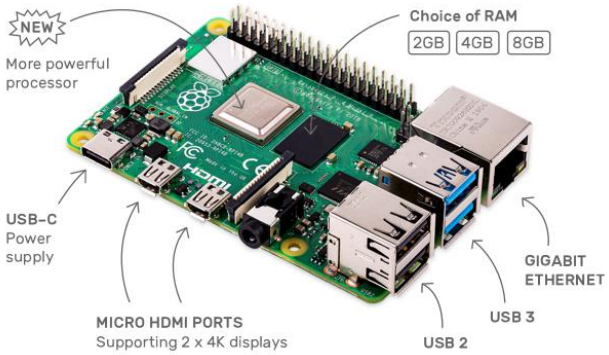

Figura A.7. Imagen de Raspberry Pi 4B [59]

ANEXO B

B. Preparación de la Raspberry Pi

Previo a realizar la programación en Python y poder ejecutar los programas para que funcione esta micro-computadora como el Video Portero deseado, es necesario inicializar la Raspberry Pi, así como instalar librerías y programas adicionales para su funcionamiento. Por lo que a continuación se detallarán los pasos requeridos para esto.

B.1. Sistema Operativo para Raspberry Pi

En primer lugar, es necesario elegir el sistema operativo que contenga los elementos necesarios para poder realizar el proyecto. Para poder elegir entre los diversos sistemas operativos para Raspberry Pi, existe en la página oficial de este producto, varias opciones para poder descargarlo [36]. En esta página se podrá observar S.O. basados en Debian, que es el S.O. oficial de este producto 'Raspbian' con el logo que se puede observar en la Figura B.1.

Raspbian

Figura B.1. Logo de SO oficial basado en Debian Buster. [36]

Pero así mismo se pueden encontrar otros S.O. desarrolladas por otras compañías como se puede observar en la Figura B.2.

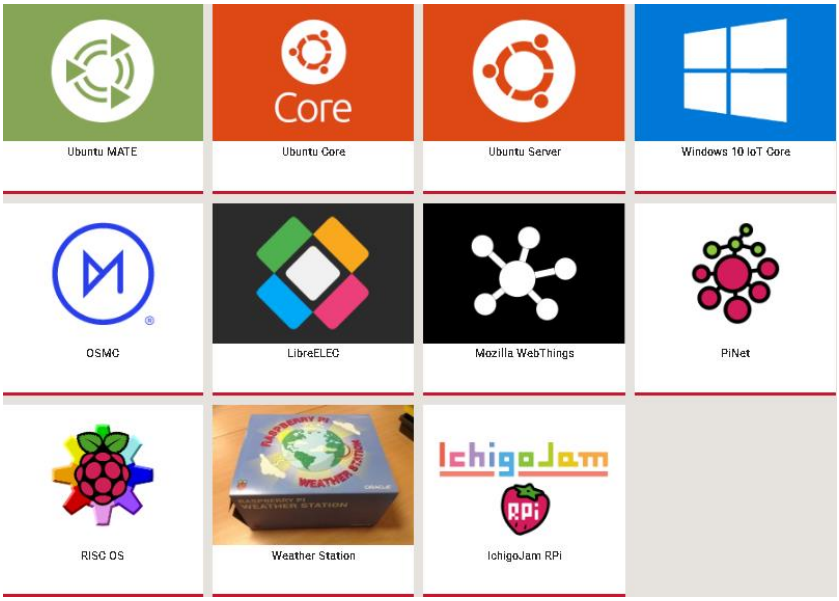

Figura B.2. Logos de SO disponibles para la instalación en la Raspberry Pi [36].

Tomando en cuenta estas opciones se escogió el S.O. Raspbian por ser un sistema operativo oficial de Raspberry Pi y por lo tanto tiene varios sitios y foros a donde se puede recurrir por ayuda en caso de ser necesario. Así mismo Raspbian Buster tiene varias opciones de descarga [37], con Escritorio y software pre instalado "Raspbian Buster with desktop and recommended software", únicamente escritorio y software básico para su funcionamiento "Raspbian Buster with Desktop", o sin escritorio (es decir únicamente para controlarlo mediante comandos) "Raspberry Buster Lite" [37]. Entre estas opciones se eligió el "Raspberry Buster Lite" ya es ligero en comparación a las otras opciones y no es requerido tener un escritorio para que este pueda funcionar como Video Portero.

Instalación de Raspbian Buster

1. Descargar la imagen del S.O. requerido, esto puede ser mediante Torrent o descarga directa de la página, estas dos opciones se encontrarán en la página oficial de Raspberry Pi [37].

2. Montar el S.O. en la tarjeta microSD a ocupar, una vez teniendo la imagen descargada, mediante el uso del programa balenaEtcher [38] recomendado por Raspberry Pi Foundation o cualquier otro programa que permita el montaje de un sistema operativo en tarjetas SD o USB driver.

3. Después de realizar la escritura del S.O. sobre la tarjeta microSD, está lista para ser colocada en la Raspberry Pi y al momento de conectarlo empezará el encendido.

B.2. Configuración inicial y configuración de Red de Raspberry Pi

Una vez instalado el S.O., es necesario conectar la Raspberry Pi a un monitor mediante un cable HDMI y un teclado para poder realizar las configuraciones iniciales.

Cuando el dispositivo se encuentra encendido ocupando el S.O. Raspbian Buster Lite, únicamente se abrirá una ventana de comandos, permitiendo ocupar únicamente el teclado para su interacción y pedirá un usuario y una contraseña después de este, en un inicio el usuario y contraseña van a ser:

login as: pi
pi's password: raspberry

Datos que podrán ser cambiados si es necesario.

Después de ingresar por primera vez a este SO es necesario realizar actualizaciones de este para su correcto funcionamiento, esto se puede realizar mediante los siguientes comandos:

~$ sudo apt-get update
~$ sudo apt-get upgrate

Una vez actualizado el S.O., será necesario conectarlo a la red en la que se desee el Video Portero, esto es posible ingresando a una ventana de configuración, desde donde se pueden realizar varias configuraciones requeridas. Para ingresar a esta ventana se lo realiza con el comando:

~$ sudo raspi-config

Después del cual se observará la siguiente ventana mostrada en la Figura B.3.

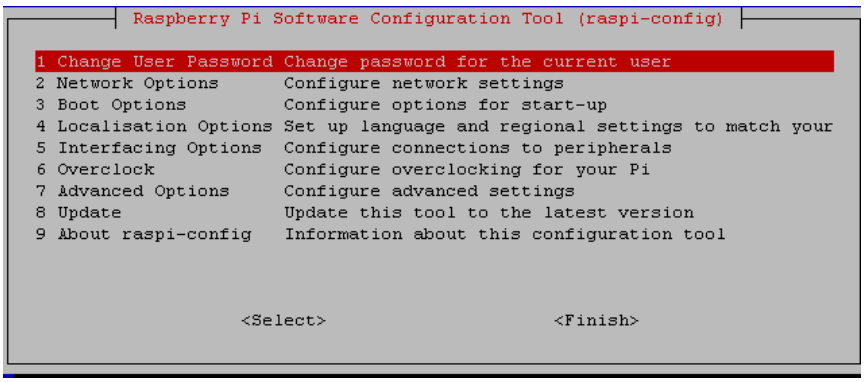

Figura B.3. Ventana de configuración de Raspberry Pi

-Para configurar una red inalámbrica en este dispositivo ingresamos a ‘Network Options’

-En la nueva ventana escogeremos la opción de Wi-fi

-Ingresamos el SSID seguido de la contraseña de este y después de colocar correctamente estos datos se presiona finalizar. Después solicitará reiniciar el dispositivo para que pueda ser configurado.

Pero antes de reiniciar es necesario colocar una dirección IP fija y de este modo evitar que cambie seguido dependiendo del DHCP ocupado. Esto es posible configurando el documento /etc/dhcpcd.conf ingresando el siguiente comando para editar el archivo:

~$ nano /etc/dhcpcd.conf

E ingresando las siguientes líneas de texto dependiendo de la dirección IP de red:

interface wlan0
static ip_address=192.168.1.150/24
static routers=192.168.1.1
static domain_name_servers=192.168.1.1 8.8.8.8

Una vez culminada la configuración del archivo se procede a guardar y cerrarlo para resetear la Raspberry.

Ya conectado a la red y con la dirección IP elegida para el dispositivo se procederá a activar el protocolo de comunicación SSH para poder continuar con la manipulación y programación del dispositivo remotamente sin la necesidad de conectarle un teclado o un monitor. Esto puede realizarse ingresando a la ventana de configuración antes mencionada (que se observa en la Figura 2.7.) e ingresando a la opción “Interfacing Options”, donde se verá la opción de SSH o simplemente ingresando los siguientes comandos:

~$ sudo systemctl enable ssh
~$ sudo systemctl start ssh

De este modo ya es posible ingresar desde una computadora con simulador de Terminal como “Putty” o “Hiperterminal” para poder comunicarse mediante SSH con la Raspberry.

Una vez realizadas estas configuraciones básicas se procederá a instalar las librerías y programas que serán necesarios antes de poder comenzar a programar en Python y para que funcionen todas las características que se requieren.

B.3. Activación de la cámara

Para poder empezar a ocupar la cámara es necesario activarla mediante la ventana de configuración. Ingresado a la ventana con el comando 'raspi-config' se selecciona la opción 'interfacing options', para ingresar a la ventana con las opciones observadas en la Figura B.4.

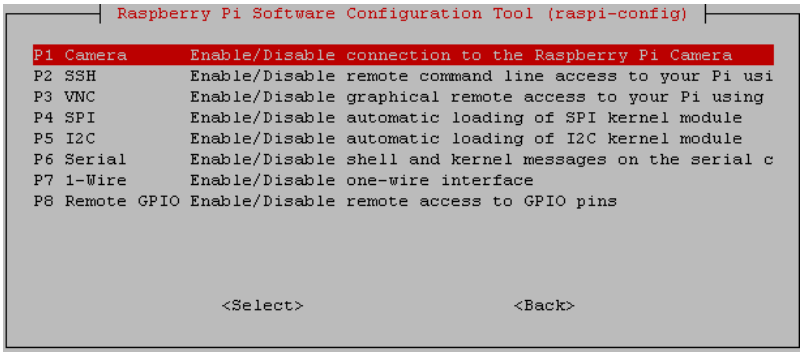

Figura B.4. Ventana de configuración de interfaces

Aquí ingresando en la opción "Camera" podemos activar o desactivar la cámara. Después de la cual tendremos que reiniciar el dispositivo.

B.4. Instalación de programas y librerías adicionales

B.4.1. Controlador de Tarjeta de Audio

Para poder usar la tarjeta de audio es necesario instalar un controlador de la tarjeta "ReSpeaker_2_Mics_pi_HAT" y es posible instalarlo con los siguientes pasos:

Primero descargamos los archivos necesarios para la instalación mediante el siguiente comando:

~$ git clone https://github.com/respeaker/seeed-voicecard.git

Después de descargarlo ingresamos a la carpeta descargada y procedemos con la instalación:

~$ cd seeed-voicecard
~$ sudo ./install.sh

una vez instalado se debe reiniciar el dispositivo para que concrete la instalación:

~$ sudo reboot

Para culminar, es necesario configurar el archivo /boot/config.txt que permite que todo el audio de ahora en adelante se reproduzca atreves de la tarjeta de audio que decíamos. Esto lo realizamos con el editor de texto nano y como usuario root de la siguiente forma:

~$ sudo nano /boot/config.txt

En este archivo se modificarán algunas de las últimas 20 líneas [39], por lo que a continuación se presenta en la Figura B.5. donde se puede observar en el lado izquierdo el antes y en el derecho el después de la parte modificada.

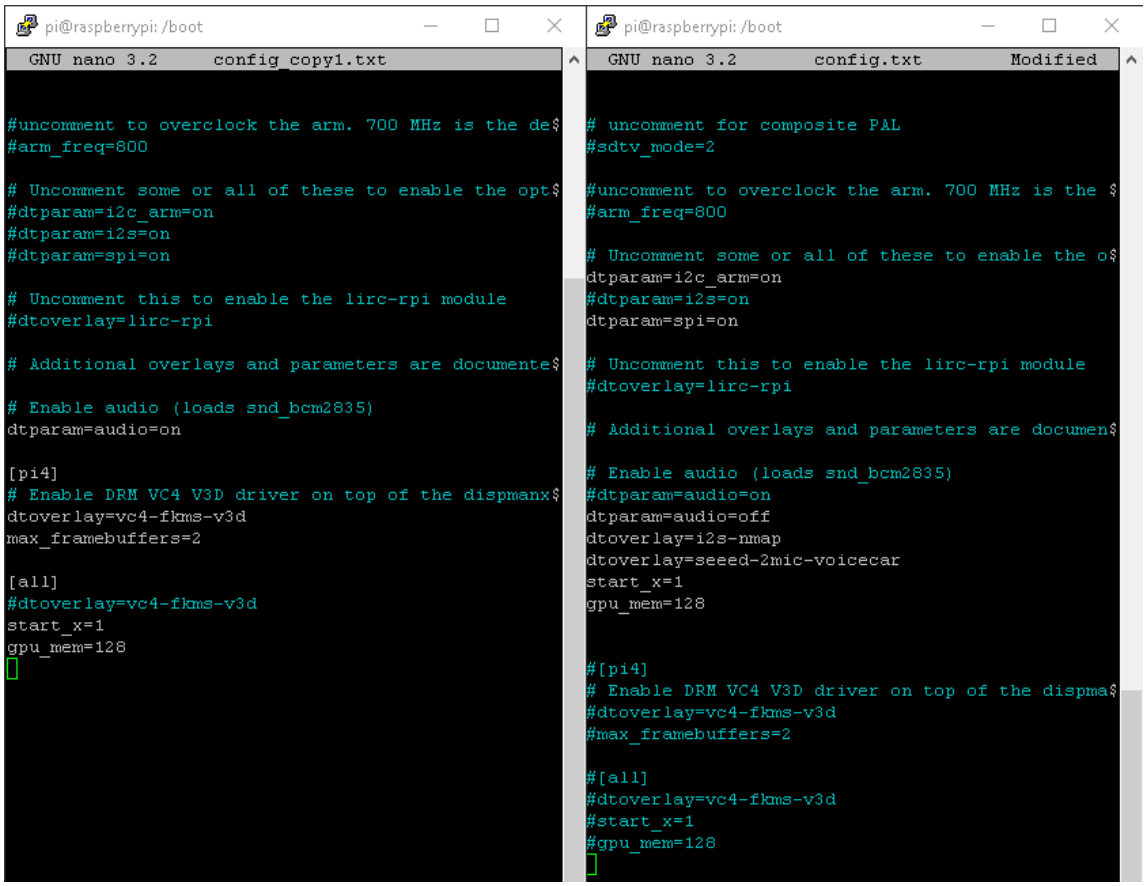

Figura B.5. Modificaciones de archivo '/boot/config.txt' antes/después.

B.4.2. Instalación de Asterisk

Un software principal que se ocupa para comunicación de voz es Asterisk y es el primero que se instalará debido a su complejidad.

En primer lugar descargamos el archivo 'asterisk-13.29.0-rc1.tar.gza' [39] a instalar de la página mencionada en la referencia cuya última actualización hasta el momento de realizar este escrito se realizó el 12 de septiembre del 2019. En esta página mantienen únicamente las últimas actualizaciones con algunos complementos adicionales por lo que es posible

que el comando que se menciona a continuación no funcione, en tal caso se deberá conseguir el url de la versión de Asterisk disponible en la página mencionada [39].

Para descargar Asterisk 13.29 se lo puede realizar con el siguiente comando:

wget http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-13.29.0-rc1.tar.gz,

antes de comenzar la instalación del programa descargado se comenzará instalando algunas dependencias necesarias para que este se instale debidamente mediante el siguiente comando:

sudo apt-get install libsqlite3-dev libncurses5-dev libxml2-dev libnewt-dev libssl-dev libiksemel-dev libgnutls28-dev libcurl4-openssl-dev libspandsp2 libspandsp-dev libmariadbd-dev mariadb-server unixodbc-dev unixodbc libportaudio-dev libical-dev libneon27-dev libportaudio-dev libspeex-dev speex libvorbis-dev libsrtp0-dev iptables-persistent libsox-dev,

Una vez iniciado el proceso de instalación solicitará la aprobación de la instalación de estos paquetes a lo que será necesario colocar “y” para confirmar. Lo mismo sucederá con el siguiente grupo de dependencias necesarias que se instalará con la siguiente línea de comando:

sudo apt-get install libuuid1 uuid uuid-dev libjansson4 libjansson-dev subversion sendmail sox ngrep

Una vez instaladas estas dependencias es posible la instalación de Asterisk, comenzando con descomprimir el archivo descargado con el siguiente comando:

tar -xf asterisk-13.29.0-rc1.tar.gz

se ingresa a la carpeta que se descomprimió. Una vez dentro de la carpeta mencionada se procede a instalar Asterisk con los siguientes comandos:

sudo ./configure --libdir=/usr/lib --with-pjproject-bundled

Culminando la ejecución del primer comando correctamente, se presentarán las siguientes líneas observadas en la Figura B.6, confirmando la primera etapa de la instalación y deberemos continuar con el resto de líneas de comando.

Figura B.6. Confirmación de instalación de Asterisk

sudo make menuconfig

Este comando permitirá la inclusión de algunos módulos en Asterisk, pero lo que nos interesa es permitir que acepte la lectura de formato MP3 por lo que señalamos la línea que dice “[ ] format_mp3” como se encuentra en la Figura B.7., y continuamos con el siguiente comando.

Figura B.7. Módulos y opciones para configuración de Asterisk

sudo contrib/scripts/get_mp3_source.sh

sudo make

Insertando el siguiente comando culmina con la instalación y configuración inicial de Asterisk, al realizarlo con éxito tendremos las líneas de texto mostradas en la Figura B.8.

sudo make install

A continuación, instalamos ejemplos de la configuración de los distintos archivos para tener una guía del formato que deben seguir:

sudo make samples

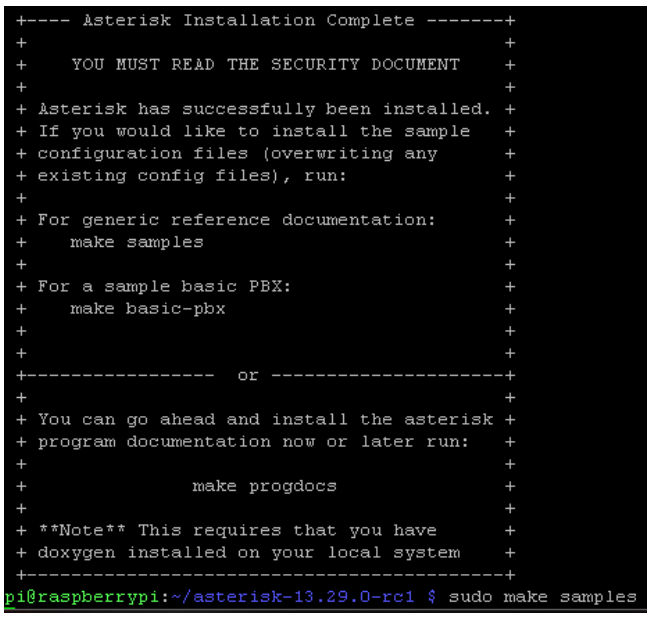

Figura B.8. Culminación con éxito de la instalación de Asterisk

Una vez culminada la instalación se procede a realizar las configuraciones necesarias e iniciar el sistema con el comando:

systemctl start Asterisk

Para culminar con la configuración de Asterisk agregando los usuarios que en este caso serán la Raspberry Pi (Video Portero) y el dispositivo Android, en el archivo "etc/asterisk/sip.conf" es necesario añadir estas líneas:

[friends_internal](!)

type=friend

host=dynamic

context=local_test

disallow=all

allow=ulaw

[Rasp](friends_internal)

secret=password

[Android](friends_internal)

secret=password

Para configurar el nombre de usuario y contraseña que tendrán en Asterisk.

Mientrasa que en el archivo “/etc/asterisk/extensions.conf”, se configuran las extenciones a ocupar y se deberán añadir de igual forma al final del archivo las siguientes lineas:

[local_test]

exten=>100,1,Dial(SIP/Rasp,20)

exten=>101,1,Dial(SIP/Android,20)

B.4.3. Instalación de Librerías adicionales para Python

Empezamos instando un programa que permite la instalación de librerías que mantienen almacenado en servidores dedicados para estas, mediante el siguiente comando:

sudo apt-get install python3-pip

después de esto ya es posible instalar otras librerías.

Por lo que continuamos instalando la librería que permite el uso de los pines GPIO de la Raspberry Pi mediante el siguiente comando [40]:

sudo pip3 install RPi.GPIO

Para poder enviar notificaciones a un celular Android de igual forma es necesario instalar una librería, de este modo se podrá instalar con el siguiente comando [41]:

sudo pip3 install pyfcm

Instalamos la librería “flask” para poder realizar una transmisión broadcast de video atreves de HTTP por lo que lo instalamos con el siguiente comando [42]:

sudo pip3 install flask

De igual forma para poder ocupar la cámara mediante Python es necesario instalar la librería “picamara” mediante el siguiente comando [43]:

sudo apt-get install python-picamera

- Instalación de la librería Most.voip.api

Esta librería para Python ocupa propiedades de otra librería llamada PJSIP que es escrita en C, por lo que es necesario instalar esta antes de proceder a la instalación de la librería que se ocupará en la programación con Python.

En primer lugar, se descargará la última versión de PJSIP que hasta el momento es la versión PJSIP 2.9 que fue liberado el 13 de junio del 2019, y se puede descargar con el siguiente comando:

wget https://www.pjsip.org/release/2.9/pjproject-2.9.tar.bz2

una vez descargada la carpeta, la descomprimimos e ingresamos a esta:

tar xjf pjproject-2.8.tar.bz2
cd pjproject-2.9

para realizar la instalación es necesario realizarlo como usuario root por lo que se deberá comenzar cada línea de comando con ‘sudo’ que permite la ejecución de esta línea de comando como usuario root, como se ve a continuación:

sudo ./configure --disable-libwebrtc
sudo make dep
sudo make
sudo make install
cd pjsip-apps/src/python/
sudo python setup.py install

una vez instalado PJSIP será posible la instalación la librería “most.voip.api” que es la librería que se usa directamente en la programación para este proyecto de titulación.

Pero antes de poder descargar el archivo requerido para esta instalación es necesario la instalación de otro programa que permite la descarga desde un servidor dedicado de este u otros programas, mediante el siguiente comando [44]:

sudo apt-get install wget git-core

después de esto se podrá descargar la carpeta necesaria para después ingresar a esta y después copiar la carpeta requerida a la librería de Python con los siguientes comandos [20]:

git clone https://github.com/crs4/most-voip.git
cd most-voip/python/src
sudo cp -R most /usr/local/lib/python2.7/dist-packages

Por ultimo para poder comprobar la correcta instalación de esta librería se lo puede hacer iniciando python2.7 e ingresando en este la siguiente línea:

import most.voip.api

y al no tener un error en la ejecución se podrá conocer su correcto funcionamiento.

ANEXO C

Diagrama de directorios y archivos desarrollados.

