



Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

LAURA RICO ÁLVAREZ

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

## **DESARROLLO DE SERVICIO SIP PARA VIDEOPORTERO**

Memoria

ENERO 2019



# Índice general

<b>1. Introducción</b> .....	5
<b>1.1. DESCRIPCIÓN DEL PROYECTO</b> .....	5
<b>1.2. OBJETIVOS Y ALCANCE</b> .....	5
<b>1.3. ANTECEDENTES</b> .....	5
Figura 1. 1.- Pantalla PPL10-G, producto en catálogo de INGENIUM S.L. [1].....	6
<b>1.4. DOCUMENTOS Y CONTENIDO DEL PROYECTO</b> .....	6
<b>2. Protocolo de inicio de sesión</b> .....	8
<b>2.1. TRANSPORTE</b> .....	8
<b>2.2. PROTOCOLO DE TRANSMISIÓN EN TIEMPO REAL</b> .....	9
<b>2.3. FUNCIONAMIENTO</b> .....	10
<b>2.4. MENSAJES</b> .....	10
<b>2.4.1. Formato</b> .....	11
<b>2.5. MENSAJERÍA INSTANTÁNEA</b> .....	12
<b>3. Sistema BUSing®</b> .....	13
<b>3.1. PROTOCOLO BUSING®</b> .....	13
<b>3.2. EQUIPOS</b> .....	15
<b>3.2.1. Actuadores</b> .....	15
<b>3.2.2. Reguladores</b> .....	15
<b>3.2.3. Climatización</b> .....	15
<b>3.2.4. Sensores</b> .....	15
<b>3.2.5. Interfaces gráficas</b> .....	15
<b>4. Programación</b> .....	17
<b>4.1. PROGRAMACIÓN ORIENTADA A OBJETOS</b> .....	18
<b>4.1.1. Mecanismos básicos</b> .....	19
<b>4.1.2. Características</b> .....	19
<b>4.2. LENGUAJE JAVA</b> .....	21
<b>4.2.1. Compilador</b> .....	22
<b>4.2.2. Kit de desarrolloJDK</b> .....	22
<b>4.2.3. Java Native Interface (JNI)</b> .....	23
<b>4.3. ANDROID</b> .....	24

4.3.1.	Arquitectura .....	24
4.3.2.	NDK.....	25
4.3.3.	Entornos de desarrollo.....	26
4.4.	LIBRERÍAS SIP.....	26
4.4.1.	Stack SIP de Android.....	27
4.4.2.	PJSIP .....	27
5.	Desarrollo de servicio SIP según la librería Android .....	28
5.1.	ENTORNO RAD STUDIO.....	29
5.2.	ENTORNO ANDROID STUDIO .....	33
6.	Desarrollo de servicio SIP según la librería PJSIP .....	35
6.1.	ENTORNO RAD STUDIO.....	36
6.2.	ENTORNO ANDROID STUDIO .....	36
6.2.1.	Endpoint.....	36
6.2.2.	Account.....	37
6.2.3.	Media .....	37
6.2.4.	Call.....	37
6.2.5.	Buddy .....	38
6.2.6.	Video.....	38
7.	Uso de servidores.....	39
7.1.	SERVIDORES GENÉRICOS.....	39
7.1.1.	Asterisk-FreePBX.....	39
7.1.2.	Elastix.....	41
7.2.	SERVIDOR IP VARIO .....	42
8.	Aplicación BUSing para comunicación.....	45
8.1.	COMUNICACIÓN.....	46
8.2.	ESPECIFICACIONES .....	46
8.3.	APLICACIÓN.....	47
8.4.	IMPLEMENTACIÓN FINAL .....	48
9.	Conclusiones .....	51
10.	Planificación.....	53
11.	Referencias.....	55

# Lista de figuras

Figura 1. 1.- Pantalla PPL10-G, producto en catálogo de INGENIUM S.L. [1].....	6
Figura 2. 1.- Diseño de modelo SIP. ....	8
Figura 2. 2.- Ejemplo de envío de mensajes SIP. [2] .....	11
Figura 3. 1.- Modelo de telegrama BUSing®. [3].....	14
Figura 4. 1.- Tipos de lenguaje de programación.....	17
Figura 4. 2.- Esquema de OOP.....	18
Figura 4. 3.- Ejemplo de una programación OOP con herencia. [4] .....	21
Figura 4. 4.- Arquitectura de Android. [7] .....	24
Figura 5. 1.- Ejemplo instanciación SipManager y SipProfile en Java. ....	29
Figura 5. 2.- Ejemplo de la “traducción” de instanciación de SipManager y SipProfile en C++.....	30
Figura 5. 3.- Instanciación válida de SipProfile en C++. ....	30
Figura 5. 4.- Instanciación correcta de SipManager en C++. ....	31
Figura 5. 5.- Código de error al registrar el usuario. ....	33
Figura 6. 1.- Diseño de la aplicación.....	38
Figura 7. 1.- Control de registros en Asterisk-FreePBX. ....	40
Figura 7. 2.- Tablero de Asterisk-PBX.....	40
Figura 7. 3.- Control de registros de Elastix.....	41
Figura 7. 4.- Tablero de conexiones y llamadas de Elastix. ....	41
Figura 7. 5.- Intercomunicados 2N IP Vario. ....	42
Figura 7. 6.- Pantalla principal del intercomunicador. ....	43
Figura 7. 7.- Tabla de eventos del intercomunicador. ....	44
Figura 8. 1.- Esquema de conexión BUSing. ....	45
Figura 8. 2.- Pantalla de conexión BUSing®. ....	47
Figura 8. 3.- Pantalla de lectura y escritura del bus.....	48
Figura 8. 4.- Pantalla principal de la aplicación. ....	49
Figura 8. 5.-Pantalla del menú de la aplicación.....	49
Figura 8. 6.- Pantalla de llamada. ....	50
Figura 8. 7.- Panel de pruebas. ....	50

# 1. Introducción

## 1.1. DESCRIPCIÓN DEL PROYECTO

<b>Título</b>	Desarrollo de servicio SIP para videoportero
<b>Tutor:</b>	Antonio Robles Álvarez
<b>Autor:</b>	Laura Rico Álvarez
<b>Titulación:</b>	Máster en Automatización e Informática Industrial
<b>Fecha:</b>	Enero de 2019
<b>Financiación:</b>	INGENIUM, Ingeniería y Domótica, S.L.

## 1.2. OBJETIVOS Y ALCANCE

El objetivo de la realización de este trabajo será el desarrollo de un servicio SIP para la recepción de llamadas enviadas a través de un videoportero. Dicho servicio será un extra dentro de las pantallas Android usadas como interfaces de usuario dentro del conjunto de dispositivos que se encuentran a la venta en INGENIUM S.L.

Para ello será necesario realizar primeramente el estudio del protocolo sobre el cual se va a implementar el servicio, así como la programación de la aplicación que va a darle soporte. Además, se testeará la aplicación en servidores SIP genéricos y en un servidor específico que contará con las funcionalidades del videoportero.

También se programará una aplicación de prueba para efectuar el control sobre una instalación BUSing® que simulará el efecto de las aplicaciones existentes en las pantallas mencionadas anteriormente.

Por último, se combinarán ambas aplicaciones y se llevarán a cabo diferentes casos simulando el uso de estas en la vida real.

## 1.3. ANTECEDENTES

Partiendo de lo mencionado en el apartado anterior, se tiene que este proyecto surge en colaboración con la empresa INGENIUM S.L, mediante la realización de la asignatura de prácticas del máster y que en él se desarrollará una nueva aplicación.

La empresa dispone de unas pantallas que sirven de interfaz a través de las cuales los usuarios pueden manejar y controlar los diferentes dispositivos que tienen en su instalación domótica. Como breve apunte, se definirá domótica como el conjunto de sistemas destinados a la automatización de la vivienda, los cuales se basan en diferentes tecnologías y arquitecturas.

Por lo tanto, a través de dichas pantallas los usuarios pueden encender o apagar la luz, regularla o hasta incluso programar escenas para que estas sigan unos procesos determinados, y al igual que con las luces, se pueden gobernar persianas, puertas... Todo equipo que se tenga conectado al sistema.

Pero ¿qué pasa en este caso con los videoporteros? Hasta el momento INGENIUM S.L. no cuenta con la incorporación del manejo del videoportero a través de estas interfaces por lo que se incluirá dicho servicio a través del protocolo SIP, de manera que la recepción, en este caso de audio y video, llegue directamente a la pantalla que cuenta con todas las instalaciones disponibles.



Figura 1. 1.- Pantalla PPL10-G, producto en catálogo de INGENIUM S.L. [1]

#### 1.4. DOCUMENTOS Y CONTENIDO DEL PROYECTO

El presente proyecto consta de los siguientes documentos:

- Memoria: Documento presente, en el cual se detallarán los estudios, análisis y desarrollo realizados para llevar a cabo este proyecto, así como las conclusiones y la planificación de este.

- Presupuesto: En él se detallará el coste total del proyecto, así como el coste de cada una de las partes.
- Anexos: En los cuales se incluirá el código fuente de las aplicaciones.
- Manual de Usuario: Documento que servirá como ayuda final para el cliente.
- Manual de Programador: Este deberá contener la información suficiente para que una persona ajena al proyecto puede modificar el código, en caso de ser necesario, para adaptarlo.
- Pruebas: Documento que registra alguna de las pruebas realizadas a la aplicación para verificar el correcto funcionamiento de todas sus funciones.



## 2. Protocolo de inicio de sesión

El protocolo de inicio de sesión o SIP, por sus siglas en inglés (Session Initiation Protocol), es un protocolo de señalización para iniciación, modificación y terminación de sesiones de comunicación multimedia. Entre las capacidades de este protocolo se encuentran, por ejemplo, la bidireccionalidad y localización del usuario, aunque como se ha mencionado anteriormente es para la comunicación multimedia, no permite el transporte de información de esta, sino que requiere de otros protocolos para ello.

Para que dicha transmisión sea posible, primeramente, se necesita una arquitectura de tipo P2P (peer-to-peer) o red de pares, que es una red de ordenadores en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí. Actúan simultáneamente como clientes y servidores respecto a los demás nodos de la red, únicamente es necesario conocer la localización del usuario y mediante SIP se realiza el encaminamiento. [2]

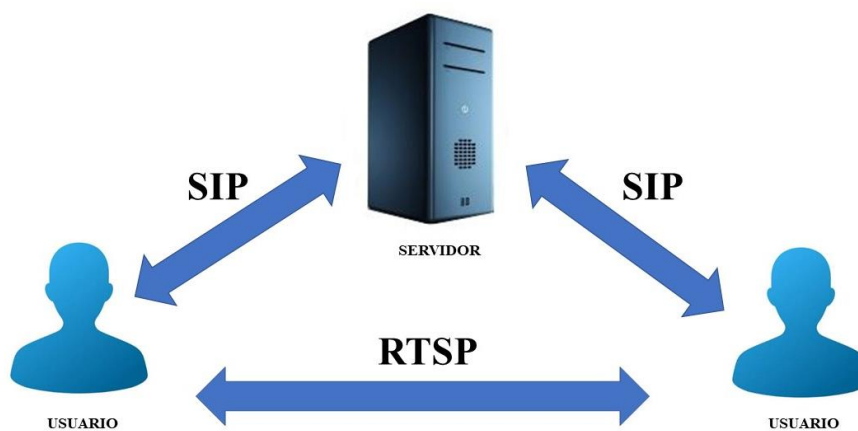


Figura 2. 1.- Diseño de modelo SIP.

### 2.1. TRANSPORTE

En el subapartado anterior se ha mencionado muchas veces la palabra transporte, pero ¿cómo se realiza? Pues bien, un mensaje SIP puede ser transportado utilizando TCP (Protocolo de control de transmisión) o UDP (Protocolo de datagramas de usuario) mayoritariamente, aunque también existe la posibilidad de utilizar otros protocolos como

TLS (Seguridad de la capa de transporte), SCTP (Protocolo de control de transmisión) ... Incluso un mensaje con múltiples saltos podría combinar diferentes tipos.

El TCP es uno de los protocolos fundamentales, ya que permite crear conexiones entre sí a través de las cuales puede enviarse un flujo de información. Este protocolo garantiza que no habrá errores al entregar los datos en el destino y que además estos se entregarán en el mismo orden en el que se transmitieron. Si es el caso de que SIP utiliza dicho protocolo, la conexión no tiene por qué permanecer activa durante toda la sesión, podría cerrarse y volverse a abrir cuando hiciese falta.

En cambio, UDP permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera. A diferencia del protocolo anterior, este no proporciona información ni control de flujo, de tal forma que los paquetes pueden llegar desordenados y además nunca se confirma su recepción por lo que es imposible saber si el paquete ha sido entregado. En el caso de que sea este el protocolo utilizado cada mensaje debe ser enviado en un datagrama, no se pueden cortar ni agrupar mensajes.

## **2.2. PROTOCOLO DE TRANSMISIÓN EN TIEMPO REAL**

Como se ha mencionado anteriormente, el protocolo SIP permite la transmisión de información multimedia, aunque necesita de otro protocolo para que esto sea posible. Dicho protocolo es el de transmisión en tiempo real o RTSP (Real Time Streaming Protocol), el cual establece y controla uno o muchos flujos sincronizados de datos, ya sean de audio o video, como pasa en este caso.

Es un protocolo que no está orientado a la conexión, ya que es el servidor el que mantiene la conexión de la sesión y asocia a esta un identificador, por lo tanto, cuando se envían datagramas con dicho identificador el servidor sabrá hacia donde redirigirlos. En la mayoría de los casos usa TCP para datos de control del reproductor y UDP para los datos de audio y vídeo.

En el transcurso de una sesión, un cliente puede abrir y cerrar varias conexiones de transporte hacia el servidor con tal de satisfacer las necesidades del protocolo. Además, tanto el servidor como el cliente pueden lanzar peticiones.

### 2.3. FUNCIONAMIENTO

Dentro del SIP existen diferentes funciones que pueden llevarse a cabo dependiendo del papel que se desempeñe en la conexión:

- **User-agent (Agentes de usuario):** Es el dispositivo final de una red SIP y origina las solicitudes de establecimiento de una sesión multimedia. Pueden comportarse como clientes, UAC, o como servidores, UAS. En el momento en el que están realizando una petición son clientes y en el momento que la reciben son servidores
- **Registrar (Servidores de registro):** Elemento puramente centrado en la señalización, y por ello no transmite nada multimedia, es un equipo de apoyo entre los dispositivos de llamada. Realiza una asociación entre la dirección física del usuario, su dirección IP, y la dirección lógica, invariante respecto de la ubicación física del usuario, la cual es de la forma usuario@dominio.
- **Servidor proxy:** Es el encargado de encaminar hacia otro sitio la petición que ha recibido. Hay dos tipos que son Inbound proxy o proxy de entrada y Outbound proxy o proxy de salida, el primero se encarga de la localización y el segundo se encarga de temas de facturación y permite filtrar llamadas.
- **Redirect server (Servidor de redirección):** Recibe peticiones de un user-agent o de un proxy y devuelve la petición indicando dónde debe dirigirse.

### 2.4. MENSAJES

Dentro de una llamada SIP existen dos tipos de mensajes, los de requerimiento, enviados como métodos SIP, y los de estado, emitidos con un indicador de estado numérico. Se mencionarán solamente los más importantes.

En cuanto a los de requerimiento, se tienen, por ejemplo:

- **REGISTER:** Utilizado para registrar o desregistrar un cliente en un Registrar.
- **INVITE:** Inicia un diálogo de sesión.
- **ACK:** Es utilizado para responder a un mensaje de estado de SIP mientras que se encuentra dentro de un diálogo INVITE.
- **BYE:** Finaliza una sesión previamente establecida.
- **CANCEL:** Cancela un requerimiento antes de que se haya atendido

Para el caso de los de estado:

- 100 Trying: Indica el estado temporal de que está intentando enviar la petición.
- 180 Ringing: Estado temporal que indica que el teléfono está sonando.
- 200 OK: Indica un estado final exitoso.
- 500 Server Internal Error: Error interno del servidor de la central telefónica.

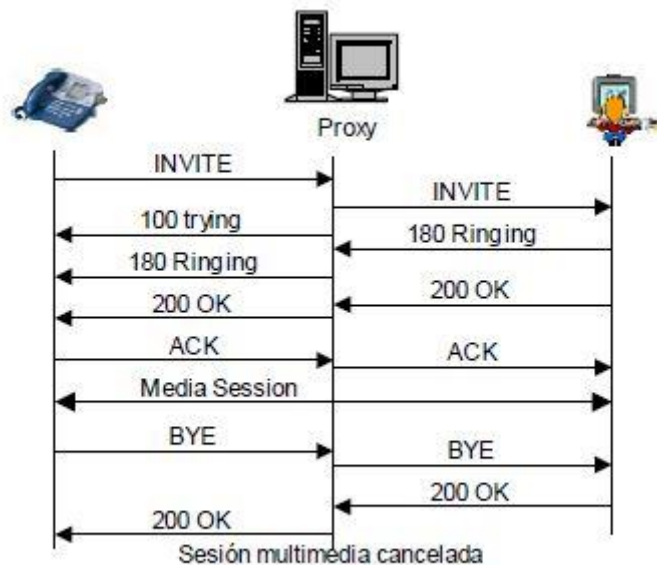


Figura 2. 2.- Ejemplo de envío de mensajes SIP. [2]

### 2.4.1. Formato

Los mensajes mencionados contienen unas cabeceras, muy parecidas a las que tienen los mensajes del correo electrónico, debido a que es necesario tener identificadores ya que de lo contrario habría caos.

Esas cabeceras han de contener los identificadores, tanto el de destinatario para saber a quién va dirigido el mensaje, como el del creador para saber con quién se va a comunicar. Además de esos dos, un identificador más, que será el de la sesión que se está realizando en ese momento.

También se necesita que el mensaje vaya acompañado de un número de secuencia en el que se determinará el orden de las transiciones de cada método.

Otros de los campos que contiene un mensaje son el número de saltos máximo que puede realizar la petición en su camino al destinatario y los puntos por los que ha pasado dicha petición hasta ser entregada.

Como extra se añade una dirección en la cual se localizará al llamante en futuras llamadas y un resumen en el que indica la naturaleza de la llamada.

## **2.5. MENSAJERÍA INSTANTÁNEA**

El SIP también permite el envío de mensajes de manera instantánea, el cual puede enviarse en cualquier momento sin que haya una sesión establecida, aunque este caso no estará contemplado en el servicio a realizar.

## 3. Sistema BUSing®

BUSing® es el primer sistema de comunicaciones distribuido diseñado íntegramente para las aplicaciones domóticas e inmóticas, desarrollado completamente por la empresa INGENIUM.

La capacidad de control de este sistema abarca desde las típicas aplicaciones domóticas de iluminación o gestión de toldos y persiana hasta el control de cámaras IP o sistemas de seguridad. [3]

### 3.1. PROTOCOLO BUSING®

BUSing® es un sistema distribuido, donde cada uno de los dispositivos conectados tiene autonomía propia, debido a que son equipos microprocesados, y alcanza sentido global por su pertenencia a un sistema que lo abarca y le hace formar parte de un todo. Los microprocesadores que presenta cada equipo permiten el envío y recepción de datos posibilitando que todos los dispositivos del BUS al que pertenecen actúen simultáneamente como maestros y esclavos.

El BUS es un bus independiente de 4 hilos, de los cuales 2 se utilizan como medio de transmisión de la información de control y los otros 2 para la alimentación de los dispositivos, aportando una gran fiabilidad, robustez y seguridad. Dicha información de control se envía a través de telegramas BUSing® que sirven como código para que ambas partes se entiendan.

Es necesario garantizar un intercambio ordenado de información entre los componentes del BUS, por lo que el tráfico de telegramas y el acceso al BUS deben estar organizados, de manera que los paquetes individuales de información se envían por la línea serie uno tras otro, lo que quiere decir que, en el bus sólo puede haber información de un solo dispositivo en cada momento.

Para asegurar la fiabilidad del sistema, se utiliza un acceso al bus descentralizado, de modo que cada componente decide cómo y cuándo accede al bus. Si dos componentes deciden acceder al bus al mismo tiempo, existe un mecanismo de acceso especial que asegura que no se perderá ninguna información y que el bus estará operativo constantemente.

El intercambio de información sucede de forma controlada, solamente cuando ocurre un evento, es decir, cuando se da un cambio de estado en una variable y por lo tanto necesita enviar dicha información para notificarlo y que se actúe en consecuencia.

Como se ha dicho anteriormente toda esta información viaja a través de telegramas, pues bien, un telegrama consta de una serie de caracteres, los cuales llevan asociada información diversa, y estos se agrupan en distintos campos.

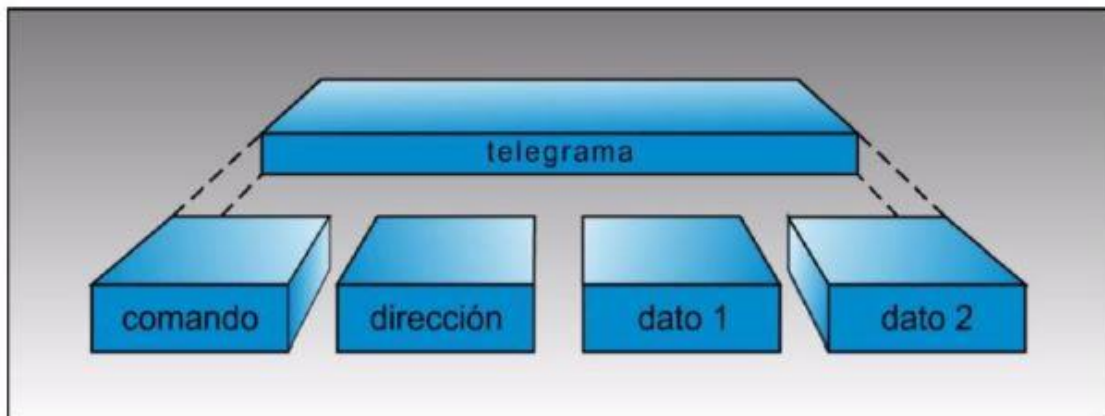


Figura 3. 1.- Modelo de telegrama BUSing®. [3]

- Campo de direcciones:
  - Dirección origen: Identificador del nodo que transmite el telegrama conformado por el resto de los campos de que se compone un telegrama.
  - Dirección destino: Identificador del nodo al cual se transmitirá el telegrama.
- Campo de comandos: Determinan la acción a ejecutar definida en el campo de datos, bien sea de lectura o de escritura.
- Campo de datos: Consta de dos bloques para la definición de acciones a ejecutar según el dispositivo al que va dirigido el telegrama. Estos son los llamados Dato 1 y Dato 2.

## **3.2. EQUIPOS**

Para llevar a cabo el control de una instalación primero es necesario conocer los principales dispositivos que pueden formar parte de ella.

### **3.2.1. Actuadores**

Estos equipos son utilizados para controlar todo tipo de dispositivo que necesite conexión a la red eléctrica. Presentan entradas configurables para el control de cada una de las salidas o incluso para el control de cualquier parte de la instalación. Además, cuentan con dos modos de programación según el uso que se le va a dar.

### **3.2.2. Reguladores**

A través de estos equipos es posible controlar de manera proporcional algún parámetro de la instalación, como por ejemplo la intensidad luminosa.

### **3.2.3. Climatización**

En estos dispositivos se incluyen sondas de temperatura que permiten el control eficiente de las calderas o de los aires acondicionados y pasarelas a otros sistemas de climas más sofisticados.

### **3.2.4. Sensores**

Un sensor sirve para captar cambios en el entorno que puedan requerir una acción del sistema domótico, como por ejemplo detectores de incendio o inundación.

### **3.2.5. Interfaces gráficas**

Las interfaces gráficas aportan gran funcionalidad ya que permiten el control de una gran parte de la instalación desde un solo punto y ayudan a aprovechar más la instalación permitiendo el control a través de móviles o tabletas.

Dicho elemento es altamente importante en este proyecto, ya que el servicio SIP a desarrollar será posteriormente implementado en una interfaz gráfica que desarrollará la empresa para el soporte de este.



En el caso de la instalación que se controlará con la aplicación a programar, contará con pocos elementos, un par de actuadores para encender y apagar un par de bombillas, otro de estos para la activación o desactivación de una persiana y un regulador para manejar la intensidad lumínica de una bombilla más potente que las anteriores.

Como interfaz gráfica se utilizará, en este caso, un smartphone con sistema Android en el que estará corriendo constantemente la aplicación diseñada.

## 4. Programación

La programación se encuentra muy asociada a la creación de aplicaciones informáticas, en las cuales una persona desarrolla un programa a través de una herramienta que le permite escribir el código, en cualquiera de los lenguajes existentes de programación, y de otra que es capaz de traducirlo a lenguaje que pueda entender un microprocesador.

Debido a que un programa no puede estar hecho al azar, se establecen unos factores que son capaces de determinar la calidad de este:

- **Correctitud:** En las fases previas al desarrollo de un programa, se establecen una serie de funcionalidades y sólo será correcto si al finalizarlo cumple con todas ellas.
- **Claridad:** Ha de ser lo más claro y legible posible para facilitar tanto su desarrollo como su posterior mantenimiento.
- **Eficiencia:** Además de realizar aquello para lo que fue creado, ha de hacerlo gestionando de la mejor forma posible los recursos que utiliza.
- **Portabilidad:** Un programa es portable cuando tiene la capacidad de poder ejecutarse en una plataforma, ya sea hardware o software, diferente a aquella en la que se desarrolló.



Figura 4. 1.- Tipos de lenguaje de programación.

A la vista de la Figura 4.1, se puede observar que existen 3 tipos de lenguaje en la programación que son el lenguaje máquina, basado en la utilización del código binario, el lenguaje de bajo nivel que depende mucho de la máquina y el lenguaje de alto nivel que usa palabras o comandos del lenguaje natural, por eso mismo, este último es el más utilizado y en el cual se centrará este desarrollo.

Dentro del lenguaje de alto nivel existe también otra clasificación de tipos de lenguaje, pero se hablará directamente del que concierne a este proyecto, que es la programación orientada a objetos.

#### 4.1. PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos o OOP, según sus siglas en inglés, es un paradigma de programación. Esto quiere decir que proporciona una visión y unos métodos al programador a la hora de plantear la solución a un problema.

No hay que confundirla con la programación estructurada tradicional, en la cual los atributos y los métodos están separados y sin relación ninguna. En esta se busca solamente el procesamiento de los datos.



Figura 4. 2.- Esquema de OOP.

#### 4.1.1. Mecanismos básicos

Como se observa en la Figura 4.2, la OOP cuenta con una serie de mecanismos básicos en los que se encuentran:

- Clase: Modelo que define un conjunto de variables y métodos apropiados para operar con ciertos tipos de datos. Una clase requiere de métodos para poder tratar los atributos con los que cuenta y el programador debe pensar indistintamente en ambos conceptos sin separarlos ya que si esto ocurre se estarían creando clases con información por un lado y clases con métodos que manejan dicha información por el otro lado. Esto produciría el efecto de una programación orientada a objetos, pero en realidad se estaría realizando una programación estructurada. Cada objeto creado a partir de una clase se define instancia de la clase.
- Objeto: Es la instancia de una clase. Entidad que manipula los datos de entrada para obtener unos específicos de salida, ha de contener toda la información que permita definirlo e identificarlo frente a otros objetos. Posee diferentes atributos, que son los datos, y realiza diferentes funcionalidades a través de sus métodos.
- Método: Algoritmo asociado a un objeto o a una clase de objetos, cuya ejecución se desencadena tras la recepción de un mensaje. Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un evento con un nuevo mensaje para otro objeto del sistema.
- Atributos: Características que tiene la clase.
- Mensaje: Una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.
- Evento: Reacción que puede desencadenar un objeto.

#### 4.1.2. Características

Dentro de las características que posee la programación orientada a objetos se ha realizado una selección de las más importantes, en las que encontramos:

- **Abstracción:** Características específicas de un objeto que hacen que se distinga de los demás y que logran definir límites conceptuales. Es la capacidad que posee un objeto para realizar su trabajo, informar, cambiar de estado y comunicarse con otros objetos sin necesidad de revelar la implementación de sus características. La abstracción es la clase en el proceso de análisis y diseño orientado a objetos, ya que mediante ella se puede llegar a armar un conjunto de clases que permitan modelar la realidad o el problema que se quiere atacar.
- **Encapsulamiento:** Mecanismo que organiza datos y métodos en una estructura evitando el acceso a datos por cualquier otro medio distinto a los especificados. Aumenta la cohesión de los componentes del sistema.
- **Herencia:** Característica específica de la OOP. Es capaz de, a partir de una clase existente, crear una nueva que contiene los atributos y métodos de la clase primaria. Es decir, los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. Cuando un objeto hereda de más de una clase, se dice que hay herencia múltiple, aunque se suele recurrir a la herencia virtual debido a la complejidad de la anterior. Al hablar de herencia, surgen algunas definiciones para clasificar los métodos o atributos que se deben conocer.
  - **Público:** Un método o atributo es público cuando puede ser accedido desde fuera de la clase
  - **Privado:** En cambio, es privado si sólo pueden ser accedidos desde dentro de la misma clase. Una clase heredada los hereda, pero se mantienen escondidos.
  - **Protegido:** Para poder acceder a un atributo o método de una clase en cualquiera de sus subclases, pero mantenerla oculta para otras clases es necesario registrar los componentes como protegidos.

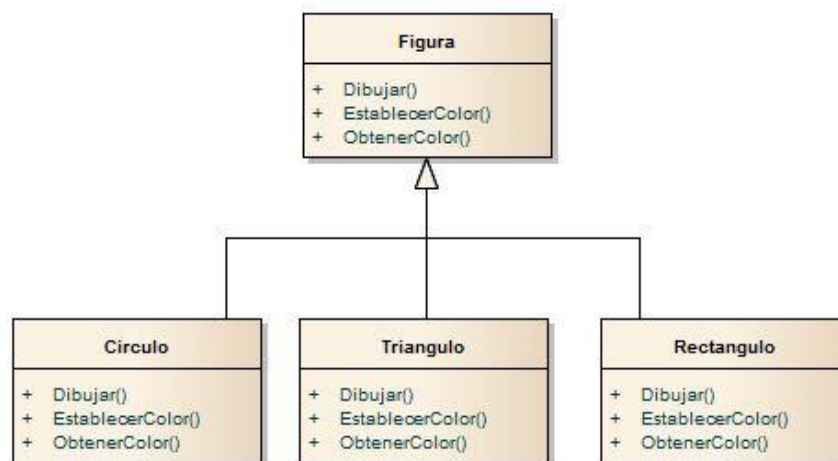


Figura 4. 3.- Ejemplo de una programación OOP con herencia. [4]

- Polimorfismo: Capacidad que tienen los objetos de una clase de comportarse frente a la recepción del mismo mensaje o evento en función de los parámetros utilizados durante su invocación.

## 4.2. LENGUAJE JAVA

Dentro de la lista de lenguajes más utilizados en el mundo, se pueden encontrar los siguientes lenguajes por orden de popularidad [5]:

- Java: Imprescindible para crear aplicaciones y usuario de la programación orientada a objetos.
- C++: El lenguaje más clásico y más universal de todos. Actualmente se utiliza cada vez más en los dispositivos IoT (Internet of things).
- Python: Utilizado para las nuevas tecnologías como la robótica, la realidad virtual y Bigdata.
- JavaScript: Lenguaje web.

Vistas las características de cada uno de ellos y enfocando el proyecto a realizar, se ve como opción más lógica la elección del lenguaje Java como el elegido para desarrollar las aplicaciones pertinentes. Por lo tanto, este apartado entrará más a fondo en las características de dicho lenguaje.