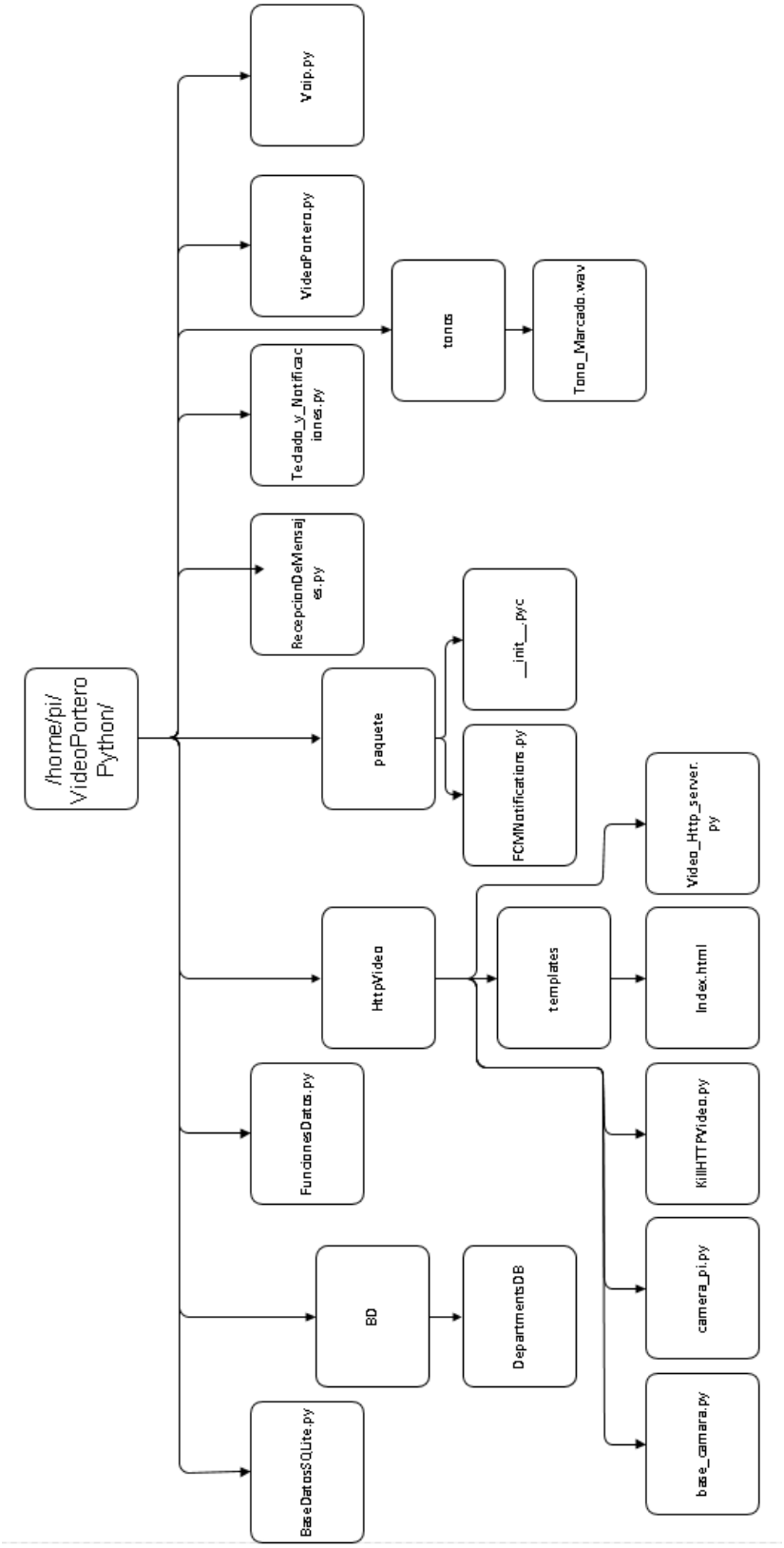

Figura C.1. Diagrama de directorios y archivos desarrollados

ANEXO D

Código del archivo “VideoPortero.py” programa inicial del Video Portero.

```
import BaseDatosSQLite
import RecepcionDeMensajes
import time
import subprocess
BaseDatosSQLite.NewDataBase()
try:
    DepID="DEP_ADMIN"
    DepPass="password"
    UserAdmin=""
    UserName="ADMIN"
    UserPass="password"
    MacAdd=""
    IPadd=""
BaseDatosSQLite.AddInformation(DepID,DepPass,UserAdmin,UserName,UserPass,Mac
Add,IPadd)
    tiempo_st = time.localtime()
    print (time.asctime(tiempo_st) + "Inicialización de Video Poetero")
except:
    tiempo_st = time.localtime()
    print (time.asctime(tiempo_st)+"Video Portero a sido reiniciado")
subprocess.Popen(["python2.7","/home/pi/VideoPorteroPython/Teclado_y_Notificaciones.
py"])
time.sleep(5)
RecepcionDeMensajes.RecepcionDeMensaje()
```

ANEXO E

Código del archivo “Teclado_y_Notificaciones.py’ programa que controla el teclado y gestiona el envío de Notificaciones al dispositivo Android.

```
import RPi.GPIO as GPIO
import BaseDatosSQLite
import time
from paquete import FCMNotifications
import subprocess
GPIO.setmode(GPIO.BCM)

out_total=[9,22,27,4]
GPIO.setup(out_total,GPIO.OUT)

in_total=[26,16,6,5]
GPIO.setup(in_total,GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
print("reading keyboard")
DepartNumber=["DEP_"]
while 1:
    GPIO.output(out_total,(1,0,0,0))
    time.sleep(0.005)
    if GPIO.input(in_total[0])==1:
        a="1"
        print(a)
        DepartNumber.append(a)
        time.sleep(0.25)
    if GPIO.input(in_total[1])==1:
        a="2"
        print(a)
        DepartNumber.append(a)
        time.sleep(0.25)
    if GPIO.input(in_total[2])==1:
        a="3"
        print(a)
        DepartNumber.append(a)
```

```
    time.sleep(0.25)
if GPIO.input(in_total[3])==1:
    a="A"
    print(a)
    DepartNumber.append(a)
    time.sleep(0.25)
GPIO.output(out_total,(0,1,0,0))
time.sleep(0.005)
if GPIO.input(in_total[0])==1:
    a="4"
    print(a)
    DepartNumber.append(a)
    time.sleep(0.25)
if GPIO.input(in_total[1])==1:
    a="5"
    print(a)
    DepartNumber.append(a)
    time.sleep(0.25)
if GPIO.input(in_total[2])==1:
    a="6"
    print(a)
    DepartNumber.append(a)
    time.sleep(0.25)
if GPIO.input(in_total[3])==1:
    a="B"
    print(a)
    DepartNumber.append(a)
    time.sleep(0.25)

GPIO.output(out_total,(0,0,1,0))
time.sleep(0.005)
if GPIO.input(in_total[0])==1:
    a="7"
    print(a)
    DepartNumber.append(a)
    time.sleep(0.25)
```

```
if GPIO.input(in_total[1])==1:
    a="8"
    print(a)
    DepartNumber.append(a)
    time.sleep(0.25)
if GPIO.input(in_total[2])==1:
    a="9"
    print(a)
    DepartNumber.append(a)
    time.sleep(0.25)
if GPIO.input(in_total[3])==1:
    a="C"
    print(a)
    subprocess.Popen(["killall","aplay"])
    DepartNumber=["DEP_"]
    time.sleep(0.25)

GPIO.output(out_total,(0,0,0,1))
time.sleep(0.005)
if GPIO.input(in_total[0])==1:
    a="*"
    print(a+"ADMIN")
    DepartNumber.append("ADMIN")
    time.sleep(0.25)
if GPIO.input(in_total[1])==1:
    a="0"
    print(a)
    DepartNumber.append(a)
    time.sleep(0.25)
if GPIO.input(in_total[2])==1:
    a="#"
    print(a+" Delete All")
    time.sleep(0.25)
    DepartNumber=["DEP_"]
if GPIO.input(in_total[3])==1:
    a="D"
```

```
        print(a+" Enter")
        time.sleep(0.25)
        if len(DepartNumber)>1 and len(DepartNumber)<4:
            DepID=''.join(DepartNumber)
            print(DepID)
            DepartNumber=["DEP_"]
            #Ingresar a base de datos para verificar Departamento
            try:
                print("Intenta leer base de datos")
                BD=BaseDatosSQLite.ReadInformationColumnOfDep("TOCKEN",DepID)
                print(BD)
                print(len(BD))
                if len(BD)>0:
                    for i in range(0,len(BD)):
                        a=list(BD[i])
                        BD[i]=a[0]
                    print(BD)
                    if len(BD)==1:
                        DIR=BD[0]
                        FCMNotifications.SendNotifucation(DIR)
                    else:
                        FCMNotifications.SendNotifucations(BD)

subprocess.Popen(["python3.5","/home/pi/VideoPorteroPython/HttpVideo/Video_Http_server.py"])

subprocess.Popen(["python3.5","/home/pi/VideoPorteroPython/HttpVideo/KillHTTPVideo.py"])

subprocess.Popen(["aplay","/home/pi/VideoPorteroPython/tones/Game_of_Thrones_Theme.wav"])

            except:
                subprocess.Popen(["killall","python3.5"])
                print("No existe departamento")
                DepartNumber=["DEP_"]
```

```
    #
      else:
         DepartNumber=["DEP_"]

GPIO.cleanup(out_total)
GPIO.cleanup(in_total)
```

Código del archivo "FCMNotifications.py" programa que controla el envío de notificaciones al celular Android.

```
from pyfcm import FCMNotification
import time
push_service = FCMNotification(api_key="AAAAA7aa8sE:APA91bGT7mYs1wxzUX0Qc7rBFQpcEQtbckKp2Qb83ic29de-McTefyyq2iO_T3olnyMTOaFPKogmE-B1nYzzromX9ORwvZdDvV3GL8F5UCQXZjtqgAq$

def SendNotifucation(registration_id):
   tiempo_st = time.localtime()
   message_title = "Llamada de la puerta"
   message_body = ("Desea contestar a la puerta?"+time.asctime(tiempo_st))
   result = push_service.notify_single_device(registration_id=registration_id, message_title=message_title, message_body=message_body)
   print (result)


def SendNotifucations(registration_ids):
   tiempo_st = time.localtime()
   message_title = "Llamada de la puerta"
   message_body = ("Desea contestar a la puerta?"+time.asctime(tiempo_st))
   result = push_service.notify_multiple_devices(registration_ids=registration_ids, message_title=message_title, message_body=message_body)
   print (result)
```

ANEXO F

Código del archivo “Recepcion_de_Mensajes.py” que permite la recepción de los mensajes de control

```
import socket
import sys
import time
import FuncionesDatos
def RecepcionDeMensaje():
    tiempo_st = time.localtime()
    HOST = ' '
    PORT = 5005
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print (time.asctime(tiempo_st)+'-socket creado')
    Skt=1
    n=0
    while Skt:
        try:
            s.bind((HOST, PORT))
            Skt=0
        except socket.error as err:
            n+=1
            tiempo_st = time.localtime()
            print (time.asctime(tiempo_st)+' Bind Failed, Error Code: ' + str(err[0]) + ', Message: ' + err[1])
            print ("Intento de conección a Socket "+str(n))
            time.sleep(5)
            # sys.exit()
            # s.close()
    #print ('Socket Bind Success!')

    #listen(): This method sets up and start TCP listener.
    s.listen(10)
    print ('Socket ya esta escuchando')

    datos=[]
    while 1:
```

```
        #try:
        conn, addr = s.accept()
        print (time.asctime(tiempo_st)+'- Conectando con ' + addr[0] + ':' + str(addr[1]))
        buf = conn.recv(500)
        buf=str(buf)
        print ("El Mensaje es "+buf)
        datos=buf.split()
        datos.insert(0,addr[0])
        Acceso=FuncionesDatos.DataType1(datos)
        #except:
                #s.close()
        try:
                if Acceso[0]=="TRUE":
                        data=Acceso[2]
                        print ("enviando mensaje de vuelta al cliente: "+data)
                        data=data.encode()
                        conn.send(data)
                        conn.close()
                datos=[]
        except:
                data="Incorrecto"
                print ("enviando mensaje de vuelta al cliente: "+data)
                data=data.encode()
                conn.send(data)
                conn.close()
                datos=[]
s.close()
```

ANEXO G

Código del archivo “FuncionesDatos.py” que gestiona la autenticación e identifica los mensajes de control.

```
import BaseDatosSQLite
import Voip
import time
import subprocess
import  RPi.GPIO as GPIO
tiempo_st = time.localtime()
global call_free
call_free=True
global call_name

GPIO.setmode(GPIO.BCM)
GPIO.setup(23,GPIO.OUT)
GPIO.output(23,1)

def DataType1(datos):
    global call_free
    global call_name
    acceso=[1,2,3]
    IPadd=datos[0]
    #IPadd=IPadd[0]
    try:
        v=datos.index("ACCION")
        w=datos.index("ID_DEP")
        try:
            x=datos.index("CLAVE_DEP")
        except:
            print (time.asctime(tiempo_st)+"-No se recibió la CLAVE_DEP")
    except:
        subprocess.Popen(["killall","python3.5"])
        acceso[0]="FALSE"
        acceso[1]="FALSE"
        acceso[2]="RONG"
        return acceso
```

```
#       print datos[w+1]
        miLista=BaseDatosSQLite.ReadInformationOfDep(datos[w+1])
#       print miLista
        try:
                y=datos.index("USER_NAME")
                z=datos.index("CLAVE_USER")
                n=len(miLista)
#               print n
#               print miLista
                if n == 0:
                        miLista=miLista[0]
                        miLista=list(miLista)

                else:
                        for k in range(0,n):
#                               print miLista
                                if datos[y+1] in miLista[k]:
                                        for i in range(0,n):
                                                if (datos[y+1] in miLista[i]):
                                                        miLista=miLista[i]
#                                                       print miLista
                                                        break
                                        break

                                else:
                                        for j in range(0,n):
                                                if ("datos[w+1]" in miLista[j] and  "" in miLista[j] ):
                                                        miLista=miLista[i]
#                                                       print "j="
#                                                       print j
#                                                       print miLista
                                                        break

                        miLista=list(miLista)
#               print miLista
```

```
        except:
            print (time.asctime(tiempo_st)+"-No se extrajo adecuadamente la lista")
        try:
            a=miLista.index(datos[w+1])
        except:
            try:
                miLista=miLista[0]
                miLista=list(miLista)
            except:
                print ("Error")
        Dep_ID1=miLista[0]
        try:
            Clave_dep1=miLista[1]
        except:
            print (time.asctime(tiempo_st)+"-No se extrajo la CLAVE_DEP de la Lista")
        try:
            User_Name1=miLista[3]
            Clave_User1=miLista[4]
        except:
            print(time.asctime(tiempo_st)+"-No se extrajo el USER_NAME o CLAVE_USER
de la Lista")

        try:
            if (call_free or User_Name1==call_name):
                if(datos[v+1]=="START_HTTP_VIDEO"):
subprocess.Popen(["python3.5","/home/pi/VideoPorteroPython/HttpVideo/Video_Http_ser
ver.py"])
                    acceso[0]="TRUE"
                    acceso[1]="TRUE"
                    acceso[2]="HTTP_RUNING"
                    return acceso
                elif(datos[v+1]=="STOP_HTTP_VIDEO"):
                    subprocess.Popen(["killall","python3.5"])
                    acceso[0]="TRUE"
                    acceso[1]="TRUE"
                    acceso[2]="HTTP_STOP"
```

```
                    return acceso
            #elif(datos[v+1]=="START_HTTP_VIDEO"):

#subprocess.Popen(["python3.5","/home/pi/VideoPorteroPython/HttpVideo/Video_Http_se
rver.py"])
                    #acceso[0]="TRUE"
                    #acceso[1]="TRUE"
                    #acceso[2]="HTTP_RUNING"
                    #return acceso
            # elif(datos[v+1]=="STOP_HTTP_VIDEO"):
                   # acceso[0]="TRUE"
                    #acceso[1]="TRUE"
                    #acceso[2]="HTTP_STOPED"
                    #return acceso
             if(datos[w+1]==Dep_ID1 and datos[x+1]==Clave_dep1):
                    print (time.asctime(tiempo_st)+"-Datos de Dep correcto")
                    if(datos[y+1]==User_Name1 and datos[z+1]==Clave_User1):
                        print (time.asctime(tiempo_st)+"-Datos de Usuario correcto")
                        if(datos[v+1]=="RECUSERACCESS"):
                            acceso[0]="TRUE"
                            acceso[1]="TRUE"
                            acceso[2]="USER_AUTORIZADO"
                            return acceso
                        elif (datos[v+1]=="SPECTING_CALL"):
                            if (call_free):
                                acceso[0]="TRUE"
                                acceso[1]="TRUE"
                                acceso[2]="RECEIVING_CALL"
                                call_free=False
                                call_name=datos[y+1]
                                print ("Call_Free: "+str(call_free))
                                subprocess.Popen(["killall","aplay"])

subprocess.Popen(["python2.7","/home/pi/VideoPorteroPython/Voip.py"])
                                return acceso
                            else:
```

```
                        acceso[0]="TRUE"
                        acceso[1]="TRUE"
                        acceso[2]="CALL_BUSY"
                        print ("Call_Free: "+str(call_free))
                        return acceso

                elif (datos[v+1]=="CLOSE_CALL" and call_name==User_Name1):
                    acceso[0]="TRUE"
                    acceso[1]="TRUE"
                    acceso[2]="CLOSE__CALL"
                    call_free=True
                    print ("Call_Free: "+str(call_free))
                    subprocess.Popen(["killall","python3.5"])
                    return acceso
                elif (datos[v+1]=="OPEN_DOOR"):
                    GPIO.output(23,0)
                    time.sleep(1)
                    GPIO.output(23,1)
                    acceso[0]="TRUE"
                    acceso[1]="TRUE"
                    acceso[2]="DOOR_IS_OPEN"
                    return acceso
##########################################################################
############
                elif(datos[v+1]=="CONFIG"):
                    if(datos[w+1]=="DEP_ADMIN"):
                        acceso[0]="TRUE"
                        acceso[1]="TRUE"
                        acceso[2]="ADMIN"
                        try:
                            xx=datos.index("NEW_DEP")
                            yy=datos.index("NEW_DEP_PASS")
                            BaseDatosSQLite.AddInformation(datos[xx+1],
datos[yy+1], datos[xx+1], datos[xx+1], datos[yy+1], datos[xx+1], datos[xx+1])
                            lista=BaseDatosSQLite.ReadInformationColum(
                            "ID_Dep")
```

```
                    b=len(lista)
                    for i in range (0,b):
                        a=list(lista[i])
                        lista[i]=a[0]
                #----se elimina los nombres de departamentos repetidos
                    lista1=[]
                    j=0
                    for i in range(0,b):
                        lista1.append(lista[i])
                        n=lista1.count(lista1[j])
                        if n>1:
                            lista1.remove(lista1[j])
                            j=j-1
                        j=j+1

                    lista1.sort()
                    lista1.insert(0,"DEPARTAMENT UPDATA")
                    lista=lista1
                    lista=','.join(lista)
                    acceso[0]="TRUE"
                    acceso[1]="TRUE"
                    acceso[2]=lista
                    return acceso
                except:
                    print ("No hay NEW_DEP,NEW_DEP_PASS, o algo
mas salió mal")
                try:
                    xx=datos.index("USER_OF_DEP")
                    lista2=
BaseDatosSQLite.ReadInformationColumnOfDep("USER_NAME",datos[xx+1])
                    b=len(lista2)
                    for i in range (0,b):
                        a=list(lista2[i])
                        lista2[i]=a[0]
                    print (lista2)
```

```
                lista2.sort()
                lista2.insert(0,"DEPARTMENT_USERS")
                lista=lista2
                lista=','.join(lista)
                acceso[0]="TRUE"
                acceso[1]="TRUE"
                acceso[2]=lista
                return acceso

            except:
                print ("no existe USER_OF_DEP o algo salió mal")
            try:
                xx=datos.index("CHANGE_DEP_PASS")
                yy=datos.index("NEW_DEP_PASS1")

                BaseDatosSQLite.ActualizarDep_Pass(
datos[xx+1],datos[yy+1])
                lista=BaseDatosSQLite.ReadInformationColum(
"ID_Dep")
                b=len(lista)
                for i in range (0,b):
                    a=list(lista[i])
                    lista[i]=a[0]

            #----se elimina los nombres de departamentos repetidos
                lista1=[]
                j=0
                for i in range(0,b):
                    lista1.append(lista[i])
                    n=lista1.count(lista1[j])
                    if n>1:
                        lista1.remove(lista1[j])
                        j=j-1
                    j=j+1

                lista1.sort()
```

```
                lista1.insert(0,"DEPARTAMENT UPDATA")
                lista=lista1

                lista=','.join(lista)
                acceso[0]="TRUE"
                acceso[1]="TRUE"
                acceso[2]=lista
                return acceso
            except:
                print ("no hay CHANGE_DEP_PASS, o
NEW_DEP_PASS1, o algo mas salió mal")
            try:
                xx=datos.index("DELETE_DEP")
                BaseDatosSQLite.BorrarDepBD(datos[xx+1])
                lista=BaseDatosSQLite.ReadInformationColum(
"ID_Dep")
                b=len(lista)
                for i in range (0,b):
                    a=list(lista[i])
                    lista[i]=a[0]

                #----se elimina los nombres de departamentos repetidos
                lista1=[]
                j=0
                for i in range(0,b):
                    lista1.append(lista[i])
                    n=lista1.count(lista1[j])
                    if n>1:
                        lista1.remove(lista1[j])
                        j=j-1
                    j=j+1
                lista1.sort()
                lista1.insert(0,"DEPARTAMENT UPDATA")
                lista=lista1

                lista=','.join(lista)
```

```
                    acceso[0]="TRUE"
                    acceso[1]="TRUE"
                    acceso[2]=lista
                    return acceso
                except:
                    print( "No Existe DELETE_DEP, o algo salió mal")


                try:
                    xx=datos.index("USER_TO_DELETE")
                    try:
                        BaseDatosSQLite.BorrarUser(datos[xx+1])
                    except:
                        print ("aqui esta la falla")
                    lista2=
BaseDatosSQLite.ReadInformationColumnOfDep("USER_NAME",datos[xx+1])
                    print (lista2)
                    b=len(lista2)
                    for i in range (0,b):
                        a=list(lista2[i])
                        lista2[i]=a[0]
                    print (lista2)

                    lista2.sort()
                    lista2.insert(0,"DEPARTMENT_USERS")
                    lista=lista2
                    lista=','.join(lista)
                    acceso[0]="TRUE"
                    acceso[1]="TRUE"
                    acceso[2]=lista
                    return acceso

                except:
                    print ("No existe USER_TO_DELETE, o almo salió
mal")
            else:
```

```
                    acceso[0]="TRUE"
                    acceso[1]="TRUE"
                    acceso[2]="USER"
                #try:
                    AdminUser1=["ADMIN",datos[w+1],str(1)]
                    AdmUser=''.join(AdminUser1)
                #yy=datos.index(AdmUser)
                    if AdmUser in miLista:
                        try:
                            acceso[0]="TRUE"
                            acceso[1]="TRUE"
                            acceso[2]="ADMIN_USER"
                            #return acceso
                        except:
                            print (" ")
                        try:
                            xx=datos.index("USER_OF_DEP")

lista2=BaseDatosSQLite.ReadInformationColumnOfDep("USER_NAME",datos[w+1])
                            b=len(lista2)
                            for i in range (0,b):
                                a=list(lista2[i])
                                lista2[i]=a[0]
                            print (lista2)
                            lista2.sort()
                            lista2.insert(0,"DEPARTMENT_USERS")
                            lista=lista2
                            lista=','.join(lista)
                            acceso[0]="TRUE"
                            acceso[1]="TRUE"
                            acceso[2]=lista
                            return acceso
                        except:
                            print ("no existe USER_OF_DEP o algo salió mal"
)
```

```
                    try:
                        xx=datos.index("USER_TO_DELETE")
                        try:
                            BaseDatosSQLite.BorrarUser(datos[xx+1])
                        except:
                            print ("aqui esta la falla")


                        lista2=
BaseDatosSQLite.ReadInformationColumnOfDep("USER_NAME",datos[w+1])
                        print (lista2)
                        b=len(lista2)
                        for i in range (0,b):
                            a=list(lista2[i])
                            lista2[i]=a[0]
                        print (lista2)

                        lista2.sort()
                        lista2.insert(0,"DEPARTMENT_USERS")
                        lista=lista2
                        lista=','.join(lista)
                        acceso[0]="TRUE"
                        acceso[1]="TRUE"
                        acceso[2]=lista
                        return acceso

                    except:
                        print ("No existe USER_TO_DELETE, o almo salió
mal")
                    try:
                        xx=datos.index("CHANGE_DEP_PASS")
                        yy=datos.index("NEW_DEP_PASS1")

BaseDatosSQLite.ActualizarDep_Pass(datos[w+1],datos[yy+1])
```

```
lista=BaseDatosSQLite.ReadInformationColumnOfDep("USER_NAME",datos[w+1])
                                        b=len(lista)

                                        for i in range (0,b):
                                            a=list(lista[i])
                                            lista[i]=a[0]

                                    #----se elimina los nombres de departamentos repetidos
                                        lista1=[]
                                        j=0
                                        for i in range(0,b):
                                            lista1.append(lista[i])
                                            n=lista1.count(lista1[j])
                                            if n>1:
                                                lista1.remove(lista1[j])
                                                j=j-1
                                            j=j+1

                                        lista1.sort()
                                        lista1.insert(0,"DEPARTMENT_USERS")
                                        lista=lista1

                                        lista=','.join(lista)
                                        acceso[0]="TRUE"
                                        acceso[1]="TRUE"
                                        acceso[2]=lista
                                        return acceso
                                    except:
                                        print ("no hay CHANGE_DEP_PASS, o NEW_DEP_PASS1, o algo mas salió mal")

                            #       except:
                            #           print " "
                            elif(datos[v+1]=="CONFIG1"):
```

```
                try:
                    u=datos.index("NEW_USER_NAME")
                    s=datos.index("NEW_CLAVE_USER")
                    t=datos.index("TOCKEN")

BaseDatosSQLite.ActualizarUser_Pass(datos[u+1],datos[y+1],datos[s+1],datos[t+1],IPad
d)
                    acceso[0]="TRUE"
                    acceso[1]="TRUE"
                    acceso[2]="USUARIO_REGISTRADO"
                    print (time.asctime(tiempo_st)+"-Se logro guardar
adecuadamente los nuevos datos del usuario "+datos[u+1])
                    return acceso

                except:
                    print (time.asctime(tiempo_st)+"-No se logro guardar
adecuadamente los nuevos datos del usuario "+datos[u+1])
                    acceso[0]="FALSE"
                    acceso[1]="FALSE"
                    acceso[2]="FALSE"
            elif(datos[v+1]=="ASKDEPIDDATA"):
                try:
                    lista=BaseDatosSQLite.ReadInformationColum("ID_Dep")
                    b=len(lista)
                    for i in range (0,b):
                        a=list(lista[i])
                        lista[i]=a[0]

                    #----se elimina los nombres de departamentos repetidos
                    lista1=[]
                    j=0
                    for i in range(0,b):
                        lista1.append(lista[i])
                        n=lista1.count(lista1[j])
                        if n>1:
                            lista1.remove(lista1[j])
```

```
                    j=j-1
                j=j+1

            lista1.sort()
            lista1.insert(0,"LIST")
            lista=lista1

            lista=','.join(lista)
            acceso[0]="TRUE"
            acceso[1]="TRUE"
            acceso[2]=lista
            return acceso
        except:
            print ("No logra extraer los ID_DEP")
            acceso[0]="FALSE"
            acceso[1]="FALSE"
            acceso[2]="FALSE"
    elif (datos[v+1]=="ASKDEPPASSWORD"):
        try:
            L=datos.index("PASS_OF_DEP")
            ID_Dep1=datos[L+1]
            BDatos=
BaseDatosSQLite.ReadInformationOfDep(ID_Dep1)
            n=len(BDatos)
            for i in range(0,n):
                if ID_Dep1 in BDatos[i]:
                    miLista=BDatos[i]
                    x=miLista[1]
                else:
                    x="FALSE"
                    print ("No extrajo los datos del departamento
correcto1")
            passw=["PASSWORDIS",x]
            passw=','.join(passw)
            acceso[0]="TRUE"
            acceso[1]="TRUE"
```

```
                    acceso[2]=passw
                    return acceso
                except:
                    print (time.asctime(tiempo_st)+"-No extrajo los datos del departamento correcto2")

            elif(datos[v+1]=="RECDEPACCESS"):
                acceso[0]="TRUE"
                acceso[1]="TRUE"
                acceso[2]="DEP_AUTORIZADO"
                return acceso
            elif (datos[v+1]=="ADDUSER"):
                miLista3=BaseDatosSQLite.ReadInformationOfDep(datos[w+1])
                n=len(miLista3)
                j=0
                for k in range(0,n+1):
                    i=str(k)
                    Addm=[datos[y+2],datos[w+1],i]
                    addm=''.join(Addm)
                    for l in range(0,n):
                        if (addm in miLista3[l]):
                            j=j+1
                            break
                        else:
                            User_Add=addm

                try:
                    t=datos.index("TOCKEN")
                    BaseDatosSQLite.AddInformation(
datos[w+1],datos[x+1],User_Add,datos[y+1],datos[z+1],datos[t+1],IPadd)
                    acceso[0]="TRUE"
                    acceso[1]="TRUE"
                    acceso[2]="USUARIO_REGISTRADO"
                    return acceso
                except:
                    acceso[0]="FALSE"
```

```
                    acceso[1]="FALSE"
                    acceso[2]="USUARIO_NO_REGISTRADO"
            else:
                acceso[0]="FALSE"
                acceso[1]="FALSE"
                acceso[2]="FALSE"
        else:
            acceso[0]="FALSE"
            acceso[1]="FALSE"
            acceso[2]="FALSE"
except:
    subprocess.Popen(["killall","python3.5"])
    acceso[0]="TRUE"
    acceso[1]="TRUE"
    acceso[2]="HTTP_STOP"
    time.sleep(0.1)
    print "Condicionales mal"
    return acceso
return acceso
```