Al igual que cualquier lenguaje de programación, Java tiene su propia estructura, reglas de sintaxis y paradigma. Este último es el mencionado en el apartado anterior.

Aunque sea un lenguaje diferente, Java es un derivado del lenguaje C, por lo que sus reglas de sintaxis se parecen mucho a las de este. Estructuralmente, comienza con paquetes, que son el mecanismo de espacio de nombres. Dentro de estos se encuentran las clases que a su vez contienen los métodos y variables entre otros.

Java permite el término WORA, “*write once, run anywhere*”, que significa que una vez que el programa de la aplicación esté escrito podrá ejecutarse en cualquier dispositivo.

Además, este lenguaje evita en gran medida, gracias al recolector de basura que posee, el problema de fugas de memoria existente en otros. El programador sigue eligiendo cuándo se crean los objetos, pero es el entorno, en tiempo de ejecución, el responsable de gestionar el ciclo de vida de los objetos. De este modo, cuando no existe referencia ninguna a ese objeto, el recolector de basura lo borra, liberando así la memoria que ocupaba.

#### **4.2.1. Compilador**

El compilador de Java, al igual que los compiladores existentes para otros lenguajes, verifica que el código escrito cumpla las reglas de sintaxis, pero, además, tiene una funcionalidad extra que los demás no poseen. Una vez verificado el código, escribe los *códigos byte* en archivos .class, estos códigos son instrucciones estándar destinadas para ejecutarse en una Java Virtual Machine (JVM).

La JVM lee e interpreta los archivos y ejecuta las instrucciones del programa en la plataforma hardware nativa para la que se escribió. Lo que diferencia a esta de la CPU, es que es un software escrito específicamente para una plataforma particular, es lo que permite el WORA.

#### **4.2.2. Kit de desarrolloJDK**

Por un lado, se tiene el JDK, también llamada Kit de desarrollo de Java, que es un software que provee herramientas de desarrollo para la creación de programas en Java y que puede instalarse en una computadora local o en una unidad de red.

Entre esas herramientas se pueden encontrar el compilador y una librería de clase completa de programas de utilidad preconstruidos que lo ayudan a cumplir cualquier tarea común al desarrollo de aplicaciones.

Dicha librería, API, Application Programming Interface por sus siglas en inglés, es una interfaz de programación de aplicaciones provista de un conjunto de clases [6] utilitarias para efectuar cualquier tarea necesaria dentro de un programa. Además, está organizada en paquetes lógicos, donde cada paquete contiene un conjunto de clases relacionadas semánticamente.

Por otro lado, se encuentra el SDK, Kit de desarrollo software, paquete genérico que reúne un grupo de herramientas que permiten la programación de aplicaciones móviles con otros idiomas.

#### **4.2.3. Java Native Interface (JNI)**

Framework de programación que permite que un programa escrito en Java pueda interactuar con programas escritos en otros lenguajes como C++.

El JNI se usa para escribir métodos nativos que permitan solventar situaciones en las que una aplicación no puede ser enteramente escrita en Java. Modifica también programas existentes escritos en otro lenguaje, permitiéndole ser accesible desde aplicaciones Java.

Muchas de las clases de la API estándar de Java dependen del JNI para proporcionar funcionalidad al desarrollador y al usuario. El desarrollador debe asegurarse que la API estándar de Java no proporciona una determinada funcionalidad antes de recurrir al JNI, ya que la primera ofrece una implementación segura e independiente de la plataforma.

Permite a un método nativo utilizar los objetos Java de la misma forma en que el propio código de Java lo hace. Un método nativo puede crear objetos Java, examinarlos y utilizarlos para que lleven a cabo su función. Puede también examinar y utilizar objetos que han sido creados por código de aplicación escrito en Java.

Se utiliza a veces como “válvula de escape” ya que permite añadir funcionalidades a sus aplicaciones que el API de Java no puede proporcionar.



### 4.3. ANDROID

Android es un sistema operativo inicialmente pensado para teléfonos. Lo que lo hace diferente de los SO, como iOS por ejemplo, es que está basado en Linux, un núcleo de sistema operativo libre, gratuito y multiplataforma. Este sistema permite programar aplicaciones en una variación de Java. Una de las mejores características de este sistema operativo es que es completamente libre.

#### 4.3.1. Arquitectura



Figura 4. 4.- Arquitectura de Android. [7]

En la Figura 4.4 pueden verse los principales componentes de los que está compuesto el sistema Android:

- **Aplicaciones:** Todas las aplicaciones están escritas en lenguaje de programación Java.

- Marco de trabajo de aplicaciones: Todo el conjunto de funciones del SO Android está disponible mediante API escritas en el lenguaje Java.
- Bibliotecas: Se incluyen un conjunto de bibliotecas de C y C++ usadas por varios componentes del sistema. Si se desarrolla una app que requiera C o C++ se puede usar el NDK (Native Development Kit) de Android para acceder a algunas de estas bibliotecas de plataformas nativas directamente desde el código nativo.
- Runtime de Android: Tiene un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java.
- Capa de abstracción de hardware (HAL): Brinda interfaces estándares que exponen las capacidades de hardware del dispositivo al framework de la API de Java de nivel más alto. Consiste en varios módulos de biblioteca y cada uno de estos implementa una interfaz para un tipo específico de componente de hardware-
- Núcleo Linux: Depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores.

#### 4.3.2. NDK

Como se ha mencionado en el apartado anterior para usar código nativo en una aplicación Android existe el Kit de desarrollo nativo que se utiliza para realizar compilaciones a partir de propio código fuente o aprovechándose de las bibliotecas existentes previamente compiladas.

Este kit puede ser útil cuando se necesite un mayor rendimiento de un dispositivo para aplicaciones de gran exigencia en términos computacionales o el uso de bibliotecas programadas en lenguaje nativo.

### 4.3.3. Entornos de desarrollo

Existe más de un entorno de desarrollo a través del cual se puede programar una aplicación Android, aunque en este caso se mencionaran y explicarán solamente 2 ya que son los que se van a utilizar para el desarrollo de las aplicaciones en este caso.

- **Android Studio:** Es el entorno de desarrollo integrado oficial para esta plataforma. Además de ser un potente editor de códigos y poseer herramientas para los desarrolladores, ofrece más funciones que aumentan la productividad durante la compilación de las aplicaciones. Tiene la posibilidad de, a través del NDK, compilar aplicaciones en código nativo, incluyendo, por supuesto, las programadas en código Java.
- **RAD Studio [8]:** Compatible con código Delphi o C++ que permite el diseño de aplicaciones nativas multiplataforma. Es decir, programando en código nativo, es capaz de compilar bajo ese mismo código aplicaciones para Windows, Android y iOS, lo que supone una eficiencia en cuanto al tiempo de programación en el caso de que se quiera que una aplicación pueda ejecutarse en cualquier entorno.

## 4.4. LIBRERÍAS SIP

Una vez vista la manera en que ha de programarse la aplicación, se verá qué es lo que hay que programar exactamente o, mejor dicho, con qué hay que hacerlo.

Teniendo en cuenta lo mencionado en el capítulo 2, en el que se realizó el estudio del protocolo SIP se tiene que hay que realizar un servicio que sea capaz de proporcionar al usuario la posibilidad de registrarse en un servidor para que cada vez que el usuario realice o reciba una llamada el servidor sepa dirigir la llamada a dónde debe. Por otra parte, esa llamada debe llevar información multimedia de audio y video por lo que habrá configurar ciertos códecs.

Un códec es un programa o dispositivo hardware capaz de codificar o decodificar una señal o flujo de datos digitales. Su uso está muy extendido para la codificación de señales de audio y vídeo dentro de un formato contenedor.

Aun así, todo esto no ha de programarse tal cual, si no que se hará uso de alguna de las librerías existentes que cumplen los requisitos de una funcionalidad SIP.

#### 4.4.1. Stack SIP de Android

Dentro de las muchas librerías propias que posee la plataforma Android, se encuentran unas clases que cuentan con las funcionalidades mencionadas anteriormente y preparadas para programar en código Java, directamente para hacer uso de ella y compilar la aplicación. [9]

#### 4.4.2. PJSIP

En cambio, como contraposición se tiene la librería PJSIP [10], stack de protocolo SIP de código abierto escrito en C.

Las características principales de PJSIP son su construcción sobre PJJIB, la cual es una biblioteca portátil que hace que PJSIP pueda ejecutarse en cualquier plataforma, que es personalizable y además modular.

A lo largo de los siguientes capítulos, se explicará el uso de cada una de estas librerías en cada uno de los entornos de programación mencionados anteriormente, así como el detalle de los errores y soluciones que se han ido encontrando a lo largo del camino hasta la obtención, finalmente, de una aplicación con servicio SIP que cumpla con los requisitos demandados.

## 5. Desarrollo de servicio SIP según la librería Android

Como primeros requisitos para el desarrollo del servicio, se tenía que la aplicación debía ser desarrollada en el entorno RAD Studio y a través del Stack de Android.

¿Por qué estos requisitos estrictos, si como se ha visto en el capítulo anterior existen otros entornos y otras librerías que puedan desempeñar la misma función?

Se escogió RAD Studio como entorno sobre el que programar la aplicación debido a que las aplicaciones que se ejecutan en las pantallas existentes en el catálogo de la empresa, estaban programadas ya en dicho entorno. Por lo tanto, para el desarrollo de la nueva pantalla que contará con la nueva funcionalidad se deseó que fuera este mismo.

El uso del stack de Android se decidió así debido a la facilidad con la que se esperaba poder programar una aplicación de Android con una librería programada directamente para ese uso.

Según lo mencionado antes sobre esta librería y entrando un poco más en detalle acerca de sus clases, se tiene lo siguiente:

- SipManager: Proporciona API para tareas SIP, como iniciar conexiones SIP y acceso a servicios relacionados.
- SipProfile: Define un perfil SIP, incluida su cuenta, dominio e información del servidor. Lleva asociada una clase heredada que la ayuda a crear el perfil.
- SipSession: Representa una sesión SIP que está asociada con un cuadro de diálogo SIP o una transacción independiente que no está dentro de un cuadro de diálogo. Tiene 2 clases heredadas, una para escuchar los eventos relacionados con la sesión y otra que define los estados en los que se encuentra dicha sesión.
- SipAudioCall: Maneja una llamada de audio de Internet a través de SIP. También tiene una clase heredada que escucha los eventos que tienen relación con la llamada.
- SipErrorCode: Define los códigos de error devueltos durante las acciones SIP.
- SipException: Indica una excepción general relacionada con SIP.

Una vez que se tienen claras las clases que tiene y que son perfectas para el uso que esperamos, es necesario comprobar los requisitos y limitaciones que supondrá utilizarla.

Los dispositivos en lo que se podrá ejecutar tendrán que tener una versión de Android 2.3 o superior, ya que existen ciertos aspectos que las versiones más antiguas no soportan. Además, dicho dispositivo ha de tener una conexión de datos debido a que la conexión será inalámbrica. Por otra parte, cada participante en la sesión deberá tener una cuenta SIP con la que registrarse en el servidor. Finalmente, el usuario deberá dar permisos al instalar la aplicación para el uso de ciertas singularidades.

## 5.1. ENTORNO RAD STUDIO

A la hora de empezar a programar en RAD Studio, hay que tener en cuenta, que, para este caso, no va a ser una cosa “directa”, ya que se quiere desarrollar una aplicación para Android a través de lenguaje C++ con una librería compilada en Java. Esto indica que es necesario hacer igual del JNI mencionado en el apartado 4.2.3, que se utiliza como interfaz entre el código Java y el código nativo.

En este entorno, debido a que también es posible programar en código Delphi, se usará el Delphi Interface para la obtención de los objetos de las clases que se han descrito. El Delphi Interface es un puente para la utilización del JNI. Dicha interfaz toma el código Java y a través de unos operadores por la línea de comandos CMD crea unos archivos (con extensión \*.pas, en pascal) que son interpretables por el programa. A través de estos últimos se consiguen los archivos que el código C++ es capaz de entender (los \*.h y \*.cpp). Estos archivos se incluyen en el proyecto y se utilizan como API. Este es el caso de “Android.JNI.Net.hpp”, archivo que se utilizará en este proyecto.

Entonces, sería tan fácil como ver la manera en que se instancian estos objetos en el código Java, tal y como está descrito en la documentación asociada al stack y pasarlo por la interfaz mencionada. Se entenderá mejor con un ejemplo:

```
1 public static SipManager manager;
2 public static SipProfile me;
3 public static SipProfile.Builder builder;
4 public static String password;
5
6 public static void initializeSip (Context context){
7     manager = SipManager.newInstance(context)
8     builder = new SipProfile.Builder("direccionURIInvalida");
9     builder.setPassword(password);
10    builder.setPort(XXXX);
11    me = builder.build();
```

Figura 5. 1.- Ejemplo instanciación SipManager y SipProfile en Java.

```

1 public:
2     _di_JSipManager manager;
3     _di_JSipProfile me;
4     _di_JSipProfile_Builder builder;
5     _di_JString password;
6     _di_JString name;
7     _di_JString domain;
8
9
10    __fastcall TMainForm::TMainForm(TComponent* Owner) : TForm(Owner){
11        manager = TJSipManager::JavaClass->newInstance(SharedActivityContext());
12        builder = new TJSipProfile_Builder(name, domain);
13        builder->setPassword(password);
14        builder->setPort(XXXX);
15        me = builder->build();
16        manager->open(me);
17    }

```

Figura 5. 2.- Ejemplo de la “traducción” de instanciación de SipManager y SipProfile en C++.

Teniendo un poco de conocimientos básicos acerca de la programación en código Java y código C++, al comparar la Figura 5.2 con la Figura 5.1, podría pensarse que esta sería la traducción más lógica, y así es, puede ser la más lógica pero no la efectiva. Si se ejecuta esa parte de código en RAD Studio dará un error de compilación, lo que significa que no se están instanciando bien los objetos.

Tras muchos intentos e investigaciones, e incluso búsquedas de ejemplos de otras librerías de Android compiladas para C++ a través de Delphi Interface, se llegó a una instanciación válida para la librería SipProfile.

```

1 public:
2     _di_JSipProfile me;
3     _di_JSipProfile_Builder builder;
4     _di_JString password;
5     _di_JString uri;
6
7
8    __fastcall TMainForm::TMainForm(TComponent* Owner) : TForm(Owner){
9        builder = TJSipProfile_Builder::JavaClass->init(uri);
10        builder->setPassword(password);
11        builder->setPort(XXXX);
12        me = builder->build();
13    }

```

Figura 5. 3.- Instanciación válida de SipProfile en C++.

Como se puede apreciar, sin contar que se ha eliminado la parte de instanciación del SipManager, las figuras 5.2 y 5.2 son iguales a excepción de la línea sombreada en esta última. Si se ejecuta de este modo la compilación es correcta, problema solucionado.



Aun así, sigue existiendo un problema, la traducción de la instanciación del SipManager, objeto sin el cual no es posible realizar el programa ya que es el encargado de abrir el perfil para que sea usado.

Investigando más y haciendo participación en algún foro de dudas de programación se obtuvo la respuesta de cuál era el fallo que acarreaba este objeto, y es que el método necesario para instanciarlo, *newInstance()*, no estaba incluido en la interfaz JSipManagerClass en Android.JNI.Net.hpp por lo tanto sería necesario reimportar la clase SipManager incluyendo el método.

Para el reimporte de la clase es necesario el Java2OP.exe, herramienta de línea de comandos que puede utilizarse para generar archivos de puente nativos de Delphi desde bibliotecas Java [11].

A fin de evitar problemas con la ambigüedad de nombre se creó una nueva API llamada “Androidapi.JNI.SIP” que tiene como interfaz a JSipManager2 en la que se incluyó el método necesario y algún que otro parámetro que necesitaba estaba y no estaba incluido tampoco en el API anterior. [12]

Una vez creado el archivo en pascal, como se explicó al comienzo de este apartado, se compiló para obtener los archivos en código C++ y así obtener los legibles por este lenguaje. Así se consiguió lo siguiente:

```
1 public:
2     _di_JSipManager2 manager;
3
4
5
6    fastcall TMainForm::TMainForm(TComponent* Owner) : TForm(Owner) {
7         manager = TJSipManager2::JavaClass->newInstance(TAndroidHelper::Context);
8     }
```

Figura 5. 4.- Instanciación correcta de SipManager en C++.

Conseguidos estos 2 pasos tan importantes, se ha podido desarrollar el código realizando pasos similares a los mencionados aquí arriba de manera que se siguieron las siguientes pautas:

- Instanciación del mánager.
- Instanciación del builder.
- Apertura del perfil con las credenciales oportunas.
- Registro del perfil a través del mánager.



Gracias a un servidor, del que se hablará en capítulos siguientes, fue posible la comprobación del registro de la cuenta y así poner en marcha el proceso de realizar una llamada de audio.

Originalmente, la aplicación no debe realizar llamadas ya que no es lógico que desde el, comúnmente, telefonillo de casa se llame al videoportero o incluso a otras casas, pero aún así se decía intentar implementarlo para desarrollar en principio un servicio SIP auténtico y seguir todos los pasos del desarrollo para no saltar ninguno y pudiera dar lugar a error. Una vez que la aplicación estuviera totalmente funcional ya se adecuarían las funciones a las requeridas.

Se siguieron los siguientes pasos:

- Instanciación del SipAudioCall\_Listener, que escuchará los eventos de llamada.
- Instanciación del SipSession\_Listener, que escuchará los eventos de la sesión.
- Creación de la sesión a través del mánager.
- Creación y realización de la llamada a través del mánager.

En este caso, gracias a un software multiplataforma, ya existente diseñado para dar soporte SIP, llamado Zoiper®, y al servidor, se crearon 2 cuentas SIP. Una se registró en el Zoiper® y otra en la aplicación bajo desarrollo y desde esta última se realizó una llamada de audio a la otra.

La llamada fue satisfactoria, aunque no había recepción ni envío de audio en ella. Esto era debido a que el audio y el micrófono estaban programados para iniciarse cuando había un evento, a través del llamado SipAudioCall\_Listener, que indicara que se estaba en una llamada. Y ¿cuál era el problema de esto? Que el entorno en el que se está trabajando no es capaz de escuchar eventos.

Por lo tanto, en vez de comprobar cuando saltaba el evento de estar en una llamada, se comprobaría el estado de la llamada para activar los dispositivos media y esto solucionó el problema. Pero sólo se solucionó el problema de enviar y recibir audio, no el de los eventos. Lo que derivó en el gran problema que no tuvo solución

La función principal de esta aplicación sería solamente la de registrarse en el servidor y ya, el usuario no tendría que hacer nada más, solamente dejar que la aplicación por si sola estuviera escuchando a través del “listener” los eventos de llamada entrante del videoportero, pero claro, no hay opción de poder utilizar los eventos.

Este problema fue decisivo para desechar la idea de realizar la aplicación en RAD Studio, aunque las demás aplicaciones de las pantallas estuvieran en este entorno. Se pasaría a programar la aplicación en Android Studio y ya se estudiaría la opción de incluir las dos en la pantalla o programar todo en este nuevo entorno.

## 5.2. ENTORNO ANDROID STUDIO

Aunque se haya decidido pasar a desarrollar la aplicación en este entorno, se seguirá usando el stack de Android, ya que se conoce su funcionamiento.

En este caso, no hará falta usar la instanciación realizada a través de Delphi, vista en el capítulo anterior, debido a que esta aplicación se desarrollará totalmente en lenguaje Java, no en C++. Por lo tanto, se podrá configurar todo de igual manera a la que se muestra en la Figura 5.1.

Siguiendo los pasos mencionados, se consigue crear un programa que compila, crea el objeto mánager y abre el perfil, pero no registra.

```
I/System.out: +++ User Profile = CREATED
I/System.out: +++ ATTEMPTING TO OPEN PROFILE: sip:101@192.168.0.15;transport=tcp
I/art: Do partial code cache collection, code=62KB, data=53KB
I/art: After code cache collection, code=62KB, data=53KB
      Increasing code cache capacity to 256KB
I/System.out: +++ IS OPEN IN CC: true
      +++ IS API SUPPORTED: true
I/System.out: +++ IS VOIP SUPORTED: true
I/System.out: +++ MANAGER INSTANCE: android.net.sip.SipManager@12f053c
I/System.out: +++ IS OPENED: true
I/System.out: null, SipService.createSession() returns null
```

Figura 5. 5.- Código de error al registrar el usuario.

Como se observa en la Figura 5.5, se crea el perfil, se abre en la dirección del dominio del servidor, comprueba que el dispositivo soporte la aplicación, abre el perfil a través de la instancia del mánager y al registrar devuelve un error que indica que el servicio SIP no puede crear la sesión y retorna un objeto nulo que no se puede registrar.

La dirección de dominio y credenciales del usuario son válidas ya que se trata de un servidor propio desarrollado y que se ha comprobado con la anterior aplicación que sí registraba.

Tras mucho investigar y no dar con una solución al problema y no entender por qué la misma librería es capaz de registrarse en un entorno y en otro no, se llegó a la conclusión de que es un problema interno de Android Studio y este no es capaz de crear una sesión SIP aun teniendo todos los permisos dados.

Se intentó, de todas formas, realizar por otro medio posible el registro, sin éxito alguno y además se localizó otro error en la librería que hubiera echado por tierra todo lo conseguido hasta entonces.

Este stack no contiene la capacidad de realizar o recibir llamadas de vídeo. Hasta la fecha, Android solamente ha creado la librería para la transmisión bidireccional de audio, mientras que el vídeo no está implementado. De haber seguido con esta aplicación adelante, al llegar a la transmisión del vídeo se hubiera llegado otra vez a un callejón sin salida.

## 6. Desarrollo de servicio SIP según la librería PJSIP

Llegado a este punto y, ahora sí que sí, desechando totalmente el stack de Android, se decide cambiar a la librería PJSIP que se ha mencionado en el apartado 4.4.2 y volver al entorno de partida, RAD Studio y volver a la idea principal de desarrollar esta aplicación juntamente con las de las pantallas existentes.

¿Qué se necesita saber antes de empezar a utilizar la librería PJSIP? Se había comentado que es de código abierto y multiplataforma, lo que quiere decir que además de hacer esta aplicación para Android, como es el caso, si se quisiera desarrollar la aplicación para Windows o iOS será posible, siempre y cuando se compila la librería para cada tipo de plataforma y se realice su programa.

Al buscar un poco más de documentación acerca de la librería se observa que compilando PJSIP para Android [13] se puede crear una API de alto nivel llamada PJSUA2 creada sobre PJSUA-LIB. Esta API unifica SIP, medios de audio/video y las mejores prácticas de aplicaciones de medios de cliente. La compilación ha sido realizada mediante línea de comandos, en una máquina Linux, para la obtención de los paquetes de librería \*.so y los archivos \*.java. Estos serán incluidos en el proyecto antes de comenzar a programar nada.

De forma resumida, se tiene que PJSUA2 es una abstracción orientada a objetos y que proporciona una API de alto nivel para la construcción de aplicaciones de agentes de usuario multimedia del protocolo SIP. [14]

Al igual que en el caso del stack de Android, esta cuenta también con una serie de clases principales sobre las que hay que desarrollar la aplicación:

- **Endpoint:** Clase principal. Es necesario crear una, y solamente una, instancia de ella, para poder inicializar e iniciar la biblioteca.
- **Account:** Sirve para identificar la identidad del cliente y a través de ella se podrán recibir y realizar llamadas.
- **Media:** Clase abstracta para reproducir medios.
- **Call:** Representa una llamada en curso.
- **Buddy:** Esta clase es un amigo remoto que puede tenerse en lista de contactos.

## 6.1. ENTORNO RAD STUDIO

Antes de empezar a programar, leyendo detalladamente la documentación de la librería [15], se estuvieron comprobando los métodos que se iban a utilizar para las diferentes clases presentadas en el inicio de este capítulo.

Aparte de métodos, se ha visto que también es necesario el uso de eventos para, por ejemplo, ver si el usuario ha sido registrado, comprobar el estado o la información de la llamada...

Por consiguiente, se desechó la idea de desarrollar la aplicación en este entorno ya que se tendría el problema del apartado 5.1, en el cual no se podían utilizar los eventos.

## 6.2. ENTORNO ANDROID STUDIO

Una vez compilada la librería para su utilización en aplicaciones Android y estudiada la documentación se empezó a programar. Pero se hablará un poco más acerca de cada clase necesaria y del proceso seguido.

### 6.2.1. Endpoint

Entrando más en detalle de lo mencionado antes, la clase principal proporciona las funciones de arranque y apagado del servicio además de la personalización de la configuración que se le quiera dar a la cuenta.

Los pasos que seguir son:

- Instanciación.
- Creación de la librería.
- Iniciación de la librería y configuración de la futura cuenta.
- Creación del transporte.
- Comienzo de la librería.

Se debe programar también la destrucción de esta en el caso de que la aplicación se cierre.

### 6.2.2. Account

Una cuenta tiene un identificador de recursos SIP asociado, este identificador es la URI. Esta actúa como dirección de registro y es del estilo *sip:User@Domain*. En la aplicación debe crearse al menos una cuenta para poder recibir las solicitudes.

Esta cuenta puede tener o no usuario, aunque una cuenta sin usuario identifica un Endpoint local en lugar del usuario en particular. La cuenta que se desarrollará contará con un usuario.

Para el uso de esta clase es necesaria la creación de una clase heredada en la que se recibirán las notificaciones de la cuenta, como del proceso de registro o de las llamadas entrantes.

- URI.
- Dirección del servidor Registrar.
- Configuración de credenciales.
- Creación de la cuenta.

### 6.2.3. Media

Dentro de esta clase existen algunas clases heredadas ya incluidas en la API de uso, la más importante de estas es AudioMedia, que representa los medios de audio. Tiene distintas formas de uso, en este caso servirá para transmitir y recibir audio en la llamada entrante.

### 6.2.4. Call

En este caso también es necesario crear una clase que herede de la original. Esta nueva subclase deberá implementar las devoluciones de llamada.

Para realizar una llamada saliente, aunque en este caso no sería necesario implementarlo, solo hace falta la dirección URI del otro usuario con el que se quiera crear la conexión y realizar la acción del método *makeCall*.

En el caso de la recepción de un evento de llamada entrante, siempre habrá que comprobar que tipo de información contiene la llamada que está llegando, para así activar solamente el audio, en caso de que sea una llamada sin video, o activar todos los servicios media en caso de que si lo lleve.

### 6.2.5. Buddy

La clase Buddy no será necesaria para esta aplicación ya que sirve para guardas contactos en una lista de llamadas a realizar, y aquí no se va a realizar ninguna llamada. También podría servir para identificar a quién está llamando, pero solo se van a recibir llamadas del videoportero, por lo tanto, su uso no hace falta.

### 6.2.6. Video

Aunque no sea una clase particular de la librería es importante destacar que una de las partes esenciales de esta aplicación es la captura del vídeo que viene del videoportero. Para ello es necesario capturar en todo momento las imágenes que se van recibiendo.

El código del programa se puede encontrar en el documento Anexos, mientras que los aspectos de la programación están detallados en el Manual del Programador.

En la siguiente figura se tiene la pantalla de registro resultante tras el desarrollo de la aplicación.

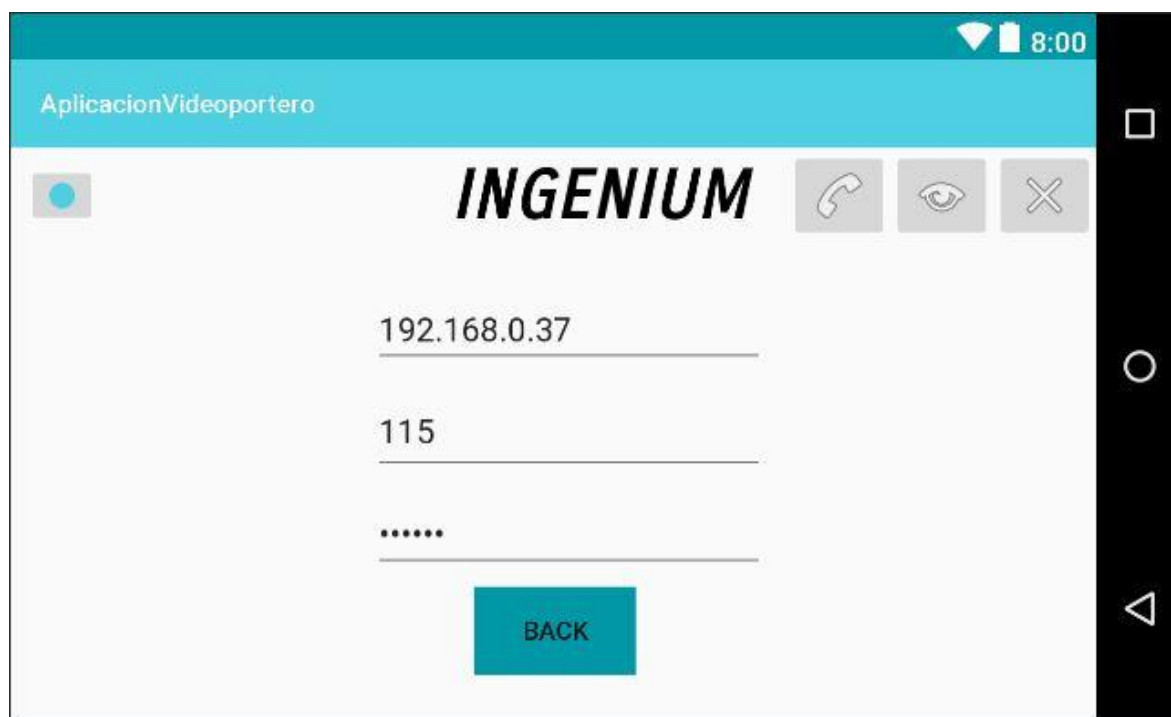


Figura 6. 1.- Diseño de la aplicación.

## 7. Uso de servidores

El uso de servidores ha sido obligatorio en este proyecto ya que se requiere de ellos para los registros de los usuarios, pero el manejo de estos ha sido clave para resolver muchos problemas que se iban encontrando en la programación, ya que gracias a ello se han podido comprobar en todo momento si los registros se estaban realizando bien, si existían llamadas...

En este capítulo se hablará de dos tipos de servidores, servidores genéricos para servicio SIP, que han sido necesario durante la fase de aprendizaje, desarrollo y pruebas, y de un servidor específico de videoportero sobre el que se han realizado las comprobaciones finales de la aplicación.

### 7.1. SERVIDORES GENÉRICOS

Estos servidores son una interfaz gráfica de usuario de código abierto que controla un servidor VoIP. Son centralitas que corren sobre el sistema operativo Linux y que gracias a esa interfaz gráfica son más fáciles de configurar. Esta configuración tiene dos partes, la parte interna entre la centralita y los teléfonos y la unión entre la centralita y el proveedor del servicio VoIP.

#### 7.1.1. Asterisk-FreePBX

Se encarga de toda la parte funcional de recepción, emisión de llamadas, configuración de extensiones y todo tipo de funcionalidades que posee el protocolo. Tiene un panel de configuración, FreePBX, con el que se realizan de forma gráfica y sencilla las configuraciones de entradas, salidas y funciones específicas de Asterisk.

La configuración de este servidor se realizó para crear en él unas extensiones que pudieran indicar la existencia de registros y llamadas en todo momento.

Se crearon 4 extensiones. de la 101 a la 104 incluidas, que se fueron probando a lo largo del desarrollo de las aplicaciones, en ambos entornos y ambas librerías.



```

Endpoint: <Endpoint/CID.....> <State.....> <Channels.>
I/OAuth: <AuthId/UserName.....>
Aor: <Aor.....> <MaxContact>
Contact: <Aor/ContactUri.....> <Hash.....> <Status> <RTT(ms)..>
Transport: <TransportId.....> <Type> <cos> <tos> <BindAddress.....>
Identify: <Identify/Endpoint.....>
Match: <ip/cidr.....>
Channel: <ChannelId.....> <State.....> <Time.....>
Exten: <DialedExten.....> CLCID: <ConnectedLineCID.....>
=====

Endpoint: 101/101                               Not in use    0 of inf
  InAuth: 101-auth/101
  Aor: 101                                       1
  Contact: 101/sip:101@192.168.0.16:38700;rinstance=3 9166539a80 Avail    6.991
  Identify: 101-identify/101

Endpoint: 102/102                               Not in use    0 of inf
  InAuth: 102-auth/102
  Aor: 102                                       1
  Contact: 102/sip:102@192.168.0.14:43736;rinstance=c c433d4ff7f Avail    23.763
  Identify: 102-identify/102

Endpoint: 103/103                               Unavailable   0 of inf
  InAuth: 103-auth/103
  Aor: 103                                       1
  Identify: 103-identify/103

```

Figura 7. 1.- Control de registros en Asterisk-FreePBX.

Las cuentas registradas se muestran en la información de registros y en ella, como se puede ver en la Figure 7.1, se muestran cuáles de las extensiones creadas están registradas y en que dirección física se las puede encontrar.

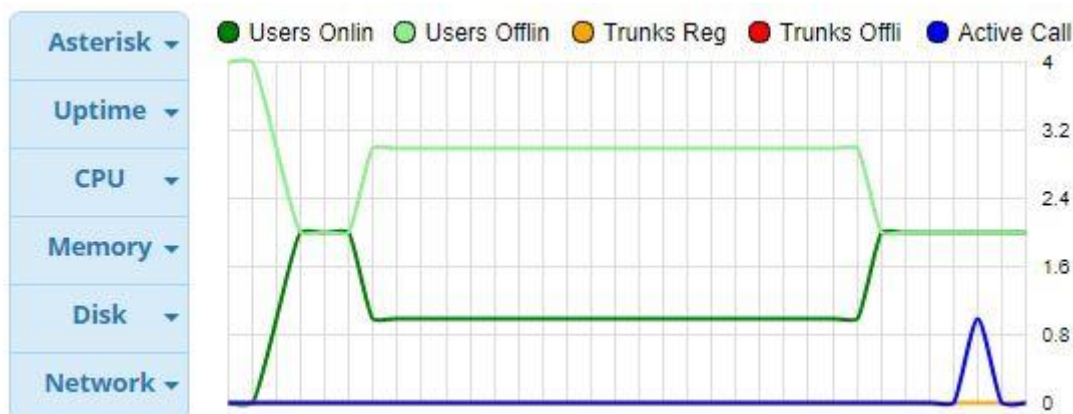


Figura 7. 2.- Tablero de Asterisk-PBX.

En la Figura 7.2 se ve una imagen del tablero del servidor que muestra una evolución temporal de los registros y llamadas por unidad.

Aunque con Asterisk-FreePBX se pudieron comprobar las cuentas de prueba que se usaron para el desarrollo de la aplicación con el stack de Android, a la hora de probar las de PJSIP había fallos que indicaban que el registro se encontraba siempre ocupado.

Tras buscar información al respecto, se encontró que el servidor actual aun no contaba con las mejoras incluidas en las nuevas versiones de la librería PJSIP y daba fallos en el registro, por lo que hubo que cambiar de servidor a Elastix y rehacer toda la configuración.

### 7.1.2. Elastix

Software de código abierto para el establecimiento de comunicaciones unificadas. Incluye dos partes en una, el servidor Asterisk para las funcionalidades de la centralita y el FreePBX para la configuración mediante la interfaz web del servidor.

Al igual que en el servidor anterior, se crearon extensiones para realizar las comprobaciones oportunas de registros y llamadas.

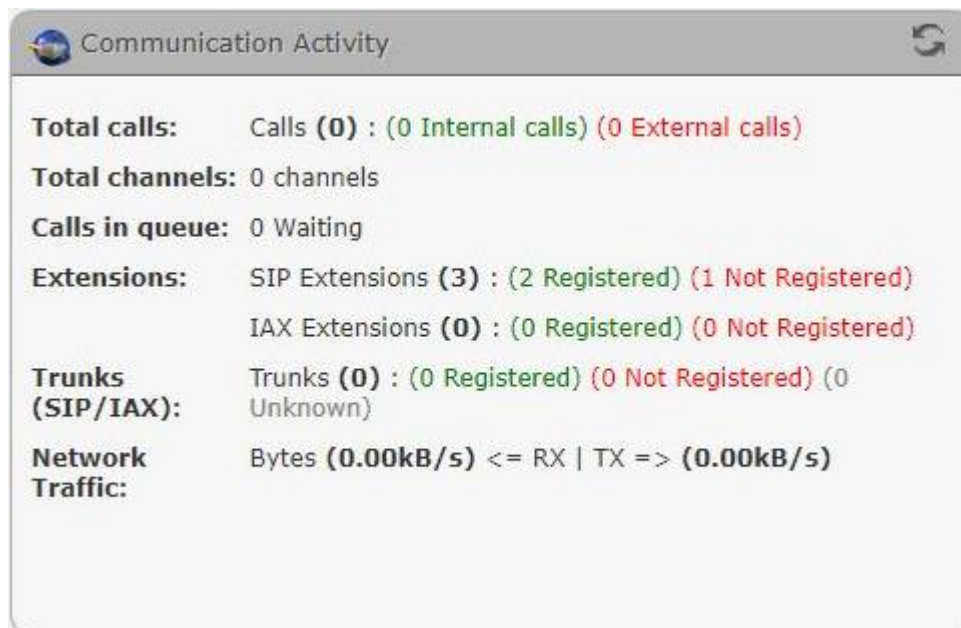


Figura 7. 3.- Control de registros de Elastix.

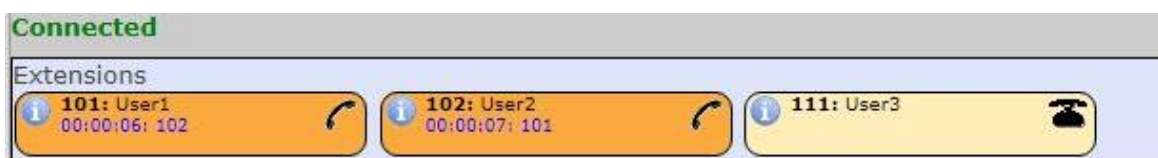


Figura 7. 4.- Tablero de conexiones y llamadas de Elastix.

## 7.2. SERVIDOR IP VARIO

Una vez probada la aplicación con los servidores genéricos y comprobar su funcionalidad básica, es decir, registros de cuentas y el envío o recepción de llamadas de audio, era necesario probar eso mismo, pero con las llamadas que también incluían vídeo, por lo tanto, se escogió un producto para ello.



Figura 7. 5.- Intercomunicados 2N IP Vario.

Este intercomunicador proporciona servicios avanzados y más amplios que los teléfonos domésticos estándar. Además, su instalación es muy sencilla, solo se necesita conectar el intercomunicador a los otros elementos de la red LAN mediante cable UTP y configurar los parámetros necesarios a través de la interfaz web que posee para el uso del servidor.

El intercomunicador, como se ve en la Figura 7.5, está equipado con un botón estándar para, en este caso, realizar la llamada a un usuario. Este es el modelo básico de prueba, en un caso futuro el intercomunicador que se fuera a utilizar podría tener tantos botones como se quisiera, un teclado alfanumérico e incluso lectores de tarjetas o huellas.

Internamente, está equipado con un interruptor de relé que controlará el bloqueo eléctrico de la puerta de acceso. Su tiempo y método de activación se puede programar de manera flexible.

En la siguiente figura se encuentra la interfaz principal que posee el intercomunicador para realizar las configuraciones necesarias.

## 2N<sup>®</sup> IP Vario



Figura 7. 6.- Pantalla principal del intercomunicador.

A la vista de la figura, se encuentran una serie de apartados para las diferentes configuraciones del intercomunicador, desde la opción de configurar el propio servidor, hasta la opción de configurar el hardware para la selección del uso del botón o el relé, pasando por la posibilidad de tener un directorio en el que se registren los usuarios que se vayan a tener en el sistema.

Al tener la versión más sencilla de este, ha habido configuraciones que han sido más sencillas de realizar, o al menos más inmediatas.

En cuanto al tema del servidor, el propio intercomunicador, será el que ejerza de esto, pero además cuenta con la posibilidad de desarrollar 2 usuarios internos, ya que sin esa posibilidad sería imposible que este pudiera realizar las llamadas a los demás usuarios.

Para los usuarios se han añadido extensiones, al igual que en los servidores genéricos, y en cuanto al uso del hardware se ha añadido que, con la pulsación del botón de marcación rápida, se llame al usuario registrado en la pantalla. La apertura de la puerta se realizará con el envío de un código por parte de la aplicación al intercomunicador.

La interfaz también cuenta con un registro de eventos en los que informa de las llamadas entrantes o salientes del servidor, los estados de dichas llamadas, los cambios en el switch del relé...

19 Nov 16:00:19	<b>CallStateChanged</b>	direction= <b>outgoing</b> , state= <b>connecting</b> , peer= <b>sip:101@192.168.0.14</b> , session= <b>22</b> , call= <b>25</b>
19 Nov 16:00:19	<b>KeyPressed</b>	key= <b>%1</b>
19 Nov 15:59:42	<b>CallStateChanged</b>	direction= <b>outgoing</b> , state= <b>terminated</b> , reason= <b>normal</b> , peer= <b>sip:101@192.168.0.15</b> , session= <b>21</b> , call= <b>24</b>
19 Nov 15:59:17	<b>OutputChanged</b>	port= <b>relay1</b> , state= <b>false</b>
19 Nov 15:59:17	<b>SwitchStateChanged</b>	switch= <b>1</b> , state= <b>false</b>
19 Nov 15:59:12	<b>OutputChanged</b>	port= <b>relay1</b> , state= <b>true</b>
19 Nov 15:59:12	<b>SwitchStateChanged</b>	switch= <b>1</b> , state= <b>true</b> , originator= <b>dtmf</b>
19 Nov 15:59:03	<b>CallStateChanged</b>	direction= <b>outgoing</b> , state= <b>connected</b> , peer= <b>sip:101@192.168.0.15</b> , session= <b>21</b> , call= <b>24</b>
19 Nov 15:59:03	<b>CallStateChanged</b>	direction= <b>outgoing</b> , state= <b>ringing</b> , peer= <b>sip:101@192.168.0.15</b> , session= <b>21</b> , call= <b>24</b>
19 Nov 15:59:03	<b>KeyReleased</b>	key= <b>%1</b>
19 Nov 15:59:03	<b>CallStateChanged</b>	direction= <b>outgoing</b> , state= <b>connecting</b> , peer= <b>sip:101@192.168.0.15</b> , session= <b>21</b> , call= <b>24</b>
19 Nov 15:59:03	<b>KeyPressed</b>	key= <b>%1</b>
19 Nov 15:57:29	<b>CallStateChanged</b>	direction= <b>incoming</b> , state= <b>terminated</b> , reason= <b>normal</b> , peer= <b>sip:101@192.168.0.15</b> , session= <b>20</b> , call= <b>23</b>

Figura 7. 7.- Tabla de eventos del intercomunicador.

## 8. Aplicación BUSing para comunicación

Se decidió ampliar el proyecto e incluir también una aplicación BUSing para simular la comunicación que tendría que tener la aplicación del videoportero con las pantallas de INGENIUM S.L, ya que además ambas estarán realizadas en entornos y con lenguajes diferentes.

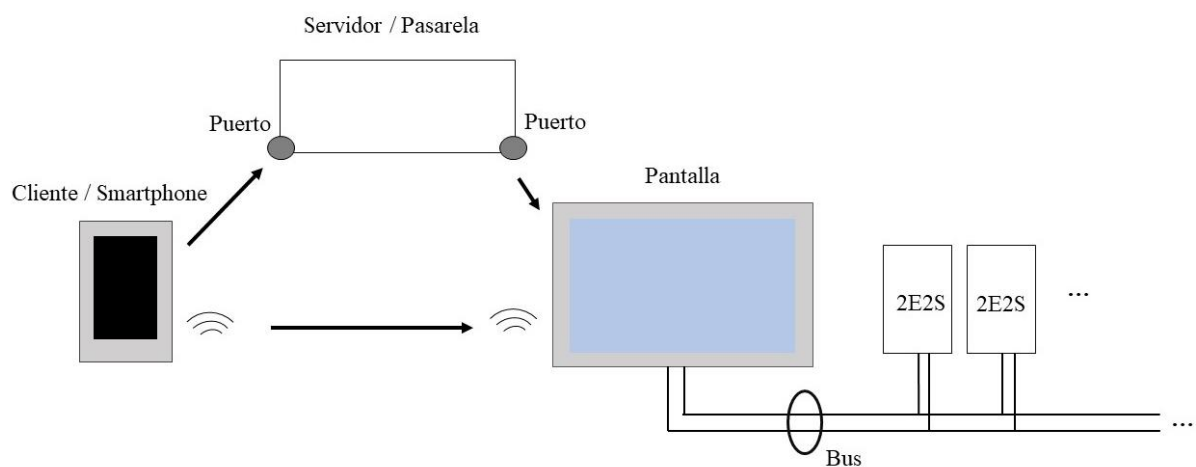


Figura 8. 1.- Esquema de conexión BUSing.

En la Figura 8.1, se muestra un esquema del conexionado BUSing en el que pueden distinguirse 4 partes importantes en la comunicación.

Se tiene el smartphone, que será el cliente dentro de la comunicación y en el cual correrán ambas aplicaciones, el servidor, que almacena los datos y actúa como pasarela dependiendo del tipo de comunicación, la pantalla que será una Smart Touch y los dispositivos conectados a ella. Dentro de lo que es la comunicación se tienen 3 modos que se explicarán en el siguiente apartado.

Para la comunicación se hará uso de un panel que contará con 2 actuadores 2E2S (2 entradas – 2 salidas), uno estará programado en modo normal y otro en modo persiana, también cuenta con un regulador conectado a una bombilla y la pantalla que funciona como termostato.



## 8.1. COMUNICACIÓN

Como se ha comentado en la introducción de este capítulo, se cuentan con 3 tipos de comunicación disponibles. Estas conexiones, sean por la forma que sean, serán a través del protocolo TCP.

- Directa: El cliente se puede conectar directamente a la pantalla si conoce la IP y el puerto de esta y además se encuentran dentro de la misma red Wifi.
- Servidor: El cliente también se puede conectar al servidor, estando este a su vez conectado a la pantalla, y esto puede ser de 2 maneras:
  - Con identificador: Conociendo la IP y el puerto por el que conectarse al servidor y además mandarle el identificador que posee la pantalla a la que se quiere conectar para que redirija el tráfico.
  - Con credenciales: De esta manera, el usuario, que estará previamente registrado en el servidor, le enviará a este su nombre de usuario y contraseña y el servidor se encargará de realizar la conexión.

## 8.2.ESPECIFICACIONES

Para la programación, se ha tenido en cuenta la funcionalidad, básica, que tienen las pantallas, y se ha concluido que la aplicación ha de realizar lo siguiente:

- Conexión TCP: Como se ha visto en el apartado anterior, la conexión TCP puede realizarse de 3 formas distintas, aunque solamente se realizarán 2, directamente a la pantalla y a través de credenciales, ya que no es normal que el usuario vaya a conectarse al servidor y darle el identificador de la pantalla.
- Lectura del bus: La lectura del bus, estará realizándose en todo momento que haya algo en él que leer y no ha de ser bloqueante para la aplicación, por lo que su programación ha de estar en un hilo aparte. Además, habrá que diferenciar el modo de conexión, ya que la lectura será diferente dependiendo del modo.
- Escritura en el bus: Para la escritura, habrá que tener en cuenta también el modo de conexión.

- Cierre de la conexión TCP: Aunque la funcionalidad de una pantalla es estar conectada el 100% del tiempo a la instalación, en este caso, al estar realizando un prototipo de aplicación móvil, contará también con la desconexión.

### 8.3. APLICACIÓN

La aplicación cuenta con navegación entre pantallas, teniendo la de registro y lectura y envío de datos, como se muestra a continuación.

El código se encuentra en el documento Anexos y la información detallada de la programación se encuentra en el Manual del Programador.



**INGENIUM**

Please enter the IP address and port...

IP: 85.152.52.212

Port: 12347

Connect

or your username and password

Username: testgoogle

Password: 1234

Send

Waiting for the connection

Videoportero

Figura 8. 2.- Pantalla de conexión BUSing®.



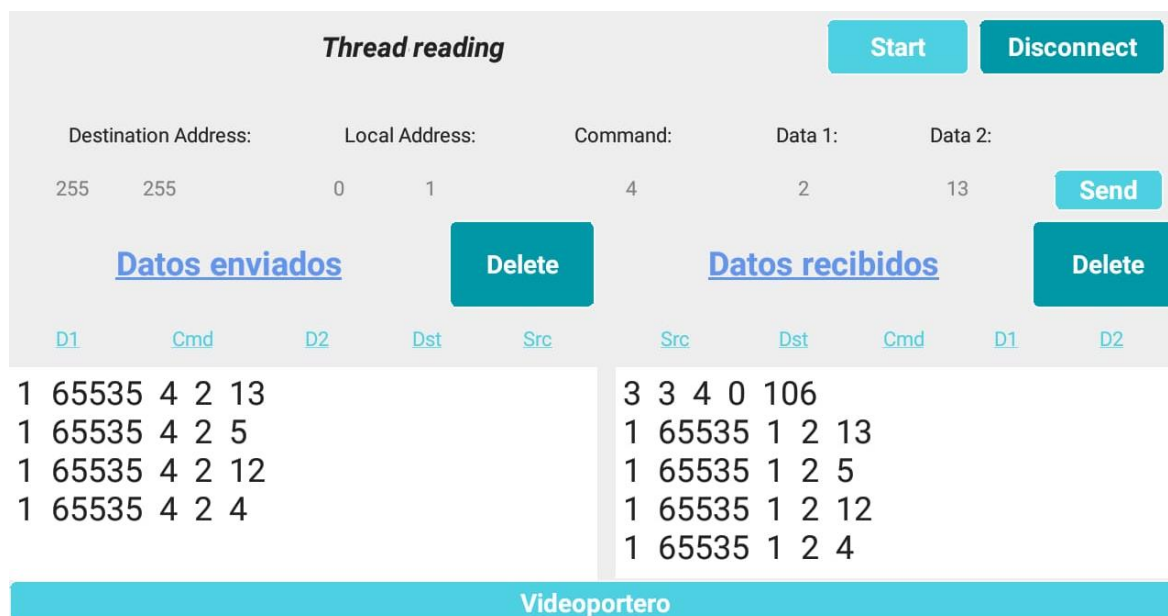


Figura 8. 3.- Pantalla de lectura y escritura del bus.

## 8.4. IMPLEMENTACIÓN FINAL

A las especificaciones originales de la pantalla, se les ha añadido la de la comunicación con el videoportero, está se hará a través de un botón en la aplicación BUSing que lanzará la del videoportero. Esta segunda habrá de contar con otro botón que permita ir hacia la pantalla anterior, aplicación anterior en este caso, sin cerrar la aplicación para que se encuentre activa en todo momento y pueda recibir las llamadas.

Para que la implementación tenga más sentido de pantalla, se han realizado ampliaciones referentes al diseño de ambas aplicaciones, así como una interfaz más intuitiva y sencilla con vistas a su utilización por el usuario final. Añadiendo además una especificación más como es la configuración de distintos aspectos de la pantalla, entre los que se puede encontrar el fondo o el idioma entre otros.

Además, se ha preparado y programado un panel para las pruebas programado con el SiDE, entorno de programación de los elementos BUSing.

En las siguientes Figuras se muestran algunas vistas que pueden considerarse de cierta importancia:



Figura 8. 4.- Pantalla principal de la aplicación.

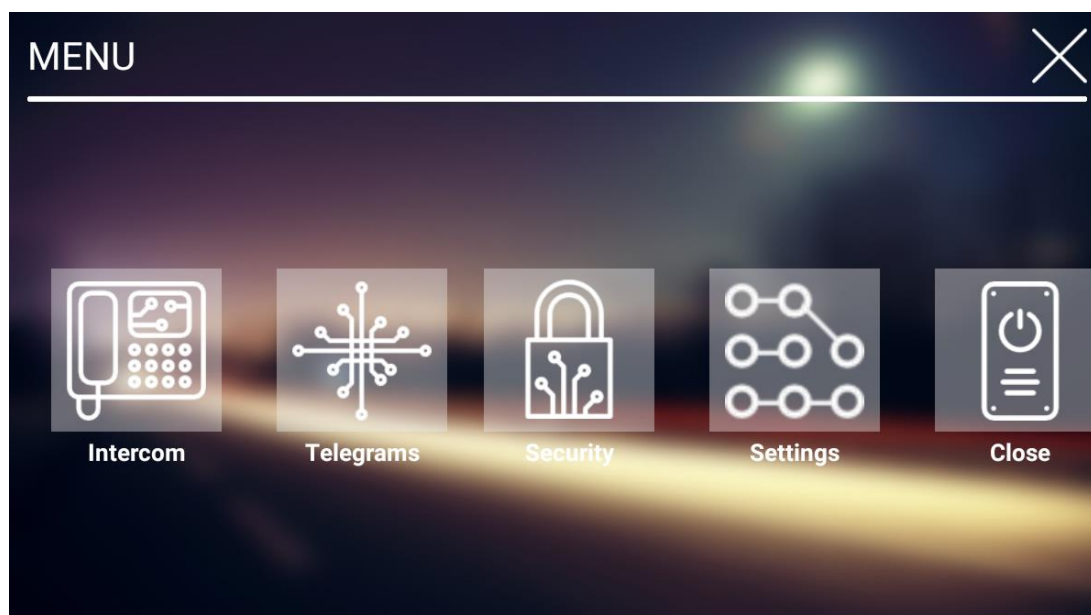


Figura 8. 5.-Pantalla del menú de la aplicación.



Figura 8. 6.- Pantalla de llamada.

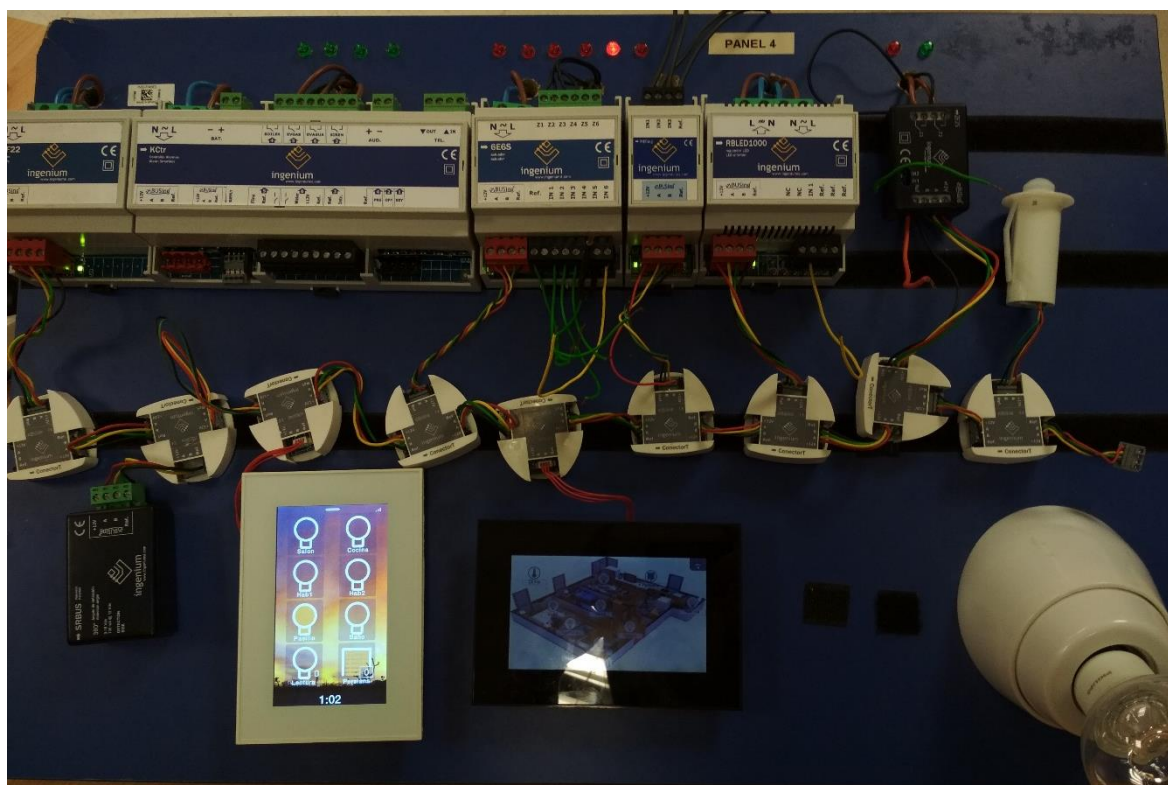


Figura 8. 7.- Panel de pruebas.

En el documento Manual de Usuario se pueden encontrar todas las imágenes referentes a la aplicación y como hacer uso de ella.

## 9. Conclusiones

El presente proyecto tenía como objetivo principal la realización de una aplicación Android que actuara como servicio SIP para la comunicación entre pantallas y videoportero. Parte de ese objetivo pedía que la aplicación estuviera desarrollada en el entorno RAD Studio en lenguaje C++ con una librería Android.

A la vista del proceso descrito en este documento, se aprecia que el entorno RAD Studio, así como la librería Android, no son los adecuados para esta característica. Las consideraciones que se han tomado para llegar a esta conclusión son las siguientes:

Si se habla del entorno RAD Studio, como se ha mencionado en el apartado 4.3.3, puede observarse, que es un entorno muy versátil con la capacidad de compilar aplicaciones para los sistemas operativos más utilizados en la actualidad, aunque como también se ha visto a lo largo del desarrollo de la aplicación, cuenta con algunos matices que hacen que la programación de esta aplicación no sea tan sencilla, llegando al punto de ser imposible, al menos sin tener unos conocimientos más profundos de la programación. Por ello mismo se toma como decisión la implementación en el entorno Android Studio, que, aunque sea solo para aplicaciones de este tipo, es más robusto.

En cuanto a la librería Android, también se ha comentado que, aunque sea la correcta para aplicaciones de este mismo lenguaje, no es la idónea para este caso, ya que no está al completo, faltándole la implementación de vídeo, que es una de las partes principales de este servicio. Debido a este motivo, se eligió por tanto la librería PJSIP, que, aunque su programación resulte más complicada, sus prestaciones son más completas.

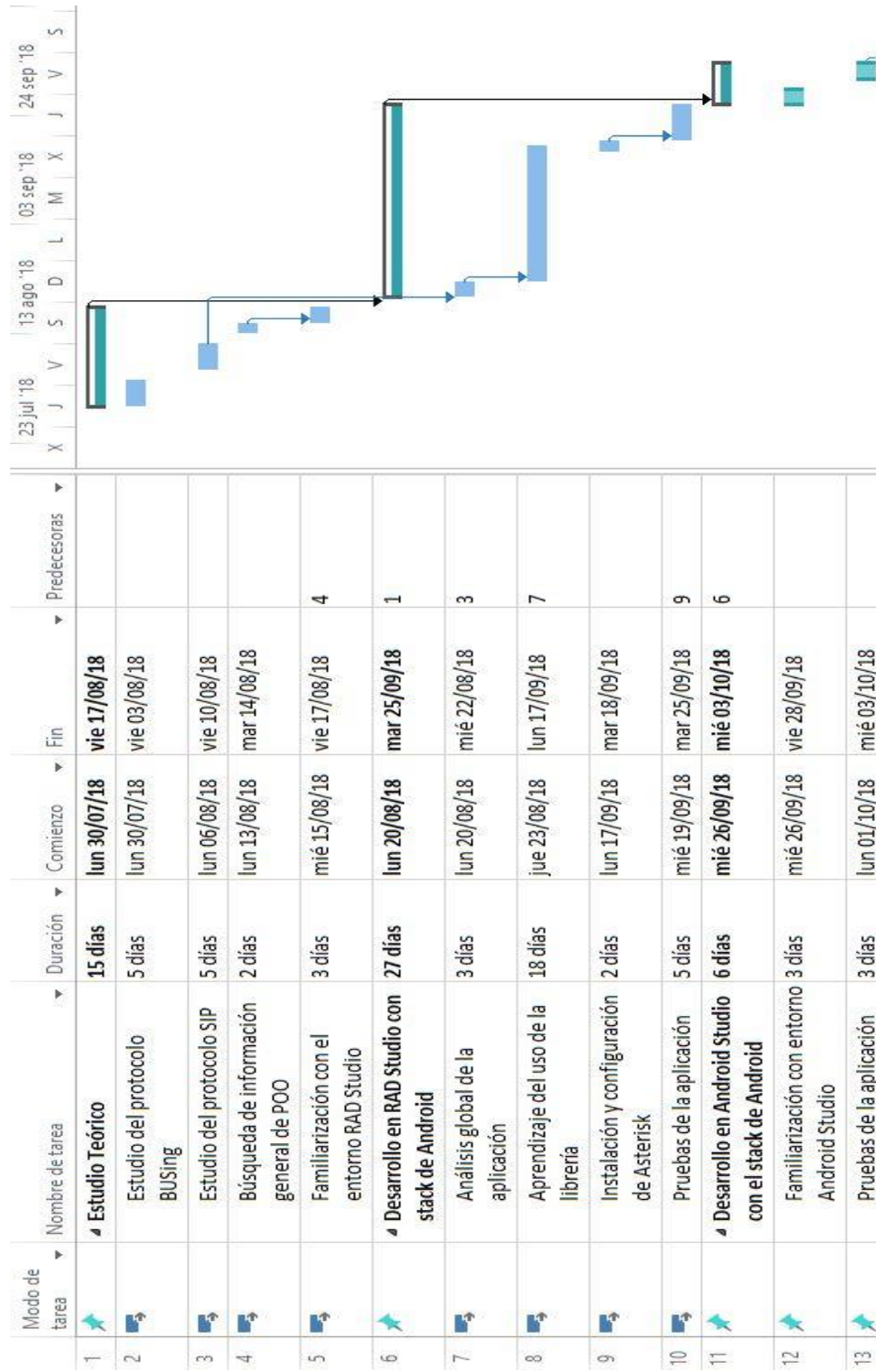
También se ha encontrado un problema, que no se ha comentado a lo largo del documento y ha sido el servidor. Este ha sido elegido por sus características, pero no se había tenido en cuenta las opciones de registro. Como se ha mencionado en el capítulo 2, el protocolo SIP es capaz de, una vez registrado, direccionar las llamadas usando solamente la extensión, pero en este caso, en las pruebas era necesario, además de la extensión, utilizar la dirección IP del teléfono, pantalla o Tablet que se estuviera utilizando como usuario destino. Esto ha sido debido a la marca del intercomunicador, este sólo permite realizar registros completos entre dispositivos de dicha marca que posee para estas aplicaciones, por lo que la elección del servidor, a la hora de poner en marcha el proyecto, también tendrá que necesitar un estudio y una investigación previa.

Acerca de la mejora añadida, de comunicar esta aplicación con una aplicación de comunicación BUSing, se puede apreciar que ha sido un trabajo menos costoso en cuanto al esfuerzo que haya podido suponer la aplicación del videoportero, ya que de base ya se conocía que en el entorno RAD Studio podían desarrollarse las funcionalidades al completo.