ANEXO H

Código del archivo “Voip.py” que gestiona la comunicación de voz sobre IP.

```
#Funcional en Python2.7
import sys
from time import sleep
from most.voip.api import VoipLib
global event
global num
global my_voip
num=True
def notify_events(voip_event_type, voip_event, params):
    global event
    event=voip_event
    print ("Received Event Type: %s -> Event: %s\n Params: %s"
        % (voip_event_type, voip_event, params))
# inicializa lib
def voipRegister():
    global my_voip
    global event
    global num
    my_voip = VoipLib()
# crea un directorio con los parametros necesaruis para inicializar la Lib
    voip_params = {  u'username': u'Rasp',  r
                u'sip_server_address':u'192.168.1.150',
                u'sip_server_user': u'Rasp',
                u'sip_server_pwd': u'password',
                u'sip_server_transport' :u'udp',
                u'log_level' : 0,
                u'debug' : False
                }
# initialize the lib passing the dictionary and the callback method defined above:
    try:
        if (num):
            x=my_voip.init_lib(voip_params, notify_events)
            print("Inicializacion: "+str(x))
```

```
            #Step2############################
                y=my_voip.register_account()
                print("Registro: "+str(y))
                num=False
        except:
            print("Error en inicialización de llamada")
def voipCall():
        global my_voip
        global num
        my_extension = "101"
        if (num):
            voipRegister()
        #Step3############################
        if (event!="VOIP_EVENT__LIB_INITIALIZATION_FAILED"):
            print("Calling: "+my_extension)
            z=my_voip.make_call(my_extension)
            print("Call: "+str(z))
            n=0
            while(True):
                sleep(1)
                n+=1
                if (event=="VOIP_EVENT__CALL_REMOTE_HANGUP" or n>120):
                    # ends the current call
                    my_voip.hangup_call()
                    sleep(5)
                    break
            print("end_call")
        else:
            print(event)
voipCall()
```

ANEXO I

Funciones de archivo “BaseDatosSQLite.py”

```
import sqlite3
import time
#-------------------Crea la Base de datos-------------------------------------------------------------------------
def NewDataBase():
    try:
        conexion=sqlite3.connect ("/home/pi/VideoPorteroPython/BD/DepartmentsDB")
        cursor=conexion.cursor()
        cursor.execute('''
            CREATE TABLE DepartmentsInf
            (ID_DEP VARCHAR(5) NOT NULL,
            CLAVE_DEP VARCHAR(10) NOT NULL,
            USER_ADMIN VARCHAR(20) UNIQUE,
            USER_NAME VARCHAR(20) UNIQUE,
            CLAVE_USER VARCHAR(5),
            TOCKEN VARCHAR,
            IP_ADD VARCHAR)
            ''')
        conexion.close()
        tiempo_st = time.localtime()
        print(time.asctime(tiempo_st)+"- Base de Datos Inicial Creada con Exito")
    except:
        conexion.close()
        tiempo_st = time.localtime()
        print(time.asctime(tiempo_st)+"- La Base de Datos ya fue Creada Previamente")


#--------------------Crea un nuevo Usuario-------------------------------------------------------------------------
def
AddInformation(ID_Dep1,Clave_Dep2,User_Admin,Usuario3,Clave_User4,TOCKEN5,IP_
ADD6):
    try:
        conexion=sqlite3.connect ("/home/pi/VideoPorteroPython/BD/DepartmentsDB")
```

```
            cursor=conexion.cursor()
            Datos=ID_Dep1, Clave_Dep2,User_Admin, Usuario3, Clave_User4, TOCKEN5,
IP_ADD6
            cursor.execute('''INSERT INTO DepartmentsInf VALUES(?,?,?,?,?,?,?)''',
                  (Datos))
            conexion.commit()
            conexion.close()
            return "USUARIO_REGISTRADO"
except:
            conexion.close()
            tiempo_st = time.localtime()
            print(time.asctime(tiempo_st)+"-AddInformation: Se repitió el usuario:
"+ID_Dep1+", "+Usuario3)
            return "USUARIO_NO_REGISTRADO"

#--------------------Lee toda la Informacion--------------------------------------------------------------
--------------------------------
def ReadInformation():
      conexion=sqlite3.connect ("/home/pi/VideoPorteroPython/BD/DepartmentsDB")
      cursor=conexion.cursor()
      cursor.execute("SELECT * FROM DepartmentsInf")
      miLista=cursor.fetchall()
#       conexion.commit()
      conexion.close()
      return miLista

#--------------------Lee la Información del departamento deceado-------------------------------------
-------------------------------------------------------------
def ReadInformationOfDep(ID_Dep1):
      conexion=sqlite3.connect ("/home/pi/VideoPorteroPython/BD/DepartmentsDB")
      cursor=conexion.cursor()
      cursor.execute("SELECT * FROM DepartmentsInf WHERE ID_Dep='"+ID_Dep1+"'")
      miLista=cursor.fetchall()
      conexion.close()
      return miLista
```

```
#-------------------Lee la Información del departamento deceado-----------------------------------
-------------------------------------------------------------
def ReadInformationColum(Column):
    conexion=sqlite3.connect ("/home/pi/VideoPorteroPython/BD/DepartmentsDB")
    cursor=conexion.cursor()
    cursor.execute("SELECT "+Column+" FROM DepartmentsInf ")
    miLista=cursor.fetchall()
    conexion.close()
    return miLista

#-------------------Lee la Información del departamento deceado-----------------------------------
-------------------------------------------------------------
def ReadInformationColumnOfDep(Column,ID_Dep1):
    conexion=sqlite3.connect ("/home/pi/VideoPorteroPython/BD/DepartmentsDB")
    cursor=conexion.cursor()
    cursor.execute("SELECT "+Column+" FROM DepartmentsInf WHERE
ID_Dep='"+ID_Dep1+"'")
    miLista=cursor.fetchall()
    conexion.close()
    return miLista


#----------Cambia Nombre de Usuario, clave, Mac_add e IP_add de Usuario dependiendo
del nombre de usuario anterior-------------------------
def ActualizarUser_Pass(Usuario3_1,Usuario3,Clave_User4,TOCKEN5,IP_ADD6):
    try:
        conexion=sqlite3.connect("/home/pi/VideoPorteroPython/BD/DepartmentsDB")
        cursor=conexion.cursor()
        Datos=Usuario3_1, Clave_User4,TOCKEN5,IP_ADD6
        cursor.execute("UPDATE DepartmentsInf SET USER_NAME=?,
CLAVE_USER=?,TOCKEN=?, IP_ADD=? WHERE USER_NAME='"+
            Usuario3+"'",(Datos))
        conexion.commit()
        conexion.close()
    except:
```

```
            print(time.asctime(tiempo_st)+"- actualizarUser_Pass: Ya nada al Actualizar Usuario y password")


def ActualizarDep_Pass(DEP_ID1,DEP_PASSWORD):
    try:
            conexion=sqlite3.connect("/home/pi/VideoPorteroPython/BD/DepartmentsDB")
            cursor=conexion.cursor()
            cursor.execute("UPDATE DepartmentsInf SET CLAVE_DEP='"+DEP_PASSWORD+"' WHERE ID_DEP='"+DEP_ID1+"'")
            conexion.commit()
            conexion.close()
    except:
            ActualizarDep_Pass
            tiempo_st = time.localtime()
            print(time.asctime(tiempo_st)+"-ActualizarDep_Pass Nada bien al Actualizar Usuario "+Usuario3+" a "+Usuario3_1+" y papassword")
#------Borra departamento o al usuario dependiendo del Dep_Id o nombre de susuario----------------------------------------
def BorrarDepBD(dep_id):
    try:
            conexion=sqlite3.connect ("/home/pi/VideoPorteroPython/BD/DepartmentsDB", timeout=10)
            cursor=conexion.cursor()
            cursor.execute("DELETE FROM DepartmentsInf WHERE ID_DEP='" + dep_id +"'")
            conexion.commit()
            conexion.close()
    except:
            conexion.close()
            tiempo_st = time.localtime()
            print(time.asctime(tiempo_st)+"-BorrarDepBD: No logro borrar "+dep_id)

def BorrarUser(User):
    try:
```

```
            conexion=sqlite3.connect   ("/home/pi/VideoPorteroPython/BD/DepartmentsDB",
timeout=10)
            cursor=conexion.cursor()
            cursor.execute("DELETE  FROM  DepartmentsInf  WHERE  USER_NAME='" +
User +"'")
            conexion.commit()
            conexion.close()
        except:
            conexion.close()
            tiempo_st = time.localtime()
            print(time.asctime(tiempo_st)+"-BorrarUser: No logro borrar "+User)
```

ANEXO J

Envío de imágenes mediante protocolo HTTP.

 # “Video_Http_server.py”

```
from flask import Flask, render_template, Response
from camera_pi import Camera
import subprocess
app = Flask(__name__)
@app.route('/')
def index():
    """Video streaming home page."""
    return render_template('index.html')
# Stream routing
@app.route('/video_feed')
def video_feed():

subprocess.Popen(["sudo","python2.7","/home/pi/VideoPorteroPython/HttpVideo/KillHTTP
Video.py"])
    """Video streaming route. Put this in the src attribute of an img tag."""
    return Response(genVideo(Camera()),
                    mimetype='multipart/x-mixed-replace; boundary=frame')
def genVideo(camera):
    """Video streaming generator function."""
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
if(__name__ == '__main__'):
        app.run(host='192.168.1.150', threaded=True)
```

 # “base_camara.py”

```
import time
import threading
try:
    from greenlet import getcurrent as get_ident
except ImportError:
    try:
        from thread import get_ident
```

```
    except ImportError:
        from _thread import get_ident


class CameraEvent(object):
    """An Event-like class that signals all active clients when a new frame is
    available.
    """
    def __init__(self):
        self.events = {}

    def wait(self):
        """Invoked from each client's thread to wait for the next frame."""
        ident = get_ident()
        if ident not in self.events:
            # this is a new client
            # add an entry for it in the self.events dict
            # each entry has two elements, a threading.Event() and a timestamp
            self.events[ident] = [threading.Event(), time.time()]
        return self.events[ident][0].wait()
    def set(self):
        """Invoked by the camera thread when a new frame is available."""
        now = time.time()
        remove = None
        for ident, event in self.events.items():
            if not event[0].isSet():
                # if this client's event is not set, then set it
                # also update the last set timestamp to now
                event[0].set()
                event[1] = now
            else:
                # if the client's event is already set, it means the client
                # did not process a previous frame
                # if the event stays set for more than 5 seconds, then assume
                # the client is gone and remove it
                if now - event[1] > 5:
```

```
                remove = ident
        if remove:
            del self.events[remove]
    def clear(self):
        """Invoked from each client's thread after a frame was processed."""
        self.events[get_ident()][0].clear()
class BaseCamera(object):
    thread = None  # background thread that reads frames from camera
    frame = None  # current frame is stored here by background thread
    last_access = 0  # time of last client access to the camera
    event = CameraEvent()
    def __init__(self):
        """Start the background camera thread if it isn't running yet."""
        if BaseCamera.thread is None:
            BaseCamera.last_access = time.time()

            # start background frame thread
            BaseCamera.thread = threading.Thread(target=self._thread)
            BaseCamera.thread.start()
            # wait until frames are available
            while self.get_frame() is None:
                time.sleep(0)
    def get_frame(self):
        """Return the current camera frame."""
        BaseCamera.last_access = time.time()

        # wait for a signal from the camera thread
        BaseCamera.event.wait()
        BaseCamera.event.clear()
        return BaseCamera.frame
    @staticmethod
    def frames():
        """"Generator that returns frames from the camera."""
        raise RuntimeError('Must be implemented by subclasses.')
    @classmethod
    def _thread(cls):
```

```
        """Camera background thread."""
        print('Starting camera thread.')
        frames_iterator = cls.frames()
        for frame in frames_iterator:
            BaseCamera.frame = frame
            BaseCamera.event.set()  # send signal to clients
            time.sleep(0)
            # if there hasn't been any clients asking for frames in
            # the last 10 seconds then stop the thread
            if time.time() - BaseCamera.last_access > 10:
                frames_iterator.close()
                print('Stopping camera thread due to inactivity.')
                break
        BaseCamera.thread = None
```

 # “camara_pi.py”

```
import io
import time
import picamera
from base_camera import BaseCamera


class Camera(BaseCamera):
    @staticmethod
    def frames():
        with picamera.PiCamera() as camera:
            # let camera warm up
            time.sleep(2)
#           camera.resolution = (1080, 720)
            stream = io.BytesIO()
            for _ in camera.capture_continuous(stream, 'jpeg',
                                               use_video_port=True,resize=(620,360)):
                # return current frame
                stream.seek(0)
                yield stream.read()
```

```
            # reset stream for next frame
            stream.seek(0)
            stream.truncate()
```

 # “index.html”

```
<!DOCTYPE html>
<html>
  <body>
    <img src="{{ url_for('video_feed') }}" >
  </body>
</html>
```

ANEXO K

Cancela el streaming de video Http.

```
import subprocess
from time import sleep
print ("Video will stop in 2min")
sleep(120)
subprocess.Popen(["killall","python3.5"])
print("Video Killed")
```

ANEXO L

Envío de notificaciones.

```
from pyfcm import FCMNotification
import time
push_service = FCMNotification(api_key="AAAAA7aa8sE:APA91bGT7mYs1wxzUX0Qc7rBFQpcEQtbckKp2Qb83ic29de-McTefyyq2iO_T3olnyMTOaFPKogmE-B1nYzzromX9ORwvZdDvV3GL8F5UCQXZjtqgAqLjHXobHXc9y9PkxVHnufzZABh")

def SendNotifucation(registration_id):
    tiempo_st = time.localtime()
    message_title = "Llamada de la puerta"
    message_body = ("Desea contestar a la puerta?"+time.asctime(tiempo_st))
    result = push_service.notify_single_device(registration_id=registration_id, message_title=message_title, message_body=message_body)
    print (result)

def SendNotifucations(registration_ids):
    tiempo_st = time.localtime()
    message_title = "Llamada de la puerta"
    message_body = ("Desea contestar a la puerta?"+time.asctime(tiempo_st))
    result = push_service.notify_multiple_devices(registration_ids=registration_ids, message_title=message_title, message_body=message_body)
    print (result)
```

ANEXO M

Programación Java de actividad “MainActivity.java”.

```
package com.example.android.applicationvideoportero;
import android.Manifest;
import android.annotation.SuppressLint;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.os.AsyncTask;
import android.support.design.widget.FloatingActionButton;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.ImageButton;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;

public class MainActivity extends AppCompatActivity {
    private Boolean SavePassword;
    String TAG="Pruebas";
    public static final String MainData="MyPrefsFile";
    private String IpAddressRBpi,Port1RBpi, DepartID, DepartPassword,
UserName, UserPassword;
    private static int MY_PERMISSIONS_REQUEST_READ_CONTACTS1 = 666;
    //Inicio del programa de la MainActivity
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        recuestPermitions();
        FloatingActionButton buttonA1 = findViewById(R.id.ButtonA1);
        ImageButton buttonA2 = findViewById(R.id.ButtonA2);
//Crea y guarda variables a usar en la aplicacion
        SharedPreferences
IPAddressRBpi=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
PorT1RBpi=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
departID=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
departPassword=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
userName=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
userPassword=getSharedPreferences(MainData,MODE_PRIVATE);
        final SharedPreferences
savePasswordSt=getSharedPreferences(MainData,MODE_PRIVATE);
```

```
        SavePassword=savePasswordSt.getBoolean("SavePassword",false);


IpAddressRBpi=IPAddressRBpi.getString("IpAddressRBpi",IpAddressRBpi);
        Port1RBpi=PorT1RBpi.getString("Port1RBpi",Port1RBpi);
        DepartID=departID.getString("DepartID",DepartID);

DepartPassword=departPassword.getString("DepartPassword",DepartPassword);
        UserName=userName.getString("UserName",UserName);
        UserPassword=userPassword.getString("UserPassword",UserPassword);

//buttonA1 permite acceder a la actividad DepartmentDataRegistration
donde ingresar datos iniciales de departamento
        buttonA1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                NextWinConfigData();
                OpenDepRegistrationLayout();
            }
        });

// Muestra ButtonA2 solo si ya existe un registro anterior guardado desde
el celular
// y si se verifican correctamente los datos de departamento
        if (DepartID!= null && DepartPassword!=null &&
IpAddressRBpi!=null && Port1RBpi!=null && UserName!=null){
//si cumple condiciones hace visible el boton
            buttonA2.setVisibility(View.VISIBLE);
            Log.i(TAG, "Si se almacenaron los datos del departamento
"+IpAddressRBpi+",  "+DepartID);
        }
//buttonA2 permite acceder a la actividad UserDataRegistration dondese
puede ingresar datos iniciales de Usuario
//Unicamente si ya tiene registrado previamente el nombre del dep. y la
clave del mismo.
        buttonA2.setImageResource(R.drawable.house);
        buttonA2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (SavePassword && UserPassword != null) {
                    String[]
DepartEip={IpAddressRBpi,Port1RBpi,"ACCION","RECUSERACCESS","ID_DEP",

DepartID,"CLAVE_DEP",DepartPassword,"USER_NAME",
                            UserName,"CLAVE_USER",UserPassword};
                    new SendMessage().execute(DepartEip);
                } else {
                    NextWinConfigData();
                    OpenUserDataVerificationLayout();
                }
            }
        });
    }

    public void NextWinConfigData(){
//Selecciona si el LayOut de UserDataVerification se comportará para
acceder
// a VideoAccess o a Configuracion.
        SharedPreferences
settings=getSharedPreferences(MainData,MODE_PRIVATE);
```

```
        SharedPreferences.Editor edit=settings.edit();
        edit.putBoolean("VideoOrConfig",true);
        edit.apply();
    }
    public void OpenDepRegistrationLayout(){
//Abre el siguiente LayOut "DepartmentDataRegistration"
// para registrar a un nuevo Departamento.
        Intent intent =new Intent(this,DepartmentDataRegistration.class);
        startActivity(intent);
    }
    public void OpenUserDataVerificationLayout(){
//Abre el siguiente LayOut "UserDataVerification"
// para confirmar a un usuario existente.
        Intent intent1 =new Intent(this,UserDataVerification.class);
        startActivity(intent1);
    }
    public void VideoAccessLayout() {
//Abre el LayOut VideoAccess si el usuario permite el almacenamiento de
la clave
        Intent intent = new Intent(this, VideoAccess.class);
        startActivity(intent);
    }
    private void recuestPermitions() {
        if (ContextCompat.checkSelfPermission(MainActivity.this,
                Manifest.permission.RECORD_AUDIO) !=
PackageManager.PERMISSION_GRANTED
                || ContextCompat.checkSelfPermission(MainActivity.this,
                Manifest.permission.USE_SIP)!=
PackageManager.PERMISSION_GRANTED ) {
            if
(ActivityCompat.shouldShowRequestPermissionRationale(MainActivity.this,
                    Manifest.permission.RECORD_AUDIO) ||

ActivityCompat.shouldShowRequestPermissionRationale(MainActivity.this,
                    Manifest.permission.USE_SIP)) {
                    AlertDialog.Builder builder = new
AlertDialog.Builder(MainActivity.this);
                    builder.setMessage("You will need to grant this two
permitions" +
                            " in orther to comunicate whith the door.");
                    builder.setTitle("Permission");
                    builder.setPositiveButton("Yes", new
DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int
which) {

ActivityCompat.requestPermissions(MainActivity.this,
                                    new
String[]{Manifest.permission.RECORD_AUDIO, Manifest.permission.USE_SIP},

MY_PERMISSIONS_REQUEST_READ_CONTACTS1);
                                }
                        });
                    builder.setNegativeButton("No", new
DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int
which) {
                            dialog.cancel();
```

```
                    }
                });
                AlertDialog dialog = builder.create();
                dialog.show();
            } else {
                ActivityCompat.requestPermissions(MainActivity.this,
                        new String[]{Manifest.permission.RECORD_AUDIO,
Manifest.permission.USE_SIP},
                        MY_PERMISSIONS_REQUEST_READ_CONTACTS1);
            }
        }
    }

    @SuppressLint("StaticFieldLeak")
    public class SendMessage extends AsyncTask<String[] , Void, String> {
//Envía mensaje de control con datos para permitir el acceso al video
portero
        @Override
        protected String doInBackground(String[]... params) {
            String[] parama=params[0];
            String ipaddress=parama[0];
            int PORT=Integer.parseInt(parama[1]);
            StringBuffer paramString=new StringBuffer();
            paramString=paramString.append(parama[2]);
            for(int i=3;i<parama.length;i++){
                paramString=paramString.append(" ");
                paramString=paramString.append(parama[i]);
            }
            String param=paramString.toString();
            try {
                Socket socket = new Socket(ipaddress, PORT);
                socket.setSoTimeout(2000);
                Log.i(TAG,"Socker Creado: "+ipaddress+":"+PORT);
                BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                PrintWriter out = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()));
                Log.i(TAG,"Buffers Creados");
                try {
                    out.print(param);
                    out.flush();
                    Log.i(TAG,"Se espera respuesta ");
                    String respuesta=in.readLine();
                    in.close();
                    socket.close();
                    Log.i(TAG,"Se recibio: "+respuesta);
                    return respuesta;
                } catch (IOException e) {
                    e.printStackTrace();
                    out.flush();
                    in.close();
                    socket.close();
                    Log.i(TAG,"Error al enviar o recibir");
                    return "FALSE";
                }
            } catch (Exception e) {
                Log.i(TAG,"Error al crear el Socket o Buffers");
                return "FALSE";
            }
        }
```

```
        @Override
        protected void onPostExecute(String MSJ) {
            if (MSJ.equals("USER_AUTORIZADO")) {
                VideoAccessLayout();
            }
            else if (MSJ.equals("HTTP_RUNING")){
                Toast.makeText(MainActivity.this,
                        "Access door Online ",
                        Toast.LENGTH_LONG).show();
            }else if(MSJ.equals("HTTP_WAS_PAUSE")) {
                Toast.makeText(MainActivity.this,
                        "Access door Online ",
                        Toast.LENGTH_LONG).show();
            }else if(MSJ.equals("Incorrecto")){
                OpenUserDataVerificationLayout();
            }
        }
    }
}
```

ANEXO N

Programación XML de actividad “activity_main.xml”.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <de.hdodenhof.circleimageview.CircleImageView
        android:layout_width="150dp"
        android:layout_height="131dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="50dp"
        android:src="@drawable/logo" />

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/ButtonA1"
        android:layout_width="68dp"
        android:layout_height="71dp"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="30dp"
        android:clickable="true"
        app:backgroundTint="@android:color/holo_green_dark"
        app:fabSize="normal"
        app:srcCompat="@android:drawable/ic_input_add" />

    <ImageButton
        android:id="@+id/ButtonA2"
        android:layout_width="130dp"
        android:layout_height="130dp"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="150dp"
        android:scaleType="fitXY"
        android:src="@drawable/house"
        android:visibility="gone" />

</RelativeLayout>
```

ANEXO O

Programación Java de actividad “DepartmentDataRegistration.java”.

```
package com.example.android.applicationvideoportero;

import android.annotation.SuppressLint;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.TextView;


import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;

import static
com.example.android.applicationvideoportero.MainActivity.MainData;

public class DepartmentDataRegistration extends AppCompatActivity {
    EditText IPAddressRBPiEdit,Port1RBPiEdit, DepartmentIdEdit,
DepartPasswordEdit;
    TextView Autentic;
    LinearLayout DataLayout;
    private String DepID, DepPassword;
    private String DepIpAddress,Port1RBpi;
    private String prue="Pruebas";
    private Boolean NewOldUser;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_department_data_registration);
        DataLayout=findViewById(R.id.DataLayout);
        Button buttonB1 = findViewById(R.id.NewUserButtonB1);
        Button buttonB2 = findViewById(R.id.OldUserButtonB2);
        Autentic =findViewById(R.id.Autenticacion);

        IPAddressRBPiEdit = findViewById(R.id.IPAddressRBPiEditText);
        Port1RBPiEdit = findViewById(R.id.PortRBPiEditText);
        DepartmentIdEdit = findViewById(R.id.DepartmentIdEditText);
        DepartPasswordEdit = findViewById(R.id.DepartPasswordEditText);

        SharedPreferences
IPAddressRBpi=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
PorT1RBpi=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
departID=getSharedPreferences(MainData,MODE_PRIVATE);
```

```
        DepIpAddress=IPAddressRBpi.getString("IpAddressRBpi","");
        Port1RBpi=PorT1RBpi.getString("Port1RBpi","");
        DepID=departID.getString("DepartID","");

//Rellena datos de de departamento automaticamente
        IPAddressRBPiEdit.setText(DepIpAddress);
        Port1RBPiEdit.setText(Port1RBpi);
        DepartmentIdEdit.setText(DepID);

        buttonB1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Log.i(prue, "Se preciono el boton New User");
                NewOldUser=true;
//Lee las 3 variables y los coloca en Strings
                DepIpAddress=IPAddressRBPiEdit.getText().toString();
                Port1RBpi=Port1RBPiEdit.getText().toString();
                DepID=DepartmentIdEdit.getText().toString();
                DepPassword=DepartPasswordEdit.getText().toString();
// Envía datos de Dep y DepPassword para confirmar que tenga los permisos
necesarios
                SafeDepartmentData();
                String[]
DepartEip={DepIpAddress,Port1RBpi,"ACCION","RECDEPACCESS","ID_DEP",DepID,

"CLAVE_DEP",DepPassword,"USER_NAME","CLAVE_USER"};
                new SendMessage().execute(DepartEip);

            }
        });
        buttonB2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Log.i("Pruebas", "Se preciono el boton User");
                NewOldUser=false;
                NextWinConfigData();
                //Lee las 4 variables y los coloca en Strings
                DepIpAddress=IPAddressRBPiEdit.getText().toString();
                Port1RBpi=Port1RBPiEdit.getText().toString();
                DepID=DepartmentIdEdit.getText().toString();
                DepPassword=DepartPasswordEdit.getText().toString();
// Envía datos de Dep y DepPassword para confirmar que tenga los permisos
necesarios
                SafeDepartmentData();
                String[]
DepartEip={DepIpAddress,Port1RBpi,"ACCION","RECDEPACCESS","ID_DEP",

DepID,"CLAVE_DEP",DepPassword,"USER_NAME","CLAVE_USER"};
                new SendMessage().execute(DepartEip);
            }
        });
    }
    public void Open_New_User(){
        Intent intent =new Intent(this,UserDataRegistration.class);
        startActivity(intent);
    }
    public void Open_Old_User(){
        Intent intent =new Intent(this,UserDataVerification.class);
        startActivity(intent);
    }
```

```
    public  void NextWinConfigData(){
        SharedPreferences
settings=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences.Editor edit=settings.edit();
        edit.putBoolean("VideoOrConfig",true);
        edit.apply();
        Log.i(prue, "Se almacenará la condicion de Switch "+"true");
    }
    public void SafeDepartmentData(){
        SharedPreferences
IPAddressRBpi=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
PorT1RBpi=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
departID=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
departPassword=getSharedPreferences(MainData,MODE_PRIVATE);
//Guarda DireccionIP
        SharedPreferences.Editor editor=IPAddressRBpi.edit();
        editor.putString("IpAddressRBpi", DepIpAddress);
        editor.apply();
//Guardar Puerto
        SharedPreferences.Editor editor0=PorT1RBpi.edit();
        editor0.putString("Port1RBpi", Port1RBpi);
        editor0.apply();
//Guarda ID de Departamento
        SharedPreferences.Editor editor1=departID.edit();
        editor1.putString("DepartID", DepID);
        editor1.apply();
//Guarda Password de Departamento
        SharedPreferences.Editor editor2=departPassword.edit();
        editor2.putString("DepartPassword", DepPassword);
        editor2.apply();
        Log.i("Pruebas", "Ingreso a guardar los datos de departamento: "
+DepIpAddress);
    }

    @SuppressLint("StaticFieldLeak")
    private class SendMessage extends AsyncTask<String[] , Void, String>
{
        @Override
        protected String doInBackground(String[]... params) {
            String[] parama=params[0];
            String ipaddress=parama[0];
            int PORT=Integer.parseInt(parama[1]);
            StringBuffer paramString=new StringBuffer();
            paramString=paramString.append(parama[2]);
            for(int i=3;i<parama.length;i++){
                paramString=paramString.append(" ");
                paramString=paramString.append(parama[i]);
            }
            String param=paramString.toString();
            try {
                Socket socket = new Socket(ipaddress, PORT);
                BufferedReader in =new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                Log.i(prue,"Socker Creado: "+ipaddress+":"+PORT);
                PrintWriter out = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()));
                Log.i(prue,"Buffers Creados");
```

```
                try {
                    out.print(param);
                    out.flush();
                    Log.i(prue,"Se espera respuesta ");
                    String respuesta=in.readLine();
                    in.close();
                    socket.close();
                    Log.i(prue,"Se recibio: "+respuesta);
                    return respuesta;
                } catch (IOException e) {
                    e.printStackTrace();
                    out.flush();
                    in.close();
                    socket.close();
                    Log.i(prue,"Error al enviar o recibir");
                    return "FALSE2";
                }
            } catch (Exception e) {
                Log.i(prue,"Error al crear el Socket o Buffers");
                return "FALSE1";
            }
        }
        @SuppressLint("SetTextI18n")
        @Override
        protected void onPostExecute(String MSJ) {
            Log.i(prue,"Return "+MSJ);
            if (MSJ.equals("DEP_AUTORIZADO")) {
                if (NewOldUser) {
                    Open_New_User();
                }else {
                    Open_Old_User();
                }
            } else {
                Autentic.setText(R.string.Error1);
                Autentic.setVisibility(View.VISIBLE);
            }
        }
    }
}
```

ANEXO P

Programación XML de actividad “activity_department_data_registration.xml”.

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".DepartmentDataRegistration">
    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        tools:ignore="UselessParent">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">
            <RelativeLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content">
                <de.hdodenhof.circleimageview.CircleImageView
                    android:layout_width="137dp"
                    android:layout_height="156dp"
                    android:layout_alignParentTop="true"
                    android:layout_centerHorizontal="true"
                    android:src="@drawable/logo" />
            </RelativeLayout>
            <LinearLayout
                android:id="@+id/DataLayout"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:orientation="vertical"
                android:padding="10dp">
                <RelativeLayout
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:padding="10dp">
                    <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:padding="10dp"
                        android:text="@string/ip_address" />
                    <EditText
                        android:id="@+id/IPAddressRBPiEditText"
                        android:layout_width="150dp"
                        android:layout_height="wrap_content"
                        android:layout_alignParentEnd="true"
                        android:digits="0123456789."
                        android:inputType="number|numberDecimal"
                        android:padding="10dp"
                        />
                </RelativeLayout>
                <RelativeLayout
                    android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:padding="10dp">
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:padding="10dp"
                android:text="Port" />
            <EditText
                android:id="@+id/PortRBPiEditText"
                android:layout_width="150dp"
                android:layout_height="wrap_content"
                android:layout_alignParentEnd="true"
                android:digits="0123456789."
                android:inputType="number|numberDecimal"
                android:padding="10dp"
                />
        </RelativeLayout>
        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="10dp">
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:padding="10dp"
                android:text="@string/department_id" />
            <EditText
                android:id="@+id/DepartmentIdEditText"
                android:layout_width="150dp"
                android:layout_height="wrap_content"
                android:layout_alignParentEnd="true"
                android:padding="10dp"/>
        </RelativeLayout>
        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="10dp">
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentTop="true"
                android:padding="10dp"
                android:text="@string/department_password" />

            <EditText
                android:id="@+id/DepartPasswordEditText"
                android:layout_width="150dp"
                android:layout_height="wrap_content"
                android:layout_alignParentEnd="true"
                android:layout_alignParentTop="true"
                android:padding="10dp"
                android:inputType="textPassword" />
        </RelativeLayout>
    </LinearLayout>
    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <TextView
            android:id="@+id/Autenticacion"
            android:layout_width="wrap_content"
```

```
                android:layout_height="wrap_content"
                android:layout_alignParentTop="true"
                android:layout_centerHorizontal="true"
                android:text="TextView"
                android:visibility="gone" />
        </RelativeLayout>
        <RelativeLayout
            android:id="@+id/buttons"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="10dp">
            <Button
                android:id="@+id/OldUserButtonB2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentStart="true"
                android:layout_centerVertical="true"
                android:layout_marginStart="40dp"
                android:text="@string/user"
                android:visibility="visible" />
            <Button
                android:id="@+id/NewUserButtonB1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentEnd="true"
                android:layout_centerVertical="true"
                android:layout_marginEnd="40dp"
                android:text="@string/new_user"
                android:visibility="visible" />
        </RelativeLayout>
    </LinearLayout>
</ScrollView>
</RelativeLayout>
```

ANEXO Q

Programación Java de actividad “UserDataRegistration.java”.

```
package com.example.android.applicationvideoportero;

import android.annotation.SuppressLint;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;

import static
com.example.android.applicationvideoportero.MainActivity.MainData;

public class UserDataRegistration extends AppCompatActivity {
    private EditText NewUserNameEdit, NewUserPasswordEdit,
UserPassword2Edit;
    private String IpAddressRBpi;
    private String DepartID;
    private String DepartPassword;
    private String UserName;
    private String UserPassword;
    private String UserPassword2;
    private String UserName1;
    private String UserPassword1;
    private String tocken;
    private TextView Autoiritation;
    Boolean VideoOrConfig;
    String prue="Pruebas";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user_data_registration);
//Da valores a las variables
        NewUserNameEdit=findViewById(R.id.NewUserNameEditText);
        NewUserPasswordEdit=findViewById(R.id.NewUserPasswordEditText);
        UserPassword2Edit=findViewById(R.id.UserPassword2EditText);
        Autoiritation=findViewById(R.id.Autoiritation);
        Button buttonCA1 = findViewById(R.id.LoginButtonCA1);
// Abre registro de datos de dep y usuario
        SharedPreferences
iPAddressRBpi=getSharedPreferences(MainData,MODE_PRIVATE);

IpAddressRBpi=iPAddressRBpi.getString("IpAddressRBpi",IpAddressRBpi);
        SharedPreferences
departID=getSharedPreferences(MainData,MODE_PRIVATE);
```

```
        DepartID=departID.getString("DepartID",DepartID);
        SharedPreferences
departPassword=getSharedPreferences(MainData,MODE_PRIVATE);

DepartPassword=departPassword.getString("DepartPassword",DepartPassword);
        SharedPreferences
userName=getSharedPreferences(MainData,MODE_PRIVATE);
        UserName=userName.getString("UserName",UserName);
        SharedPreferences
userPassword=getSharedPreferences(MainData,MODE_PRIVATE);
        UserPassword=userPassword.getString("UserPassword",UserPassword);
        SharedPreferences
GetVariable=getSharedPreferences(MainData,MODE_PRIVATE);
        VideoOrConfig=GetVariable.getBoolean("VideoOrConfig",true);
        SharedPreferences
GetTocken=getSharedPreferences(MainData,MODE_PRIVATE);
        tocken=GetTocken.getString("tocken",tocken);

        Log.i(prue, "Recuperado de base de datos-"+IpAddressRBpi+" DepID:
"+DepartID+" Depar Passw: "+DepartPassword);
        if(UserName!=null){
            NewUserNameEdit.setText(UserName);
            Log.i("Pruebas","Si existe un registro previo de usuario
"+UserName+" "+UserPassword);
        }else{
            Log.i("Pruebas","No existe un registro previo de usuario
"+UserName+" "+UserPassword);
        }
        buttonCA1.setOnClickListener(new View.OnClickListener() {
            @SuppressLint("SetTextI18n")
            @Override
            public void onClick(View v) {
                UserName1=NewUserNameEdit.getText().toString();
                UserPassword1=NewUserPasswordEdit.getText().toString();
                UserPassword2=UserPassword2Edit.getText().toString();

                if(UserPassword1.equals(UserPassword2)){
// /Solo si las dos contraseñas son iguales
                    if (VideoOrConfig) {
                        String[] DepartEip = {IpAddressRBpi, "5005",
"ACCION", "ADDUSER", "ID_DEP",
                                DepartID, "CLAVE_DEP", DepartPassword,
"USER_NAME",UserName1, "CLAVE_USER", UserPassword1,"TOCKEN",tocken};
                        new SendMessage().execute(DepartEip);
                    }else {
                        String[] DepartEip = {IpAddressRBpi, "5005",
"ACCION", "CONFIG1", "ID_DEP",
                                DepartID, "CLAVE_DEP", DepartPassword,
"USER_NAME",UserName,
                                "NEW_USER_NAME",UserName1, "CLAVE_USER",
UserPassword, "NEW_CLAVE_USER", UserPassword1,"TOCKEN",tocken};
                        new SendMessage().execute(DepartEip);
                    }
// Despues almacena nombre de usuario en el registro del celular
                    SafeUserData();
                }else{
                    Autoiritation.setText(R.string.PasswError);
                    Autoiritation.setVisibility(View.VISIBLE);
                    Log.i("Pruebas","las contraseñas no son compatibles
"+UserPassword+", "+UserPassword2);
```

```
                }
            }
        });
    }
    public void VideoAccessLayout() {
        Intent intent = new Intent(this, VideoAccess.class);
        startActivity(intent);
    }
    public void SafeUserData(){
//Guarda DireccionIP
        SharedPreferences
userName=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences.Editor editor=userName.edit();
        editor.putString("UserName", UserName1);
        editor.apply();
//Guarda ID de Departamento
        SharedPreferences
userPassword=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences.Editor editor1=userPassword.edit();
        editor1.putString("UserPassword", UserPassword1);
        editor1.apply();
        Log.i("Pruebas","se almaceno UserName:"+UserName1+" y la
password: "+UserPassword1);
    }
    @SuppressLint("StaticFieldLeak")
    public class SendMessage extends AsyncTask<String[] , Void, String> {
        @Override
        protected String doInBackground(String[]... params) {
            String[] parama=params[0];
            String ipaddress=parama[0];
            Log.i(prue,parama[1]);
            int PORT=Integer.parseInt(parama[1]);
            StringBuffer paramString=new StringBuffer();
            paramString=paramString.append(parama[2]);
            for(int i=3;i<parama.length;i++){
                paramString=paramString.append(" ");
                paramString=paramString.append(parama[i]);
            }
            String param=paramString.toString();
            try {
                Socket socket = new Socket(ipaddress, PORT);
                BufferedReader in =new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                Log.i(prue,"Socker Creado: "+ipaddress+":"+PORT);
                PrintWriter out = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()));
                Log.i(prue,"Buffers Creados");
                try {
                    out.print(param);
                    out.flush();
                    Log.i(prue,"Se espera respuesta ");
                    String respuesta=in.readLine();
                    in.close();
                    socket.close();
                    Log.i(prue,"Se recibio: "+respuesta);
                    return respuesta;
                } catch (IOException e) {
                    e.printStackTrace();
                    out.flush();
                    in.close();
```

```
                    socket.close();
                    Log.i(prue,"Error al enviar o recibir");
                    return "FALSE";
                }
            } catch (Exception e) {
                Log.i(prue,"Error al crear el Socket o Buffers");
                return "FALSE";
            }
        }
        @SuppressLint("SetTextI18n")
        @Override
        protected void onPostExecute(String MSJ) {
             if (MSJ.equals("USUARIO_REGISTRADO")) {
                 VideoAccessLayout();
             }
                else{
                Autoiritation.setText(R.string.ErrorInformation);
                Autoiritation.setVisibility(View.VISIBLE);

            }
        }
    }
}
```

ANEXO R

Programación XML de actividad “activity_user_data_registration.xml”.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".UserDataRegistration">
     <ScrollView
         android:layout_width="match_parent"
         android:layout_height="wrap_content">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">
            <RelativeLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content">
                <de.hdodenhof.circleimageview.CircleImageView
                    android:layout_width="129dp"
                    android:layout_height="152dp"
                    android:layout_centerInParent="true"
                    android:src="@drawable/logo" />
            </RelativeLayout>
            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical"
                android:padding="10dp"
                android:visibility="visible">
                <RelativeLayout
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:padding="10dp">
                    <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:padding="10dp"
                        android:text="@string/create_user_name" />
                    <EditText
                        android:id="@+id/NewUserNameEditText"
                        android:layout_width="150dp"
                        android:layout_height="wrap_content"
                        android:layout_alignParentEnd="true"
                        android:padding="10dp" />
                </RelativeLayout>
                <RelativeLayout
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:padding="10dp">
                    <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:padding="10dp"
                        android:text="@string/create_password" />
                    <EditText
```

```
                    android:id="@+id/NewUserPasswordEditText"
                    android:layout_width="150dp"
                    android:layout_height="wrap_content"
                    android:layout_alignParentEnd="true"
                    android:padding="10dp"
                    android:inputType="textPassword" />
            </RelativeLayout>
            <RelativeLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:padding="10dp">
                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_alignParentStart="true"
                    android:padding="10dp"
                    android:text="@string/repeat_password" />
                <EditText
                    android:id="@+id/UserPassword2EditText"
                    android:layout_width="150dp"
                    android:layout_height="wrap_content"
                    android:layout_alignParentEnd="true"
                    android:padding="10dp"
                    android:inputType="textPassword" />
            </RelativeLayout>
        </LinearLayout>
        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
            <TextView
                android:id="@+id/Autoiritation"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentTop="true"
                android:layout_centerHorizontal="true"
                android:text="Text"
                android:visibility="gone" />
        </RelativeLayout>
        <RelativeLayout
            android:id="@+id/buttons"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="5dp">
            <Button
                android:id="@+id/LoginButtonCA1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerHorizontal="true"
                android:layout_margin="10dp"
                android:text="@string/next" />
        </RelativeLayout>
    </LinearLayout>
</ScrollView>
</RelativeLayout>
```

ANEXO S

Programación Java de actividad “UserDataVerification.java”.