Como implementaciones futuras a realizar una vez finalizado este proyecto, sería la creación de la pantalla que sea capaz de integrar ambas aplicaciones y la mejora de la aplicación BUSing para que sea reutilizable en cada instalación que se utilice, como es el caso de las aplicaciones que incorporan ahora las pantallas, que están configuradas para leer el archivo de cada instalación y crear tanto iconos como dispositivos haya que configurar. Además de un diseño propio de servidor/intercomunicador que permita los registros totales de las aplicaciones diseñadas.



# 10. Planificación



Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predesoras
14	▲ Aprendizaje de librería PJSIP	10 días	jue 04/10/18	mié 17/10/18	13
15	Búsqueda de documentación	4 días	jue 04/10/18	mar 09/10/18	
16	Compilación para Android	4 días	mar 09/10/18	vie 12/10/18	
17	Estudio de posibilidades	5 días	jue 11/10/18	mié 17/10/18	
18	▲ Desarrollo en Android Studio con PJSIP	17 días	jue 18/10/18	vie 09/11/18	14
19	Creación de clases	4 días	jue 18/10/18	mar 23/10/18	
20	Instalación de Elastix	2 días	lun 22/10/18	mar 23/10/18	
21	Aplicación	14 días	mar 23/10/18	vie 09/11/18	
22	▲ Videoportero	9 días	mar 23/10/18	vie 02/11/18	
23	Montaje	4 días	mar 23/10/18	vie 26/10/18	
24	Estudio y preparación del servidor	5 días	lun 29/10/18	vie 02/11/18	
25	▲ Desarrollo aplicación BUSing	15 días	lun 12/11/18	vie 30/11/18	
26	Estudio de especificaciones	4 días	lun 12/11/18	jue 15/11/18	
27	Programación del panel	4 días	mié 14/11/18	lun 19/11/18	
28	Aplicación	7 días	jue 15/11/18	vie 23/11/18	
29	Implementación conjunta de aplicaciones	5 días	lun 26/11/18	vie 30/11/18	28
30	Diseño de aplicaciones	20 días	lun 03/12/18	vie 28/12/18	25
31	Documentación	31 días	lun 03/12/18	lun 14/01/19	25

# 11. Referencias

- [1] <http://ingeniumsl.com/website/> Última visita 09/01/2019
- [2] García Pañeda, Xabiel. *Servicios Multimedia e Interactivos*. Año 2015 **Universidad de Oviedo**.
- [3] *Libro BUSing®*. **INGENIUM** Ingeniería y Domótica, S.L.
- [4] <https://ortizol.blogspot.com/2014/02/constructores-y-herencia-en-c.html> Última visita 09/01/2019
- [5] <https://cipsa.net/lenguajes-programacion-mas-populares-2018-ranking-tiobe/> Última visita 09/01/2019
- [6] <https://docs.oracle.com/javase/6/docs/api/> Última visita 09/01/2019
- [7] <https://developer.android.com/guide/platform/> Última visita 09/01/2019
- [8] <https://www.embarcadero.com/> Última visita 09/01/2019
- [9] <https://developer.android.com/reference/android/net/sip/package-summary> Última visita 09/01/2019
- [10] [https://www.pjsip.org/sip\\_media\\_features.htm](https://www.pjsip.org/sip_media_features.htm) Última visita 09/01/2019
- [11] [http://docwiki.embarcadero.com/RADStudio/Tokyo/en/Java2OP.exe,\\_the\\_Native\\_Bridge\\_File\\_Generator\\_for\\_Android](http://docwiki.embarcadero.com/RADStudio/Tokyo/en/Java2OP.exe,_the_Native_Bridge_File_Generator_for_Android) Última visita 09/01/2019
- [12] <https://stackoverflow.com/questions/52291105/instantiate-android-sipmanager-in-c-builder> Última visita 09/01/2019
- [13] <https://trac.pjsip.org/repos/wiki/Getting-Started/Android#BuildPreparation> Última visita 09/01/2019
- [14] <https://www.pjsip.org/docs/book-latest/html/index.html> Última visita 09/01/2019
- [15] <https://www.pjsip.org/docs/book-latest/html/genindex.html> Última visita 09/01/2019
- [16] [https://www.2n.cz/es\\_ES/productos/intercomunicadores/2n-helios-ip-vario](https://www.2n.cz/es_ES/productos/intercomunicadores/2n-helios-ip-vario) Última visita 09/01/2019





Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

LAURA RICO ÁLVAREZ

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

## **DESARROLLO DE SERVICIO SIP PARA VIDEOPORTERO**

Anexos

ENERO 2019





# Índice general

1.- Código fuente de la clase “MainActivity” .....	5
2.- Código fuente de la clase “MainAction” .....	11
3.- Código fuente de la clase “CallActivity” .....	13
4.- Código fuente de la clase “CallAction” .....	21
5.- Código fuente de la clase “MyConfiguration” .....	24
6.- Código fuente de la clase “MyAccount” .....	27
7.- Código fuente de la clase “MyCall” .....	28
8.- Código de la clase “MyMessages” .....	30
9.- Código fuente de la pantalla “activity_main” .....	31
10.- Código fuente de la pantalla “activity_call” .....	35
11.- Código fuente del “Manifest” de la aplicación del videoportero .....	37
12.- Código fuente de la clase “TMainForm” .....	38
12.1.- ARCHIVO .H.....	38
12.2.- ARCHIVO .CPP .....	51
13.- Código fuente de la clase “myThread” .....	93
13.1.- ARCHIVO .H.....	93
13.2.- ARCHIVO .CPP .....	94
14.- Código fuente del archivo “global” .....	102
14.1.- ARCHIVO .H.....	102
14.2.- ARCHIVO .CPP .....	103
15.- Código fuente del archivo “global_var” .....	108
15.1.- ARCHIVO .H.....	108
15.2.- ARCHIVO .CPP .....	109
16.- Código fuente del archivo “global_struct” .....	111
17.- Código fuente del “Manifest” de la aplicación BUSing® .....	113

# 1.- Código fuente de la clase “MainActivity”

```
package com.example.laura.videoporteropantalla;
//Librerías importadas de Android
import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.os.Handler;
import android.os.Message;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.view.Window;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;
//Librerías importadas de pjsip, compiladas previamente
import org.pjsip.pjsua2.CallInfo;
import org.pjsip.pjsua2.CallOpParam;
import org.pjsip.pjsua2.pjsip_inv_state;
import org.pjsip.pjsua2.pjsip_status_code;

/**CLASE MAINACTIVITY**/
//Clase única y principal de este archivo
//En esta clase se configuran todos los aspectos relativos al cambio
físico de la pantalla principal
public class MainActivity extends AppCompatActivity implements
Handler.Callback, MyMessages{

    /**DECLARACIÓN DE VARIABLES**/
    //Objetos físicos de la pantalla
    private EditText mServer, mUsername, mPassword;
    private TextView mTitle, mTitleView;
    private Button mRegister;
    private ImageView mView, mUnreg, mBack, mRectangle;

    //Objetos de las clases creadas
    public MyConfiguration config = null;
    private MainAction action;
    public static MyCall currentCall = null;

    //Variables auxiliares
    private String lastRegStatus = "";
    public Message m;
    private final Handler handler = new Handler(this);
    private boolean checking;
    public static boolean interaction = false;

    //Variables estáticas
    public class MSG_TYPE {
        public final static int REG_STATE = 1;
        public final static int CALL_STATE = 2;
        public final static int CALL_MEDIA_STATE = 3;
        public final static int INCOMING_CALL = 4;
    }
}
```

```
}

/**EVENTO onCreate**/
//Este evento se ejecuta cuando se crea la aplicación
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(R.layout.activity_main);

    checkingOfPermits(); //Función que realiza la comprobación de
los permisos

    //Objetos de la pantalla
    mServer = findViewById(R.id.editTextServer);
    mUsername = findViewById(R.id.editTextUser);
    mPassword = findViewById(R.id.editTextPass);
    mTitle = findViewById(R.id.textViewTitle);
    mTitleView = findViewById(R.id.textView);
    mRegister = findViewById(R.id.buttonReg);
    mBack = findViewById(R.id.buttonBack);
    mView = findViewById(R.id.imageButtonView);
    mUnreg = findViewById(R.id.imageButtonUnreg);
    mRectangle = findViewById(R.id.imageRectangle);

    mServer.setVisibility(View.VISIBLE);
    mUsername.setVisibility(View.VISIBLE);
    mPassword.setVisibility(View.VISIBLE);
    mTitle.setVisibility(View.VISIBLE);
    mTitleView.setVisibility(View.VISIBLE);
    mRegister.setVisibility(View.VISIBLE);
    mBack.setVisibility(View.GONE);
    mView.setVisibility(View.GONE);
    mUnreg.setVisibility(View.GONE);
    mRectangle.setVisibility(View.GONE);

    //Instanciación de la nueva configuración a crear
    if (config == null) {
        config = new MyConfiguration();
        config.init(this);
    }
}

/**FUNCIÓN PARA LA COMPROBACIÓN DE LOS PERMISOS**/
private void checkingOfPermits() {
    int permsRequestCode = 100;
    String[] perms = {
        //En este caso es necesario comprobar que los
servicios de grabación de audio y de
        //cámara han sido permitidos
        Manifest.permission.RECORD_AUDIO,
        Manifest.permission.CAMERA};
    int audioPermission =
ContextCompat.checkSelfPermission(this, Manifest.permission.RECORD_AUDI
O);
    int cameraPermission =
ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA);

    if (cameraPermission == PackageManager.PERMISSION_GRANTED &&
audioPermission == PackageManager.PERMISSION_GRANTED)
{
```

```
    } else {
        ActivityCompat.requestPermissions(this, perms,
permsRequestCode);
    }
}

/**EVENTO DE RESPUESTA DE LOS PERMISOS**/
@Override
public void onRequestPermissionsResult(int requestCode, String
permissions[], int[] grantResults) {
    switch (requestCode) {
        case 100:
            break;
    }
}

/**FUNCIÓN PARA REGISTRAR LA CUENTA**/
//Esta función se ejecuta cuando se pulsa el botón de "REGISTER"
public void registerAccount(View view){

    //Se obtienen las variables de la pantalla
    String server = mServer.getText().toString();
    String username = mUsername.getText().toString();
    String password = mPassword.getText().toString();

    String idUri = "sip:"+username+"@"+server;
    String registrarUri = "sip:"+server;

    //Instanciación en la sub-clase creada que realiza las
acciones de la pantalla principal
    action = new MainAction();
    //Se comprueba la conexión
    checking =
action.registration(idUri, registrarUri, username, password);
}

/**FUNCIÓN PARA DESREGISTRAR LA CUENTA Y CERRAR LA APLICACIÓN**/
//Esta función se ejecuta cuando se pulsa el botón "X"
public void unregisterAccount(View view){
    config.deinit();
    finish();
    Runtime.getRuntime().gc();
    android.os.Process.killProcess(android.os.Process.myPid());
}

/**FUNCIÓN PARA VISUALIZAR LA CÁMARA AUNQUE NO LLAMEN**/
//Esta función se ejecuta cuando se pulsa el botón del ojo.
public void watchCamera (View view){
    if (currentCall != null) {
        return;
    }
    //Se llama al servidor
    currentCall = action.outgoing("sip:111@192.168.0.37");
    //Se muestra la pantalla de la cámara
    showCallActivity();
}

/**FUNCIÓN QUE MANEJA LOS MENSAJES RECIBIDOS DE LAS OTRAS
CLASES**/
//Esta función se ejecuta conjuntamente con la interfaz
MyMessages.java
```

```
public boolean handleMessage(Message m) {
    //Si el mensaje es 0, significa que ha habido un error o que
    la aplicación se ha cerrado
    if (m.what == 0) {
        config.deinit();
        finish();
        Runtime.getRuntime().gc();

        android.os.Process.killProcess(android.os.Process.myPid());

        //Si el mensaje es REG_STATE significa que el estado es
        CONECTADO
    } else if (m.what == MSG_TYPE.REG_STATE) {
        lastRegStatus = (String) m.obj;
        if (checking){
            //Se realizan cambios físicos en la pantalla
            mServer.setVisibility(View.GONE);
            mUsername.setVisibility(View.GONE);
            mPassword.setVisibility(View.GONE);
            mTitle.setVisibility(View.GONE);
            mTitleView.setVisibility(View.GONE);
            mRegister.setVisibility(View.GONE);
            mView.setVisibility(View.VISIBLE);
            mUnreg.setVisibility(View.VISIBLE);
            mBack.setVisibility(View.VISIBLE);
            mRectangle.setVisibility(View.VISIBLE);

        } else{
            mTitle.setVisibility(View.GONE);
        }

        //Si el mensaje es CALL_STATE significa que hay una
        llamada en curso
    } else if (m.what == MSG_TYPE.CALL_STATE) {
        //Se obtiene la información de la llamada
        CallInfo ci = (CallInfo) m.obj;
        if (CallActivity.handler_ != null) {
            Message m2 = Message.obtain(CallActivity.handler_,
            MSG_TYPE.CALL_STATE, ci);
            m2.sendToTarget();
        }

        //Si el mensaje es CALL_MEDIA_STATE significa que hay una
        llamada media en curso
    } else if (m.what == MSG_TYPE.CALL_MEDIA_STATE) {
        if (CallActivity.handler_ != null) {
            Message m2 = Message.obtain(CallActivity.handler_,
            MSG_TYPE.CALL_MEDIA_STATE, null);
            m2.sendToTarget();
        }

        //Si el mensaje es INCOMING_CALL significa que se está
        recibiendo una llamada
    } else if (m.what == MSG_TYPE.INCOMING_CALL) {

        //Se instancia la llamada y sus parámetros
        final MyCall call = (MyCall) m.obj;
        CallOpParam prm = new CallOpParam();

        if (currentCall != null){
            call.delete();
        }
    }
}
```



```
        return true;
    }

    //Código de llamada: SONANDO
    prm.setStatusCode(pjsip_status_code.PJSIP_SC_RINGING);
    /** PARA CONTESTAR DIRECTAMENTE **/
    try {
        //Código de llamada: CONTESTADO
        prm.setStatusCode(pjsip_status_code.PJSIP_SC_OK);
        call.answer(prm);
    } catch (Exception e) {
        System.out.println(e.getCause() + "," +
e.getMessage());
    }
    /** PARA EL PASO DE LLAMADAS Y LAS LLAMADAS EN PARALELO
**/
    /*
    try {
        //Código de llamada: CONTESTADO
        prm.setStatusCode(pjsip_status_code.PJSIP_SC_OK);
        call.answer(prm);
    } catch (Exception e) {
        System.out.println(e.getCause() + "," +
e.getMessage());
    }*/

    //Se pasa la llamada a la pantalla de la llamada
    currentCall = call;
    showCallActivity();

    } else {
        return false;
    }
    return true;
}

/**FUNCIÓN QUE MUESTRA LA PANTALLA DE LA LLAMADA**/
//Esta función se ejecuta cuando hay una llamada entrante o cuando
se pulsa el botón del ojo
//para ver que hay al otro lado de la cámara
public void showCallActivity(){
    Intent intent = new Intent(this, CallActivity.class);
    intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(intent);
}

/**FUNCIÓN PARA LAS NOTIFICACIONES DE REGISTRO**/
public void notifyRegState(pjsip_status_code code, String reason,
int expiration) {
    String msg_str = "";
    if (expiration == 0) {
        msg_str += "Unregistration";
    }else {
        msg_str += "Registration";
    }
    if (code.swigValue()/100 == 2) {
        msg_str += " successful";
    }else {
        msg_str += " failed: " + reason;
    }
    Message m = Message.obtain(handler, MSG_TYPE.REG_STATE,
```

```
msg_str);
    m.sendToTarget();
}

/**FUNCIÓN PARA LAS NOTIFICACIONES DE ESTADO DE LLAMADA**/
public void notifyCallState(MyCall call) {
    if (currentCall == null || call.getId() !=
currentCall.getId())
        return;

    CallInfo ci;
    try {
        //Información de la llamada
        ci = call.getInfo();
    } catch (Exception e) {
        ci = null;
    }
    Message m = Message.obtain(handler, MSG_TYPE.CALL_STATE, ci);
    m.sendToTarget();

    //Si no hay información y el estado es desconectado, no hay
    llamada
    if (ci != null && ci.getState() ==
pjsip_inv_state.PJSIP_INV_STATE_DISCONNECTED) {
        currentCall = null;
    }
}

/**FUNCIÓN PARA LAS NOTIFICACIONES DE ESTADO DE LLAMADA MEDIA**/
public void notifyCallMediaState(MyCall call) {
    Message m = Message.obtain(handler, MSG_TYPE.CALL_MEDIA_STATE,
null);
    m.sendToTarget();
}

/**FUNCIÓN PARA LA NOTIFICACIÓN DE LLAMADA ENTRANTE**/
@Override
public void notifyIncomingCall(MyCall call) {
    m = Message.obtain(handler, MSG_TYPE.INCOMING_CALL, call);
    m.sendToTarget();
}

/**EVENTO onRestart**/
//Esta función se ejecuta cuando se realiza el Restart de la
aplicación y hace que esta no se
//cierre si no se sale de la aplicación desregistrándose
@Override
public void onRestart() {
    super.onRestart();
    if(interaction) {
        moveTaskToBack(true);
    }
    interaction = false;
}

/**FUNCIÓN PARA VOLVER ATRÁS SIN CERRAR LA APLICACIÓN**/
public void returnBack(View view){
    moveTaskToBack(true);
}
}
```

## 2.- Código fuente de la clase “MainAction”

```
package com.example.laura.videoportero Pantalla;
//Librerías importadas de Android
import android.util.Log;

//Librerías importadas de pjsip, compiladas previamente
import org.pjsip.pjsua2.AccountConfig;
import org.pjsip.pjsua2.AuthCredInfo;
import org.pjsip.pjsua2.CallOpParam;
import org.pjsip.pjsua2.StringVector;

/**CLASE MAINACTION**/
//Clase única y principal de este archivo
//En esta clase se configuran todos los aspectos relativos a la
ejecución de funciones internas de
//la pantalla principal
public class MainAction {

    /**DECLARACIÓN DE VARIABLES**/
    //Objetos de las clases creadas
    public static MyAccount account = null;
    public static AccountConfig accountConfig;

    //Variables auxiliares
    public boolean verification = false;
    public MyCall outgoingCall = null;

    /**FUNCIÓN PARA REGISTRAR LA PANTALLA EN EL SERVIDOR**/
    public boolean registration(String myUri, String myServer, String
myUsername, String myPassword) {

        //Instanciación de la configuración de la cuenta
        accountConfig = new AccountConfig();
        accountConfig.getNatConfig().setIceEnabled(true);
        accountConfig.getMediaConfig().getSrtpUse();
        accountConfig.getVideoConfig().setAutoTransmitOutgoing(true);
        accountConfig.getVideoConfig().setAutoShowIncoming(true);

        //Paso de los parámetros introducidos por pantalla
        accountConfig.setIdUri(myUri);
        accountConfig.getRegConfig().setRegistrarUri(myServer);

        StringVector proxies =
accountConfig.getSipConfig().getProxies();
        proxies.add(myServer);
        accountConfig.getSipConfig().setProxies(proxies);

        AuthCredInfo cred = new AuthCredInfo("digest", "*",
myUsername, 0, myPassword);
        accountConfig.getSipConfig().getAuthCreds().add(cred);
        accountConfig.getNatConfig().setIceEnabled(true);

        //Instanciación de la cuenta
        account = new MyAccount(accountConfig);
        try {
            //Creación de la cuenta
```

```
        account.create(accountConfig);
    } catch (Exception e) {
        System.out.println(e.getCause() + "," + e.getMessage());
    }
    try {
        Thread.sleep(500);
    } catch (Exception e) {
        System.out.println(e.getCause() + "," + e.getMessage());
    }
    if (account.isValid()){
        try{
            account.setRegistration(true);
            verification = true;
        } catch (Exception e) {
            System.out.println(e.getCause() + "," +
e.getMessage());
        }
    }
    else{
        verification = false;
        Log.d("Register", "Invalid Account");
    }
    return verification;
}

/**FUNCIÓN DE LLAMADA SALIENTE**/
public MyCall outgoing(String numberCall) {

    //Instanciación de la llamada y sus parámetros
    MyCall call = new MyCall(account, -1);
    CallOpParam prm = new CallOpParam(true);
    try {
        //Realización de la llamada
        call.makeCall(numberCall, prm);
    } catch (Exception e) {
        System.out.println(e.getCause() + "," + e.getMessage());
        call.delete();
        return outgoingCall = null;
    }
    return outgoingCall = call;
}
}
```

## 3.- Código fuente de la clase “CallActivity”

```
package com.example.laura.videoporteropantalla;

//Librerías importadas de Android
import android.app.Activity;
import android.content.Context;
import android.content.res.Configuration;
import android.media.MediaPlayer;
import android.os.Handler;
import android.os.Message;
import android.os.Bundle;
import android.view.Display;
import android.view.Surface;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.WindowManager;
import android.widget.ImageView;

//Librerías importadas de pjsip, compiladas previamente
import org.pjsip.pjsua2.AccountConfig;
import org.pjsip.pjsua2.CallInfo;
import org.pjsip.pjsua2.Endpoint;
import org.pjsip.pjsua2.VideoWindowHandle;
import org.pjsip.pjsua2.pjmedia_orient;
import org.pjsip.pjsua2.pjsip_inv_state;
import org.pjsip.pjsua2.pjsip_role_e;

/**CLASE MAINACTIVITY**/
//Clase única y principal de este archivo
//En esta clase se configuran todos los aspectos relativos al cambio
físico de la pantalla de llamada
public class CallActivity extends Activity implements
Handler.Callback, SurfaceHolder.Callback{

    /**DECLARACIÓN DE VARIABLES**/
    //Objetos físicos de la pantalla
    private SurfaceView cSurface;
    private ImageView cAnswer, cSpeak, cDontSpeak;

    //Objetos de las clases creadas
    private CallAction actionCall = null;
    private MyCall cCall;
    private Endpoint cEp;
    private MyAccount cAccount;
    private AccountConfig cAccountConfig;

    //Objetos necesarios para la configuración
    private static CallInfo lastCallInfo;
    private MediaPlayer player;

    //Variables auxiliares
    public static Handler handler_;
    private final Handler handler = new Handler(this);
    private int cMSG_callState;
    private int cMSG_callStateMedia;
```

```
private boolean inCall = false;
private boolean ringing = false;

/**EVENTO onCreate**/
//Este evento se ejecuta cuando se crea la pantalla de llamada
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_call);

    //Objetos de la pantalla
    cSurface = findViewById(R.id.surfaceView);
    cAnswer = findViewById(R.id.imageButtonAnswer);
    cSpeak = findViewById(R.id.imageButtonSpeak);
    cDontSpeak = findViewById(R.id.imageButtonDontSpeak);

    cDontSpeak.setVisibility(View.GONE);

    //Objetos de la pantalla principal
    cCall = MainActivity.currentCall;
    cMSG_callState = MainActivity.MSG_TYPE.CALL_STATE;
    cMSG_callStateMedia = MainActivity.MSG_TYPE.CALL_MEDIA_STATE;
    cEp = MyConfiguration.ep;
    cAccount = MainAction.account;
    cAccountConfig = MainAction.accountConfig;

    //Instanciación en la sub-clase creada que realiza las
    acciones de la pantalla de llamada
    actionCall = new CallAction();
    MainActivity.interaction = true;

    //Creación del player que ejecutará el tono de llamada
    player = MediaPlayer.create(this,R.raw.ringtone);

    //Creación de la pantalla de recepción del vídeo de la cámara
    if (cCall == null || cCall.vidWin == null)
    {
        cSurface.setVisibility(View.GONE);
    }
    cSurface.getHolder().addCallback(this);
    handler_ = handler;

    //Obtención de la información de la llamada y actualización
    del estado según esta
    if (cCall != null) {
        lastCallInfo = actionCall.getInformation(cCall);
        updateCallState(lastCallInfo);
    } else {
        updateCallState(lastCallInfo);
    }
}

/**EVENTO onConfigurationChanged**/
//Esta función se ejecuta cuando la configuración de la imagen
cambia
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);

    //Declaración de variables internas de la función
    WindowManager wm;
```

```
Display display;
int rotation;
pjmedia_orient orient;

//Se obtiene el servicio de la pantalla y sus características
wm =
(WindowManager) this.getSystemService(Context.WINDOW_SERVICE);
display = wm.getDefaultDisplay();
rotation = display.getRotation();

//Se rota dependiendo de su orientación
switch (rotation) {
    case Surface.ROTATION_0:
        orient = pjmedia_orient.PJMEDIA_ORIENT_ROTATE_270DEG;
        break;
    case Surface.ROTATION_90:
        orient = pjmedia_orient.PJMEDIA_ORIENT_NATURAL;
        break;
    case Surface.ROTATION_180:
        orient = pjmedia_orient.PJMEDIA_ORIENT_ROTATE_90DEG;
        break;
    case Surface.ROTATION_270:
        orient = pjmedia_orient.PJMEDIA_ORIENT_ROTATE_180DEG;
        break;
    default:
        orient = pjmedia_orient.PJMEDIA_ORIENT_UNKNOWN;
}

//Se captura el dispositivo
if (cEp != null && cAccount != null) {
    try {
        AccountConfig cfg = cAccountConfig;
        int cap_dev =
cfg.getVideoConfig().getDefaultCaptureDevice();
        cEp.vidDevManager().setCaptureOrient(cap_dev, orient,
true);
    } catch (Exception e) {
        System.out.println();
    }
}

/**FUNCIÓN QUE MANEJA LOS MENSAJES RECIBIDOS DE LAS OTRAS
CLASES**/
//Esta función se ejecuta conjuntamente con la interfaz
MyMessages.java
@Override
public boolean handleMessage(Message m) {

    //Si la llamada no es media
    if (m.what == cMSG_callState)
    {

        //Obtiene su información
        lastCallInfo = (CallInfo) m.obj;
        updateCallState(lastCallInfo);

        //Si la llamada es media
    } else if (m.what == cMSG_callStateMedia) {

        //Actualiza siempre la visualización de la pantalla
```

```
        if (cCall.vidWin != null) {
onConfigurationChanged(getResources().getConfiguration());
            setupVideoSurface();
        }
    } else {
        return false;
    }
    return true;
}

/**Evento surfaceCreated**/
//Esta función se ejecuta cuando se crea la superficie de la
pantalla
@Override
public void surfaceCreated(SurfaceHolder holder) {

}

/**Evento surfaceChanged**/
//Esta función se ejecuta cuando cambia la superficie de la
pantalla
@Override
public void surfaceChanged(SurfaceHolder holder, int format, int
width, int height) {

    //Actualiza la ventana de vídeo
    updateVideoWindow(true);
}

/**Evento surfaceDestroyed**/
//Esta función se ejecuta cuando se destruye la superficie de la
pantalla
@Override
public void surfaceDestroyed(SurfaceHolder holder) {

    //Actualiza la ventana de vídeo
    updateVideoWindow(false);
}

/**FUNCIÓN DE CONFIGURACIÓN DEL VÍDEO**/
private void setupVideoSurface() {

    //Siempre visible
    cSurface.setVisibility(View.VISIBLE);
}

/**FUNCIÓN DE ACTUALIZACIÓN DE LA VENTANA DE VÍDEO**/
private void updateVideoWindow(boolean show) {

    //Si hay llamada y hay ventana de vídeo
    if (cCall != null && cCall.vidWin != null)
    {
        VideoWindowHandle vidWH = new VideoWindowHandle();
        if (show) {

            //Mostrar la captura del vídeo
vidWH.getHandle().setWindow(cSurface.getHolder().getSurface());

            //Si no, hacer la ventana nula
```



```

    } else {
        vidWH.getHandle().setWindow(null);
    }
    try {
        cCall.vidWin.setWindow(vidWH);
    } catch (Exception e) {
        System.out.println();
    }
}
}

/**FUNCIÓN DE ACTUALIZACIÓN DEL ESTADO DE LA LLAMADA**/
private void updateCallState(CallInfo ci) {

    /**LLAMANTE**/
    if (ci.getRole() == pjsip_role_e.PJSIP_ROLE_UAC) {

        //Pantalla física
        cSpeak.setVisibility(View.VISIBLE);
        cAnswer.setVisibility(View.GONE);
    }

    /**LLAMANDO**/
    if (ci.getState().swigValue() <
pjsip_inv_state.PJSIP_INV_STATE_CONFIRMED.swigValue())
    {

        //Pantalla física
        cSpeak.setVisibility(View.GONE);
        /**LLAMADO**/
        if (ci.getRole() == pjsip_role_e.PJSIP_ROLE_UAS)
        {
            /**INCOMING CALL**/
            //Pantalla física
            cAnswer.setVisibility(View.VISIBLE);

            //Suena el tono
            player.start();
            ringing = true;
        }
        /*
        //LLAMANTE
        else {
            //QUE SE ESCUCHE EL PI, PII ,PIII...
            //PLAYER2.START
        }*/
    }

    /** PARA LLAMADA DIRECTAMENTE CON CÁMARA **/
    /** LLAMADA CONFIRMADA**/
    else if (ci.getState().swigValue() >=
pjsip_inv_state.PJSIP_INV_STATE_CONFIRMED.swigValue())
    {
        if (ci.getState() ==
pjsip_inv_state.PJSIP_INV_STATE_CONFIRMED)
        {
            /**LLAMADO**/
            if (ci.getRole() == pjsip_role_e.PJSIP_ROLE_UAS) {
                if(!inCall) {

                    //Pantalla física

```

```
        cAnswer.setVisibility(View.VISIBLE);
        cSpeak.setVisibility(View.GONE);
        cDontSpeak.setVisibility(View.GONE);

        //Suena el tono
        player.start();
        ringing = true;
    }
}
/**LLAMANTE**/
if (ci.getRole() == pjsip_role_e.PJSIP_ROLE_UAC) {

    //Pantalla física
    cSpeak.setVisibility(View.VISIBLE);
}

/** PARA TRASPASO DE LLAMADA O LLAMADA EN PARALELO **/
/**LLAMADA CONFIRMADA**/
/*
    else if (ci.getState().swigValue() >=
pjsip_inv_state.PJSIP_INV_STATE_CONFIRMED.swigValue())
    {
        if (ci.getState() ==
pjsip_inv_state.PJSIP_INV_STATE_CONFIRMED)
        {
            //LLAMADO
            if (ci.getRole() == pjsip_role_e.PJSIP_ROLE_UAS) {

                //Pantalla física
                mAnswer.setVisibility(View.GONE);
                mSpeak.setVisibility(View.GONE);
                mDontSpeak.setVisibility(View.VISIBLE);

                //Micrófono activado
                actionCall.setSpeakerOn(lastCallInfo);
            }
            //LLAMANTE
            if (ci.getRole() == pjsip_role_e.PJSIP_ROLE_UAC) {

                //Pantalla física
                mSpeak.setVisibility(View.VISIBLE);
            }
        }
    }
*/

/**LLAMADA DESCONECTADA**/
else if (ci.getState() ==
pjsip_inv_state.PJSIP_INV_STATE_DISCONNECTED)
{
    if (ringing){
        //Deja de sonar el tono
        player.pause();
    }
    inCall = false;
    finish();
}
}

/**FUNCIÓN PARA ACEPTAR LA LLAMADA**/
```

```
//Esta función se ejecuta cuando se pulsa el botón verde para
aceptar la llamada
public void acceptCall(View view) {
    /** PARA CONTESTAR DIRECTAMENTE **/
    //Deja de sonar el tono
    player.pause();
    inCall = true;

    //Pantalla física
    cAnswer.setVisibility(View.GONE);
    cSpeak.setVisibility(View.GONE);
    cDontSpeak.setVisibility(View.VISIBLE);
    actionCall.setSpeakerOn(lastCallInfo);

    /** PARA PASAR LLAMADAS Y HACER LLAMADAS EN PARALELO **/
    /*
    //Deja de sonar el tono
    player.pause();

    //Se contesta a la llamada
    actionCall.answerCall();
    inCall = true;
    */
}

/**FUNCIÓN PARA ACTIVAR EL MICRÓFONO**/
//Esta función se ejecuta cuando se pulsa el botón del micrófono
apagado
public void speakOn(View view){

    //Pantalla física
    actionCall.setSpeakerOn(lastCallInfo);
    cSpeak.setVisibility(View.GONE);
    cDontSpeak.setVisibility(View.VISIBLE);
}

/**FUNCIÓN PARA DESACTIVAR EL MICRÓFONO**/
//Esta función se ejecuta cuando se pulsa el botón del micrófono
encendido
public void speakOff(View view){

    //Pantalla física
    actionCall.setSpeakerOff(lastCallInfo);
    cSpeak.setVisibility(View.VISIBLE);
    cDontSpeak.setVisibility(View.GONE);
}

/**FUNCIÓN PARA ABRIR LA PUERTA DEL PORTAL**/
//Esta función se ejecuta cuando se pulsa el botón de la llave
public void openDoor(View view){

    //Deja de sonar el tono
    player.pause();

    //Se activa el micrófono
    actionCall.setSpeakerOn(lastCallInfo);

    //Se mandan los comandos para abrir la puerta
    try {
        cCall.dialDtmf("101");
    } catch (Exception e) {
```

```
e.printStackTrace();
}

//Si no se está en una llamada
if (!inCall) {

    //El micrófono se apaga
    actionCall.setSpeakerOff(lastCallInfo);
}
}

/**FUNCIÓN PARA COLGAR LA LLAMADA**/
//Esta función se ejecuta cuando se pulsa el botón rojo para
colgar la llamada
public void hangupCall(View view) {

    //Finalización de la llamada
    actionCall.endCall();
    handler_ = null;
    finish();
}

/**EVENTO onDestroy**/
//Este evento se ejecuta cuando se destruye la pantalla porque la
llamada ha finalizado
@Override
protected void onDestroy() {

    //Se destruye la pantalla
    super.onDestroy();

    if (ringing){
        //Deja de sonar el tono
        player.stop();
    }
    ringing = false;
    handler_ = null;
}
}
```