```
package com.example.android.applicationvideoportero;

import android.annotation.SuppressLint;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.Switch;
import android.widget.TextView;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;

import static
com.example.android.applicationvideoportero.MainActivity.MainData;

public class UserDataVerification extends AppCompatActivity {
    private EditText UserNameEdit, UserPasswordEdit;
    private String IpAddressRBpi, Port1RBpi, DepartID, DepartPassword,
UserName, UserPassword;
    private Boolean SavePassword, VideoOrConfig;
    private TextView AdvertText;
    public String prue="Pruebas";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user_data_verification);

        UserNameEdit=findViewById(R.id.UserNameEditText);
        UserPasswordEdit=findViewById(R.id.UserPasswordEditText);
        Switch passwordSwitch = findViewById(R.id.PasswordSwitch);
        AdvertText=findViewById(R.id.AdvertText);
        Button buttonCB1 = findViewById(R.id.LoginButtonCB1);
        SharedPreferences
GetVariable=getSharedPreferences(MainData,MODE_PRIVATE);
        VideoOrConfig=GetVariable.getBoolean("VideoOrConfig",true);
        passwordSwitch.setVisibility(View.VISIBLE);
//******Extraer datos almacenados en telefono
        SharedPreferences
iPAddressRBpi=getSharedPreferences(MainData,MODE_PRIVATE);

IpAddressRBpi=iPAddressRBpi.getString("IpAddressRBpi",IpAddressRBpi);
        SharedPreferences
PorT1RBpi=getSharedPreferences(MainData,MODE_PRIVATE);
        Port1RBpi=PorT1RBpi.getString("Port1RBpi",Port1RBpi);
        SharedPreferences
```

```
departID=getSharedPreferences(MainData,MODE_PRIVATE);
        DepartID=departID.getString("DepartID",DepartID);
        SharedPreferences
departPassword=getSharedPreferences(MainData,MODE_PRIVATE);

DepartPassword=departPassword.getString("DepartPassword",DepartPassword);
        SharedPreferences
userName=getSharedPreferences(MainData,MODE_PRIVATE);
        UserName=userName.getString("UserName",UserName);
        SharedPreferences
userPassword=getSharedPreferences(MainData,MODE_PRIVATE);
        UserPassword=userPassword.getString("UserPassword",UserPassword);
//almacena la opcion del Shitch si se mantiene activo o no
        SharedPreferences
savePasswordSt=getSharedPreferences(MainData,MODE_PRIVATE);
        SavePassword=savePasswordSt.getBoolean("SavePassword",false);
        passwordSwitch.setChecked(SavePassword);
        if(UserName!=null){
            UserNameEdit.setText(UserName);
        }
        passwordSwitch.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {
                SavePassword=isChecked;
                Log.i("Pruebas","El estado del Switch es: "+isChecked);
            }
        });
        buttonCB1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                UserName=UserNameEdit.getText().toString();
                UserPassword=UserPasswordEdit.getText().toString();
                SafeUserData();
                PermanentPassword();
                if (VideoOrConfig) {
                    String[]
DepartEip={IpAddressRBpi,Port1RBpi,"ACCION","RECUSERACCESS","ID_DEP",

DepartID,"CLAVE_DEP",DepartPassword,"USER_NAME",
                            UserName,"CLAVE_USER",UserPassword};
                    new SendMessage().execute(DepartEip);
                }else{
                    String[]
DepartEip={IpAddressRBpi,Port1RBpi,"ACCION","CONFIG","ID_DEP",

DepartID,"CLAVE_DEP",DepartPassword,"USER_NAME",
                            UserName,"CLAVE_USER",UserPassword};
                    new SendMessage().execute(DepartEip);
                }
            }
        });
    }
    public void SafeUserData(){
//Guarda DireccionIP
        SharedPreferences
userName=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences.Editor editor=userName.edit();
        editor.putString("UserName", UserName);
```

```
        editor.apply();
//Guarda ID de Departamento
        SharedPreferences
userPassword=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences.Editor editor1=userPassword.edit();
        editor1.putString("UserPassword", UserPassword);
        editor1.apply();
        Log.i("Pruebas","se almaceno UserName:"+UserName+" y la password:
"+UserPassword);
    }
    public void VideoAccessLayout() {
        Intent intent = new Intent(this, VideoAccess.class);
        startActivity(intent);
    }
    public void PermanentPassword(){
        SharedPreferences
settings=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences.Editor edit=settings.edit();
        edit.putBoolean("SavePassword",SavePassword);
        edit.apply();
        Log.i("Pruebas","estado guardado "+SavePassword);
    }
    public void ConfigUsuario(){
        Intent intent = new Intent(this, UserDataRegistration.class);
        startActivity(intent);
    }
    public void ConfigAdmin(Boolean DepAdm_Ad){
        SharedPreferences
setting=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences.Editor edit=setting.edit();
        edit.putBoolean("DepAdm_Adm",DepAdm_Ad);
        edit.apply();
        Intent intent = new Intent(this, AdminConfiguration.class);
        startActivity(intent);
    }
    @SuppressLint("StaticFieldLeak")
    public class SendMessage extends AsyncTask<String[] , Void, String> {
        @Override
        protected String doInBackground(String[]... params) {
            String[] parama=params[0];
            String ipaddress=parama[0];
            int PORT=Integer.parseInt(parama[1]);
            StringBuffer paramString=new StringBuffer();
            paramString=paramString.append(parama[2]);
            for(int i=3;i<parama.length;i++){
                paramString=paramString.append(" ");
                paramString=paramString.append(parama[i]);
            }
            String param=paramString.toString();
            try {
                Socket socket = new Socket(ipaddress, PORT);
                BufferedReader in =new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                Log.i(prue,"Socker Creado: "+ipaddress+":"+PORT);
                PrintWriter out = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()));
                Log.i(prue,"Buffers Creados");
                try {
                    out.print(param);
                    out.flush();
```

```
                    Log.i(prue,"Se espera respuesta ");
                    String respuesta=in.readLine();
                    in.close();
                    socket.close();
                    Log.i(prue,"Se recibio: "+respuesta);
                    return respuesta;
                } catch (IOException e) {
                    e.printStackTrace();
                    out.flush();
                    in.close();
                    socket.close();
                    Log.i(prue,"Error al enviar o recibir");
                    return "FALSE";
                }
            } catch (Exception e) {
                Log.i(prue,"Error al crear el Socket o Buffers");
                return "FALSE";
            }
        }
        @SuppressLint("SetTextI18n")
        @Override
        protected void onPostExecute(String MSJ) {
            switch (MSJ) {
                case "USER_AUTORIZADO":
                    VideoAccessLayout();
                    break;
                case "FALSE":
                    AdvertText.setText("Access not granted");
                    AdvertText.setVisibility(View.VISIBLE);
                    break;
                case "TRUE1":
                    AdvertText.setText("Access not granted");
                    AdvertText.setVisibility(View.VISIBLE);
                    break;
                case "USER":
                    ConfigUsuario();
                    break;
                case "ADMIN":
                    ConfigAdmin(true);
                    break;
                case "ADMIN_USER":
                    ConfigAdmin(false);
                    break;
                default:
                    AdvertText.setText("Access not granted, check the
user name and password");
                    AdvertText.setVisibility(View.VISIBLE);
                    break;
            }
        }
    }
}
```

ANEXO T

Programación XML de actividad “activity_user_data_verification.xml”.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".UserDataVerification">
    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        tools:ignore="UselessParent">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">
            <RelativeLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content">
                <de.hdodenhof.circleimageview.CircleImageView
                    android:layout_width="150dp"
                    android:layout_height="131dp"
                    android:layout_alignParentTop="true"
                    android:layout_centerHorizontal="true"
                    android:layout_marginTop="50dp"
                    android:src="@drawable/logo" />
            </RelativeLayout>
            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginBottom="10dp"
                android:orientation="vertical"
                android:padding="10dp">
                <RelativeLayout
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:padding="10dp">
                    <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:padding="10dp"
                        android:text="@string/user_name" />
                    <EditText
                        android:id="@+id/UserNameEditText"
                        android:layout_width="175dp"
                        android:layout_height="wrap_content"
                        android:layout_alignParentEnd="true"
                        android:padding="10dp" />
                </RelativeLayout>
                <RelativeLayout
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:padding="10dp">
                    <TextView
                        android:layout_width="wrap_content"
```

```
                    android:layout_height="wrap_content"
                    android:padding="10dp"
                    android:text="@string/password" />
                <EditText
                    android:id="@+id/UserPasswordEditText"
                    android:layout_width="175dp"
                    android:layout_height="wrap_content"
                    android:layout_alignParentEnd="true"
                    android:padding="10dp"
                    android:inputType="textPassword" />
            </RelativeLayout>
            <RelativeLayout
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:padding="10dp">
                <Switch
                    android:id="@+id/PasswordSwitch"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:padding="10dp"
                    android:text="@string/save_password"
                    android:visibility="gone" />
            </RelativeLayout>
        </LinearLayout>
        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
            <TextView
                android:id="@+id/AdvertText"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentTop="true"
                android:layout_centerHorizontal="true"
                android:text="text"
                android:visibility="gone" />
        </RelativeLayout>
        <RelativeLayout
            android:id="@+id/buttons"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="5dp">
            <Button
                android:id="@+id/LoginButtonCB1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentTop="true"
                android:layout_centerHorizontal="true"
                android:text="@string/next" />
        </RelativeLayout>
    </LinearLayout>
</ScrollView>
</RelativeLayout>
```

ANEXO U

Diagrama de flujo de botones de actividad “UserDataVerification.java”.

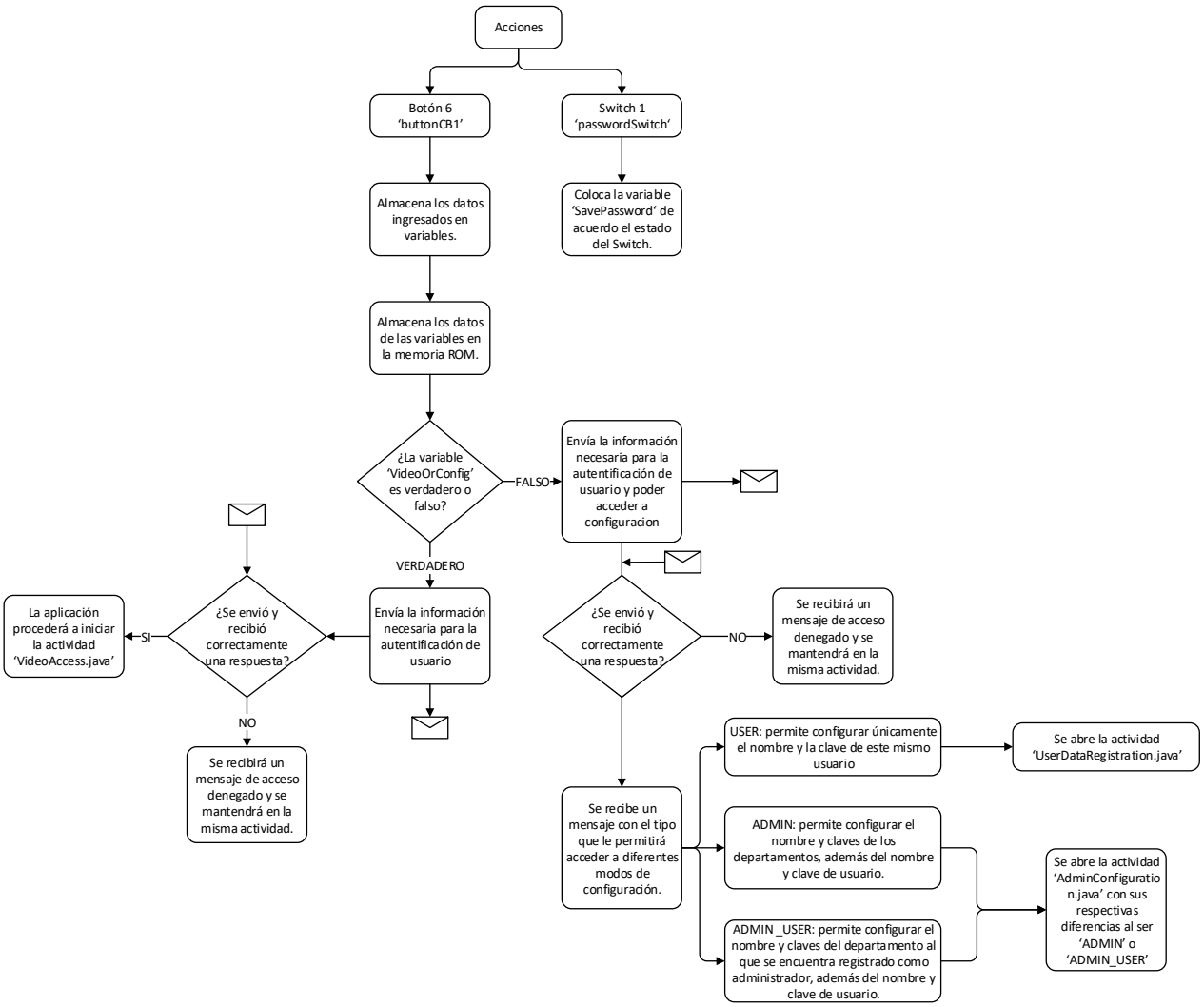

Figura U.1. Diagrama de flujo de botones de actividad “UserDataVerification.java”

ANEXO v

Programación Java de actividad “AdminConfiguration.java”.

```
package com.example.android.applicationvideoportero;

import android.annotation.SuppressLint;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.AsyncTask;
import android.support.design.widget.FloatingActionButton;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.ListView;
import android.widget.RelativeLayout;
import android.widget.TextView;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Random;


import static
com.example.android.applicationvideoportero.MainActivity.MainData;

public class AdminConfiguration extends AppCompatActivity {
    private LinearLayout LinearLayout1,LinearLayout2;
    private RelativeLayout NewDepPasswL;
    private ListView List1;
    private EditText DepDATA,NewDepID,NewDepPassw,NewDepPassw2 ;
    private TextView DepPassw;
    private Button AskPassword;
    private Button EditDepartData;
    private Button Delete_user;
    private Button EditAdmin;
    private Button randomPass;
    private Button addDepartment;
    private FloatingActionButton NewDepartment;
    private String
IpAddressRBpi,DepID,DepartPassword,UserName,UserPassword;
    private String depiD;
    private Boolean List1OrList2=true, SetNewPassword=true;
    private Boolean DepAdm_Adm;
    String prue="Pruebas";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_admin_configuration);

        LinearLayout1=findViewById(R.id.LinearLayout1);
        LinearLayout2=findViewById(R.id.LinearLayout2);
        DepDATA=findViewById(R.id.DepDATA);
        DepPassw=findViewById(R.id.DepPassw);
        Delete_user=findViewById(R.id.Delete_user);
        AskPassword=findViewById(R.id.AskPassword);
        EditDepartData=findViewById(R.id.EditDepartData);
        NewDepID=findViewById(R.id.NewDepID);
        NewDepPassw=findViewById(R.id.NewDepPassw);
        randomPass = findViewById(R.id.RandomPass);
        addDepartment = findViewById(R.id.AddDepartment);
        NewDepartment=findViewById(R.id.NewDepartment);
        NewDepPassw2=findViewById(R.id.NewDepPassw2);
        NewDepPasswL=findViewById(R.id.NewDepPassw1);
        EditAdmin=findViewById(R.id.EditAdmin);

        SharedPreferences
IPAddressRBpi=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
departID=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
departPassword=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
userName=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
userPassword=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences
DepAd_Ad=getSharedPreferences(MainData,MODE_PRIVATE);


IpAddressRBpi=IPAddressRBpi.getString("IpAddressRBpi",IpAddressRBpi);
        DepID=departID.getString("DepartID","");

DepartPassword=departPassword.getString("DepartPassword",DepartPassword);
        UserName=userName.getString("UserName",UserName);
        UserPassword=userPassword.getString("UserPassword",UserPassword);
        DepAdm_Adm=DepAd_Ad.getBoolean("DepAdm_Adm",true);
        Log.i (prue,"1+ "+DepAdm_Adm);
        if(DepAdm_Adm) {
            String[] DepartEip = {IpAddressRBpi, "5005", "ACCION",
"ASKDEPIDDATA", "ID_DEP", DepID, "CLAVE_DEP",
                    DepartPassword, "USER_NAME", UserName, "CLAVE_USER",
UserPassword};
            new SendMessage().execute(DepartEip);
        }else{

            String[] DepartEip = {IpAddressRBpi, "5005", "ACCION",
"CONFIG", "ID_DEP", DepID, "CLAVE_DEP",
                    DepartPassword, "USER_NAME", UserName, "CLAVE_USER",
UserPassword, "USER_OF_DEP"};
            new SendMessage().execute(DepartEip);
        }
        AskPassword.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(List1OrList2){
                    String depID1=DepDATA.getText().toString();
```

```
                    String[]
DepartEip={IpAddressRBpi,"5005","ACCION","ASKDEPPASSWORD","ID_DEP",DepID,
"CLAVE_DEP",

DepartPassword,"USER_NAME",UserName,"CLAVE_USER",UserPassword,"PASS_OF_DE
P",depID1};
                    new SendMessage().execute(DepartEip);
                }else if(SetNewPassword) {
                        String str=randomPassword();
                        NewDepPassw2.setText(str);
                        NewDepPasswL.setVisibility(View.VISIBLE);
                        SetNewPassword=false;
                }else{
                        AlertDialog.Builder builder=new
AlertDialog.Builder(AdminConfiguration.this);
                        builder.setTitle("WARNING");
                        builder.setMessage("Are you sure you want to
change the department password? If it is " +
                                "the department you are in, you will have
to set this new password.");
                        builder.setPositiveButton("YES", new
DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog,
int which) {
                                String
NewPass=NewDepPassw2.getText().toString();
                                String[] DepartEip = {IpAddressRBpi,
"5005", "ACCION", "CONFIG", "ID_DEP",
                                        DepID,
"CLAVE_DEP",DepartPassword, "USER_NAME", UserName,
                                        "CLAVE_USER", UserPassword,
"CHANGE_DEP_PASS", depiD,"NEW_DEP_PASS1",NewPass};
                                new SendMessage().execute(DepartEip);
                            }
                        });
                        builder.setNegativeButton("NO", new
DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog,
int which) {
                                dialog.cancel();
                            }
                        });
                        AlertDialog dialog=builder.create();
                        dialog.show();
                }
            }
        });
        EditDepartData.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(List1OrList2) {
                    depiD = DepDATA.getText().toString();
                    String[] DepartEip = {IpAddressRBpi, "5005",
"ACCION", "CONFIG", "ID_DEP", DepID, "CLAVE_DEP",
                            DepartPassword, "USER_NAME", UserName,
"CLAVE_USER", UserPassword, "USER_OF_DEP", depiD};
                    new SendMessage().execute(DepartEip);
                }else if(DepAdm_Adm) {
```

```
                        AlertDialog.Builder builder=new
AlertDialog.Builder(AdminConfiguration.this);
                        builder.setTitle("Configuration");
                        builder.setMessage("Are you sure you want to
delete the department?");
                        builder.setPositiveButton("YES", new
DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog,
int which) {
                                String[]
DepartEip={IpAddressRBpi,"5005","ACCION","CONFIG","ID_DEP",DepID,"CLAVE_D
EP",

DepartPassword,"USER_NAME",UserName,"CLAVE_USER",UserPassword,"DELETE_DEP
",depiD};
                                new SendMessage().execute(DepartEip);
                            }
                        });
                        builder.setNegativeButton("NO", new
DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog,
int which) {
                                dialog.cancel();
                            }
                        });
                        AlertDialog dialog=builder.create();
                        dialog.show();
                    }else{
                        ConfigUsuario();
                    }
                }
            });
            Delete_user.setOnClickListener(new View.OnClickListener() {

                @Override
                public void onClick(View v) {
                    depiD = DepDATA.getText().toString();
                    AlertDialog.Builder builder = new
AlertDialog.Builder(AdminConfiguration.this);
                    builder.setMessage("Are you shore you want to delete the
user?");
                    builder.setTitle("Configuration");
                    builder.setPositiveButton("Yes", new
DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int
which) {
                            String[] DepartEip = {IpAddressRBpi, "5005",
"ACCION", "CONFIG", "ID_DEP", DepID, "CLAVE_DEP",
                                    DepartPassword, "USER_NAME", UserName,
"CLAVE_USER", UserPassword, "USER_TO_DELETE", depiD};
                            new SendMessage().execute(DepartEip);
                        }
                    });
                    builder.setNegativeButton("No", new
DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int
```

```
which) {
                        dialog.cancel();
                    }
                });
                AlertDialog dialog=builder.create();
                dialog.show();
            }
        });
        NewDepartment.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                LinearLayout1.setVisibility(View.GONE);
                LinearLayout2.setVisibility(View.VISIBLE);
                NewDepartment.setVisibility(View.GONE);
            }
        });
        randomPass.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String str=randomPassword();
                NewDepPassw.setText(str);
            }
        });
        addDepartment.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String new_dep=NewDepID.getText().toString();
                String new_dep_pass=NewDepPassw.getText().toString();
                String[] DepartEip = {IpAddressRBpi, "5005", "ACCION",
"CONFIG", "ID_DEP", DepID, "CLAVE_DEP",
                        DepartPassword, "USER_NAME", UserName,
"CLAVE_USER", UserPassword, "NEW_DEP", new_dep,
"NEW_DEP_PASS",new_dep_pass};
                new SendMessage().execute(DepartEip);
            }
        });
        EditAdmin.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                ConfigUsuario();
            }
        });
    }
    public void List(String[] MSJ1){
        List1=findViewById(R.id.List1);
        ArrayAdapter<String> adapter=new
ArrayAdapter<>(this,R.layout.text,MSJ1);
        List1.setAdapter(adapter);
        List1.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
                String DEPDATA=(String)
List1.getItemAtPosition(position);
                DepDATA.setText(DEPDATA);
            }
        });
    }
    private String randomPassword(){
```

```
    Random r=new Random();
    String alphabet =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
    char x;
    String[] y={"1","2","3","4","5","6","7","8","9","0"};
                for (int i = 0; i < 10; i++) {
        x=alphabet.charAt(r.nextInt(alphabet.length()));
        y[i]=Character.toString(x);
        Log.i(prue, Character.toString(x));
    }
    StringBuilder builder = new StringBuilder();
                for(String s : y) {
        builder.append(s);
    }
    String str = builder.toString();
    Log.i(prue,"es lo que sale aleatorio "+str);
    return str;
    }
    private void ConfigUsuario(){
        Intent intent = new Intent(this, UserDataRegistration.class);
        startActivity(intent);
    }
    @SuppressLint("StaticFieldLeak")
    private class SendMessage extends AsyncTask<String[] , Void, String>
{
        @Override
        protected String doInBackground(String[]... params) {
            String[] parama=params[0];
            String ipaddress=parama[0];
            int PORT=Integer.parseInt(parama[1]);
            StringBuffer paramString=new StringBuffer();
            paramString=paramString.append(parama[2]);
            for(int i=3;i<parama.length;i++){
                paramString=paramString.append(" ");
                paramString=paramString.append(parama[i]);
            }
            String param=paramString.toString();
            try {
                Socket socket = new Socket(ipaddress, PORT);
                BufferedReader in =new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                Log.i(prue,"Socker Creado: "+ipaddress+":"+PORT);
                PrintWriter out = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()));
                Log.i(prue,"Buffers Creados");
                try {
//                          try {
  //                            Thread.sleep(100);
    //                        } catch (InterruptedException ex) {
      //                      }
                        out.print(param);
                        out.flush();
                    Log.i(prue,"Se espera respuesta ");
                    String respuesta=in.readLine();
                    in.close();
                    socket.close();
                    Log.i(prue,"Se recibio: "+respuesta);
                    return respuesta;
                } catch (IOException e) {
                    e.printStackTrace();
```

```
                    out.flush();
                    in.close();
                    socket.close();
                    Log.i(prue,"Error al enviar o recibir");
                    return "FALSE";
                }
            } catch (Exception e) {
                Log.i(prue,"Error al crear el Socket o Buffers");
                return "FALSE";
            }
        }
        @SuppressLint("SetTextI18n")
        @Override
        protected void onPostExecute(String MSJ) {
            String[] MSJ1= MSJ.split(",");
            switch (MSJ1[0]) {
                case "LIST":
                    List(MSJ1);
                    break;
                case "PASSWORDIS":
                    DepPassw.setText(MSJ1[1]);
                    break;
                case "DEPARTMENT_UPDATA":
                    LinearLayout2.setVisibility(View.GONE);
                    NewDepPasswL.setVisibility(View.GONE);
                    Delete_user.setVisibility(View.GONE);
                    AskPassword.setText("Password Request");
                    EditDepartData.setText("Edit department DATA");
                    DepPassw.setVisibility(View.VISIBLE);
                    LinearLayout1.setVisibility(View.VISIBLE);
                    NewDepartment.setVisibility(View.VISIBLE);
                    EditAdmin.setVisibility(View.VISIBLE);
                    List1OrList2 = true;
                    SetNewPassword = true;
                    List(MSJ1);
                    Toast.makeText(AdminConfiguration.this,
                            "The configuration was successful",
                            Toast.LENGTH_LONG).show();
                    break;
                case "DEPARTMENT_USERS":
                    LinearLayout2.setVisibility(View.GONE);
                    NewDepartment.setVisibility(View.GONE);
                    DepPassw.setVisibility(View.GONE);
                    EditAdmin.setVisibility(View.GONE);
                    AskPassword.setText("Change department password");
                    if (DepAdm_Adm) {
                        EditDepartData.setText("Delete Department");
                    } else {
                        EditDepartData.setText("Edit Department Admin
Data");
                    }
                    LinearLayout1.setVisibility(View.VISIBLE);
                    Delete_user.setVisibility(View.VISIBLE);
                    List1OrList2 = false;
                    List(MSJ1);
                    Toast.makeText(AdminConfiguration.this,
                            "The configuration was successful ",
                            Toast.LENGTH_LONG).show();
                    break;
                default:
```

```
                        Log.i(prue, "somting rong");
                        Toast.makeText(AdminConfiguration.this,
                                "The configuration was not successful.",
                                Toast.LENGTH_LONG).show();
                        break;
                }
            }
        }
    }
```