## 4.- Código fuente de la clase “CallAction”

```
package com.example.laura.videoporteropantalla;

//Librerías importadas de pjsip, compiladas previamente
import org.pjsip.pjsua2.AudDevManager;
import org.pjsip.pjsua2.AudioMedia;
import org.pjsip.pjsua2.CallInfo;
import org.pjsip.pjsua2.CallMediaInfo;
import org.pjsip.pjsua2.CallMediaInfoVector;
import org.pjsip.pjsua2.CallOpParam;
import org.pjsip.pjsua2.Endpoint;
import org.pjsip.pjsua2.Media;
import org.pjsip.pjsua2.pjmedia_type;
import org.pjsip.pjsua2.pjsip_status_code;
import org.pjsip.pjsua2.pjsua_call_media_status;

/**CLASE CALLACTION**/
//Clase única y principal de este archivo
// En esta clase se configuran todos los aspectos relativos a la
ejecución de funciones internas de
//la pantalla de llamada
public class CallAction {

    /**DECLARACIÓN DE VARIABLES**/
    //Objetos de las clases creadas
    public CallInfo myInfo;
    private MyCall cCall = MainActivity.currentCall;
    private Endpoint cEp = MyConfiguration.ep;

    /**FUNCIÓN PARA OBTENER INFORMACIÓN DE LA LLAMADA**/
    public CallInfo getInformation(MyCall call){
        try {
            myInfo = call.getInfo();
        } catch (Exception e) {
            System.out.println();
        }
        return myInfo;
    }

    /**FUNCIÓN PARA CONTESTAR LA LLAMADA**/
    public void answerCall() {
        CallOpParam prm = new CallOpParam();
        prm.setStatusCode(pjsip_status_code.PJSIP_SC_OK);
        try {
            cCall.answer(prm);
        } catch (Exception e) {
            System.out.println();
        }
    }

    /**FUNCIÓN PARA FINALIZAR LA LLAMADA**/
    public void endCall() {
        if (cCall != null)
        {
```

```
//Instanciación de los parámetros de la llamada
CallOpParam prm = new CallOpParam();

//Cambio del estado de la llamada a declinada
prm.setStatusCode(pjsip_status_code.PJSIP_SC_DECLINE);

//Colgar la llamada
try {
    cCall.hangup(prm);
} catch (Exception e) {
    System.out.println();
}
}

/**FUNCIÓN PARA ACTIVAR EL MICRÓFONO**/
public void setSpeakerOn(CallInfo ci){

    //Obtención de la información media de la llamada
    CallMediaInfoVector cmiv = ci.getMedia();
    for (int i = 0; i < cmiv.size(); i++) {
        CallMediaInfo cmi = cmiv.get(i);

        //Si la llamada es de tipo audio
        if (cmi.getType() == pjmedia_type.PJMEDIA_TYPE_AUDIO &&
            (cmi.getStatus() ==
pjsua_call_media_status.PJSUA_CALL_MEDIA_ACTIVE ||
            cmi.getStatus() ==
pjsua_call_media_status.PJSUA_CALL_MEDIA_REMOTE_HOLD)) {

            //Se activa el audio y el micrófono
            Media m = cCall.getMedia(i);
            AudioMedia am = AudioMedia.typecastFromMedia(m);
            AudDevManager mgr = cEp.audDevManager();
            try {
                am.startTransmit(mgr.getPlaybackDevMedia());
            } catch (Exception e) {
                e.printStackTrace();
            }
            try {
                mgr.getCaptureDevMedia().startTransmit(am);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

/**FUNCIÓN PARA APAGAR EL MICRÓFONO**/
public void setSpeakerOff(CallInfo ci){

    //Se obtiene la información de la llamada
    CallMediaInfoVector cmiv = ci.getMedia();
    for (int i = 0; i < cmiv.size(); i++) {
        CallMediaInfo cmi = cmiv.get(i);

        //Si la llamada es de tipo audio
        if (cmi.getType() == pjmedia_type.PJMEDIA_TYPE_AUDIO &&
            (cmi.getStatus() ==
pjsua_call_media_status.PJSUA_CALL_MEDIA_ACTIVE ||
```

```
        cmi.getStatus() ==  
pjsua_call_media_status.PJSUA_CALL_MEDIA_REMOTE_HOLD)) {  
  
    //Se desactiva el audio y el micrófono  
    Media m = cCall.getMedia(i);  
    AudioMedia am = AudioMedia.typecastFromMedia(m);  
    AudDevManager mgr = cEp.audDevManager();  
    try {  
        am.stopTransmit(mgr.getPlaybackDevMedia());  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    try {  
        mgr.getCaptureDevMedia().stopTransmit(am);  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    }  
    }  
    }  
}
```

## 5.- Código fuente de la clase “MyConfiguration”

```
package com.example.laura.videoportero pantalla;

//Librerías importadas de pjsip, compiladas previamente
import org.pjsip.pjsua2.Endpoint;
import org.pjsip.pjsua2.EpConfig;
import org.pjsip.pjsua2.TransportConfig;
import org.pjsip.pjsua2.pjsip_transport_type_e;

/**CLASE MYCONFIGURATION**/
//Clase única y principal de este archivo
//En esta clase se realiza la carga de librerías y del endpoint,
necesario para la programación
//con PJSIP
public class MyConfiguration {
    static {

        //Carga de la librería OpenH264 que incluye el códec H264
        try{
            System.loadLibrary("openh264");
            System.out.println("Library OPENH264 loaded");
        } catch (UnsatisfiedLinkError e) {
            System.out.println("UnsatisfiedLinkError: " +
e.getMessage());
            System.out.println("This could be safely ignored if you "
+
                "don't need video.");
        }

        //Carga de la librería Libyuv necesaria para el endpoint
        try{
            System.loadLibrary("yuv");
            System.out.println("Library LIBYUV loaded");
        } catch (UnsatisfiedLinkError e) {
            System.out.println("UnsatisfiedLinkError: " +
e.getMessage());
            System.out.println("This could be safely ignored if you "
+
                "don't need video.");
        }

        //Carga de la librería Pjsua2 que incluye todas las clases
necesarias
        try{
            System.loadLibrary("pjsua2");
            System.out.println("Library PJSUA2 loaded");
        } catch (UnsatisfiedLinkError e) {
            System.out.println("UnsatisfiedLinkError: " +
e.getMessage());
        }
    }

    /**DECLARACIÓN DE VARIABLES**/
    //Objetos necesarios de configuración
```



```
public static Endpoint ep;
public static MyMessages messages;
private EpConfig epConfig;
private TransportConfig sipTpConfig;

/**FUNCIÓN DE INICIO DE UN PARÁMETRO**/
public void init(MyMessages msg)
{
    init(msg, false);
}

/**FUNCIÓN DE INICIO DE DOS PARÁMETROS**/
public void init(MyMessages msg, boolean own_worker_thread)
{
    messages = msg;

    //Instanciación de variables
    ep = new Endpoint();
    epConfig = new EpConfig();
    sipTpConfig = new TransportConfig();

    //Creación del Endpoint
    try {
        ep.libCreate();
    } catch (Exception e) {
        System.out.println(e.getCause() + "," + e.getMessage());
        return;
    }

    /*
    UaConfig ua_cfg = epConfig.getUaConfig();
    ua_cfg.setUserAgent("Pjsua2 Android " +
ep.libVersion().getFull());
    StringVector stun_servers = new StringVector();
    ua_cfg.setStunServer(stun_servers);
    if (own_worker_thread) {
        ua_cfg.setThreadCnt(0);
        ua_cfg.setMainThreadOnly(true);
    }
    */

    //Inicio del Endpoint
    try {
        ep.libInit(epConfig);
    } catch (Exception e) {
        System.out.println(e.getCause() + "," + e.getMessage());
        return;
    }

    //Configuración del puerto
    try {
        sipTpConfig.setPort(5060);
    }

    ep.transportCreate(pjsip_transport_type_e.PJSIP_TRANSPORT_UDP,
        sipTpConfig);
    } catch (Exception e) {
        System.out.println(e.getCause() + "," + e.getMessage());
    }

    //Comienzo del Endpoint
    try {
        ep.libStart();
    }
```

```
    } catch (Exception e) {
        System.out.println(e.getCause() + "," + e.getMessage());
        return;
    }
}

/**FUNCIÓN DE DESTRUCCIÓN DEL ENDPOINT**/
public void deinit()
{
    Runtime.getRuntime().gc();

    //Destrucción del endpoint
    try {
        ep.libDestroy();
    } catch (Exception e) {
        System.out.println(e.getCause() + "," + e.getMessage());
    }
    ep.delete();
    ep = null;
}
}
```

## 6.- Código fuente de la clase “MyAccount”

```
package com.example.laura.videoporteropantalla;

//Librerías importadas de pjsip, compiladas previamente
import org.pjsip.pjsua2.*;

/**CLASE MYACCOUNT**/
//Clase única y principal de este archivo
//En esta clase se realizan las funciones de registro y llamada
entrante
public class MyAccount extends Account {

    /**DECLARACIÓN DE VARIABLES**/
    //Objetos necesarios de configuración
    public AccountConfig accConfig;

    /**CONSTRUCTOR**/
    MyAccount(AccountConfig aConfig) {
        super();
        accConfig = aConfig;
    }

    /**EVENTO DE REGISTRO DE LA CUENTA**/
    //Esta función se ejecuta cuando hay algún evento de registro
    @Override
    public void onRegState(OnRegStateParam prm) {

        //Notificación del registro a través de la interfaz MyMessages
        try {
            MyConfiguration.messages.notifyRegState(prm.getCode(),
            prm.getReason(), prm.getExpiration());
        } catch (Exception e) {
            System.out.println(e.getCause() + "," + e.getMessage());
        }
    }

    /**EVENTO DE LLAMADA ENTRANTE**/
    @Override
    public void onIncomingCall(OnIncomingCallParam iprm) {

        //Instanciación de la llamada
        MyCall Call = new MyCall(this, iprm.getCallId());
        //Notificación de la llamada a través de la interfaz
        MyMessages
        MyConfiguration.messages.notifyIncomingCall(Call);
    }
}
```

## 7.- Código fuente de la clase “MyCall”

```
package com.example.laura.videoporteropantalla;

//Librerías importadas de pjsip, compiladas previamente
import org.pjsip.pjsua2.*;

/**CLASE MYCALL**/
//Clase única y principal de este archivo
//En esta clase se realizan las configuraciones de las llamadas
public class MyCall extends Call{

    /**DECLARACIÓN DE VARIABLES**/
    //Objetos necesarios de configuración
    public VideoWindow vidWin;
    public AudioMedia am;

    /**CONSTRUCTOR**/
    MyCall(MyAccount acc, int call_id){
        super(acc, call_id);
        vidWin = null;
    }

    /**EVENTO onCallState**/
    //Esta función se ejecuta cuando hay una llamada
    @Override
    public void onCallState(OnCallStateParam prm)
    {
        //Notificación de la llamada a través de la interfaz
        MyMessages
        MyConfiguration.messages.notifyCallState(this);

        //Obtención de la información de la llamada
        try {
            CallInfo ci = getInfo();

            //Si el estado es desconectado
            if (ci.getState() ==
            pjsip_inv_state.PJSIP_INV_STATE_DISCONNECTED)
            {
                //Se borra la llamada
                this.delete();
            }
        } catch (Exception e) {
            System.out.println(e.getCause() + "," + e.getMessage());
            return;
        }
    }

    /**EVENTO onCallMediaState**/
    //Esta función se ejecuta cuando hay una llamada media
    @Override
    public void onCallMediaState(OnCallMediaStateParam prm)
    {
        CallInfo ci;
```

```
//Se obtiene la información de la llamada
try {
    ci = getInfo();
} catch (Exception e) {
    e.printStackTrace();
    return;
}

//Se obtiene la información media de la llamada
CallMediaInfoVector cmiv = ci.getMedia();

//Para esa información, se va comprobando que estado tiene
for (int i=0; i<cmiv.size(); i++){
    CallMediaInfo cmi = cmiv.get(i);

    //Si el estado es desconectado
    if (ci.getState() ==
pjsip_inv_state.PJSIP_INV_STATE_DISCONNECTED) {

        //Se borra la llamada
        this.delete();

        //Si el tipo de la llamada es de audio
    }else if (cmi.getType() == pjmedia_type.PJMEDIA_TYPE_AUDIO
&&
        (cmi.getStatus() ==
pjsua_call_media_status.PJSUA_CALL_MEDIA_ACTIVE ||
        cmi.getStatus() ==
pjsua_call_media_status.PJSUA_CALL_MEDIA_REMOTE_HOLD)){
        //No se hace nada

        //Si el tipo de la llamada es de vídeo
    }else if (cmi.getType() ==
pjmedia_type.PJMEDIA_TYPE_VIDEO)
    {
        //Se instancia la ventana sobre la que se va a ver el
contenido de la cámara
        vidWin = new
VideoWindow(cmi.getVideoIncomingWindowId());
    }
}

//Notificación de la llamada a través de la interfaz
MyMessages
MyConfiguration.messages.notifyCallMediaState(this);
}
}
```

## 8.- Código de la clase “MyMessages”

```
package com.example.laura.videoporteropantalla;

//Librerías importadas de pjsip, compiladas previamente
import org.pjsip.pjsua2.pjsip_status_code;

/**INTERFAZ MYMESSAGES**/
//Hace de interfaz entre las clases creadas y la pantalla principal
para notificar los eventos
interface MyMessages {

    //Notificación del estado del registro
    abstract void notifyRegState(pjsip_status_code code, String
reason, int expiration);

    //Notificación del estado de la llamada
    abstract void notifyCallState(MyCall call);

    //Notificación del estado de la llamada media
    abstract void notifyCallMediaState(MyCall call);

    //Notificación del estado de la llamada entrante
    abstract void notifyIncomingCall(MyCall call);
}
```

## 9.- Código fuente de la pantalla “activity\_main”

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
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"
android:background="@drawable/fondo"
tools:context=".MainActivity">

<EditText
    android:id="@+id/editTextServer"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="4dp"
    android:layout_marginLeft="4dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="14dp"
    android:layout_marginRight="14dp"
    android:layout_marginBottom="8dp"
    android:ems="10"
    android:inputType="textPersonName"
    android:singleLine="false"
    android:text="@string/server"
    android:textColorLink="@color/colorPrimary"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.537"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView"
    app:layout_constraintVertical_bias="0.060000002" />

<EditText
    android:id="@+id/editTextUser"
    android:layout_width="wrap_content"
    android:layout_height="50dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:ems="10"
    android:inputType="textPersonName"
    android:text="@string/user"
    android:textColorLink="@color/colorPrimary"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.523"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/editTextServer"
    app:layout_constraintVertical_bias="0.0" />
```

```
<Button
    android:id="@+id/buttonReg"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="24dp"
    android:alpha="1"
    android:background="@android:color/darker_gray"
    android:onClick="registerAccount"
    android:text="@string/register"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="1.0" />

<EditText
    android:id="@+id/editTextPass"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:ems="10"
    android:inputType="textPassword"
    android:text="@string/pass"
    android:textColorLink="@color/colorPrimary"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.523"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/editTextUser"
    app:layout_constraintVertical_bias="0.0" />

<TextView
    android:id="@+id/textViewTitle"
    android:layout_width="0dp"
    android:layout_height="49dp"
    android:layout_marginStart="157dp"
    android:layout_marginLeft="157dp"
    android:layout_marginEnd="157dp"
    android:layout_marginRight="157dp"
    android:layout_marginBottom="250dp"
    android:text="@string/title"
    android:textAllCaps="false"
    android:textColor="@android:color/background_light"
    android:textSize="36sp"
    android:textStyle="bold"
    android:typeface="monospace"
    app:fontFamily="sans-serif-smallcaps"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
```



```
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.0" />

<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:text="@string/credentials"
    android:textColor="@android:color/background_light"
    android:textColorLink="@color/colorPrimary"
    android:textSize="18sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textViewTitle" />

<ImageView
    android:id="@+id/imageButtonView"
    android:layout_width="178dp"
    android:layout_height="164dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:onClick="watchCamera"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.743"
    app:srcCompat="@drawable/look" />

<ImageButton
    android:id="@+id/buttonBack"
    android:layout_width="35dp"
    android:layout_height="36dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:background="@android:color/transparent"
    android:onClick="returnBack"
    android:scaleType="centerInside"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.014"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.016"
    app:srcCompat="@drawable/back" />

<ImageButton
    android:id="@+id/imageButtonUnreg"
```

```
android:layout_width="33dp"  
android:layout_height="34dp"  
android:layout_marginStart="8dp"  
android:layout_marginLeft="8dp"  
android:layout_marginTop="8dp"  
android:layout_marginEnd="8dp"  
android:layout_marginRight="8dp"  
android:layout_marginBottom="8dp"  
android:background="@android:color/transparent"  
android:onClick="unregisterAccount"  
android:scaleType="centerInside"  
app:layout_constraintBottom_toBottomOf="parent"  
app:layout_constraintEnd_toEndOf="parent"  
app:layout_constraintHorizontal_bias="0.985"  
app:layout_constraintStart_toStartOf="parent"  
app:layout_constraintTop_toTopOf="parent"  
app:layout_constraintVertical_bias="0.022"  
app:srcCompat="@drawable/cancel" />
```

```
<ImageButton  
  android:id="@+id/imageRectangle"  
  android:layout_width="600dp"  
  android:layout_height="3dp"  
  android:layout_marginStart="8dp"  
  android:layout_marginLeft="8dp"  
  android:layout_marginTop="60dp"  
  android:layout_marginEnd="8dp"  
  android:layout_marginRight="8dp"  
  android:background="@android:color/transparent"  
  android:scaleType="centerCrop"  
  android:tint="@android:color/background_light"  
  app:layout_constraintEnd_toEndOf="parent"  
  app:layout_constraintStart_toStartOf="parent"  
  app:layout_constraintTop_toTopOf="@+id/textViewTitle"  
  app:srcCompat="@drawable/rectangle"  
  tools:srcCompat="@android:color/background_light" />
```

```
</android.support.constraint.ConstraintLayout>
```

# 10.- Código fuente de la pantalla “activity\_call”

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
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"
android:background="@drawable/fondo"
tools:context=".CallActivity">

    <SurfaceView
        android:id="@+id/surfaceView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="62dp"
        android:layout_height="50dp"
        android:layout_marginEnd="16dp"
        android:layout_marginRight="16dp"
        android:layout_marginBottom="80dp"
        android:clickable="true"
        android:onClick="openDoor"
        android:src="@drawable/key"
        app:layout_constraintBottom_toTopOf="@+id/imageButtonHangup"
        app:layout_constraintEnd_toEndOf="parent" />

    <ImageView
        android:id="@+id/imageButtonAnswer"
        android:layout_width="55dp"
        android:layout_height="55dp"
        android:layout_marginEnd="20dp"
        android:layout_marginRight="20dp"
        android:layout_marginBottom="16dp"
        android:onClick="acceptCall"
        android:src="@drawable/hangup"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/imageButtonHangup" />

    <ImageView
        android:id="@+id/imageButtonHangup"
        android:layout_width="55dp"
        android:layout_height="55dp"
        android:layout_marginEnd="16dp"
        android:layout_marginRight="16dp"
        android:layout_marginBottom="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginStart="16dp" />
```

```
        android:onClick="hangupCall"
        android:src="@drawable/hangout"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

<ImageView
    android:id="@+id/imageButtonSpeak"
    android:layout_width="40dp"
    android:layout_height="30dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:layout_marginRight="16dp"
    android:onClick="speakOn"
    android:src="@drawable/muted"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<ImageView
    android:id="@+id/imageButtonDontSpeak"
    android:layout_width="40dp"
    android:layout_height="30dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:layout_marginRight="16dp"
    android:onClick="speakOff"
    android:src="@drawable/microphone"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

# 11.- Código fuente del “Manifest” de la aplicación del videoportero

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.laura.videoporteropantalla">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission
android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
    <uses-permission
android:name="android.permission.PROCESS_OUTGOING_CALLS" />
    <uses-permission android:name="android.permission.WRITE_SETTINGS"
/>
    <uses-permission
android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission
android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.READ_LOGS" />
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="false" />
    <uses-feature android:name="android.hardware.camera" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat.NoActionBar">
        <activity
            android:name=".MainActivity"
            android:screenOrientation="landscape">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".CallActivity"
            android:screenOrientation="landscape">
        </activity>
    </application>

</manifest>
```

# 12.- Código fuente de la clase “TMainForm”

## 12.1.- ARCHIVO .H

```
//-----  
  
#ifndef MainH  
#define MainH  
  
#include "global_struct.h"  
#include "global_var.h"  
#include "global_def.h"  
#include "global.h"  
#include "ReadingThread.h"  
  
//-----  
  
#include <System.Classes.hpp>  
#include <FMX.Controls.hpp>  
#include <FMX.Forms.hpp>  
#include <FMX.Controls.Presentation.hpp>  
#include <FMX.Edit.hpp>  
#include <FMX.Layouts.hpp>  
#include <FMX.StdCtrls.hpp>  
#include <FMX.TabControl.hpp>  
#include <FMX.Types.hpp>  
#include <FMX.Objects.hpp>  
#include <IdBaseComponent.hpp>  
#include <IdComponent.hpp>  
#include <IdTCPClient.hpp>  
#include <IdTCPConnection.hpp>  
#include <FMX.Memo.hpp>  
#include <FMX.ScrollBox.hpp>
```

```
#include <FMX.ImgList.hpp>
#include <System.ImageList.hpp>

#if defined (_PLAT_ANDROID)
    #include "Androidapi.JNI.SIP.hpp"
    #include <Androidapi.Helpers.hpp>
#endif

//-----
class TMainForm : public TForm
{
    __published: // IDE-managed Components
        TTabControl *TabControlInit;
        TTabItem *Tab_Reg;
        TTabItem *Tab_Main;
        TTabItem *Tab_Menu;
        TLabel *lbl_Title;
        TLabel *lbl_IpPort;
        TEdit *edt_IP;
        TEdit *edt_Port;
        TCornerButton *btt_IpPort;
        TLabel *lbl_UserPass;
        TEdit *edt_User;
        TEdit *edt_Pass;
        TCornerButton *btn_UserPass;
        TLayout *lay_Credentials;
        TLayout *lay_IpPortBtn;
        TLayout *lay_UserPassBtn;
        TLayout *lay_IpPort;
        TLayout *lay_UserPass;
        TImage *img_ilu;
        TImage *img_pers;
        TImage *img_termo;
        TImage *img_fondo;
```

```
TTabControl *TabControlMain;  
TTabItem *Pantalla_Principal;  
TTabItem *Pantalla_Luces;  
TTabItem *Pantalla_Persianas;  
TTabControl *TabControlMenu;  
TTabItem *Pantalla_Menu;  
TTabItem *Pantalla_Paquetes;  
TTabItem *Pantalla_Configuracion;  
TLayout *lay_img1;  
TTabItem *Pantalla_Clima;  
TRectangle *Rectangle2;  
TLayout *lay_img2;  
TRectangle *Rectangle3;  
TRectangle *Rectangle4;  
TImage *image_sip;  
TImage *image_datag;  
TImage *image_config;  
TImage *image_close;  
TLayout *lay_images5;  
TRectangle *Rectangle5;  
TLabel *lbl_menu;  
TTabControl *TabControlConfig;  
TTabItem *Pantalla_Config;  
TTabItem *Pantalla_DateTime;  
TTabItem *Pantalla_Language;  
TTabItem *Pantalla_Appearance;  
TRectangle *Rectangle6;  
TCornerRadius *button_date_time;  
TCornerRadius *button_language;  
TCornerRadius *button_appearance;  
TRectangle *Rectangle7;  
TRectangle *Rectangle8;  
TRectangle *Rectangle9;  
TImage *image_appearance;
```



TImage \*image\_date\_time;  
TImage \*image\_language;  
TRectangle \*Rectangle10;  
TLayout \*cabecera1;  
TLayout \*cabecera1\_buttons;  
TLayout \*lay\_light;  
TLayout \*lay\_pers;  
TLayout \*lay\_termo;  
TLayout \*cabecera2;  
TLayout \*cabecera2\_button;  
TLayout \*cabecera3;  
TLayout \*cabecera3\_button;  
TLayout \*cabecera4;  
TLayout \*cabecera4\_button;  
TLayout \*cabecera5;  
TLayout \*cabecera5\_button;  
TLayout \*lay\_image\_SIP;  
TLayout \*lay\_image\_datag;  
TLayout \*lay\_image\_config;  
TLayout \*lay\_image\_close;  
TLayout \*cabecera6;  
TLayout \*cabecera6\_button;  
TLayout \*cabecera7;  
TLayout \*cabecera7\_button;  
TLayout \*lay\_date\_time;  
TLayout \*lay\_language;  
TLayout \*lay\_appearance;  
TLayout \*cabecera8;  
TLayout \*cabecera8\_button;  
TLayout \*cabecera9;  
TLayout \*cabecera9\_button;  
TLayout \*cabecera10;  
TLayout \*cabecera10\_button;  
TIdTCPClient \*Client;

TIdTCPClient \*Client1;  
TCornerButton \*button\_enviar;  
TCornerButton \*button\_delete;  
TMemo \*MemoEnviado;  
TEdit \*edt\_DirLcl;  
TEdit \*edt\_DirDst;  
TEdit \*edt\_Cmm;  
TEdit \*edt\_Dt1;  
TEdit \*edt\_Dt2;  
TLayout \*cabecera\_datos;  
TLabel \*Source;  
TLabel \*Destiny;  
TLabel \*Command;  
TLabel \*Data1;  
TLabel \*Data2;  
TLayout \*envio\_datos;  
TLayout \*datos;  
TLayout \*lay\_contraseña;  
TLabel \*lbl\_contraseña;  
TEdit \*edt\_contraseña;  
TLayout \*lay\_edt;  
TLayout \*lay\_buttons;  
TCornerButton \*button\_ok;  
TCornerButton \*button\_cancel;  
TLayout \*lay\_fondos;  
TLayout \*lay\_fondos\_left;  
TLayout \*lay\_fondos\_right;  
TImage \*img\_fondo0;  
TImage \*img\_fondo1;  
TImage \*img\_fondo3;  
TImage \*img\_fondo2;  
TLabel \*lbl\_appearance;  
TLabel \*lbl\_language;  
TLabel \*lbl\_datetime;

```
TLabel *lbl_configuration;  
TLabel *lbl_communication;  
TLabel *lbl_ilumination;  
TLabel *lbl_blinds;  
TLabel *lbl_climate;  
TLabel *lbl_time;  
TLabel *lbl_date;  
TTimer *TimerTime;  
TLayout *lay_dates;  
TPanel *Panel1;  
TPanel *Panel2;  
TPanel *Panel3;  
TRectangle *Rectangle1;  
TLayout *lay_ilu2A;  
TPanel *panel_salon;  
TImage *image_salon;  
TLayout *lay_ilu2B;  
TPanel *panel_baño;  
TImage *image_baño;  
TPanel *Panel6;  
TImage *image_persiana;  
TPanel *Panel7;  
TImage *image_clima;  
TPanel *Panel8;  
TPanel *Panel9;  
TPanel *Panel10;  
TPanel *Panel11;  
TLayout *lay_fondos_centerleft;  
TLayout *lay_fondos_centerright;  
TPanel *Panel12;  
TPanel *Panel13;  
TPanel *Panel14;  
TPanel *Panel15;  
TTimer *TimerVisu;
```

TPanel \*Panel16;  
TImage \*imageBig;  
TLayout \*lay\_imageBig;  
TPanel \*Panel17;  
TVertScrollBar \*macro;  
TLayout \*lay\_macro;  
TImage \*back5;  
TImage \*back2;  
TImage \*back3;  
TImage \*back4;  
TImage \*back6;  
TImage \*back7;  
TImage \*back8;  
TImage \*back9;  
TImage \*back10;  
TImage \*menu1;  
TImage \*image\_fondo\_gris;  
TLayout \*lay\_salon;  
TLayout \*lay\_cocina;  
TLayout \*lay\_pasillo;  
TLayout \*lay\_baño;  
TImage \*image\_cocina;  
TPanel \*panel\_cocina;  
TImage \*image\_pasillo;  
TPanel \*panel\_pasillo;  
TLayout \*lay\_hab1;  
TPanel \*panel\_hab1;  
TImage \*image\_hab1;  
TLayout \*lay\_hab2;  
TPanel \*panel\_hab2;  
TImage \*image\_hab2;  
TLayout \*lay\_lectura;  
TPanel \*panel\_lectura;  
TImage \*image\_regulation;

TLayout \*Layout9;  
TLayout \*lay\_image\_security;  
TPanel \*Panel4;  
TImage \*image\_security;  
TLabel \*lbl\_salon;  
TLabel \*lbl\_baño;  
TLabel \*lbl\_cocina;  
TLabel \*lbl\_pasillo;  
TLabel \*lbl\_hab1;  
TLabel \*lbl\_hab2;  
TLabel \*lbl\_lectura;  
TLabel \*lbl\_salon1;  
TLabel \*lbl\_salon2;  
TLabel \*lbl\_ilumination\_main;  
TLabel \*lbl\_blinds\_main;  
TLabel \*lbl\_clima\_main;  
TLabel \*lbl\_intercom;  
TLabel \*lbl\_close;  
TLabel \*lbl\_settings\_menu;  
TLabel \*lbl\_telegrams;  
TLabel \*lbl\_security\_menu;  
TTabItem \*Pantalla\_Seguridad;  
TLayout \*cabecera12;  
TLayout \*cabecera12\_buttons;  
TLabel \*lbl\_security;  
TImage \*back12;  
TRectangle \*Rectangle11;  
TLayout \*lay\_pers0;  
TLayout \*lay\_pers1;  
TLayout \*lay\_pers2;  
TLayout \*Layout5;  
TLayout \*Layout6;  
TLayout \*Layout7;  
TLayout \*Layout8;