ANEXO w

Programación XML de actividad “activity_admin_configuration.xml”.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">
            <LinearLayout
                android:id="@+id/LinearLayout1"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical"
                android:visibility="visible">
                <ListView
                    android:id="@+id/List1"
                    android:layout_width="match_parent"
                    android:layout_height="245dp"
                    android:padding="20dp"
                    android:scrollbars="vertical"
                    tools:layout_editor_absoluteX="0dp"
                    tools:layout_editor_absoluteY="0dp"
                    tools:ignore="NestedScrolling" />
                <RelativeLayout
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content">
                    <EditText
                        android:id="@+id/DepDATA"
                        android:layout_width="130dp"
                        android:layout_height="wrap_content"
                        android:layout_alignParentStart="true"
                        android:layout_alignParentTop="true"
                        android:layout_marginStart="30dp"
                        android:padding="10dp"
                        android:textSize="8pt"
                        tools:ignore="LabelFor,TextFields" />
                    <TextView
                        android:id="@+id/DepPassw"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_alignParentEnd="true"
                        android:layout_alignParentTop="true"
                        android:layout_marginEnd="30dp"
                        android:padding="10dp"
                        android:textSize="8pt"
                        tools:ignore="RelativeOverlap" />
                    <Button
                        android:id="@+id/Delete_user"
```

```
                    android:layout_width="130dp"
                    android:layout_height="wrap_content"
                    android:layout_alignParentEnd="true"
                    android:layout_alignParentTop="true"
                    android:layout_marginEnd="30dp"
                    android:text="Delete User"
                    android:visibility="invisible"
                    tools:ignore="HardcodedText" />
            </RelativeLayout>
            <RelativeLayout
                android:id="@+id/NewDepPassw1"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:visibility="gone">
                <EditText
                    android:id="@+id/NewDepPassw2"
                    android:layout_width="145dp"
                    android:layout_height="wrap_content"
                    android:layout_alignParentTop="true"
                    android:layout_centerHorizontal="true"
                    android:text="text"
                    tools:ignore="HardcodedText,LabelFor,TextFields"
/>
            </RelativeLayout>
            <RelativeLayout
                android:layout_width="match_parent"
                android:layout_height="168dp">
                <Button
                    android:id="@+id/AskPassword"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_alignParentTop="true"
                    android:layout_centerHorizontal="true"
                    android:text="Password Request" />
                <Button
                    android:id="@+id/EditDepartData"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_centerInParent="true"
                    android:text="Edit department DATA" />
                <Button
                    android:id="@+id/EditAdmin"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_alignParentBottom="true"
                    android:layout_centerHorizontal="true"
                    android:text="Edit Admin DAta" />
            </RelativeLayout>
        </LinearLayout>
        <LinearLayout
            android:id="@+id/LinearLayout2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="80dp"
            android:orientation="vertical"
            android:visibility="gone">
            <RelativeLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content">
                <TextView
```

```
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentStart="true"
                android:layout_centerVertical="true"
                android:layout_marginStart="30dp"
                android:text="@string/new_department"
                tools:ignore="RelativeOverlap" />
            <EditText
                android:id="@+id/NewDepID"
                android:layout_width="135dp"
                android:layout_height="wrap_content"
                android:layout_alignParentEnd="true"
                android:layout_alignParentTop="true"
                android:layout_marginEnd="30dp"
                android:padding="10dp"
                android:textSize="8pt"
                tools:ignore="LabelFor,TextFields" />
        </RelativeLayout>
        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentStart="true"
                android:layout_centerVertical="true"
                android:layout_marginStart="30dp"
                android:text="Password" />
            <EditText
                android:id="@+id/NewDepPassw"
                android:layout_width="135dp"
                android:layout_height="wrap_content"
                android:layout_alignParentEnd="true"
                android:layout_alignParentTop="true"
                android:layout_marginEnd="30dp"
                android:padding="10dp"
                android:textSize="8pt"
                tools:ignore="LabelFor,TextFields" />
        </RelativeLayout>
        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="105dp">
            <Button
                android:id="@+id/RandomPass"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentTop="true"
                android:layout_centerHorizontal="true"
                android:text="Generate a random password" />
            <Button
                android:id="@+id/AddDepartment"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentBottom="true"
                android:layout_centerHorizontal="true"
                android:text="add department data" />
        </RelativeLayout>
    </LinearLayout>
</LinearLayout>
</ScrollView>
```

```
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/NewDepartment"
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentEnd="true"
        android:layout_marginBottom="15dp"
        android:layout_marginEnd="15dp"
        android:clickable="true"
        app:backgroundTint="@android:color/holo_green_dark"
        app:srcCompat="@android:drawable/ic_input_add" />
</RelativeLayout>
```

ANEXO X

Programación Java de actividad “VideoAccess.java”.

```
package com.example.android.applicationvideoportero;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.PendingIntent;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.SharedPreferences;
import android.content.res.ColorStateList;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.net.sip.SipAudioCall;
import android.net.sip.SipException;
import android.net.sip.SipManager;
import android.net.sip.SipProfile;
import android.net.sip.SipRegistrationListener;
import android.os.AsyncTask;
import android.os.PowerManager;
import android.support.design.widget.FloatingActionButton;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.Window;
import android.webkit.WebResourceError;
import android.webkit.WebResourceRequest;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Button;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;
import java.text.ParseException;

import static
com.example.android.applicationvideoportero.MainActivity.MainData;

public class VideoAccess extends Activity { //AppCompatActivity {
    private WebView WebDisplay;
    private FloatingActionButton AudioButton;
    private FloatingActionButton MicrophoneButton;
    private FloatingActionButton ChooseVideo6;
    private Button ReloadVideo;
    private boolean AudioOnOff, MicOnOff,VideoOnOff=true;
    private final String
STATE_MIC="MicState",STATE_AUDIO="AudioState",STATE_VIDEO="VideoStare";
    public SipManager mSipManager = null;
    public SipProfile mSipProfile = null;
    public SipAudioCall call = null;
    public String username = "Android", domain , password = "password";
```

```
    String
IpAddressRBpi,Port1RBpi,DepartID,DepartPassword,UserName,UserPassword;
    public IncomingCallReceiver callReceiver;

    SensorManager senManager;
    Sensor sensor;
    SensorEventListener sensorEventListener;

    private PowerManager.WakeLock wakeLock;

    Integer port = 5000;
    Boolean Receiver=true;
    String Logs="Pruebas";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_video_access);
        WebDisplay= findViewById(R.id.WebDisplay);

        SharedPreferences
iPAddressRBpi=getSharedPreferences(MainData,MODE_PRIVATE);

IpAddressRBpi=iPAddressRBpi.getString("IpAddressRBpi",IpAddressRBpi);
        SharedPreferences
PorT1RBpi=getSharedPreferences(MainData,MODE_PRIVATE);
        Port1RBpi=PorT1RBpi.getString("Port1RBpi",Port1RBpi);
        SharedPreferences
departID=getSharedPreferences(MainData,MODE_PRIVATE);
        DepartID=departID.getString("DepartID",DepartID);
        SharedPreferences
departPassword=getSharedPreferences(MainData,MODE_PRIVATE);

DepartPassword=departPassword.getString("DepartPassword",DepartPassword);
        SharedPreferences
userName=getSharedPreferences(MainData,MODE_PRIVATE);
        UserName=userName.getString("UserName",UserName);
        SharedPreferences
userPassword=getSharedPreferences(MainData,MODE_PRIVATE);
        UserPassword=userPassword.getString("UserPassword",UserPassword);

        domain=IpAddressRBpi;
        Log.i(Logs,IpAddressRBpi);
        SEND_ACCTION("START_HTTP_VIDEO");
        PowerManager powerManager = (PowerManager)
getSystemService(POWER_SERVICE);
        assert powerManager != null;
        int field = 0x00000020;
        wakeLock = powerManager.newWakeLock(field, getLocalClassName());
        senManager=(SensorManager) getSystemService(SENSOR_SERVICE);
        assert senManager != null;
        sensor=senManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
        if (sensor!=null){
            sensorEventListener=new SensorEventListener() {
                @Override
                public void onSensorChanged(SensorEvent event) {
                    if(call!=null) {
                        if (event.values[0] < sensor.getMaximumRange()) {
                            call.setSpeakerMode(false);
                            WebDisplay.stopLoading();
```

```
                            if(!wakeLock.isHeld()) {
                                wakeLock.acquire(10*60*1000L /*10
minutes*/);
                            }
                        } else {
                            WebDisplay.setWebViewClient(new MyClient());
                            WebDisplay.loadUrl("http://" + domain + ":" +
port);
                            call.setSpeakerMode(true);
                            if(wakeLock.isHeld()) {
                                wakeLock.release();
                            }
                        }
                    }
                }
                @Override
                public void onAccuracyChanged(Sensor sensor, int
accuracy) {
                }
            };
            start();
        }
        if (mSipManager == null) {
            mSipManager = SipManager.newInstance(this);
            Log.i(Logs, "mSipManager == null");
        }
        try {
            Log.i(Logs, "Sip Profilr");
            SipProfile.Builder builder = new SipProfile.Builder(username,
domain);
            builder.setPassword(password);
            mSipProfile = builder.build();
            Log.i(Logs, "Sip Profile Build");
        } catch (ParseException e) {
            e.printStackTrace();
            Log.e(Logs, "Error");
        }
        Intent intent = new Intent();
        intent.setAction("android.SipDemo.INCOMING_CALL");
        PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0,
                intent, Intent.FILL_IN_DATA);
        try {
            mSipManager.open(mSipProfile, pendingIntent, null);
            Log.i(Logs, "Open Sip Profile");
        } catch (SipException e) {
            e.printStackTrace();
            Log.e(Logs, "Error 2");
        }
        IntentFilter filter = new IntentFilter();
        filter.addAction("android.SipDemo.INCOMING_CALL");
        callReceiver = new IncomingCallReceiver();
        this.registerReceiver(callReceiver, filter);
        try {

mSipManager.setRegistrationListener(mSipProfile.getUriString(),
                    new SipRegistrationListener() {
                        @Override
                        public void onRegistering(String localProfileUri)
{
                            updateStatus("Registering with SIP
```

```
Server...");
                                Log.i(Logs, "onRegistration ");
                            }
                            @Override
                            public void onRegistrationDone(String
localProfileUri, long expiryTime) {
                                updateStatus("Ready");
                                Log.i(Logs, "onRegistrationDone");
                            }
                            @Override
                            public void onRegistrationFailed(String
localProfileUri, int errorCode,
                                                             String
errorMessage) {
                                updateStatus("Registration failed.");
                                Log.i(Logs, "onRegistrationFailed");
                            }
                        });
            } catch (SipException e) {
                e.printStackTrace();
            }
            if (savedInstanceState !=null){
                AudioOnOff=(Boolean)
savedInstanceState.getSerializable(STATE_AUDIO);
                MicOnOff=(Boolean)
savedInstanceState.getSerializable(STATE_MIC);
                VideoOnOff=(Boolean)
savedInstanceState.getSerializable(STATE_VIDEO);
            }
            // Asignacion de Botones a las variables
            AudioButton= findViewById(R.id.AudioButtonD1);
            MicrophoneButton=findViewById(R.id.MicrophoneButtonD2);
            ChooseVideo6=findViewById(R.id.ChooseVideo6);
            FloatingActionButton openDoorButton =
findViewById(R.id.OpenDoorButtonD3);
            FloatingActionButton closeCallButton =
findViewById(R.id.CloseCallButtonD4);
            FloatingActionButton configButton =
findViewById(R.id.ConfigButtonD5);
            if (AudioOnOff){

AudioButton.setImageTintList(ColorStateList.valueOf(getResources().getCol
or(R.color.colorLightGreen)));
            }else{

AudioButton.setImageTintList(ColorStateList.valueOf(getResources().getCol
or(R.color.colorRed)));
            }
            if (MicOnOff){

MicrophoneButton.setImageTintList(ColorStateList.valueOf(getResources().g
etColor(R.color.colorLightGreen)));
            }else{

MicrophoneButton.setImageTintList(ColorStateList.valueOf(getResources().g
etColor(R.color.colorRed)));
            }
            if (VideoOnOff){

ChooseVideo6.setImageTintList(ColorStateList.valueOf(getResources().getCo
```

```
lor(R.color.colorLightGreen)));
        }else{

ChooseVideo6.setImageTintList(ColorStateList.valueOf(getResources().getCo
lor(R.color.colorRed)));
        }
//***************AudioButton
Action*******************************************
        AudioButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (Receiver) {

AudioButton.setImageTintList(ColorStateList.valueOf(getResources().getCol
or(R.color.colorLightGreen)));
                    SEND_ACCTION("SPECTING_CALL");
                    Receiver = false;
                }
                if (call != null) {
                    if (AudioOnOff) {
                        AudioOnOff = false;

AudioButton.setImageTintList(ColorStateList.valueOf(getResources().getCol
or(R.color.colorRed)));
                        try {
                            call.holdCall(30);
                        } catch (SipException e) {
                            e.printStackTrace();
                        }
                    } else {
                        AudioOnOff = true;//todo Comando para activar el
audio

AudioButton.setImageTintList(ColorStateList.valueOf(getResources().getCol
or(R.color.colorLightGreen)));
                        if (call.isOnHold()) {
                            try {
                                call.continueCall(5);
                            } catch (SipException e) {
                                e.printStackTrace();
                            }
                        }
                    }
                }
            }
        });
//**************MicrophoneButton Action
*****************************************
        MicrophoneButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(call!=null) {
                    if (MicOnOff) {
                        MicOnOff = false;//todo Comandos para desactivar
el audio

MicrophoneButton.setImageTintList(ColorStateList.valueOf(getResources().g
etColor(R.color.colorRed)));
                        if (!call.isMuted()) {
                            call.toggleMute();
```

```
                        }
                    } else {
                        MicOnOff = true;//todo Comando para activar el
audio

MicrophoneButton.setImageTintList(ColorStateList.valueOf(getResources().g
etColor(R.color.colorLightGreen)));
                        if (call.isMuted()) {
                            call.toggleMute();
                        }
                    }
                }
            }
        });

//*************Cámara Video
*****************************************
        ChooseVideo6.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (VideoOnOff){
                    SEND_ACCTION("STOP_HTTP_VIDEO");
                }else{
                    SEND_ACCTION("START_HTTP_VIDEO");
                    ReloadVideo.setVisibility(View.VISIBLE);
                    WebDisplay.setVisibility(View.GONE);
                }
            }
        });

        ReloadVideo=findViewById(R.id.ReloadVideo);
        ReloadVideo.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                WebDisplay.setWebViewClient(new MyClient());
                WebDisplay.loadUrl("http://" + domain + ":" + port);
            }
        });

//*************OpenDoorButton
Accion*********************************************
        openDoorButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
            SEND_ACCTION("OPEN_DOOR");
            }
        });
//*************CloseCallButton
Accion*********************************************
        closeCallButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(call!=null){
                    try {
                        call.endCall();
                    } catch (SipException e) {
                        e.printStackTrace();
                    }
                }
                WebDisplay.stopLoading();
```

```
                Receiver=true;
                closeLocalProfile();
                goback();
            }
        });
//*************ConfigButton
Accion*******************************************
        configButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                WebDisplay.stopLoading();
                NextWinConfigData();
                openConfig();
            }
        });

    }
    //Guarda los datos deceados al tener una interrupción como al
voltearse el celular
    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putBoolean(STATE_AUDIO, AudioOnOff);
        savedInstanceState.putBoolean(STATE_MIC, MicOnOff);
        savedInstanceState.putBoolean(STATE_VIDEO, VideoOnOff);
        super.onSaveInstanceState(savedInstanceState);
    }
    public void openConfig() {
        Intent int1 = new Intent(this, UserDataVerification.class);
        startActivity(int1);
    }
    public  void NextWinConfigData(){
        SharedPreferences
settings=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences.Editor edit=settings.edit();
        edit.putBoolean("VideoOrConfig",false);
        edit.apply();
    }
    private class MyClient extends WebViewClient {
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, String url)
{
            view.loadUrl(url);
            return true;
        }
        @Override
        public void onReceivedError(WebView view, WebResourceRequest
request,
                                                WebResourceError error) {
            super.onReceivedError(view, request, error);
            ReloadVideo.setVisibility(View.VISIBLE);
            WebDisplay.setVisibility(View.GONE);
        }
        @Override
        public void onPageFinished(WebView view, String url) {
            ReloadVideo.setVisibility(View.GONE);
            WebDisplay.setVisibility(View.VISIBLE);
        }
    }
    @SuppressLint("LongLogTag")
    public void closeLocalProfile() {
```

```
        SEND_ACCTION("CLOSE_CALL");
        try {
            if (mSipProfile != null) {
                mSipManager.close(mSipProfile.getUriString());
            }
        } catch (Exception ee) {
            Log.d("WalkieTalkieActivity/onDestroy", "Failed to close
local profile.", ee);
        }
    }
    public void updateStatus(final String status) {
        this.runOnUiThread(new Runnable() {
            public void run() {
                Log.i("Pruebas:",status);
                Toast.makeText(VideoAccess.this,
                        status,
                        Toast.LENGTH_LONG).show();
            }
        });
    }
    private void goback() {
        Intent int1 = new Intent(this, MainActivity.class);
        startActivity(int1);
    }
    public void start(){
        senManager.registerListener(sensorEventListener,sensor,1000);
    }
    public void stop(){
        senManager.unregisterListener(sensorEventListener);
    }
    @Override
    protected void onPause(){
        stop();
        closeLocalProfile();
        super.onPause();
    }
    @Override
    protected void onResume(){
        start();
        if (VideoOnOff){

ChooseVideo6.setImageTintList(ColorStateList.valueOf(getResources().getCo
lor(R.color.colorLightGreen)));
        }else{

ChooseVideo6.setImageTintList(ColorStateList.valueOf(getResources().getCo
lor(R.color.colorRed)));
        }
        super.onResume();
    }
    @Override
    public void onDestroy() {
        if (call != null) {
            call.close();
        }
        super.onDestroy();
    }
    private void SEND_ACCTION(String ACCION) {
        String[]
DepartEip={IpAddressRBpi,Port1RBpi,"ACCION",ACCION,"ID_DEP",
```

```
                DepartID,"CLAVE_DEP",DepartPassword,"USER_NAME",
                UserName,"CLAVE_USER",UserPassword};
        new SendMessage().execute(DepartEip);
    }
    @SuppressLint("StaticFieldLeak")
    public class SendMessage extends AsyncTask<String[] , Void, String> {
        @Override
        protected String doInBackground(String[]... params) {
            String[] parama=params[0];
            String ipaddress=parama[0];
            int PORT=Integer.parseInt(parama[1]);
            StringBuffer paramString=new StringBuffer();
            paramString=paramString.append(parama[2]);
            for(int i=3;i<parama.length;i++){
                paramString=paramString.append(" ");
                paramString=paramString.append(parama[i]);
            }
            String param=paramString.toString();
            String prue = "Pruebas";
            try {
                Socket socket = new Socket(ipaddress, PORT);
                socket.setSoTimeout(3000);
                BufferedReader in =new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                Log.i(prue,"Socker Creado: "+ipaddress+":"+PORT);
                PrintWriter out = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()));
                Log.i(prue,"Buffers Creados");
                try {
                    out.print(param);
                    out.flush();
                    Log.i(prue,"Se espera respuesta ");
                    String respuesta=in.readLine();
                    in.close();
                    socket.close();
                    Log.i(prue,"Se recibio: "+respuesta);
                    return respuesta;
                } catch (IOException e) {
                    e.printStackTrace();
                    out.flush();
                    in.close();
                    socket.close();
                    Log.i(prue,"Error al enviar o recibir");
                    return "FALSE";
                }
            } catch (Exception e) {
                Log.i(prue,"Error al crear el Socket o Buffers");
                return "FALSE";
            }
        }
        @SuppressLint("SetTextI18n")
        @Override
        protected void onPostExecute(String MSJ) {
            switch (MSJ) {
                case "RECEIVING_CALL":

AudioButton.setImageTintList(ColorStateList.valueOf(getResources().getCol
or(R.color.colorLightGreen)));
                    Toast.makeText(VideoAccess.this,
                            "Comunicating whit the access door ",
```

```
                        Toast.LENGTH_LONG).show();
                break;
            case "CLOSE_CALL":

AudioButton.setImageTintList(ColorStateList.valueOf(getResources().getCol
or(R.color.colorRed)));
                Toast.makeText(VideoAccess.this,
                        "The Comunication finish",
                        Toast.LENGTH_LONG).show();
                break;
            case "CALL_BUSY":

AudioButton.setImageTintList(ColorStateList.valueOf(getResources().getCol
or(R.color.colorRed)));
                Receiver = true;
                Toast.makeText(VideoAccess.this,
                        "Door phone is busy",
                        Toast.LENGTH_LONG).show();
                break;
            case "DOOR_IS_OPEN":
                Toast.makeText(VideoAccess.this,
                        "The door is open!",
                        Toast.LENGTH_LONG).show();
                break;
            case "HTTP_RUNING":

ChooseVideo6.setImageTintList(ColorStateList.valueOf(getResources().getCo
lor(R.color.colorLightGreen)));
                VideoOnOff=true;
                Toast.makeText(VideoAccess.this,
                        "Video Online",
                        Toast.LENGTH_LONG).show();
                break;
            case "HTTP_STOP":

ChooseVideo6.setImageTintList(ColorStateList.valueOf(getResources().getCo
lor(R.color.colorRed)));
                VideoOnOff=false;
                Toast.makeText(VideoAccess.this,
                        "Video Offline",
                        Toast.LENGTH_LONG).show();
                break;
        }
    }
}
}
```

ANEXO Y

Programación XML de actividad “activity_video_access.xml”.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".VideoAccess">
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/AudioButtonD1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignTop="@+id/OpenDoorButtonD3"
        android:layout_marginStart="59dp"
        android:clickable="true"
        android:tint="@color/colorRed"
        app:backgroundTint="@android:color/darker_gray"
        app:srcCompat="@android:drawable/ic_lock_silent_mode_off"
        tools:focusable="auto"
        tools:layout_editor_absoluteX="172dp"
        tools:layout_editor_absoluteY="352dp"
        android:focusable="true" />
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/MicrophoneButtonD2"
        android:layout_width="65dp"
        android:layout_height="61dp"
        android:layout_alignTop="@+id/AudioButtonD1"
        android:layout_centerHorizontal="true"
        android:clickable="true"
        android:tint="@color/colorRed"
        app:backgroundTint="@android:color/darker_gray"
        app:srcCompat="@android:drawable/presence_audio_online"
        tools:focusable="true"
        tools:layout_editor_absoluteX="172dp"
        tools:layout_editor_absoluteY="352dp"
        android:focusable="true" />
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/OpenDoorButtonD3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentEnd="true"
        android:layout_marginBottom="100dp"
        android:layout_marginEnd="60dp"
        android:clickable="true"
        android:focusable="true"
        app:backgroundTint="@android:color/darker_gray"
        app:srcCompat="@drawable/door"
        tools:layout_editor_absoluteX="300dp"
        tools:layout_editor_absoluteY="352dp"
        tools:ignore="RelativeOverlap" />
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/CloseCallButtonD4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_alignTop="@+id/ConfigButtonD5"
        android:layout_centerHorizontal="true"
        android:clickable="true"
        android:focusable="true"
        android:tint="@android:color/background_light"
        app:backgroundTint="@android:color/holo_red_dark"
        app:srcCompat="@android:drawable/sym_action_call"
        tools:layout_editor_absoluteX="172dp"
        tools:layout_editor_absoluteY="426dp" />
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/ConfigButtonD5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignStart="@+id/OpenDoorButtonD3"
        android:layout_marginBottom="35dp"
        android:clickable="true"
        android:focusable="true"
        android:tint="@android:color/black"
        app:backgroundTint="@android:color/darker_gray"
        app:srcCompat="@android:drawable/ic_menu_manage"
        tools:layout_editor_absoluteX="281dp"
        tools:layout_editor_absoluteY="425dp" />
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/ChooseVideo6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignStart="@+id/AudioButtonD1"
        android:layout_marginBottom="35dp"
        android:clickable="true"
        android:focusable="true"
        android:tint="@android:color/holo_green_light"
        app:backgroundTint="@android:color/darker_gray"
        app:srcCompat="@android:drawable/presence_video_online"
        tools:layout_editor_absoluteX="281dp"
        tools:layout_editor_absoluteY="425dp" />
    <Button
        android:id="@+id/ReloadVideo"
        android:layout_width="336dp"
        android:layout_height="328dp"
        android:layout_below="@+id/WebDisplay"
        android:layout_centerHorizontal="true"
        android:background="@android:drawable/ic_popup_sync"
        android:visibility="visible" />
    <WebView
        android:id="@+id/WebDisplay"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:clickable="true"
        android:focusable="true" />
</RelativeLayout>
```

ANEXO Z

IncomingCallReceiver.java

```
package com.example.android.applicationvideoportero;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.net.sip.SipAudioCall;
import android.net.sip.SipProfile;
import android.util.Log;

public class IncomingCallReceiver extends BroadcastReceiver {
    SipAudioCall incomingCall = null;
    @Override
    public void onReceive(Context context, Intent intent) {
        try {
            SipAudioCall.Listener listener = new SipAudioCall.Listener()
{
                @Override
                public void onRinging(SipAudioCall call, SipProfile
caller) {
                    try {
                        call.answerCall(30);
                        Log.i("Log","Incaming call");
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            };
            VideoAccess wtActivity = (VideoAccess) context;
            incomingCall = wtActivity.mSipManager.takeAudioCall(intent,
listener);
            incomingCall.answerCall(30);
            incomingCall.startAudio();
            incomingCall.setSpeakerMode(true);
            if(!incomingCall.isMuted()){
                incomingCall.toggleMute();
            }
            wtActivity.call = incomingCall;
        } catch (Exception e) {
            if (incomingCall != null) {
                incomingCall.close();
            }
        }
    }
}
```

ANEXO AA

CHannelID

```
package com.example.android.applicationvideoportero;

import android.app.Application;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.os.Build;

public class ChannelID extends Application {
    public static final String Channel_ID_1="Channel1";
    @Override
    public void onCreate() {
        super.onCreate();
        createNotificationChannel();
    }
    private void createNotificationChannel(){
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel channel1=new NotificationChannel(
                    Channel_ID_1,
                    "Calls From the Door",
                    NotificationManager.IMPORTANCE_HIGH
            );
            channel1.setDescription("This will notified you if someone is
locking for you");
            NotificationManager
manager=getSystemService(NotificationManager.class);
            assert manager != null;//
            manager.createNotificationChannel(channel1);
        }
    }
}
```

ANEXO BB

MyFarebaseMessagingService.java

```
package com.example.android.applicationvideoportero;

import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.media.RingtoneManager;
import android.net.Uri;
import android.support.v4.app.NotificationCompat;
import android.util.Log;

import com.google.firebase.messaging.FirebaseMessagingService;
import com.google.firebase.messaging.RemoteMessage;

import static
com.example.android.applicationvideoportero.ChannelID.Channel_ID_1;

public class MyFarebaseMessagingService extends FirebaseMessagingService{
    public static final String TAG="Pruebas";
    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        super.onMessageReceived(remoteMessage);
        String remitente=remoteMessage.getFrom();
        Log.i(TAG,"Remitente: "+remitente);

        if ( remoteMessage.getNotification() !=null){
            Log.i (TAG,"Notificacion:
"+remoteMessage.getNotification().getBody());

mostrarNotificacion(remoteMessage.getNotification().getTitle(),remoteMess
age.getNotification().getBody());
        }
    }
    private void mostrarNotificacion(String title, String body) {
        Uri sound=
RingtoneManager.getDefaultUri(RingtoneManager.TYPE_RINGTONE);
        Intent intent=new Intent(this, VideoAccess.class);
        intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        PendingIntent pendingIntent=PendingIntent.getActivity(this,
                0,intent,PendingIntent.FLAG_ONE_SHOT);
        NotificationCompat.Builder notif=new
NotificationCompat.Builder(this,Channel_ID_1).
                setSmallIcon(R.drawable.ic_launcher_foreground).
                setContentTitle(title).
                setContentText(body).
                setAutoCancel(true).
                setSound(sound).
                setOngoing(true).
                setFullScreenIntent(pendingIntent,true);
        NotificationManager notificationManager=(NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
        assert notificationManager != null;
        notificationManager.notify(0,notif.build());
        startActivity(intent);
    }
}
```

ANEXO CC

MyFirebaseInstanceIdService.java

```
package com.example.android.applicationvideoportero;

import android.content.SharedPreferences;
import android.util.Log;

import com.google.firebase.iid.FirebaseInstanceId;
import com.google.firebase.iid.FirebaseInstanceIdService;

import static
com.example.android.applicationvideoportero.MainActivity.MainData;

public class MyFirebaseInstanceIdService extends
FirebaseInstanceIdService{
    public static final String TAG = "Pruebas";

    @Override
    public void onTokenRefresh() {
        String tocken= FirebaseInstanceId.getInstance().getToken();
        Log.i (TAG, "Tocken: "+tocken);
        // Get updated InstanceID token.
        String refreshedToken =
FirebaseInstanceId.getInstance().getToken();
        Log.i(TAG, "Refreshed token: " + refreshedToken);

        SharedPreferences
token=getSharedPreferences(MainData,MODE_PRIVATE);
        SharedPreferences.Editor editor=token.edit();
        editor.putString("tocken", tocken);
        editor.apply();

        // If you want to send messages to this application instance or
        // manage this apps subscriptions on the server side, send the
        // Instance ID token to your app server.
//        sendRegistrationToServer(refreshedToken);
```

ANEXO DD

DD. Manual de Instalación y de Usuario

Contenido

1. Configuración Inicial de Video Portero

Para poder realizar las configuraciones iniciales del Video Portero es necesario tener adicionalmente los siguientes elementos:

Un monitor con puerto de video de entrada HDMI.

Un cable HDMI y adaptador HDMI a microHDMI.

Un teclado con conexión USB y adaptador USB a microUSB.

Una vez se hayan conectado los cables correspondientes, como se puede observar en la Figura DD.1.,

Figura DD.1. Modo de conexión de cables para configuración Inicial.

Se puede observar los cables negros conectados en el siguiente orden de izquierda a derecha: Cable HDMI, Teclado USB conectado con adaptador, fuente de energía de 5[V].

Después se podrá conectar a la toma corriente el adaptador de energía para que la microcomputadora empiece el encendido.

Después de encenderse por completo se podrá observar la siguiente línea.

login as:

donde se tendrá que colocar el nombre de usuario: pi

después de presionar enter, la pantalla solicitará la clave

password:

y la clave por defecto que se deberá colocar será: password

ya ingresado a la línea de comandos se podrá ver la ventana como se ve en la Figura DD.2.

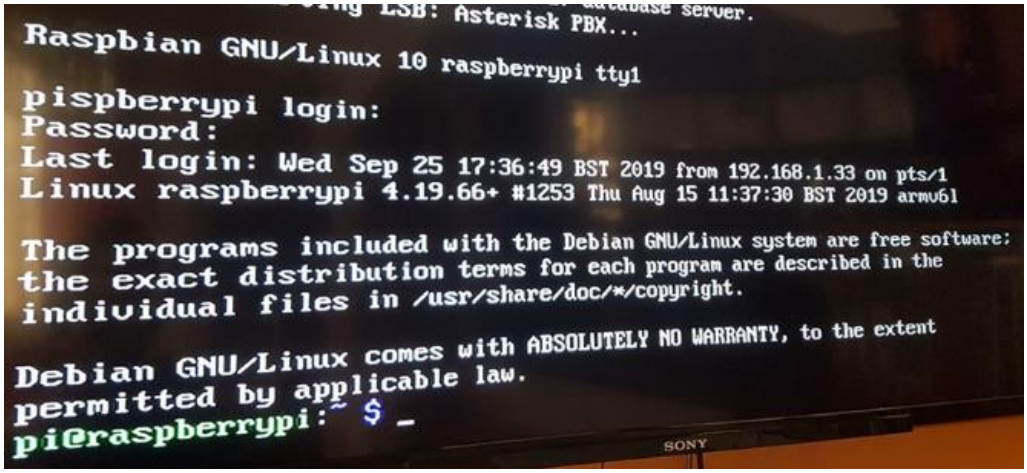

Figura DD.2. Línea de comandos inicial de Video Portero

Por lo que se procederá a conectar el Video Portero a la red.

1.1. Conexión del Video Portero a la Red inalámbrica

En la línea de comandos se escribirá lo siguiente: sudo raspi-config

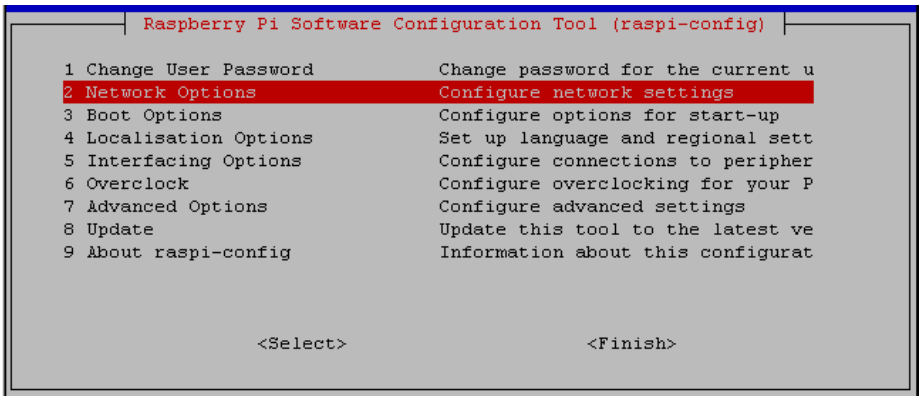

Figura DD.3. Ventana de Configuración de software.

Lo que le dará acceso a la ventana de configuración que se puede observar en la Figura DD.3 donde se deberá seleccionar con las flechas la segunda opción “Network Options” y presionando la tecla ‘ENTER’. Lo que permitirá ingresar a la ventana observada en la Figura DD.4.

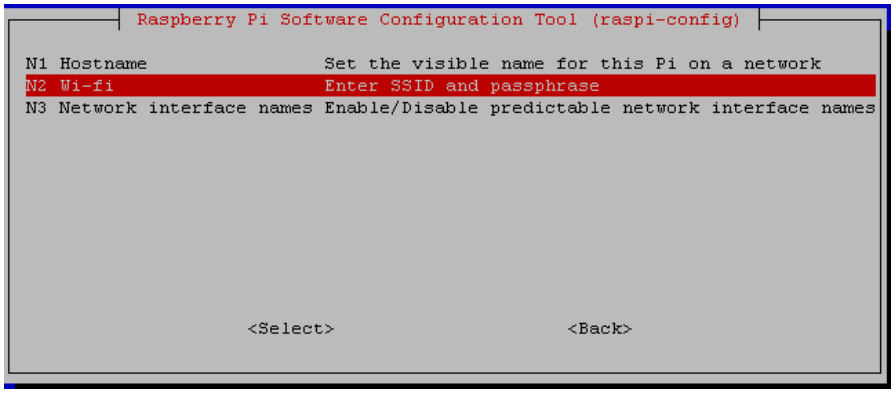

Figura DD.4. Ventana de configuración de red.

Y del mismo modo seleccionamos la segunda opción “Wi-Fi” para proceder a ingresar la red a la que deseamos conectar el Video Portero.

Se abrirá una nueva ventana para ingresar el SSID de la red inalámbrica, es decir el nombre de la red inalámbrica al que se desea conectar como por ejemplo se puede observar en la Figura DD.5.

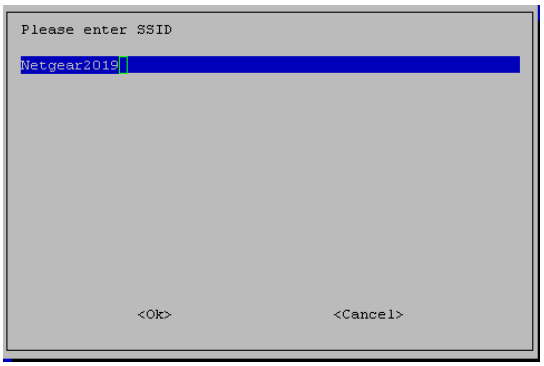

Figura DD.5. Ventana para ingresar el SSID de la red inalámbrica.

Para después presionar la tecla “ENTER” y poder ingresar la contraseña como se observa en la Figura DD.6.

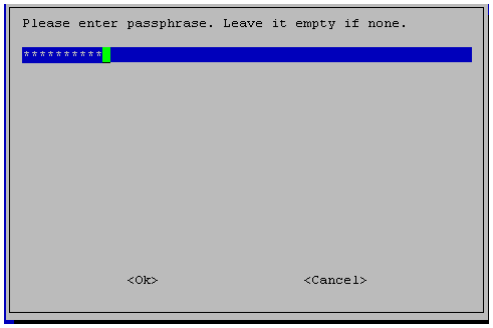

Figura DD.6. Ventana para ingresar la contraseña de la red inalámbrica.

Después de haber ingresado correctamente el SSID y la clave de la red inalámbrica se procederá a colocar una dirección IP fija en el video portero, IP que se usará para el registro en la aplicación del usuario.

1.2. Configuración dirección IP de Video Portero

La configuración de una dirección IP fija en el Video Portero es fundamental para mantener una comunicación con la aplicación por lo que se la debe de realizar antes de comenzar su uso.

En primer lugar, es necesario averiguar en qué rango de direcciones IP se encuentra la red LAN a la que se va a conectar, esto lo puede realizar ocupando una computadora que se encuentre conectada a esta red y ejecutando el programa “Símbolo del sistema” como se puede observar en la Figura DD.7.

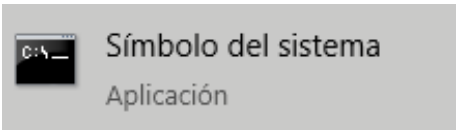

Figura DD.7 Icono de programa CMD para averiguar la dirección IP

Ejecutando este programa se abrirá una ventana de comandos en el que se deberá ingresar el comando: ipconfig

Y al oprimir la tecla “ENTER” se podrán observar los diferentes adaptadores de red entre los que se podrá observar el adaptador de Ethernet en el caso de estar conectado la computadora con cable UTP o el adaptador de LAN inalámbrico Wi-Fi, como se puede ver en la Figura DD.8.

```
Adaptador de Ethernet Ethernet 4:

   Estado de los medios. . . . . . . . . . . : medios desconectados
   Sufijo DNS específico para la conexión. . :

Adaptador de LAN inalámbrica Wi-Fi:

   Sufijo DNS específico para la conexión. . : local
   Dirección IPv6 . . . . . . . . . . . . . : 2800:370:8a:d8f0:bc9b:caf7:5260:dca7
   Dirección IPv6 temporal. . . . . . . . . : 2800:370:8a:d8f0:7c1a:fd45:13f8:ad95
   Vínculo: dirección IPv6 local. . . . . . : fe80::bc9b:caf7:5260:dca7%16
   Dirección IPv4. . . . . . . . . . . . . . : 192.168.1.36
   Máscara de subred . . . . . . . . . . . . : 255.255.255.0
   Puerta de enlace predeterminada . . . . . : fe80::1000%16
                                       192.168.1.1

C:\Users\jose9>
```

Figura DD.8. Imagen de la obtención de la dirección IP de una computadora

Aquí podremos observar las filas que dicen

Dirección IPv4, con la dirección IP…. 192.168.1.36

Máscara de subred……. 255.255.255.0

Lo que nos indica que el rango de direcciones IP posible de usar en esta red es desde 192.168.1.1 hasta 192.168.1.254

De este rango de direcciones se recomienda escoger una dirección IP alta, esto debido a que, si no se configura el servidor DHCP, podría existir un conflicto de direcciones, es decir que otro dispositivo sea asignado la dirección IP que se ha seleccionado, y al escoger una dirección IP alta se reduce esta posibilidad.

En este caso se escogerá la dirección IP: 192.168.1.150, que se encuentra dentro del rango de direcciones IP válidas. Y se colocará la misma máscara de subred, es decir 255.255.255.0

Una vez determinada la dirección IP que se colocará en el Video Portero, se procede a la configuración en la ventada de comandos de este, ingresando el comando:

sudo nano /etc/dhcpcd.config

que abrirá una hoja de texto en la que se deberá colocar la dirección IP requerida del siguiente modo:

interface wlan0

static ip_address=192.168.1.150/24

static routers=192.168.1.1

static domain_name_servers=192.168.1.1 8.8.8.8

ya configurado esto, se procede a guardar el archivo con las teclas 'CTRL+o' y se sale de este con las teclas 'CTRL+x'. y para culminar con esta configuración se procederá a reiniciar el dispositivo con el siguiente comando

sudo reboot

una vez reiniciado, se podrá proceder a registrar a los usuarios de administración general y estos podrán añadir departamentos para que se registren el resto de usuarios.

2. Registro inicial dependiendo el tipo de usuario

Sin importar el tipo de usuario que se quiera registrar en el Video Portero a través de la aplicación será necesario que acepte los permisos solicitados en la ventana inicial que se pueden observar en la Figura DD.9., caso contrario la aplicación no tendrá el correcto funcionamiento.

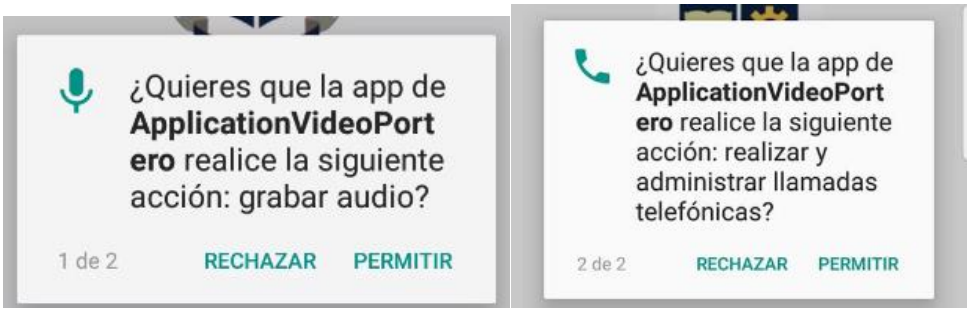

Figura DD.9. Ventanas auxiliares de solicitud de permisos.

Una vez aceptados los permisos, se podrá proceder presionando el botón con el símbolo + que se observa en la parte inferior de la pantalla del celular, pantalla que se puede ver en la Figura DD.10.