TLayout \*Layout10;  
TLayout \*Layout11;  
TLayout \*Layout12;  
TLayout \*Layout13;  
TLayout \*lay\_clima0;  
TLayout \*lay\_clima1;  
TLayout \*lay\_clima2;  
TLayout \*Layout1;  
TLayout \*Layout15;  
TLayout \*Layout16;  
TLayout \*Layout17;  
TLayout \*Layout18;  
TLayout \*Layout19;  
TLayout \*Layout20;  
TLayout \*Layout21;  
TLayout \*lay\_alarms;  
TCornerButton \*button\_alarms;  
TImage \*img\_alarms;  
TLayout \*lay\_passwords;  
TCornerButton \*button\_passwords;  
TImage \*img\_passwords;  
TLabel \*porcentaje\_ilu;  
TTimer \*TimerIllumination;  
TLayout \*lay\_regulacion;  
TTimer \*TimerPersiana;  
TLayout \*lay\_reg\_persiana;  
TLabel \*porcentaje\_pers;  
TLabel \*temp\_ambiente;  
TSpeedButton \*btn\_date\_left;  
TSpeedButton \*btn\_date\_right;  
TLayout \*lay\_date\_bottom;  
TLabel \*lbl\_date\_format;  
TLayout \*lay\_date\_top;  
TLayout \*lay\_date;

```
TLabel *lbl_date_format_title;
TLayout *lay_time;
TLayout *lay_time_bottom;
TSpeedButton *btn_time_left;
TSpeedButton *btn_time_right;
TLabel *lbl_time_format;
TLayout *lay_time_top;
TLabel *lbl_time_format_title;
TLayout *lay_language_select;
TLayout *lay_language_bottom;
TSpeedButton *button_language_left;
TSpeedButton *button_language_right;
TLabel *lbl_choose_language;
TLayout *lay_language_top;
TLabel *lbl_select_language;
TImageList *ImageList_Regulation;
TImageList *ImageList_Blind;
TImageList *ImageList_Generica;
TTimer *TimerConnection;
TCornerButton *navigate;
void __fastcall btn_UserPassClick(TObject *Sender);
void __fastcall img_iluClick(TObject *Sender);
void __fastcall img_persClick(TObject *Sender);
void __fastcall img_termoClick(TObject *Sender);
void __fastcall back2Click(TObject *Sender);
void __fastcall menu1Click(TObject *Sender);
void __fastcall image_datagClick(TObject *Sender);
void __fastcall image_configClick(TObject *Sender);
void __fastcall button_date_timeClick(TObject *Sender);
void __fastcall button_languageClick(TObject *Sender);
void __fastcall button_appearanceClick(TObject *Sender);
void __fastcall back6Click(TObject *Sender);
void __fastcall back8Click(TObject *Sender);
void __fastcall image_sipClick(TObject *Sender);
```

```
void __fastcall btt_IpPortClick(TObject *Sender);  
void __fastcall ClientDisconnected(TObject *Sender);  
void __fastcall Client1Disconnected(TObject *Sender);  
void __fastcall button_enviarClick(TObject *Sender);  
void __fastcall button_deleteClick(TObject *Sender);  
void __fastcall image_closeClick(TObject *Sender);  
void __fastcall ClientConnected(TObject *Sender);  
void __fastcall button_okClick(TObject *Sender);  
void __fastcall button_cancelClick(TObject *Sender);  
void __fastcall TimerTimeTimer(TObject *Sender);  
void __fastcall img_fondoMouseDown(TObject *Sender, TMouseButton
```

Button, TShiftState Shift,

```
float X, float Y);
```

```
void __fastcall TimerVisuTimer(TObject *Sender);
```

```
void __fastcall img_fondoMouseUp(TObject *Sender, TMouseButton
```

Button, TShiftState Shift,

```
float X, float Y);
```

```
void __fastcall imageBigClick(TObject *Sender);
```

```
void __fastcall lay_imageBigClick(TObject *Sender);
```

```
void __fastcall lay_macroClick(TObject *Sender);
```

```
void __fastcall imageClick(TObject *Sender);
```

```
void __fastcall image_securityClick(TObject *Sender);
```

```
void __fastcall image_regulationMouseDown(TObject *Sender,
```

TMouseButton Button, TShiftState Shift,

```
float X, float Y);
```

```
void __fastcall TimerIlluminationTimer(TObject *Sender);
```

```
void __fastcall image_regulationMouseUp(TObject *Sender,
```

TMouseButton Button, TShiftState Shift,

```
float X, float Y);
```

```
void __fastcall lay_regulacionMouseMove(TObject *Sender, TShiftState
```

Shift, float X, float Y);

```
void __fastcall image_persianaMouseDown(TObject *Sender,
```

TMouseButton Button, TShiftState Shift,

```
float X, float Y);
```



```
void __fastcall TimerPersianaTimer(TObject *Sender);
void __fastcall lay_reg_persianaMouseMove(TObject *Sender,
TShiftState Shift, float X,
float Y);
void __fastcall image_persianaMouseUp(TObject *Sender,
TMouseButton Button, TShiftState Shift,
float X, float Y);
void __fastcall btn_date_rightClick(TObject *Sender);
void __fastcall btn_date_leftClick(TObject *Sender);
void __fastcall btn_time_leftClick(TObject *Sender);
void __fastcall btn_time_rightClick(TObject *Sender);
void __fastcall button_language_rightClick(TObject *Sender);
void __fastcall button_language_leftClick(TObject *Sender);
void __fastcall TimerConnectionTimer(TObject *Sender);
void __fastcall image_climaClick(TObject *Sender);
void __fastcall navigateClick(TObject *Sender);

private: // User declarations
public: // User declarations
__fastcall TMainForm(TComponent* Owner);

/*FUNCIONES DEFINIDAS*/

void dateAndTime_settings(); //Actualización de fecha y hora
void inicio(); //Iniciación del hilo de lectura
void sendPacket(Packet p); //Envío del telegrama
void visuActuadores(); //Visualización del estado de los actuadores
void ejemplo_formato(); //Elección de formato de fecha y hora

/*VARIABLES DEFINIDAS*/

//BOOLEANOS
bool pulsacion_larga_iluminacion; //Indica si se realizó una pulsación
//larga en la regulación de la iluminación
```

```
bool pulsacion_larga_persiana; //Indica si se realizó una pulsación
                                //larga en la regulación de la persiana
bool visualizar; //Indica si se visualiza la imagen del fondo mediano

//INTS
int DayType; //Día de la semana
int communicationTag; //Tag que indica que imagen de iluminación fija se
está
                                //seleccionando
int imageTag; //Tag que indica que imagen de fondo en miniatura se está
                                //seleccionando

//FLOAT
float porcentaje_iluminacion; //Porcentaje de iluminación regulada a
enviar
float porcentaje_persiana; //Porcentaje de apertura de persiana a enviar

//BUFFERS
TByteArray buffer_info; //Buffer dinámico de información
TByteArray buffer_out; //Buffer de salida

//ESTRUCTURAS
//struct Packet p1;
struct Packet pSend; //Paquete de envío por pantalla
struct Packet pCommunication; //Paquete de envío por Click

//HILOS
myThread *reading; //Hilo de lectura

//INTENTS
_di_JIntent intentLaunch; //Intento de lanzamiento

//UNSIGNED SHORT
unsigned short Year; //Año actual
```

```
unsigned short Month; //Mes actual
unsigned short Day; //Día actual
unsigned short hour; //Hora actual
unsigned short min; //Minuto actual
unsigned short sec; //Segundo actual
unsigned short msec; //Milisegundo actual
```

```
//DATES
```

```
TDate fecha; //Fecha actual
```

```
//TIMES
```

```
TTime hora; //Hora actual
```

```
//UNICODESTRINGS
```

```
UnicodeString MonthName; //Nombre del mes
```

```
UnicodeString DayName; //Día de la semana
```

```
};
```

```
//-----
```

```
extern PACKAGE TMainForm *MainForm;
```

```
//-----
```

```
#endif
```

## 12.2.- ARCHIVO .CPP

```
//-----
```

```
#include <fmx.h>
```

```
#pragma hdrstop
```

```
#include "Main.h"
```

```
//-----
```

```
#pragma package(smart_init)
```

```
#pragma resource "*.fmx"  
#pragma resource ("*.NmXhdpiPh.fmx", _PLAT_ANDROID)  
#pragma resource ("*.XLgXhdpiTb.fmx", _PLAT_ANDROID)  
#pragma resource ("*.SmXhdpiPh.fmx", _PLAT_ANDROID)  
  
TMainForm *MainForm;  
//-----  
/**  
FUNCIÓN QUE SE EJECUTA AL ABRIR LA APLICACIÓN  
**/  
__fastcall TMainForm::TMainForm(TComponent* Owner)  
    : TForm(Owner)  
{  
    /*Inicilización de variables generales*/  
    reading = NULL; //Hilo de lectura  
    isReading = false; //NO se está leyendo  
    //connectionPantalla = false;  
    visualizar = false; //NO se muestran los fondos  
    mostrar = false; //NO se muestra el Memo  
  
    /*Inicialización de variables referidas a la vivienda*/  
    //Las luces se dan por supuestas todas apagadas  
    salon = false;  
    cocina = false;  
    pasillo = false;  
    baño = false;  
    hab1 = false;  
    hab2 = false;  
    //Variables para la regulación de la iluminación  
    pulsacion_larga_iluminacion = false;  
    porcentaje_iluminacion = 0;  
    dato_regulation = 0;  
    //Variables para la regulación de la persiana  
    pulsacion_larga_persiana = false;
```

```
porcentaje_persiana = 0;
dato_persiana = 0;
//Variable para clima
clima = false;

/*Inicialización de variables de fecha e idioma*/
typedate = 0;
typetime = 0;
translation = 0;

//Formato de fecha y hora
ejemplo_formato();

//Timer que comprueba la hora activado
TimerTime->Enabled = true;

//Pantalla a mostrar
TabControlInit->TabPosition = TTabPosition::None;
TabControlInit->Padding->Top = 0;
TabControlInit->TabIndex = 0;
}
//-----
/**
FUNCIÓN QUE ACTUALIZA LA FECHA Y LA HORA
**/
void TMainForm::dateAndTime_settings()
{
    if (Month == 1){
        MonthName = "January";
    }
    if (Month == 2){
        MonthName = "February";
    }
    if (Month == 3){
```

```
        MonthName = "March";
    }
    if (Month == 4){
        MonthName = "April";
    }
    if (Month == 5){
        MonthName = "May";
    }
    if (Month == 6){
        MonthName = "June";
    }
    if (Month == 7){
        MonthName = "July";
    }
    if (Month == 8){
        MonthName = "August";
    }
    if (Month == 9){
        MonthName = "September";
    }
    if (Month == 10){
        MonthName = "October";
    }
    if (Month == 11){
        MonthName = "November";
    }
    if (Month == 12){
        MonthName = "December";
    }

    DayType = DayOfWeek(fechar);
    switch(DayType)
    {
        case 1:
```

```
        DayName = "Sunday";
        break;
    case 2:
        DayName = "Monday";
        break;
    case 3:
        DayName = "Tuesday";
        break;
    case 4:
        DayName = "Wednesday";
        break;
    case 5:
        DayName = "Thursday";
        break;
    case 6:
        DayName = "Friday";
        break;
    case 7:
        DayName = "Saturday";
        break;
    }

switch(typedate)
{
    case 0 :
        lbl_date->Text = tr(DayName) + ("\n") + IntToStr(Day) +
" " + tr(MonthName) + " " + IntToStr(Year);
        break;
    case 1:
        lbl_date->Text = IntToStr(Day) + " " + tr(MonthName) + "
" + IntToStr(Year);
        break;
    case 2:
        lbl_date->Text = fecha.FormatString("dd/mm/YYYY");
```

```
        break;
    case 3:
        lbl_date->Text = fecha.FormatString("dd/mm/YY");
        break;
    case 4:
        lbl_date->Text = fecha.FormatString("mm/dd/YYYY");
        break;
    case 5:
        lbl_date->Text = fecha.FormatString("mm/dd/YY");
        break;
}

switch(typtime)
{
    case 0 :
        lbl_time->Text = hora.FormatString("HH:nn");
        break;
    case 1:
        lbl_time->Text = hora.FormatString("H:nn");
        break;
}
}
//-----
/**
FUNCIÓN DE TIMER QUE COMPRUEBA CADA SEGUNDO CUANDO
CAMBIA EL MINUTO
**/
void __fastcall TMainForm::TimerTimeTimer(TObject *Sender)
{
    static int prev_min = 0;

    hora = Time();//Manera de obtener la hora
    hora.DecodeTime(&hour, &min, &sec, &msec);
    fecha = Date();//Manera de obtener la fecha
```



```
fecha.DecodeDate(&Year, &Month, &Day);
```

```
if (prev_min != min)
{
prev_min = min;
dateAndTime_settings();
}
}
/**
```

FUNCIÓN QUE PERMITE NAVEGAR POR LAS PANTALLAS SIN  
CONEXION

```
**/
//-----
void __fastcall TMainForm::navigateClick(TObject *Sender)
{
//Manera de acceder sin conexión
TabControlInit->TabPosition = TTabPosition::None;
TabControlInit->Padding->Top = 0;
TabControlInit->TabIndex = 1;

TabControlMain->TabPosition = TTabPosition::None;
TabControlMain->Padding->Top = 0;
TabControlMain->TabIndex = 0;
}
//-----
/**
```

FUNCIÓN QUE REALIZA LA CONEXIÓN DEL CLIENTE A LA PANTALLA  
A TRAVÉS DE

LA DIRECCIÓN IP Y DEL PUERTO

```
**/
void __fastcall TMainForm::btt_IpPortClick(TObject *Sender)
{
Client->Host = edt_IP->Text;
```

```
Client->Port = StrToInt(edt_Port->Text);
try
{
    Client->Connect();
    connectionPantalla = true;
}
catch(...)
{
    Client->Disconnect();
}
}
//-----
/**
```

FUNCIÓN QUE REALIZA LA CONEXIÓN DEL CLIENTE A LA PANTALLA  
A TRAVÉS DEL

SERVIDOR CON EL USUARIO Y LA CONTRASEÑA

\*\*/

```
void __fastcall TMainForm::btn_UserPassClick(TObject *Sender)
```

```
{
    //Primero hay que conectarse al servidor a través del puerto
    //correcto
    Client1->Host = "85.152.52.212";
    Client1->Port = 2024;
    try
    {
        Client1->Connect();
        connectionPantalla = false;
    }
    catch(...)
    {
        Client1->Disconnect();
    }

    //Se le envían las credenciales introducidas por teclado
```

```
Ansistring credentials;
credentials = edt_User->Text + "\n" + edt_Pass->Text + "\n";
Client1->IOHandler->WriteLn(credentials);
int respuesta;
respuesta = Client1->IOHandler->ReadInt32();

//Si el servidor responde con algo distinto de 0
    //significa que ha habido un error
if (respuesta != 0)
{
    Client1->Disconnect();
}

//Si responde con un 0, las credenciales eran correctas
else
{
    //Se desconecta el cliente del servidor
    try
    {
        Client1->Disconnect();
    }
    catch(...)
    {
        Client1->Disconnect();
    }

    //Y se vuelve a conectar a través de otro puerto
    Client1->Host = "85.152.52.212";
    Client1->Port = 2023;
    try
    {
        Client1->Connect();
    }
    catch(...)
```

```
{
    Client1->Disconnect();
}

//Se prepara la conexión a la pantalla
if (Client1->Port == 2023 & Client1->Connected())
{
    Client1->IOHandler->WriteLn("IDETHBUS\r");
    UnicodeString sresponse = "";
    sresponse = Client1->IOHandler->ReadString(9);
    UnicodeString ident;
    ident = sresponse.SubString0(2,7);
    UTF8String ident_utf8;
    ident_utf8 = ident;
    buffer_info.set_length(9);
    buffer_info[0] = 0xEE;
    buffer_info[1] = 0xEE;
    memcpy(&buffer_info[2],ident_utf8.c_str(),7);

    //Se conecta a la pantalla
    Client->Host = "85.152.52.212";
    Client->Port = 12347;
    try
    {
        Client->Connect();
    }
    catch(...)
    {
        Client->Disconnect();
    }
}
}

//-----
```

/\*\*

FUNCIÓN QUE SE EJECUTA SI EL CLIENTE SE HA CONECTADO

\*\*/

```
void __fastcall TMainForm::ClientConnected(TObject *Sender)
{
    Client->IOHandler->Open(); //Abre el buffer de entrada/salida
    Client->IOHandler->InputBuffer->Capacity = 9;
    Client->IOHandler->Write(buffer_info,9,0);
    inicio();
}
```

//-----

/\*\*

FUNCIÓN QUE INSTANCIA EL HILO DE LECTURA Y LO LANZA

\*\*/

```
void TMainForm::inicio()
{
    if (isReading)
    {
        reading->Suspend();
        isReading = false;
    }
    else
    {
        reading = new myThread(true);
        reading->Start();
        isReading = true;
    }

    //Envío del comando de Polling para realizar una lectura del estado
    //de todas las salidas
    Packet polling;

    polling.source = 0xFFFF;
    polling.destiny = 0xFFFF;
```

```
polling.command = 10;
polling.data1 = 0;
polling.data2 = 0;

sendPacket(polling);

//Envío del comando de Polling para realizar una lectura del estado
//del clima
Packet polling_clima;

polling_clima.source = 0xFFFF;
polling_clima.destiny = 9;
polling_clima.command = 10;
polling_clima.data1 = 0;
polling_clima.data2 = 0;

sendPacket(polling_clima);

//TimerConnection->Enabled = true; //Se activa el Timer que controla
//la conexión

//Cambio de pantalla
TabControlInit->TabPosition = TTabPosition::None;
TabControlInit->Padding->Top = 0;
TabControlInit->TabIndex = 1;

TabControlMain->TabPosition = TTabPosition::None;
TabControlMain->Padding->Top = 0;
TabControlMain->TabIndex = 0;
}
//-----
/**
FUNCIÓN QUE ENVÍA LOS TELEGRAMAS
**/
```

```
void TMainForm::sendPacket(Packet p)
{
    //Variables auxiliares
    int aux;
    UnicodeString str="";

    //Telegrama dirigido a la pantalla a través del servidor:
    if (Client->Host == "85.152.52.212")
    {
        buffer_out.set_length(9);
        //Cabecera
        buffer_out[0] = 0xFF;
        buffer_out[1] = 0xFF;
        buffer_out[2] = (p.destiny & 0xFF00) >> 8; //H
        buffer_out[3] = p.destiny & 0xFF; //L
        buffer_out[4] = 0xFE; //H
        buffer_out[5] = 0xFE; //L
        buffer_out[6] = p.command;
        buffer_out[7] = p.data1;
        buffer_out[8] = p.data2;
        Client->IOHandler->Write(buffer_out,9,false); //Envío
    }

    //DTelegrama dirigido directamente a la pantalla
    else
    {
        buffer_out.set_length(7);
        //Se envía sin cabecera
        buffer_out[0] = 0xFF;
        buffer_out[1] = 0xFF;
        buffer_out[2] = (p.destiny & 0xFF00) >> 8;
        buffer_out[3] = p.destiny & 0xFF;
        buffer_out[4] = p.command;
        buffer_out[5] = p.data1;
```

```
        buffer_out[6] = p.data2;
        Client->IOHandler->Write(buffer_out,7,false); //Envío
    }

//Escritura del buffer en el Memo
if(mostrar)
{
    str = UnicodeString("\t<-\t" +
                        IntToStr(p.destiny) +
                        "\t" + IntToStr(p.source) +
                        "\t" + IntToStr(p.command) +
                        "\t" + IntToStr(p.data1) +
                        "\t" + IntToStr(p.data2));
    MemoEnviado->Lines->Insert(0,str);
}
}

//-----
/**
FUNCIÓN DE TIMER QUE COMPRUEBA CADA MINUTO SI HA
FALLADO LA CONEXIÓN
**/
void __fastcall TMainForm::TimerConnectionTimer(TObject *Sender)
{
    if(!Client->Connected())
    {
        ShowMessage(tr("Connection failed"));
        TimerConnection->Enabled = false;
    }

//Cambio de pantalla
TabControlInit->TabPosition = TTabPosition::None;
TabControlInit->Padding->Top = 0;
TabControlInit->TabIndex = 0;
}
}
```



//-----

/\*\*

FUNCIÓN PARA ACCEDER AL APARTADO DE ILUMINACIÓN

\*\*/

```
void __fastcall TMainForm::img_iluClick(TObject *Sender)
```

```
{
```

```
    //Cambio de pantalla
```

```
    TabControlInit->TabPosition = TTabPosition::None;
```

```
    TabControlInit->Padding->Top = 0;
```

```
    TabControlInit->TabIndex = 1;
```

```
    TabControlMain->TabPosition = TTabPosition::None;
```

```
    TabControlMain->Padding->Top = 0;
```

```
    TabControlMain->TabIndex = 1;
```

```
}
```

//-----

/\*\*

FUNCIÓN PARA EL ENCENDIDO O APAGADO DE LAS DIFERENTES  
LUCES FIJAS

DE LA CASA

\*\*/

```
void __fastcall TMainForm::imageClick(TObject *Sender)
```

```
{
```

```
    communicationTag = ((TImage*)Sender)->Tag;
```

```
    //Telegrama
```

```
    pCommunication.source = 255;
```

```
    pCommunication.destiny = 2;
```

```
    pCommunication.command = 4;
```

```
    pCommunication.data1 = 2;
```

```
    switch(communicationTag)
```

```
    {
```

```
        case 0:
```

```
        pCommunication.data2 = (salon ? 8 : 0);
        break;
    case 1:
        pCommunication.data2 = (cocina ? 9 : 1);
        break;
    case 2:
        pCommunication.data2 = (hab1 ? 10 : 2);
        break;
    case 3:
        pCommunication.data2 = (hab2 ? 11 : 3);
        break;
    case 4:
        pCommunication.data2 = (pasillo ? 12 : 4);
        break;
    case 5:
        pCommunication.data2 = (baño ? 13 : 5);
        break;
}

sendPacket(pCommunication); //Envío del telegrama
reading->simulatePacket(pCommunication); //Simulación del envío
}
//-----
/**
FUNCIÓN DE EVENTO QUE INDICA CUANDO SE PULSA LA IMAGEN
DE REGULACIÓN
LUMÍNICA
**/
void __fastcall TMainForm::image_regulationMouseDown(TObject *Sender,
TMouseButton Button,
TShiftState Shift, float X, float Y)
{
    //Activación del timer que controla la duración de la pulsación de
    //la imagen de regulación lumínica
```

```
TimerIllumination->Enabled = true;
}
//-----
/**
FUNCIÓN DE TIMER PARA CONTROLAR LA DURACIÓN DE LA
PULSACIÓN DE LA
IMAGEN DE REGULACIÓN LUMÍNICA
**/
void __fastcall TMainForm::TimerIlluminationTimer(TObject *Sender)
{
    pulsacion_larga_iluminacion = true;
    TimerIllumination->Enabled = false;
    lay_regulacion->Visible = true;
}
//-----
/**
FUNCIÓN DE EVENTO QUE INDICA CUANDO SE MUEVE EL
RATÓN/DEDO EN LA IMAGEN
DE REGULACIÓN LUMÍNICA
**/
void __fastcall TMainForm::lay_regulacionMouseMove(TObject *Sender,
TShiftState Shift, float X,
float Y)
{
    porcentaje_ilu->Visible = true;
    //Porcentaje de iluminación según la posición del ratón/dedo
    if(Y >= (0.90 * lay_regulacion->Height))
    {
        porcentaje_iluminacion = 0;
    }
    else if(Y <= (0.10 * lay_regulacion->Height))
    {
        porcentaje_iluminacion = 255;
    }
}
```

```

else if(((0.10 * lay_regulacion->Height) <= Y <= (0.90 * lay_regulacion-
>Height))
{
    int altura = lay_regulacion->Height;
    porcentaje_iluminacion = 255 - (((Y-(0.10 * lay_regulacion-
>Height)) / (altura -(0.20 * lay_regulacion->Height))) * 255);
}
porcentaje_ilu->Text = FloatToStr(round((porcentaje_iluminacion / 255)
* 100)) + "%";
}
//-----
/**
FUNCIÓN DE EVENTO QUE INDICA CUANDO SE SUELTA LA IMAGEN
DE REGULACIÓN
LUMÍNICA
**/
void __fastcall TMainForm::image_regulationMouseUp(TObject *Sender,
TMouseButton Button,
TShiftState Shift, float X, float Y)
{
    porcentaje_ilu->Visible = false;

    //Telegrama
    pCommunication.source = 255;
    pCommunication.destiny = 5;
    pCommunication.command = 4;
    pCommunication.data1 = 0;

    TimerIllumination->Enabled = false;

    lay_regulacion->Visible = false;

    //Pulsación corta -> Va a los extremos
    if(!pulsacion_larga_iluminacion)

```

```
{  
    //pCommunication.data2 = (dato_regulation ? 255 | 0);  
  
    if(dato_regulation == 0)  
    {  
        pCommunication.data2 = 255;  
    }  
    else  
    {  
        pCommunication.data2 = 0;  
    }  
}  
  
//Pulsación larga -> Manda según la posición del ratón/dedo  
else  
{  
    pCommunication.data2 = porcentaje_iluminacion;  
}
```

```
pulsacion_larga_iluminacion = false;
```

```
sendPacket(pCommunication); //Envío del telegrama  
reading->simulatePacket(pCommunication); //Simulación del telegrama
```

```
}  
//-----  
/**
```

**FUNCIÓN QUE ACTUALIZA LAS IMÁGENES DE LAS SALIDAS SEGÚN  
SU ESTADO**

```
**/  
void TMainForm::visuActuadores()  
{  
    TSizeF Size(90,90);  
  
    //Iluminación fija
```

```
image_salon->Bitmap = ImageList_Generica->Bitmap(Size, (salon ? 1 :
0));
image_cocina->Bitmap = ImageList_Generica->Bitmap(Size, (cocina ? 1
: 0));
image_hab1->Bitmap = ImageList_Generica->Bitmap(Size, (hab1 ? 1 :
0));
image_hab2->Bitmap = ImageList_Generica->Bitmap(Size, (hab2 ? 1 :
0));
image_pasillo->Bitmap = ImageList_Generica->Bitmap(Size, (pasillo ? 1
: 0));
image_baño->Bitmap = ImageList_Generica->Bitmap(Size, (baño ? 1 :
0));

//Iluminación regulable
image_regulation->Bitmap = ImageList_Regulation->Bitmap(Size,
round((dato_regulation * 10) / 255));

//Persianas
image_persiana->Bitmap = ImageList_Blind->Bitmap(Size,
round(dato_persiana / 10));

//Clima
image_clima->Bitmap = ImageList_Generica->Bitmap(Size, (clima ? 3 :
2));
temp_ambiente->Text = FloatToStr((dato_clima * 51) / 255.0) + (L"°C");
}
//-----
/**
FUNCIÓN PARA VOLVER A LA PANTALLA PRINCIPAL
**/
void __fastcall TMainForm::back2Click(TObject *Sender)
{
//Cambio de pantalla
TabControlInit->TabPosition = TTabPosition::None;
```

```
TabControlInit->Padding->Top = 0;
```

```
TabControlInit->TabIndex = 1;
```

```
TabControlMain->TabPosition = TTabPosition::None;
```

```
TabControlMain->Padding->Top = 0;
```

```
TabControlMain->TabIndex = 0;
```

```
}
```

```
//-----
```

```
/**
```

```
FUNCIÓN PARA ACCEDER AL APARTADO DE PERSIANAS
```

```
**/
```

```
void __fastcall TMainForm::img_persClick(TObject *Sender)
```

```
{
```

```
    //Cambio de pantalla
```

```
    TabControlInit->TabPosition = TTabPosition::None;
```

```
    TabControlInit->Padding->Top = 0;
```

```
    TabControlInit->TabIndex = 1;
```

```
    TabControlMain->TabPosition = TTabPosition::None;
```

```
    TabControlMain->Padding->Top = 0;
```

```
    TabControlMain->TabIndex = 2;
```

```
}
```

```
//-----
```

```
/**
```

```
FUNCIÓN DE EVENTO QUE INDICA CUANDO SE PULSA LA IMAGEN  
DE REGULACIÓN
```

```
DE PERSIANAS
```

```
**/
```

```
void __fastcall TMainForm::image_persianaMouseDown(TObject *Sender,  
TMouseButton Button,
```

```
TShiftState Shift, float X, float Y)
```

```
{
```

```
    //Activación del timer que controla la duración de la pulsación de
```

```
    //la imagen de regulación de persianas
```

```
TimerPersiana->Enabled = true;
}
//-----
/**
FUNCIÓN DE TIMER PARA CONTROLAR LA DURACIÓN DE LA
PULSACIÓN DE LA
IMAGEN DE REGULACIÓN DE PERSIANAS
**/
void __fastcall TMainForm::TimerPersianaTimer(TObject *Sender)
{
    pulsacion_larga_persiana = true;
    TimerPersiana->Enabled = false;
    lay_reg_persiana->Visible = true;
}
//-----
/**
FUNCIÓN DE EVENTO QUE INDICA CUANDO SE MUEVE EL
RATÓN/DEDO EN LA IMAGEN
DE REGULACIÓN DE PERSIANAS
**/
void __fastcall TMainForm::lay_reg_persianaMouseMove(TObject *Sender,
TShiftState Shift,
float X, float Y)
{
    porcentaje_pers->Visible = true;
    //Porcentaje de posición de la persiana según la posición del
    //ratón/dedo
    if(Y >= (0.90 * lay_reg_persiana->Height))
    {
        porcentaje_persiana = 0;
    }
    else if(Y <= (0.10 * lay_reg_persiana->Height))
    {
        porcentaje_persiana = 100;
    }
}
```



```

}
else if((0.10 * lay_reg_persiana->Height) <= Y <= (0.90 *
lay_reg_persiana->Height))
{
    int altura = lay_reg_persiana->Height;
    porcentaje_persiana = round(100 - (((Y-(0.10 * lay_reg_persiana-
>Height)) / (altura -(0.20 * lay_reg_persiana->Height))) * 100));
}

porcentaje_pers->Text = FloatToStr(porcentaje_persiana) + "%";
}
//-----
/**
FUNCIÓN DE EVENTO QUE INDICA CUANDO SE SUELTA LA IMAGEN
DE REGULACIÓN
DE PERSIANAS
**/
void __fastcall TMainForm::image_persianaMouseUp(TObject *Sender,
TMouseButton Button,
TShiftState Shift, float X, float Y)
{
    porcentaje_pers->Visible = false;
    TimerPersiana->Enabled = false;
    lay_reg_persiana->Visible = false;

    //Telegrama
    pCommunication.source = 255;
    pCommunication.destiny = 4;
    pCommunication.command = 4;
    pCommunication.data1 = 6;

    //Pulsación corta -> Va a los extremos
    if(!pulsacion_larga_persiana)
    {