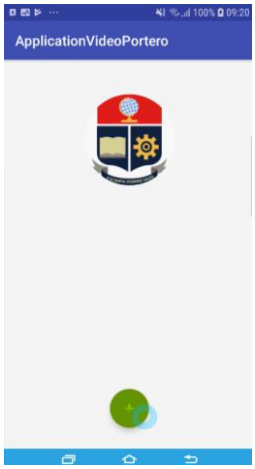

Figura DD.10. Pantalla inicial de la aplicación sin datos previos almacenados.

Los siguientes pasos de registro dependerá del tipo de usuario que se encuentre registrándose, que pueden ser:

Administrador general: administrará la base de datos entera, con el permiso de editar y borrar el identificador de un departamento y su respectiva clave, así como los usuarios de los departamentos.

Administrador de departamento: administra el departamento al que se encuentra registrado, permitiendo cambiar de clave o eliminar usuarios de este.

Usuario: le permite cambiar únicamente sus propios datos de usuario, es decir su nombre y clave de usuario.

2.1. Administrador general

Para poder registrarse como administrador general el usuario tendrá que seguir los siguientes pasos:

a. Una vez se encuentre en la ventana para registrar el departamento que se ve en la Figura DD.11., deberá colocar los siguientes datos:

Figura DD.11. Ventana de registro de departamento.

Dirección IP: Se coloca la dirección IP asignada al Video Portero.

Ejemplo: 192.168.1.150

Port: El número de puerto por defecto será **5005**, pero de haberlo cambiado se requerirá colocar este nuevo número.

ID de Departamento: Para registrarse como administrador requerirá ingresar el departamento registrado por defecto. **DEP_ADMIN**

Clave de Departamento: la clave por defecto de este departamento será: **password**

b. Una vez ingresados esto datos como se observa en la Figura DD.12, se podrá presionar el botón **NUEVO USUARIO,** para acceder a la siguiente ventana donde colocar su nombre de usuario y contraseña deseada, o el botón **USUARIO** para ingresar con el nombre de administrador predeterminado.

Figura DD.12. Ventana de registro de departamento llena con la información para registrarse como administrador general.

c. Si se presiona el botón Nuevo usuario se ingresará a la ventana que se puede observar en la Figura DD.13. a continuación.

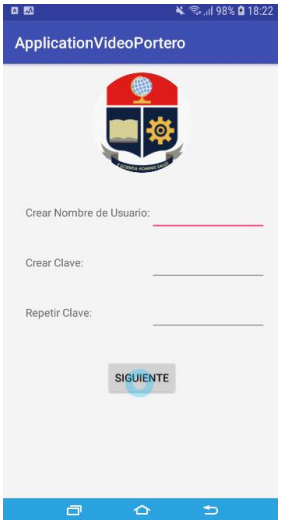

Figura DD.13. Ventana de registro de nuevo usuario

En esta ventana se ingresarán los datos que el usuario crea convenientes, tomando en cuenta que no deberá olvidarse de la clave de usuario.

Una vez llenos los campos se presiona siguiente y su registro quedará completo.

d. Por otro lado, si se presiona el botón **USUARIO** haciendo referencia a que ya existe un usuario registrado en la base de datos se podrá observar la ventana en la Figura DD.14., y podrá ingresar con el usuario que allá configurado anteriormente o con el usuario administrador que estará por defecto, siendo los siguientes datos necesario de ingresar:

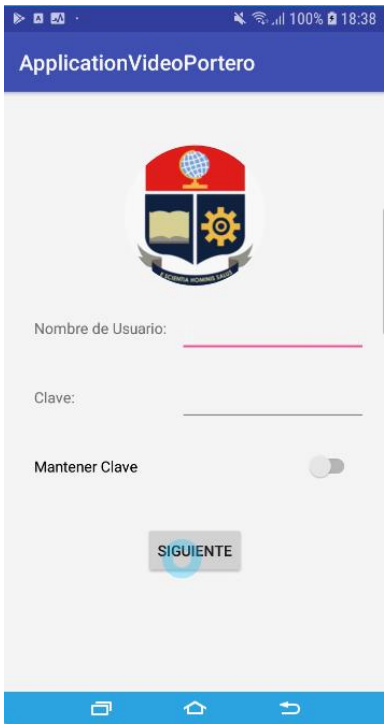

Figura DD.14. Ventana de ingreso de usuario previamente registrado.

Nombre de Usuario: ADMIN

Clave: password

Clave que se recomienda cambiar para mayor seguridad, la forma de configuración se la vera más adelante.

2.2. Administrador de departamento

a. Para poder registrar a un administrador de departamento en primer lugar deberá existir en la base de datos el departamento deseado, esto únicamente lo podrá añadir el Administrador General, por lo que los datos del departamento para ingresar en la ventana de la figura DD.11. deberán ser solicitados a este.

b. Una vez ingresados los datos del departamento obligadamente se deberá continuar presionando el botón Nuevo Usuario, ya que el administrador del departamento deberá ser el primer usuario en registrarse del modo que se va a demostrar, caso contrario quedará únicamente como usuario común.

c. Ya en la ventana de registro de un nuevo usuario, se deberá ingresar un nombre de usuario seguido de la palabra ADMIN, como, el ejemplo de la Figura DD.15.

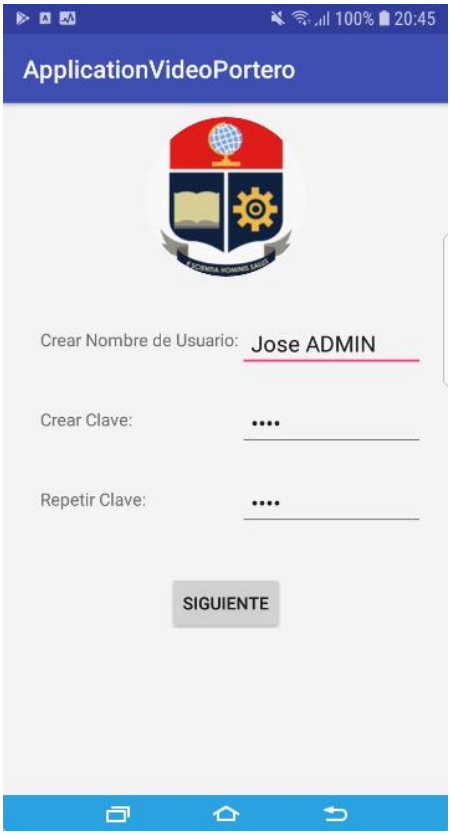

Figura DD.15. Configuración de un nuevo usuario como administrador del departamento.

Solo el primer usuario que se registre de este modo quedará como administrador del departamento, y en las futuras conexiones no tendrá que añadir la palabra ADMIN para tener acceso.

Después de presionar **SIGUIENTE** se almacenará la información en la Base de Datos y podrá ocupar el Video Portero.

2.3. Usuario regular

En el caso de querer acceder como un usuario regular al video portero únicamente deberá tener la información del departamento proporcionada por el administrador general o el administrador del departamento, para que después de esto ingrese su nombre y clave de usuario, y culminar el proceso.

Vale recalcar que el nombre de usuario usado deberá ser único, y no permitirá que se ocupe un nombre de usuario similar a otro ya registrado en cualquiera del resto de departamentos.

3. Configuración de base de datos

Para poder ingresar a una de las ventanas de configuración el usuario primero tendrá que acceder a la ventana principal del Video Portero, esta se puede acceder mediante uno de los procesos anteriores o directamente presiono el switch que permitirá almacenar la clave. De cualquier forma, después de ingresar a esta ventana que se puede observar en la Figura DD.16 será necesario presionar el botón de configuración que se encuentra en la esquina derecha inferior.

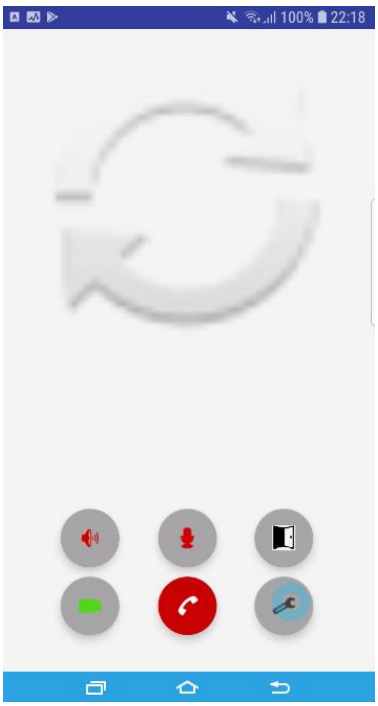

Figura DD.16. Ventana principal de comunicación del Video Portero.

Una vez presionado este botón de configuración se abrirá una nueva ventana de confirmación de usuario donde tendrá que ingresar la clave, aunque se encuentre el Switch de almacenar la clave encendida como se muestra en la Figura DD.17.

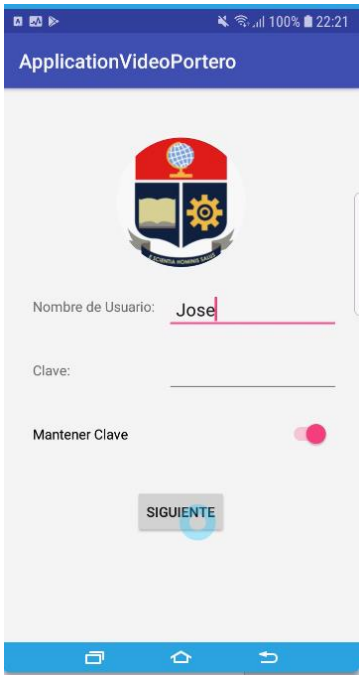

Figura DD.17. Ventana de autentificación de usuario.

Ya ingresada esta clave se presionará en el botón **SIGUIENTE** que desde este punto variará, dependiendo del tipo de usuario que haya ingresado. Por lo que se verá a continuación las tres opciones.

3.1. Administrador general

Una vez presionado el botón **SIGUIENTE** como se mencionó anteriormente, al ser administrador general se abrirá una ventana parecida a que se encuentra en la Figura DD.18.

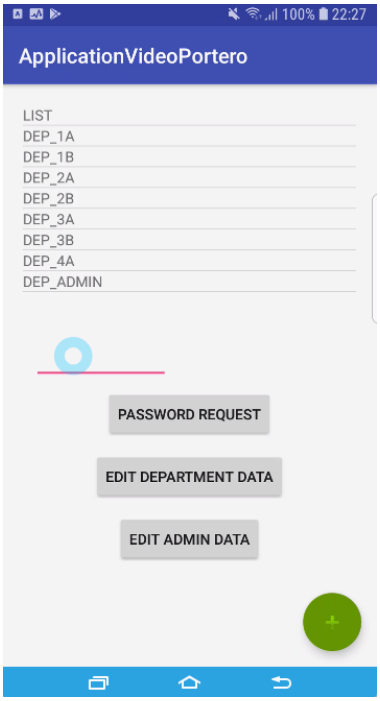

Figura DD.18. Ventana de configuración para Administradores Generales.

Desde esta ventana se podrá ingresar a otra ventana para añadir más departamentos, revisar la calve de un departamento, ingresar a editar un departamento específico y editar los datos de usuario del administrador. Acciones que se podrán ver a continuación.

a. Añadir un nuevo departamento

Para añadir un nuevo departamento primero se deberá presionar el botón con el signo de + que se encuentra en la parte inferior derecha como se puede observar en la Figura DD.18 esto permitirá acceder a una nueva ventana donde se podrá ingresar el nombre del departamento y la clave de este como se observa en la Figura DD.19.

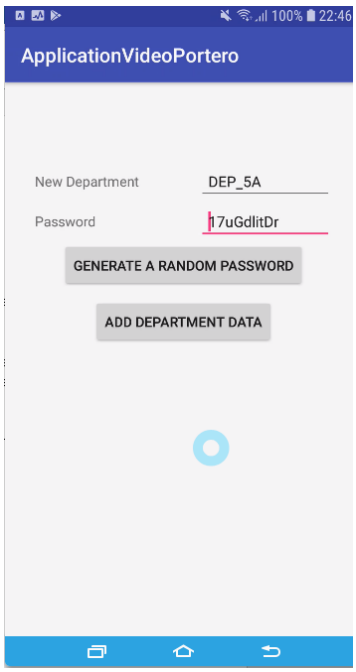

Figura DD.19. Ventana para agregar departamentos a la base de datos.

Los únicos requisitos que se tendrá que cumplir para el correcto funcionamiento de este es que deberá iniciar con las letras en mayúscula DEP seguido de un guión bajo y este seguido del número de departamento, con la posibilidad de poder incluir las letras mayúsculas A o B. como se puede observar en el ejemplo de la Figura DD.19.

Después de esto se podrá agregar una clave aleatoria o la que desee el administrador, este número de departamento y clave los tendrá que dar a los residentes de este departamento para que se puedan registrar como usuarios. Y finalizando se tendrá que presionar el botón **ADD DEPARTMENT DATA** para almacenar este en la base de datos.

b. Revisar la clave de un departamento

Para poder revisar la clave de un departamento y de este modo poder darles esta información a los usuarios del departamento es necesario presionar en la lista de departamentos el que se desee obtener la clave.

Esto permitirá que el nombre del departamento se coloque en el editor de texto inferior a la lista. Una vez se tenga el número de departamento en este lugar solo será necesario presionar el botón **PASSWORD REQUEST** y la clave se mostrará al lado del número de departamento como se puede observar en la Figura DD.20.

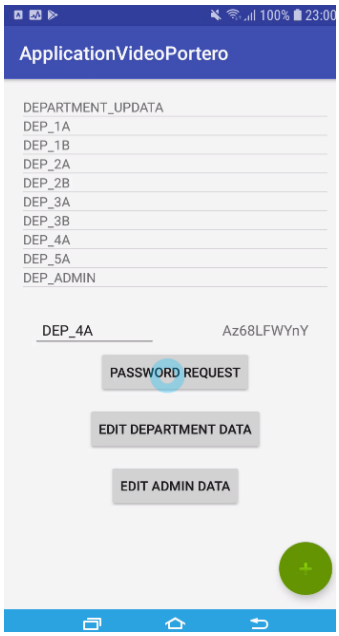

Figura DD.20. Solicitud de la clave de un departamento

c. Editar un departamento

Por otro lado, para editar un departamento determinado de deberá presionar en la lista el departamento deseado, y a continuación de este presionar el botón **EDIT DEPARTMENT DATA**, lo que dará acceso a otra ventana que desplegará en una lista con los nombres los usuarios registrados en este departamento como se puede observar en la Figura DD.21.

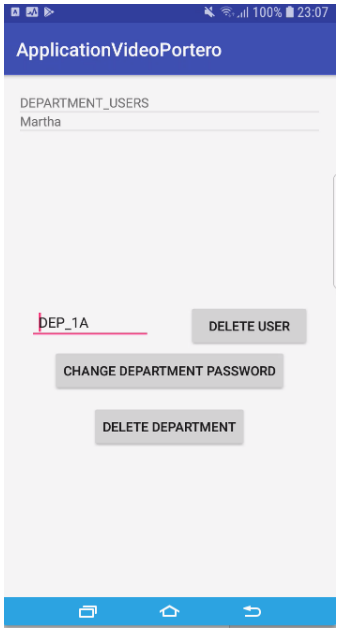

Figura DD.21. Ventana para editar datos de un departamento.

Desde donde se podrá eliminar a un usuario indicando el nombre del usuario y presionando el botón **DELETE USER.**

Cambiar la clave del departamento presionando el botón **CHANGE DEPARTMENT PASSWORD** que mostrará una línea adicional con un texto alfanumérico sugerido como clave como se puede observar en la Figura DD.22., pero del mismo modo el usuario podrá cambiar este y seleccionando nuevamente el botón **CHANGE DEPARTMENT PASSWORD** se cambiará esta clave en la base de datos.

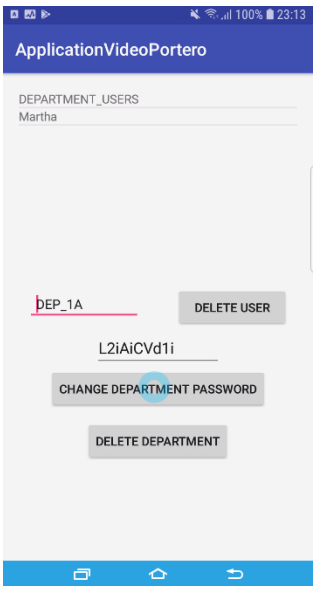

Figura DD.22. Configuración de la clave de un departamento.

d. Editar cambiar los datos de usuario del administrador

Por último, el poder editar la información del mismo administrador se lo realiza presionando el botón **EDIT ADMIN DATA,** que abrirá una nueva ventana donde se podrá ingresar el nuevo nombre de usuario y la nueva clave como se puede observar en la Figura DD.23.

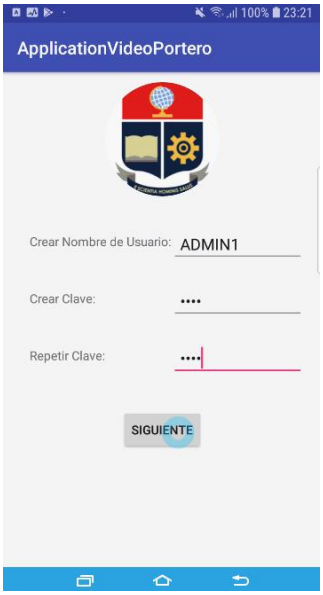

Figura DD.23. Ventana para ingresar un nuevo nombre de usuario y calve.

3.2. Administrador de departamento

En este caso se ingresará a una ventana que presentará la lista de usuarios del departamento en el que se encuentra registrado el administrador como se puede observar en la Figura DD.24.

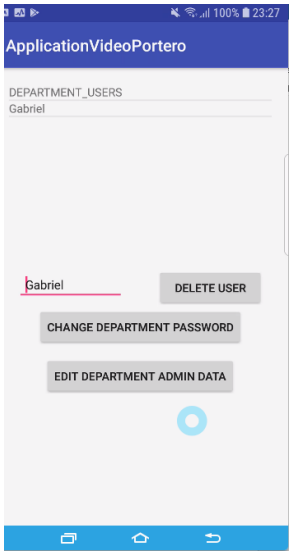

Figura DD.24. Ventana de configuración para administrador de departamento.

Desde esta ventana se podrá eliminar usuarios, cambiar la clave del departamento o editar los datos de usuario del administrador.

a. Eliminar usuarios

Para esto es necesario elegir el nombre del usuario que se desea eliminar, y presionar el botón DELETE USER. Lo que eliminará este nombre de la base de datos.

b. Cambiar la clave de departamento

Para esto es necesario presionar el botón **CHANGE DEPARTMENT PASSWORD** lo que presentará un nuevo editor de texto en el que se encuentra un texto alfanumérico como se observa en la Figura DD.25., que podrá ser editado colocando la clave deseada, y presionando nuevamente el botón **CHANGE DEPARTMENT PASSWORD** se almacenará esta nueva clave.

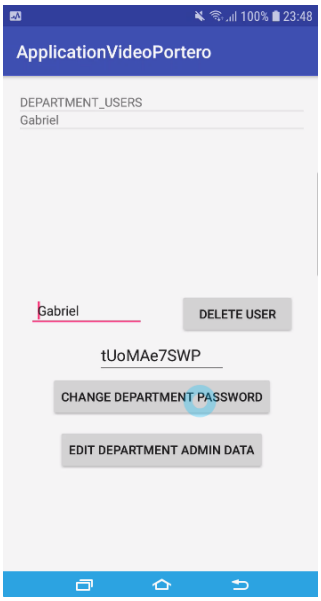

Figura DD.25. Ventana configurando la clave de departamento.

c. Editar los datos de usuario del administrador

Al igual que en administrador general, se puede editar la información del mismo administrador presionando el botón **EDIT DEPARTMENT ADMIN DATA,** que abrirá una nueva ventana donde se podrá ingresar el nuevo nombre de usuario y la nueva clave como se puede observar en la Figura DD.23.

3.3. Usuario regular

En este caso una vez autentificado y dado permiso de ingresar a la ventana de configuración, este usuario irá directo a la ventana donde podrá modificar su nombre y clave de usuario como se puede ver en la Figura DD.23.

4. Comunicación de audio y video con el Video Portero.

Una vez se ingresa a la ventana principal de comunicación, se podrá observar el video transmitido por el Video Portero, así como empezar la comunicación de voz. Esta parte del manual indicará el uso de cada uno de los botones que se ven en la Figura DD.16.

Permitirá inicializar la llamada si es la primera vez que se presiona este botón, caso contrario si el botón se encuentra en verde, al presionarlo nuevamente colocará la comunicación en espera, colocando nuevamente en rojo este botón. Y al presionarlo otra vez se reanudará la comunicación.

Permite silenciar o activar el micrófono para que la persona del lado del Video Portero pueda o no escuchar al usuario de la aplicación de acuerdo a lo que se desee. Si se encuentra el botón color rojo se encontrará el micrófono en silencio, mientras que si es de color verde significa que el micrófono está activo.

Permite activar o desactivar la cámara, el estado predeterminado de este será activo.

Permite culminar con la comunicación, saliendo de esta ventana y regresando a la ventana inicial.

Permite abrir el seguro electromecánico de la puerta.

Ingresa a la configuración de la base de datos como se mencionó anteriormente.

5. Diagrama de conexión del seguro electromecánico.

Para conectar el seguro electromecánico de la puerta exterior únicamente se tendrá que realizar la siguiente conexión que se observa en la Figura DD.26.

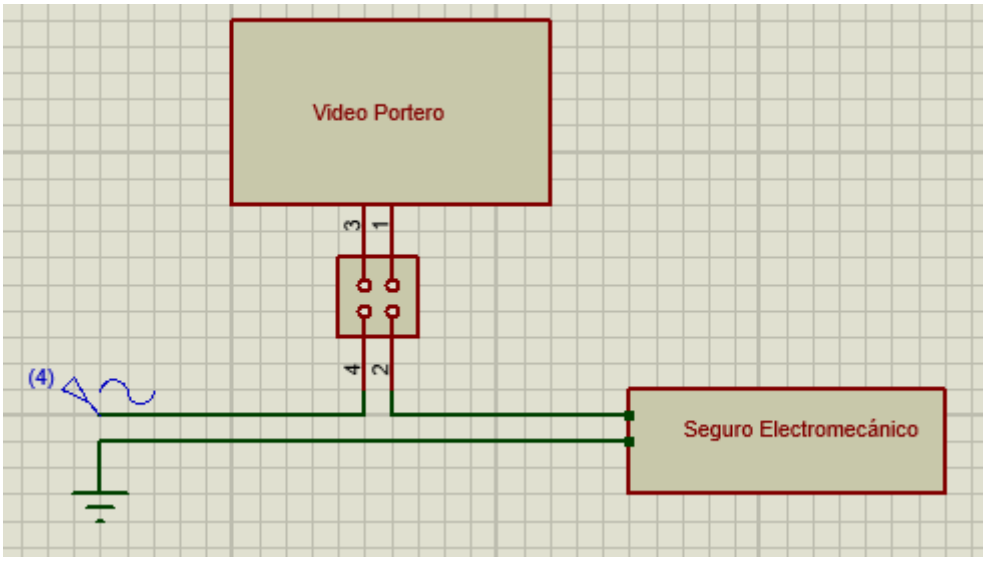

Figura DD.26. Diagrama de conexión de Seguro electromecánico a Video Portero.

ORDEN DE EMPASTADO