```

```
        if(dato_persiana == 0)
        {
            pCommunication.data2 = 100;
        }
        else
        {
            pCommunication.data2 = 0;
        }
    }

    //Pulsación larga -> Manda según la posición del ratón/dedo
    else
    {
        pCommunication.data2 = porcentaje_persiana;
    }

    pulsacion_larga_persiana = false;

    sendPacket(pCommunication); //Envío del telegrama
    reading->simulatePacket(pCommunication); //Simulación del telegrama
}

//-----
/**
FUNCIÓN PARA ACCEDER AL APARTADO DE CLIMATIZACIÓN
**/
void __fastcall TMainForm::img_termoClick(TObject *Sender)
{
    //Cambio de pantalla
    TabControlInit->TabPosition = TTabPosition::None;
    TabControlInit->Padding->Top = 0;
    TabControlInit->TabIndex = 1;

    TabControlMain->TabPosition = TTabPosition::None;
    TabControlMain->Padding->Top = 0;
```

```
TabControlMain->TabIndex = 3;
}
//-----
/**
FUNCIÓN DE CLICK QUE ENCIENDE O APAGA EL TERMOSTATO
**/
void __fastcall TMainForm::image_climaClick(TObject *Sender)
{
    //Telegrama
    pCommunication.source = 255;
    pCommunication.destiny = 9;
    pCommunication.command = 4;
    pCommunication.data1 = 10;
    pCommunication.data2 = (clima ? 0 : 1);

    sendPacket(pCommunication); //Envío del telegrama
    reading->simulatePacket(pCommunication); //Simulación del envío
}
//-----
/**
FUNCIÓN PARA ACCEDER A LA PANTALLA DEL MENU
**/
void __fastcall TMainForm::menu1Click(TObject *Sender)
{
    //Cambio de pantalla
    TabControlInit->TabPosition = TTabPosition::None;
    TabControlInit->Padding->Top = 0;
    TabControlInit->TabIndex = 2;

    TabControlMenu->TabPosition = TTabPosition::None;
    TabControlMenu->Padding->Top = 0;
    TabControlMenu->TabIndex = 0;
}
//-----
```

```
/**
FUNCIÓN PARA ACCEDER A LA APLICACIÓN DEL VIDEOPORTERO
**/
void __fastcall TMainForm::image_sipClick(TObject *Sender)
{
    intentLaunch = TIntent::Create();
    intentLaunch      =      SharedActivity()->getPackageManager()-
>getLaunchIntentForPackage(StringToJString("com.example.laura.videoporteropantalla
"));
    SharedActivity()->startActivity(intentLaunch);
}
//-----
/**
FUNCIÓN PARA ACCEDER A LA PANTALLA DE TELEGRAMAS
**/
void __fastcall TMainForm::image_datagClick(TObject *Sender)
{
    lay_images5->Visible = false;
    lay_macro->Visible = true;
}
//-----
/**
FUNCIÓN PARA INTRODUCIR LA CONTRASEÑA QUE DA EL ACCESO
A LA PANTALLA
DE TELEGRAMAS
**/
void __fastcall TMainForm::button_okClick(TObject *Sender)
{
    UnicodeString contraseña = edt_contraseña->Text;

    //Si la contraseña es correcta se accede a la pantalla de telegramas
    if(contraseña == "Ingen1um")
    {
        lay_images5->Visible = true;;
    }
}
```

```
lay_macro->Visible = false;

//Cambio de pantalla
TabControlInit->TabPosition = TTabPosition::None;
TabControlInit->Padding->Top = 0;
TabControlInit->TabIndex = 2;

TabControlMenu->TabPosition = TTabPosition::None;
TabControlMenu->Padding->Top = 0;
TabControlMenu->TabIndex = 1;

//Se muestran los telegramas en el Memo
mostrar = true;
}

//Si no, se vuelve a la pantalla de menu
else
{
    lay_images5->Visible = true;
    lay_macro->Visible = false;

//Cambio de pantalla
TabControlInit->TabPosition = TTabPosition::None;
TabControlInit->Padding->Top = 0;
TabControlInit->TabIndex = 2;

TabControlMenu->TabPosition = TTabPosition::None;
TabControlMenu->Padding->Top = 0;
TabControlMenu->TabIndex = 0;
}
}

//-----
/**
FUNCIÓN PARA RETROCEDER A LA PANTALLA DEL MENU
```

\*/

```
void __fastcall TMainForm::lay_macroClick(TObject *Sender)
```

```
{
```

```
    lay_images5->Visible = true;
```

```
    lay_macro->Visible = false;
```

```
    //Cambio de pantalla
```

```
    TabControlInit->TabPosition = TTabPosition::None;
```

```
    TabControlInit->Padding->Top = 0;
```

```
    TabControlInit->TabIndex = 2;
```

```
    TabControlMenu->TabPosition = TTabPosition::None;
```

```
    TabControlMenu->Padding->Top = 0;
```

```
    TabControlMenu->TabIndex = 0;
```

```
}
```

```
//-----
```

```
/**
```

FUNCIÓN PARA CANCELAR LA OPCIÓN DE INTRODUCIR LA  
CONTRASEÑA

```
*/
```

```
void __fastcall TMainForm::button_cancelClick(TObject *Sender)
```

```
{
```

```
    lay_images5->Visible = true;
```

```
    lay_macro->Visible = false;
```

```
    //Cambio de pantalla
```

```
    TabControlInit->TabPosition = TTabPosition::None;
```

```
    TabControlInit->Padding->Top = 0;
```

```
    TabControlInit->TabIndex = 2;
```

```
    TabControlMenu->TabPosition = TTabPosition::None;
```

```
    TabControlMenu->Padding->Top = 0;
```

```
    TabControlMenu->TabIndex = 0;
```

```
}
```

```
//-----  
/**  
FUNCIÓN QUE ENVÍA EL TELEGRAMA QUE SE INTRODUCE POR  
PANTALLA  
**/  
void __fastcall TMainForm::button_enviarClick(TObject *Sender)  
{  
    pSend.source = StrToInt(edt_DirLcl->Text);  
    pSend.destiny = StrToInt(edt_DirDst->Text);  
    pSend.command = StrToInt(edt_Cmm->Text);  
    pSend.data1 = StrToInt(edt_Dt1->Text);  
    pSend.data2 = StrToInt(edt_Dt2->Text);  
  
    sendPacket(pSend);  
}  
//-----  
/**  
FUNCIÓN QUE BORRA EL CONTENIDO DEL MEMO  
**/  
void __fastcall TMainForm::button_deleteClick(TObject *Sender)  
{  
    MemoEnviado->Lines->Clear();  
}  
//-----  
/**  
FUNCIÓN PARA VOLVER A LA PANTALLA DEL MENU  
**/  
void __fastcall TMainForm::back6Click(TObject *Sender)  
{  
    //Cambio de pantalla  
    TabControlInit->TabPosition = TTabPosition::None;  
    TabControlInit->Padding->Top = 0;  
    TabControlInit->TabIndex = 2;  
}
```

```
TabControlMenu->TabPosition = TTabPosition::None;  
TabControlMenu->Padding->Top = 0;  
TabControlMenu->TabIndex = 0;
```

```
MemoEnviado->Lines->Clear();
```

```
    mostrar = false;
```

```
}
```

```
//-----
```

```
/**
```

```
FUNCIÓN PARA ACCEDER A LA PANTALLA DE SEGURIDAD
```

```
**/
```

```
void __fastcall TMainForm::image_securityClick(TObject *Sender)
```

```
{
```

```
    //Cambio de pantalla
```

```
    TabControlInit->TabPosition = TTabPosition::None;
```

```
    TabControlInit->Padding->Top = 0;
```

```
    TabControlInit->TabIndex = 2;
```

```
    TabControlMenu->TabPosition = TTabPosition::None;
```

```
    TabControlMenu->Padding->Top = 0;
```

```
    TabControlMenu->TabIndex = 3;
```

```
}
```

```
//-----
```

```
/**
```

```
FUNCIÓN PARA ACCEDER A LA PANTALLA DE CONFIGURACIÓN
```

```
**/
```

```
void __fastcall TMainForm::image_configClick(TObject *Sender)
```

```
{
```

```
    //Cambio de pantalla
```

```
    TabControlInit->TabPosition = TTabPosition::None;
```

```
    TabControlInit->Padding->Top = 0;
```

```
    TabControlInit->TabIndex = 2;
```

```
    TabControlMenu->TabPosition = TTabPosition::None;
```



```
TabControlMenu->Padding->Top = 0;
```

```
TabControlMenu->TabIndex = 2;
```

```
TabControlConfig->TabPosition = TTabPosition::None;
```

```
TabControlConfig->Padding->Top = 0;
```

```
TabControlConfig->TabIndex = 0;
```

```
}
```

```
//-----
```

```
/**
```

FUNCIÓN PARA ACCEDER A LA PANTALLA DE AJUSTES DE FECHA Y

HORA

```
**/
```

```
void __fastcall TMainForm::button_date_timeClick(TObject *Sender)
```

```
{
```

```
    //Cambio de pantalla
```

```
    TabControlInit->TabPosition = TTabPosition::None;
```

```
    TabControlInit->Padding->Top = 0;
```

```
    TabControlInit->TabIndex = 2;
```

```
    TabControlMenu->TabPosition = TTabPosition::None;
```

```
    TabControlMenu->Padding->Top = 0;
```

```
    TabControlMenu->TabIndex = 2;
```

```
    TabControlConfig->TabPosition = TTabPosition::None;
```

```
    TabControlConfig->Padding->Top = 0;
```

```
    TabControlConfig->TabIndex = 1;
```

```
}
```

```
//-----
```

```
/**
```

FUNCIÓN PARA VOLVER A LA PANTALLA DE CONFIGURACIÓN

```
**/
```

```
void __fastcall TMainForm::back8Click(TObject *Sender)
```

```
{
```

```
    //Cambio de pantalla
```

```
TabControlInit->TabPosition = TTabPosition::None;  
TabControlInit->Padding->Top = 0;  
TabControlInit->TabIndex = 2;
```

```
TabControlMenu->TabPosition = TTabPosition::None;  
TabControlMenu->Padding->Top = 0;  
TabControlMenu->TabIndex = 2;
```

```
TabControlConfig->TabPosition = TTabPosition::None;  
TabControlConfig->Padding->Top = 0;  
TabControlConfig->TabIndex = 0;
```

```
}  
//-----  
/**
```

FUNCIÓN PARA LA ELECCIÓN DE FORMATO DE FECHA HACIA LA  
DERECHA

```
*/  
void __fastcall TMainForm::btn_date_rightClick(TObject *Sender)  
{  
    typedate++;  
  
    if (typedate>5)  
    {  
        typedate = 0;  
    }  
  
    ejemplo_formato();  
}  
//-----  
/**
```

FUNCIÓN PARA LA ELECCIÓN DE FORMATO DE FECHA HACIA LA  
IZQUIERDA

```
*/  
void __fastcall TMainForm::btn_date_leftClick(TObject *Sender)
```

```
{
    typedate--;

    if (typedate<0)
    {
        typedate = 5;
    }

    ejemplo_formato();
}
//-----
/**
FUNCIÓN PARA LA ELECCIÓN DE FORMATO DE HORA HACIA LA
DERECHA
**/
void __fastcall TMainForm::btn_time_leftClick(TObject *Sender)
{
    typetime--;

    if (typetime<0)
    {
        typetime = 1;
    }

    ejemplo_formato();
}
//-----
/**
FUNCIÓN PARA LA ELECCIÓN DE FORMATO DE HORA HACIA LA
IZQUIERDA
```

```
**/
void __fastcall TMainForm::btn_time_rightClick(TObject *Sender)
{
    typetime++;
```

```
if (typetime>1)
{
    typetime = 0;
}

ejemplo_formato();
}
//-----
/**
FUNCIÓN QUE MUESTRA LOS DISTINTOS TIPOS DE FORMATO DE
FECHA Y HORA
**/
void TMainForm::ejemplo_formato()
{
    switch(typedate)
    {
        case 0 :
            lbl_date_format->Text = tr("Tuesday") + ",\n25 " +
tr("December") + " 2018";
            break;
        case 1:
            lbl_date_format->Text = "25 " + tr("December") + " 2018";
            break;
        case 2:
            lbl_date_format->Text = ("25/12/2018");
            break;
        case 3:
            lbl_date_format->Text = ("25/12/18");
            break;
        case 4:
            lbl_date_format->Text = ("12/25/2018");
            break;
        case 5:
```

```
        lbl_date_format->Text = ("12/25/18");
        break;
    }

    switch(typtime)
    {
        case 0 :
            lbl_time_format->Text = ("09:50");
            break;
        case 1:
            lbl_time_format->Text = ("9:50");
            break;
    }

    //Actualización de la fecha y la hora según formato
    dateAndTime_settings();
}

//-----
/**
FUNCIÓN PARA ACCEDER A LA PANTALLA DEL IDIOMA
**/
void __fastcall TMainForm::button_languageClick(TObject *Sender)
{
    //Cambio de pantalla
    TabControlInit->TabPosition = TTabPosition::None;
    TabControlInit->Padding->Top = 0;
    TabControlInit->TabIndex = 2;

    TabControlMenu->TabPosition = TTabPosition::None;
    TabControlMenu->Padding->Top = 0;
    TabControlMenu->TabIndex = 2;

    TabControlConfig->TabPosition = TTabPosition::None;
    TabControlConfig->Padding->Top = 0;
```

```
TabControlConfig->TabIndex = 2;

}//-----
/**
FUNCIÓN PARA LA ELECCIÓN DE IDIOMA HACIA LA DERECHA
**/
void __fastcall TMainForm::button_language_rightClick(TObject *Sender)
{
    translation++;

    if (translation>2)
    {
        translation = 0;
    }

    //Carga del fichero según idioma
    load_file();
}
//-----
/**
FUNCIÓN PARA LA ELECCIÓN DE IDIOMA HACIA LA IZQUIERDA
**/
void __fastcall TMainForm::button_language_leftClick(TObject *Sender)
{
    translation--;

    if (translation<0)
    {
        translation = 2;
    }

    //Carga del fichero según idioma
    load_file();
}
//-----
```

/\*\*

FUNCIÓN PARA ACCEDER A LA PANTALLA DE FONDOS DE  
PANTALLA

\*\*/

```
void __fastcall TMainForm::button_appearanceClick(TObject *Sender)
```

```
{
```

```
    //Cambio de pantalla
```

```
    TabControlInit->TabPosition = TTabPosition::None;
```

```
    TabControlInit->Padding->Top = 0;
```

```
    TabControlInit->TabIndex = 2;
```

```
    TabControlMenu->TabPosition = TTabPosition::None;
```

```
    TabControlMenu->Padding->Top = 0;
```

```
    TabControlMenu->TabIndex = 2;
```

```
    TabControlConfig->TabPosition = TTabPosition::None;
```

```
    TabControlConfig->Padding->Top = 0;
```

```
    TabControlConfig->TabIndex = 3;
```

```
    lay_imageBig->Visible = false;
```

```
}
```

```
//-----
```

/\*\*

FUNCIÓN DE EVENTO QUE INDICA CUANDO SE PULSA CUALQUIER  
IMAGEN DE FONDO

EN MINIATURA

\*\*/

```
void __fastcall TMainForm::img_fondoMouseDown(TObject *Sender,
```

```
TMouseButton Button,
```

```
TShiftState Shift, float X, float Y)
```

```
{
```

```
    imageTag = ((TImage*)Sender)->Tag;
```

```
    //Activación del timer que controla la duración de la pulsación de
```

```
    //cualquier imagen de fondo en minuatara
```

```
TimerVisu->Enabled = true;
}
//-----
/**
FUNCIÓN DE TIMER PARA CONTROLAR LA DURACIÓN DE LA
PULSACIÓN DE CUALQUIER
IMAGEN DE FONDO EN MINIATURA
**/
void __fastcall TMainForm::TimerVisuTimer(TObject *Sender)
{
    switch(imageTag)
    {
        case 10:
            imageBig->Bitmap = img_fondo0->Bitmap;
            break;
        case 11:
            imageBig->Bitmap = img_fondo1->Bitmap;
            break;
        case 12:
            imageBig->Bitmap = img_fondo2->Bitmap;
            break;
        case 13:
            imageBig->Bitmap = img_fondo3->Bitmap;
            break;
    }

    //Si salta el Timer indica pulsación larga -> Aparece la imagen de
    //fondo en miniatura en tamaño medio en el medio de la pantalla
    lay_imageBig->Visible = true;
    visualizar = true;
    TimerVisu->Enabled = false;
}
//-----
/**
```



FUNCIÓN DE EVENTO QUE INDICA CUANDO SE SUELTA CUALQUIER  
IMAGEN DE FONDO

EN MINIATURA

\*/

```
void __fastcall TMainForm::img_fondoMouseUp(TObject *Sender,  
TMouseButton Button,
```

```
TShiftState Shift, float X, float Y)
```

```
{
```

```
TimerVisu->Enabled = false;
```

```
//Pulsación corta -> Cambio de fondo actual por la imagen en
```

```
//miniatura seleccionada
```

```
if(!visualizar)
```

```
{
```

```
switch(imageTag)
```

```
{
```

```
case 10:
```

```
img_fondo->Bitmap = img_fondo0->Bitmap;
```

```
break;
```

```
case 11:
```

```
img_fondo->Bitmap = img_fondo1->Bitmap;
```

```
break;
```

```
case 12:
```

```
img_fondo->Bitmap = img_fondo2->Bitmap;
```

```
break;
```

```
case 13:
```

```
img_fondo->Bitmap = img_fondo3->Bitmap;
```

```
break;
```

```
}
```

```
}
```

```
visualizar = false;
```

```
}
```

```
//-----
```

```
/**
```

FUNCIÓN DE CLICK QUE INDICA CUANDO SE HA PULSADO LA  
IMAGEN DE FONDO

MEDIO

\*/

```
void __fastcall TMainForm::imageBigClick(TObject *Sender)
```

```
{
```

```
    visualizar = false;
```

```
    //Cambio de fondo
```

```
    img_fondo->Bitmap = imageBig->Bitmap;
```

```
    lay_imageBig->Visible = false;
```

```
}
```

```
//-----
```

\*/

FUNCIÓN QUE INDICA QUE NO SE QUIERE CAMBIAR LA IMAGEN DE  
FONDO POR LA

DE TAMAÑO MEDIO

\*/

```
void __fastcall TMainForm::lay_imageBigClick(TObject *Sender)
```

```
{
```

```
    visualizar = false;
```

```
    lay_imageBig->Visible = false;
```

```
}
```

```
//-----
```

\*/

FUNCIÓN DE CLICK PARA LA CONEXIÓN/DESCONEXIÓN DEL  
CLIENTE

\*/

```
void __fastcall TMainForm::image_closeClick(TObject *Sender)
```

```
{
```

```
    if(!Client->Connected())
```

```
    {
```

```
        //Cambio de pantalla
```

```
        TabControlInit->TabPosition = TTabPosition::None;
```

```
        TabControlInit->Padding->Top = 0;
```

```
        TabControlInit->TabIndex = 0;

        return;
    }

    if(Client->Connected())
    {
        Client->Disconnect();
    }
    if (Client1->Connected())
    {
        Client1->Disconnect();
    }

    //Termina la aplicación
    Application->Terminate();
}

//-----
/**
FUNCIÓN DE DESCONEXIÓN DEL CLIENTE
**/
void __fastcall TMainForm::ClientDisconnected(TObject *Sender)
{
    //Termina el hilo de lectura
    reading->Terminate();
    isReading = false;
    MemoEnviado->Lines->Clear();
}

//-----
/**
FUNCIÓN DE DESCONEXIÓN DEL CLIENTE1
**/
void __fastcall TMainForm::Client1Disconnected(TObject *Sender)
{
```

```
//Termina el hilo de lectura
reading->Terminate();
isReading = false;
MemoEnviado->Lines->Clear();
}
//-----
```

# 13.- Código fuente de la clase “myThread”

## 13.1.- ARCHIVO .H

```
//-----  
  
#ifndef ReadingThreadH  
#define ReadingThreadH  
//-----  
#include "global_struct.h"  
#include "global_var.h"  
#include <System.Classes.hpp>  
//-----  
class myThread : public TThread  
{  
private:  
protected:  
    void __fastcall Execute();  
public:  
  
    /*FUNCIONES DEFINIDAS*/  
  
    __fastcall myThread(bool CreateSuspended); //Creación del hilo  
    void __fastcall read(); //Lectura del hilo  
    void __fastcall simulatePacket(Packet p); //Simulación de telegrama  
  
    /*VARIABLES DEFINIDAS*/  
  
    //BUFFERS  
    TByteDynArray buffer; //Buffer de lectura
```

```
//PACKETS
Packet pLectura;
};
//-----
#endif
```

### 13.2.- ARCHIVO .CPP

```
//-----

#include <System.hpp>
#pragma hdrstop

#include "ReadingThread.h"
#include "Main.h"

#pragma package(smart_init)
//-----

// Important: Methods and properties of objects in VCL can only be
// used in a method called using Synchronize, for example:
//
//   Synchronize(&UpdateCaption);
//
// where UpdateCaption could look like:
//
//   void __fastcall myThread::UpdateCaption()
//   {
//       Form1->Caption = "Updated in a thread";
//   }
//-----

/**
```

### FUNCIÓN DE CREACIÓN DEL HILO

```

**/
__fastcall myThread::myThread(bool CreateSuspended)
    : TThread(CreateSuspended)
{
    //El hilo se crea vacío
}
//-----
/**

```

### FUNCIÓN DE EJECUCIÓN DEL HILO

```

**/
void __fastcall myThread::Execute()
{
    FreeOnTerminate = true; //Cuando termine se libera

    //Si el cliente está conectado cada vez que el buffer de lectura no esté
    //vacío se leerá su contenido
    while (true)
    {
        try
        {
            if (MainForm->Client->Connected())
            {
                while (!MainForm->Client->IOHandler-
>InputBufferIsEmpty())
                {
                    MainForm->Client->IOHandler-
>ReadBytes(buffer,9,false);
                    Synchronize(read);
                }
            }
        }
        catch(...)
        {

```

```
        break;
    }

    Sleep(400);
}
}

//-----
/**
FUNCIÓN QUE LEE SEGÚN LA CONEXIÓN QUE TENGA EL CLIENTE
CON LA PANTALLA
**/
void __fastcall myThread::read()
{
    UnicodeString str="";

    if(MainForm->Client->Host == "85.152.52.212")
    {
        pLectura.source = (buffer[4] << 8) + buffer[5];
        pLectura.destiny = (buffer[2] << 8) + buffer[3];
        pLectura.command = buffer[6];
        pLectura.data1 = buffer[7];
        pLectura.data2 = buffer[8];
    }

    else
    {
        pLectura.source = (buffer[5] << 8) + buffer[6];
        pLectura.destiny = (buffer[3] << 8) + buffer[4];
        pLectura.command = buffer[2];
        pLectura.data1 = buffer[7];
        pLectura.data2 = buffer[8];
    }
}
```



```

simulatePacket(pLectura); //Se simula el telegrama

//Muestra si se está en la pantalla adecuada
if(mostrar)
{
    str = UnicodeString("\t->\t" +
                                                                IntToStr(pLectura.source)) +
"\t\t" +
                                                                IntToStr(pLectura.destiny) +
"\t\t" +
                                                                IntToStr(pLectura.command)
+ "\t\t" +
                                                                IntToStr(pLectura.data1) +
"\t\t" +
                                                                IntToStr(pLectura.data2);
    MainForm->MemoEnviado->Lines->Insert(0,str);
}
}
//-----
/**
FUNCIÓN QUE SIMULA LOS TELEGRAMAS
**/
void __fastcall myThread::simulatePacket(Packet p)
{
    //Actuador 6E6S -> Iluminación fija
    if((p.destiny == 2) && (p.command == 4))
    {
        if(p.data1 == 2)
        {
            if (p.data2 < 8 )
            {
                // Encender
                int mascara = 0x01 << p.data2;

```

```
        dato_actuador = dato_actuador | mascara;
    }
    else
    {
        // Apagar
        int mascara = 0x01 << (p.data2 - 8);
        mascara = 0xFF ^ mascara;

        dato_actuador = dato_actuador & mascara;
    }
}

if(p.data1 == 1)
{
//Modificar
    dato_actuador = p.data2;
}

//Guardar el estado
if ((dato_actuador & (0x01 << 0))!= 0)
{
    salon = true;
}
else
{
salon = false;
}

if ((dato_actuador & (0x01 << 1))!= 0)
{
    cocina = true;
}
else
{
```

```
        cocina = false;
    }

    if ((dato_actuador & (0x01 << 2))!= 0)
    {
        hab1 = true;
    }
    else
    {
        hab1 = false;
    }

    if ((dato_actuador & (0x01 << 3))!= 0)
    {
        hab2 = true;
    }
    else
    {
        hab2 = false;
    }

    if ((dato_actuador & (0x01 << 4))!= 0)
    {
        pasillo = true;
    }
    else
    {
        pasillo = false;
    }

    if ((dato_actuador & (0x01 << 5))!= 0)
    {
        baño = true;
    }
```

```
        else
        {
baño = false;
        }
    }

//Actuador 2E2S -> Persiana
if((p.destiny == 4) && (p.command == 4) && (p.data1 == 6))
{
    dato_persiana = p.data2;
}

//Regulador 2Canales -> Iluminación regulada
if((p.destiny == 5) && (p.command == 4) && (p.data1 == 0))
{
    dato_regulation = p.data2;
}

//Clima
if((p.destiny == 9) && (p.command == 4) && (p.data1 == 10))
{
    if(p.data2 == 0)
    {
        clima = false;
        //MainForm->temp_ambiente->Text = "";
    }
    else
    {
        clima = true;
    }
}

if((p.destiny == 9) && (p.command == 4) && (p.data1 == 0))
{
    dato_clima = p.data2;
}
```

```
        //MainForm->temp_ambiente->Text = dato_clima;  
    }  
  
    MainForm->visuActuadores();  
}  
  
//-----
```

# 14.- Código fuente del archivo “global”

## 14.1.- ARCHIVO .H

```
//-----  
  
#ifndef globalH  
#define globalH  
  
#include "global_var.h"  
  
//-----  
  
/*FUNCIONES GLOBALES*/  
  
//Carga de ficheros  
void load_file();  
//Búsqueda de la traducción de inglés a ...  
void load_translation(UnicodeString file);  
//Búsqueda de la traducción de inglés a inglés  
void load_en_translation(UnicodeString file);  
//Copia del texto sin traducción  
UnicodeString tr(UnicodeString text);  
//Traducción  
void translate();  
//-----  
#endif
```

## 14.2.- ARCHIVO .CPP

```
//-----  
  
#pragma hdrstop  
  
#include "global.h"  
#include "Main.h"  
#include <System.IOUtils.hpp>  
#include <System.StrUtils.hpp>  
  
//-----  
  
/**  
FUNCIÓN QUE CARGA EL FICHERO DE TRADUCCIÓN  
**/  
void load_file()  
{  
    switch(translation)  
    {  
        case 0:  
            MainForm->lbl_choose_language->Text =  
UnicodeString(L"English");  
            load_en_translation("es_trans.csv");  
            break;  
        case 1:  
            MainForm->lbl_choose_language->Text =  
UnicodeString(L"Español");  
            load_translation("es_trans.csv");  
            break;  
        case 2:  
            MainForm->lbl_choose_language->Text =  
UnicodeString(L"Français");  
            load_translation("fr_trans.csv");
```

```

        break;
    }
}
//-----
/**
FUNCIÓN QUE BUSCA LA TRADUCCIÓN DE INGLÉS A LO QUE SE LE
INDIQUE
**/
void load_translation(UnicodeString file)
{
    TStringList *lines = new TStringList;
    lines->LoadFromFile(System::Iutils::TPath::GetDocumentsPath() +
PathDelim + file);
    for (int i=0; i<lines->Count; i++)
    {
        UnicodeString ansi_str1 = lines->Strings[i];
        System::TStringDynArray tokens =
System::Strutils::SplitString(ansi_str1, ";");
        trans[tokens[0]] = tokens[1];
    }

    //Traducción
    translate();
    return;
}
//-----
/**
FUNCIÓN QUE BUSCA LA TRADUCCIÓN DE INGLÉS A INGLÉS
**/
void load_en_translation(UnicodeString file)
{
    TStringList *lines = new TStringList;
    lines->LoadFromFile(System::Iutils::TPath::GetDocumentsPath() +
PathDelim + file);

```



```
for (int i=0; i<lines->Count; i++)
{
    UnicodeString ansi_str1 = lines->Strings[i];
    System::TStringDynArray tokens =
System::Strutils::SplitString(ansi_str1, ";");
    trans[tokens[0]] = tokens[0];
}

//Traducción
translate();
return;
}

//-----
/**
    FUNCIÓN QUE COPIA EL TEXTO QUE INTENTÓ TRADUCIR SI NO
EXISTE TRADUCCIÓN
    PARA ESE TEXTO
    **/
UnicodeString tr(UnicodeString text)
{
    if (trans[text] != "")
    {
        return trans[text];
    }
    else
    {
        return text;
    }
}

//-----
/**
    FUNCIÓN PARA TRADUCIR
    **/
```

```
void translate()
{
    MainForm->lbl_Title->Text = tr("WELCOME HOME");
    MainForm->lbl_IpPort->Text = tr("Please enter the IP address and
port...");

    MainForm->btt_IpPort->Text = tr("CONNECT");
    MainForm->lbl_UserPass->Text = tr("or your username and password");
    MainForm->btn_UserPass->Text = tr("SEND");
    MainForm->lbl_iluminacion_main->Text = tr("Illumination");
    MainForm->lbl_blinds_main->Text = tr("Blinds");
    MainForm->lbl_clima_main->Text = tr("Climate");
    MainForm->lbl_iluminacion->Text = tr("illumination");
    MainForm->lbl_salon->Text = tr("Living room");
    MainForm->lbl_cocina->Text = tr("Kitchen");
    MainForm->lbl_pasillo->Text = tr("Hall");
    MainForm->lbl_baño->Text = tr("Bathroom");
    MainForm->lbl_hab1->Text = tr("Room") + "1";
    MainForm->lbl_hab2->Text = tr("Room") + "2";
    MainForm->lbl_lectura->Text = tr("Regulation");
    MainForm->lbl_blinds->Text = tr("blinds");
    MainForm->lbl_salon1->Text = tr("Living room");
    MainForm->lbl_climate->Text = tr("climate");
    MainForm->lbl_salon2->Text = tr("Living room");
    MainForm->lbl_menu->Text = tr("MENU");
    MainForm->lbl_intercom->Text = tr("Intercom");
    MainForm->lbl_telegrams->Text = tr("Telegrams");
    MainForm->lbl_security_menu->Text = tr("Security");
    MainForm->lbl_settings_menu->Text = tr("Settings");
    MainForm->lbl_close->Text = tr("Close");
    MainForm->lbl_communication->Text = tr("telegrams");
    MainForm->Source->Text = tr("Source") + ":";
    MainForm->Destiny->Text = tr("Destiny") + ":";
    MainForm->Command->Text = tr("Command") + ":";
    MainForm->Data1->Text = tr("Data") + "1:";
}
```

```
MainForm->Data2->Text = tr("Data") + "2:";
MainForm->button_enviar->Text = tr("SEND");
MainForm->button_delete->Text = tr("DELETE");
MainForm->lbl_configuration->Text = tr("settings");
MainForm->button_date_time->Text = " " + tr("Setting date/time");
MainForm->button_language->Text = " " + tr("Language");
MainForm->button_appearance->Text = " " + tr("Appearance");
MainForm->lbl_datetime->Text = tr("date/time");
MainForm->lbl_date_format_title->Text = tr("Date format");
MainForm->lbl_time_format_title->Text = tr("Time format");
MainForm->lbl_language->Text = tr("language");
MainForm->lbl_appearance->Text = tr("appearance");
MainForm->lbl_security->Text = tr("security");
MainForm->button_alarms->Text = " " + tr("Alarms");
MainForm->button_passwords->Text = " " + tr("Passwords");
MainForm->lbl_contraseña->Text = tr("Password");
MainForm->button_ok->Text = tr("Ok");
MainForm->button_cancel->Text = tr("Cancel");
MainForm->lbl_select_language->Text = tr("Select language");

MainForm->dateAndTime_settings();
}
//-----
#pragma package(smart_init)
//-----
```

# 15.- Código fuente del archivo “global\_var”

## 15.1.- ARCHIVO .H

```
//-----  
  
#ifndef global_varH  
#define global_varH  
  
#include <System.Classes.hpp>  
  
#include <map>  
  
//-----  
/*DECLARACIÓN DE VARIABLES GLOBALES*/  
  
//BOOLEANAS  
extern bool isReading; //Hilo leyendo  
extern bool connectionPantalla; //Conexión directa a la pantalla  
extern bool salon; //Estado de la iluminación del salón  
extern bool cocina; //Estado de la iluminación de la cocina  
extern bool pasillo; //Estado de la iluminación del pasillo  
extern bool baño; //Estado de la iluminación del baño  
extern bool hab1; //Estado de la iluminación de la habitación 1  
extern bool hab2; //Estado de la iluminación de la habitación 2  
extern bool persiana_open; //Estado anterior de la persiana  
extern bool clima; //Estado del clima  
extern bool mostrar; //Mostrar por pantalla los telegramas en el Memo  
  
//INT  
extern int dato_actuador; //Lectura de la iluminación fija  
extern int dato_regulation; //Lectura de la regulación de la iluminación
```

```
extern int dato_persiana; //Lectura de la persiana
extern int dato_clima; //Lectura del clima
extern int typedate; //Elección del formato de fecha
extern int typetime; //Elección del formato de hora
extern int translation; //Elección del idioma

//MAP
extern std::map<UnicodeString,UnicodeString> trans; //Traducción
//-----
#endif
```

## 15.2.- ARCHIVO .CPP

```
//-----

#pragma hdrstop

#include "global_var.h"
//-----
/*DECLARACIÓN DE VARIABLES GLOBALES*/

//BOOLEANAS
bool isReading; //Hilo leyendo
bool connectionPantalla; //Conexión directa a la pantalla
bool salon; //Estado de la iluminación del salón
bool cocina; //Estado de la iluminación de la cocina
bool pasillo; //Estado de la iluminación del pasillo
bool baño; //Estado de la iluminación del baño
bool hab1; //Estado de la iluminación de la habitación 1
bool hab2; //Estado de la iluminación de la habitación 2
bool persiana_open; //Estado anterior de la persiana
bool clima; //Estado del clima
```

```
bool mostrar; //Mostrar por pantalla los telegramas en el Memo
```

```
//INT
```

```
int dato_actuador; //Lectura de la iluminación fija
```

```
int dato_regulation; //Lectura de la regulación de la iluminación
```

```
int dato_persiana; //Lectura de la persiana
```

```
int dato_clima; //Lectura del clima
```

```
int typedate; //Elección del formato de fecha
```

```
int typetime; //Elección del formato de hora
```

```
int translation; //Elección del idioma
```

```
//MAP
```

```
std::map<UnicodeString,UnicodeString> trans; //Traducción
```

```
//-----
```

```
#pragma package(smart_init)
```

## 16.- Código fuente del archivo “global\_struct”

```
//-----
```

```
#ifndef global_structH
```

```
#define global_structH
```

```
#include <System.Classes.hpp>
```

```
//-----
```

```
/**
```

```
ESTRUCTURA DEL PACKET == TELEGRAMA
```

```
**/
```

```
struct Packet
```

```
{
```

```
    /*
```

Dentro de la estructura se va a tener el paquete de los mensajes  
que se pueden enviar o que se pueden leer.

De esta manera se simplifica el uso de los datos.

Como datos necesarios se tienen los siguientes:

-Dirección local o fuente: Source -> source

-Dirección destino: Destiny -> destiny

-Comando: Command -> command

-Dato 1: Data 1 -> data1

-Dato 2: Data 2 -> data2

Los tipos de datos dependerán de la longitud que tenga cada uno,  
para las dos direcciones se necesitará utilizar "Unsigned

short"

ya que el número va de 0 a 65536, por lo que tiene 16 bits,

y los

demás bastará que sean "unsigned char" que equivale a un byte ,

por lo tanto 8 bits.

\*/

```
unsigned short source;  
unsigned short destiny;  
unsigned char command;  
unsigned char data1;  
unsigned char data2;
```

```
Packet()
```

```
{
```

```
    source = 0xFFFF;  
    destiny = 0xFFFF;  
    command = 0xFF;  
    data1 = 0xFF;  
    data2 = 0xFF;
```

```
}
```

```
};
```

```
//-----
```

```
#endif
```



## 17.- Código fuente del “Manifest” de la aplicación BUSing®

```
<?xml version="1.0" encoding="utf-8"?>
<!-- BEGIN_INCLUDE(manifest) -->
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="%package%"
    android:versionCode="%versionCode%"
    android:versionName="%versionName%"
    android:installLocation="%installLocation%">

    <!-- This is the platform API where NativeActivity was introduced. -->
    <uses-sdk
        android:minSdkVersion="%minSdkVersion%"
    android:targetSdkVersion="%targetSdkVersion%" />
    <%uses-permission%>
    <uses-feature android:glEsVersion="0x00020000" android:required="True"/>
    <application android:persistent="%persistent%"
        android:restoreAnyVersion="%restoreAnyVersion%"
        android:label="%label%"
        android:debuggable="%debuggable%"
        android:largeHeap="%largeHeap%"
        android:icon="%icon%"
        android:theme="%theme%"
        android:hardwareAccelerated="%hardwareAccelerated%">

    <%application-meta-data%>
        <%services%>
    <!-- Our activity is a subclass of the built-in NativeActivity framework class.
        This will take care of integrating with our NDK code. -->
    <activity
    android:name="com.embarcadero.firemonkey.FMXNativeActivity"
        android:label="%activityLabel%"
```

```
android:configChanges="orientation|keyboard|keyboardHidden|screenSize"
    android:launchMode="singleTask">
<!-- Tell NativeActivity the name of our .so -->
<meta-data android:name="android.app.lib_name"
    android:value="%libNameValue%" />
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<%activity%>
<%receivers%>
</application>
</manifest>
<!-- END_INCLUDE(manifest) -->
```



Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

LAURA RICO ÁLVAREZ

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

## **DESARROLLO DE SERVICIO SIP PARA VIDEOPORTERO**

Manual de programador

ENERO 2019



# Índice general

1.- Introducción .....	5
1.1.- DESCRIPCIÓN DEL PROYECTO.....	5
1.2.- VISIÓN DEL DOCUMENTO .....	5
2.- Clases .....	6
2.1.- MAIN ACTIVITY .....	6
2.2.- MAIN ACTION .....	7
2.3.- CALL ACTIVITY .....	8
2.4.- CALL ACTION .....	9
2.5.- MY CONFIGURATION.....	10
2.6.- MY ACCOUNT.....	10
2.7.- MY CALL.....	11
2.8.- MAIN FORM.....	11
2.9.- MY THREAD .....	15
3.- Otros archivos .....	16
3.1.- INTERFAZ .....	16
3.2.- ARCHIVOS GLOBALES .....	16
3.2.1.- Global .....	16
3.2.2.- Global_var.....	17
3.2.3.- Global_struct.....	17

# Índice de tablas

Tabla 2. 1.- Métodos de MainActivity. ....	7
Tabla 2. 2.- Métodos de MainAction.....	8
Tabla 2. 3.- Métodos de CallActivity.....	9
Tabla 2. 4.- Métodos de CallAction. ....	10
Tabla 2. 5.- Métodos de MyConfiguration. ....	10
Tabla 2. 6.- Métodos de MyAccount. ....	11
Tabla 2. 7.- Métodos de MyCall. ....	11
Tabla 2. 8.- Métodos de MainForm.....	15
Tabla 2. 9.- Métodos de MyThread.....	15
Tabla 3. 1.- Notificaciones de MyMessages. ....	16
Tabla 3. 2.- Métodos del archivo Global. ....	17
Tabla 3. 3.- Variables del archivo global_var.....	17

# 1.- Introducción

## 1.1.- DESCRIPCIÓN DEL PROYECTO

<b>Título</b>	Desarrollo de servicio SIP para videoportero
<b>Tutor:</b>	Antonio Robles Álvarez
<b>Autor:</b>	Laura Rico Álvarez
<b>Titulación:</b>	Máster en Automatización e Informática Industrial
<b>Fecha:</b>	Enero de 2019
<b>Financiación:</b>	INGENIUM, Ingeniería y Domótica, S.L.

## 1.2.- VISIÓN DEL DOCUMENTO

En el presente documento, se pretende explicar cada clase de la aplicación para permitir que una persona especializada pueda realizar modificaciones en el programa.

## 2.- Clases

En este capítulo se desglosarán los métodos y propiedades de cada clase de la aplicación, de esta manera será más sencilla la comprensión individual de cada una de ellas.

Además, todos los archivos de código fuente del programa se encuentran en el documento Anexos.

### 2.1.- MAIN ACTIVITY

El MainActivity, vista principal de la aplicación del intercomunicador, está compuesta, en mayor medida, por los componentes de la pantalla, aunque también cuenta con los objetos de las clases creadas, que se explicarán más adelante, y también de unas cuantas variables auxiliares para facilitar la programación.

Cuenta con una serie de eventos, propios de las activities de Android Studio, que sirven para los diferentes estados por los que pasa la pantalla. Tiene también alguna que otra función creada por el usuario para desempeñar las acciones oportunas.

Nombre	Descripción	Parámetros de entrada	Parámetros de salida
onCreate	Este método está anclado al evento de creación de la pantalla principal. Prepara la pantalla y además realiza la comprobación de los permisos.	<ul style="list-style-type: none"> <li>savedInstanceState: Bundle</li> </ul>	-
checkingOsPermits	Función creada para la comprobación de los permisos necesarios para desempeñar el funcionamiento correcto de SIP.	-	-
onRequestPermissionsResult	Evento de respuesta de la petición de los permisos.	<ul style="list-style-type: none"> <li>requestCode: int</li> <li>permissions []: String</li> <li>grantResults: int[]</li> </ul>	-
registerAccount	Esta función se ejecuta cuando se pulsa el botón de REGISTER de la pantalla y sirve para registrar al usuario en el servidor.	<ul style="list-style-type: none"> <li>view: View</li> </ul>	-
unregisterAccount	Esta función se ejecuta cuando se pulsa el botón X de la pantalla y sirve para	<ul style="list-style-type: none"> <li>view: View</li> </ul>	-



	desregistrar el usuario del servidor.		
watchCamera	Esta función se ejecuta cuando se pulsa el botón del ojo y hace que se muestre por pantalla las imágenes que capta la cámara.	<ul style="list-style-type: none"> <li>• view : View</li> </ul>	-
handleMessage	Esta función se ejecuta juntamente con la interfaz de la aplicación y maneja los mensajes de una clase a otra.	<ul style="list-style-type: none"> <li>• m: message</li> </ul>	<ul style="list-style-type: none"> <li>• boolean</li> </ul>
showCallActivity	Esta función se ejecuta cuando hay una llamada entrante o cuando se pulsa el botón del ojo. Muestra por pantalla las imágenes que capta la cámara y permite la comunicación con el otro extremo.	-	-
notifyRegState	Es una función para las notificaciones de registro del usuario.	<ul style="list-style-type: none"> <li>• code: pjsip_status_code</li> <li>• reason: String</li> <li>• expiration: int</li> </ul>	-
notifyCallState	Es una función para las notificaciones de estado de llamada.	<ul style="list-style-type: none"> <li>• call: MyCall</li> </ul>	-
notifyCallMediaState	Es una función para las notificaciones de estado de llamada media.	<ul style="list-style-type: none"> <li>• call: MyCall</li> </ul>	-
notifyIncomingCall	Es una función para la notificación de llamada entrante.	<ul style="list-style-type: none"> <li>• call: MyCall</li> </ul>	-
onRestart	Método anclado al evento de restart de la aplicación. Sirve para que la aplicación no se cierre cuando se cambia de una aplicación a otra.	-	-
returnBack	Función para volver atrás sin cerrar la aplicación.	<ul style="list-style-type: none"> <li>• view: View</li> </ul>	-

Tabla 2. 1.- Métodos de MainActivity.

## 2.2.- MAIN ACTION

Esta clase se podría ver como una subclase de MainActivity, ya que es la clase que realiza las acciones que suceden en la pantalla principal. Podría tratarse de una sola clase, pero se ha preferido separar las acciones que ocurren en la pantalla, acciones “físicas”, de las acciones que realiza internamente el programa.

Por lo tanto, en este caso está compuesta por objetos y funciones que se utilizan para temas relacionados con la cuenta de usuario, como pueden ser el registro o des registro y con algunas variables auxiliares para comunicarse con la actividad principal.

Nombre	Descripción	Parámetros de entrada	Parámetros de salida
registration	Está función es la encargada de registrar al usuario en el servidor del intercomunicador.	<ul style="list-style-type: none"> <li>• myUri: String</li> <li>• myServer: String</li> <li>• myUsername: String</li> <li>• myPassword: String</li> </ul>	<ul style="list-style-type: none"> <li>• boolean</li> </ul>
Outgoing	Función para realizar la llamada saliente de la pantalla al intercomunicador.	<ul style="list-style-type: none"> <li>• numberCall: String</li> </ul>	<ul style="list-style-type: none"> <li>• outgoingCall: MyCall</li> </ul>

Tabla 2. 2.- Métodos de MainAction.

### 2.3.- CALL ACTIVITY

Esta clase pertenece a la segunda pantalla que se muestra en la aplicación del videoportero, en la que se muestran las imágenes captadas por la cámara de este, por lo tanto, cuando se utiliza esta clase significa que el usuario se encuentra dentro de una llamada.

Al igual que el Main Activity, sus componentes principales son esas que forman la pantalla, pero, además, también cuenta con los objetos propios del protocolo SIP utilizados en las otras clases que se están viendo en estos apartados y alguna que otra variable auxiliar.

Los eventos también son parte de la configuración de la pantalla y, en este caso, de la ventana en la que se muestran las imágenes.

Nombre	Descripción	Parámetros de entrada	Parámetros de salida
onCreate	Método anclado al evento de creación de la pantalla de llamada, coloca los objetos en ella.	<ul style="list-style-type: none"> <li>• savedInstanceState: Bundle</li> </ul>	-
onConfigurationChanged	Método anclado al evento de cambio de configuración de la pantalla. Reconfigura la imagen.	<ul style="list-style-type: none"> <li>• newConfig: Configuration</li> </ul>	-
handleMessage	Función que maneja los mensajes recibidos de las otras	<ul style="list-style-type: none"> <li>• m: message</li> </ul>	<ul style="list-style-type: none"> <li>• boolean</li> </ul>

	clases, se ejecuta juntamente con la interfaz de la aplicación.		
surfaceCreate	Método anclado al evento de creación de la superficie de la pantalla.	<ul style="list-style-type: none"> <li>holder: SurfaceHolder</li> </ul>	-
surfaceChanged	Método anclado al evento de cambio en la superficie de la pantalla.	<ul style="list-style-type: none"> <li>holder: surfaceHolder</li> <li>format: int</li> <li>width: int</li> <li>height: int</li> </ul>	-
surfaceDestroy	Método anclado al evento de destrucción de la superficie de la pantalla.	<ul style="list-style-type: none"> <li>holder: SurfaceHolder</li> </ul>	-
setupVideoSurface	Función de configuración del vídeo.	-	-
updateVideoWindow	Función de actualización de la ventana de vídeo.	<ul style="list-style-type: none"> <li>show: boolean</li> </ul>	-
updateCallState	Función de actualización del estado de la llamada. Este método comprueba cualquier estado de la llamada, desde la posición que ocupa la pantalla, si llamante o llamada, hasta el estado propio en el que se encuentra.	<ul style="list-style-type: none"> <li>ci: CallInfo</li> </ul>	-
acceptCall	Esta función se ejecuta cuando se pulsa el botón verde para aceptar la llamada.	<ul style="list-style-type: none"> <li>view: View</li> </ul>	-
speakOn	Esta función se ejecuta cuando se pulsa el botón del micrófono apagado, para activar el audio	<ul style="list-style-type: none"> <li>view: View</li> </ul>	-
speakOff	Esta función se ejecuta cuando se pulsa el botón del micrófono encendido, para desactivar el audio.	<ul style="list-style-type: none"> <li>view: View</li> </ul>	-
openDoor	Esta función se ejecuta cuando se pulsa el botón de la llave y sirve para mandar el comando DTMF que activa el interruptor y abre la puerta del portal.	<ul style="list-style-type: none"> <li>view: View</li> </ul>	-
hangupCall	Esta función se ejecuta cuando se pulsa el botón rojo para colgar la llamada.	<ul style="list-style-type: none"> <li>view: View</li> </ul>	-
onDestroy	Método anclado al evento de destrucción de la pantalla de llamada.	-	-

Tabla 2. 3.- Métodos de CallActivity.

## 2.4.- CALL ACTION

Esta clase se podría ver como una subclase de CallActivity, ya que es la clase que realiza las acciones que suceden en la pantalla de llamada. Podría tratarse de una sola

clase, pero se ha preferido separar las acciones que ocurren en la pantalla, acciones “físicas”, de las acciones que realiza internamente el programa, al igual que ha pasado con las clases del Main.

En este caso está compuesta por objetos y funciones que se utilizan para la llamada.

Nombre	Descripción	Parámetros de entrada	Parámetros de salida
getInformacion	Función para obtener información de la llamada.	<ul style="list-style-type: none"> <li>• call: MyCall</li> </ul>	<ul style="list-style-type: none"> <li>• CallInfo</li> </ul>
answerCall	Función para contestar la llamada.	-	-
endCall	Función para finalizar la llamada.	-	-
setSpeakerOn	Función para activar el micrófono.	<ul style="list-style-type: none"> <li>• ci: CallInfo</li> </ul>	-
setSpeakerOff	Función para desactivar el micrófono.	<ul style="list-style-type: none"> <li>• ci: CallInfo</li> </ul>	-

Tabla 2. 4.- Métodos de CallAction.

## 2.5.- MY CONFIGURATION

Esta clase se utiliza para cargar y configurar las librerías propias del protocolo SIP dentro de la librería PJSIP, por lo que será instanciada al principio del programa principal para iniciar la configuración.

Nombre	Descripción	Parámetros de entrada	Parámetros de salida
init	Función de inicio de la configuración de un solo parámetro.	<ul style="list-style-type: none"> <li>• msg: MyMessages</li> </ul>	-
init	Función de inicio de la configuración de 2 parámetros que realiza la configuración de las librerías.	<ul style="list-style-type: none"> <li>• msg: MyMessages</li> <li>• own_worker_thread: boolean</li> </ul>	-
deinit	Función de destrucción del endpoint.	-	-

Tabla 2. 5.- Métodos de MyConfiguration.

## 2.6.- MY ACCOUNT

Extiende de Account, que es una clase -propia de la librería PJSIP que se encarga de la realización de eventos propios de la cuenta, como en este caso el estado de registro y la llamada entrante.

Nombre	Descripción	Parámetros de entrada	Parámetros de salida
MyAccount	Constructor.	<ul style="list-style-type: none"> <li>aConfig: AccountConfig</li> </ul>	-
onRegState	Evento de registro de la cuenta que se ejecuta cuando hay algún evento de registro.	<ul style="list-style-type: none"> <li>prm: OnRegStatePara</li> </ul>	-
onIncomingCall	Evento de llamada entrante que se ejecuta cuando otro usuario quiere comunicarse con esta cuenta.	<ul style="list-style-type: none"> <li>iprm: OnIncomingCallParam</li> </ul>	-

Tabla 2. 6.- Métodos de MyAccount.

## 2.7.- MY CALL

Al igual que en el apartado anterior, esta clase extiende de la clase Call de la librería PJSIP y se utiliza para los eventos de llamada.

Nombre	Descripción	Parámetros de entrada	Parámetros de salida
MyCall	Constructor	<ul style="list-style-type: none"> <li>acc: MyAccount</li> <li>call_id: int</li> </ul>	-
onCallState	Esta función se ejecuta cuando hay una llamada para ver qué información contiene.	<ul style="list-style-type: none"> <li>prm: OnCallStateParam</li> </ul>	-
onCallMediaState	Esta función se ejecuta cuando hay una llamada media para ver qué información contiene.	<ul style="list-style-type: none"> <li>prm: OnCallStateParam</li> </ul>	-

Tabla 2. 7.- Métodos de MyCall.

## 2.8.- MAIN FORM

Clase principal de la aplicación de comunicación BUSing®. Es la clase en la que realizan todas las acciones exceptuando la de leer el bus ya que puede bloquear el programa mientras realiza dicha acción.

Cuenta con todos los objetos que componen la pantalla, desde marcos y botones, hasta las imágenes de todos los componentes. La mayoría de las funciones con las que cuenta son las de click de los botones.

Nombre	Descripción	Parámetros de entrada	Parámetros de salida
dateAndTime_settings	Función que actualiza la fecha y la hora de la pantalla principal de la aplicación.	-	-
TimerTimeTimer	Función de timer que comprueba cada segundo cuando cambia el minuto que se muestra por pantalla.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
navigateClick	Función que permite navegar por las pantallas sin conexión, para que se puedan visualizar todos los apartados programados.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
btt_IpPortClick	Función que realiza la conexión del cliente a la pantalla a través de la dirección IP y del puerto.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
btn_UserPassClick	Función que realiza la conexión del cliente a la pantalla a través del servidor con el usuario y la contraseña previamente registrados en el servidor.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
ClientConnected	Función que se ejecuta si el cliente se ha conectado bien a la pantalla, bien sea por medio del servidor o directamente a ella.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
inicio	Función que instancia el hilo de lectura y lo lanza para no bloquear el programa.	-	-
sendPacket	Función que envía los telegramas al bus, tanto si hay una conexión como la otra.	<ul style="list-style-type: none"> <li>• p: Packet</li> </ul>	-
TimerConnectionTimer	Función de timer que comprueba cada minuto si ha fallado la conexión para volver a conectarse.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
img_iluClick	Función para acceder al apartado de iluminación de la casa.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
imageClick	Función para el encendido o apagado de las diferentes luces fijas de la casa.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
image_regulationMouseDown	Función de evento que indica cuando se pulsa la imagen de regulación lumínica.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> <li>• Button: TMouseButton</li> <li>• Shift: TShiftState</li> <li>• X: float</li> <li>• Y: float</li> </ul>	-
TimerIlluminationTimer	Función de timer para controlar la duración de la pulsación de la imagen de regulación lumínica.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
lay_regulacionMouseMove	Función de evento que indica cuando se mueve el ratón/dedo en la imagen de regulación lumínica.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> <li>• Shift: TShiftState</li> <li>• X: float</li> </ul>	-

		<ul style="list-style-type: none"> <li>• Y: float</li> </ul>	
image_regulationMouseUp	Función de evento que indica cuando se suelta la imagen de regulación lumínica.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> <li>• Button: TMouseButton</li> <li>• Shift: TShiftState</li> <li>• X: float</li> <li>• Y: float</li> </ul>	-
visuActuadores	Función que actualiza las imágenes de las salidas según su estado.	<ul style="list-style-type: none"> <li>• -</li> </ul>	-
back2Click	Función para volver a la pantalla principal.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
img_persClick	Función para acceder al apartado de persianas.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
image_persianaMouseDown	Función de evento que indica cuando se pulsa la imagen de regulación de persianas.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> <li>• Button: TMouseButton</li> <li>• Shift: TShiftState</li> <li>• X: float</li> <li>• Y: float</li> </ul>	-
TimerPersianaTimer	Función de timer para controlar la duración de la pulsación de la imagen de regulación de persianas.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
lay_reg_persianaMouseMove	Función de evento que indica cuando se mueve el ratón/dedo en la imagen de regulación de persianas.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> <li>• Shift: TShiftState</li> <li>• X: float</li> <li>• Y: float</li> </ul>	-
image_persianaMouseUp	Función de evento que indica cuando se suelta la imagen de regulación de persianas.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> <li>• Button: TMouseButton</li> <li>• Shift: TShiftState</li> <li>• X: float</li> <li>• Y: float</li> </ul>	-
img_termoClick	Función para acceder al apartado de climatización.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
image_climaClick	Función de click que enciende o apaga el termostato.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
menu1Click	Función para acceder a la pantalla del menu.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
image_sipClick	Función para acceder a la aplicación del videoportero.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
image_datagClick	Función para acceder a la pantalla de telegramas.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
button_okClick	Función para introducir la contraseña que da el acceso a la pantalla de telegramas.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-

lay_macroClick	Función para retroceder a la pantalla del menú.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
button_cancelClick	Función para cancelar la opción de introducir la contraseña.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
button_enviarClick	Función que envía el telegrama que se introduce por pantalla.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
button_deleteClick	Función que borra el contenido del memo.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
back6Click	Función para volver a la pantalla del menú.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
image_securityClick	Función para acceder a la pantalla de seguridad.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
image_configClick	Función para acceder a la pantalla de configuración.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
button_date_timeClick	Función para acceder a la pantalla de ajustes de fecha y hora.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
back8Click	Función para volver a la pantalla de configuración.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
btn_date_rightClick	Función para la elección de formato de fecha hacia la derecha.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
btn_date_leftClick	Función para la elección de formato de fecha hacia la izquierda.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
btn_time_leftClick	Función para la elección de formato de hora hacia la derecha.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
btn_time_rightClick	Función para la elección de formato de hora hacia la izquierda.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
ejemplo_formato	Función que muestra los distintos tipos de formato de fecha y hora.	<ul style="list-style-type: none"> <li>• -</li> </ul>	-
button_languageClick	Función para acceder a la pantalla del idioma.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
button_language_rightClick	Función para la elección de idioma hacia la derecha.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
button_language_leftClick	Función para la elección de idioma hacia la izquierda.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
button_appearanceClick	Función para acceder a la pantalla de fondos de pantalla.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
img_fondoMouseDown	Función de evento que indica cuando se pulsa cualquier imagen de fondo en miniatura.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> <li>• Button: TMouseButton</li> <li>• Shift: TShiftState</li> <li>• X: float</li> <li>• Y: float</li> </ul>	-
TimerVisuTimer	Función de timer para controlar la duración de la pulsación de cualquier imagen de fondo en miniatura.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
img_fondoMouseUp	Función de evento que indica cuando se suelta cualquier imagen de fondo en miniatura.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> <li>• Button: TMouseButton</li> <li>• Shift: TShiftState</li> </ul>	-



		<ul style="list-style-type: none"> <li>• X: float</li> <li>• Y: float</li> </ul>	
imageBigClick	Función de click que indica cuando se ha pulsado la imagen de fondo medio.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
lay_imageBigClick	Función que indica que no se quiere cambiar la imagen de fondo por la de tamaño medio.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
image_closeClick	Función de click para la conexión/desconexión del cliente.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
ClientDisconnected	Función de desconexión del cliente.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-
Client1Disconnected	Función de desconexión del cliente 1.	<ul style="list-style-type: none"> <li>• Sender: TObject</li> </ul>	-

Tabla 2. 8.- Métodos de MainForm.

## 2.9.- MY THREAD

Clase para el hilo de lectura de la aplicación BUSing®. Se utiliza para que la lectura del bus no bloquee ninguna parte del programa y siga su ejecución normal.

Utiliza principalmente buffers para enviar el telegrama y funciones de lectura y simulación.

Nombre	Descripción	Parámetros de entrada	Parámetros de salida
Execute	Función de creación del hilo	-	-
myThread	Función de ejecución del hilo que comprueba la conexión del cliente y lee lo que pasa por el bus.	<ul style="list-style-type: none"> <li>• CreateSuspended: bool</li> </ul>	-
read	Función de lectura del hilo que pasa los elementos que se tienen en el buffer del bus a un paquete para su comprensión más sencilla.	-	-
simulatePacket	Función que simula los telegramas leídos para que se vea la repercusión de estos en la pantalla.	<ul style="list-style-type: none"> <li>• p: Packet</li> </ul>	-

Tabla 2. 9.- Métodos de MyThread.

## 3.- Otros archivos

En este apartado se hablará de otros archivos que puedan resultar interesantes al programados, como son la interfaz del videoportero, los archivos globales de la aplicación BUSing® y los manifest de ambas.

### 3.1.- INTERFAZ

La interfaz MyMessages se utiliza para enviar notificaciones desde las clases propias de la librería PJSIP a la pantalla física.

Tiene 4 notificaciones diferentes.

Nombre	Descripción	Parámetros de entrada
notifyRegState	Notificación del estado del registro.	<ul style="list-style-type: none"> <li>code: pjsip_status_code</li> <li>reason: String</li> <li>expiration: int</li> </ul>
notifyCallState	Notificación del estado de la llamada.	<ul style="list-style-type: none"> <li>call: MyCall</li> </ul>
notifyCallMediaState	Notificación del estado de la llamada media.	<ul style="list-style-type: none"> <li>call: MyCall</li> </ul>
notifyIncomingCall	Notificación de la llamada entrante.	<ul style="list-style-type: none"> <li>call: MyCall</li> </ul>

Tabla 3. 1.- Notificaciones de MyMessages.

### 3.2.- ARCHIVOS GLOBALES

Dentro de los archivos del programa de comunicación BUSing® se encuentran 3 archivos globales que sirven para la definición de funciones, variables o estructuras que puedan usarse en todo el programa sin problema de tener que definirlos en todos los archivos.

#### 3.2.1.- Global

Este archivo se utiliza para las funciones de traducción de la aplicación y cuenta con varios métodos para ello.

Nombre	Descripción	Parámetros de entrada	Parámetros de salida
load_file	Realiza la carga de los ficheros donde se encuentran las traducciones.	-	-
load_translation	Realiza la búsqueda de la traducción de inglés al idioma que se le indique en el parámetro de entrada.	<ul style="list-style-type: none"> <li>file: UnicodeString</li> </ul>	-

load_en_translation	Realiza la búsqueda de la traducción del idioma que esté en ese momento al inglés.	<ul style="list-style-type: none"> <li>file: UnicodeString</li> </ul>	-
tr	Realiza una copia del texto que ha encontrado para el caso de que no encuentre traducción de este.	<ul style="list-style-type: none"> <li>text: UnicodeString</li> </ul>	<ul style="list-style-type: none"> <li>unicodeString</li> </ul>
translate	Realiza directamente la traducción.	-	-

Tabla 3. 2.- Métodos del archivo Global.

### 3.2.2.- Global\_var

En este archivo se declaran las variables que se utilizan tanto en el archivo principal Main, como en el hilo de lectura.

Nombre	Tipo	Descripción
isReading	Bool	Hilo leyendo.
connectionPantalla	Bool	Conexión directa a la pantalla.
salon	Bool	Estado de la iluminación del salón.
ocina	Bool	Estado de la iluminación de la cocina
pasillo	Bool	Estado de la iluminación del pasillo.
baño	Bool	Estado de la iluminación del baño.
hab1	Bool	Estado de la iluminación de la habitación 1.
hab2	Bool	Estado de la iluminación de la habitación 2.
persiana_open	Bool	Estado anterior de la persiana.
clima	Bool	Estado del clima.
mostrar	Bool	Mostrar por pantalla los telegramas en el Memo.
dato_actuador	Int	Lectura de la iluminación fija.
dato_regulation	int	Lectura de la regulación de la iluminación.
dato_persiana	Int	Lectura de la persiana.
dato_clima	Int	Lectura del clima.
typedate	Int	Elección del formato de fecha.
typetime	Int	Elección del formato de hora.
translation	Int	Elección del idioma.
trans	map	Traducción.

Tabla 3. 3.- Variables del archivo global\_var.

### 3.2.3.- Global\_struct

El archivo global\_struct contiene la estructura del paquete que se envía o se lee por el bus, llamado telegrama.



Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

LAURA RICO ÁLVAREZ

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

## **DESARROLLO DE SERVICIO SIP PARA VIDEOPORTERO**

Manual de usuario

ENERO 2019



# Índice general

1.- Introducción .....	5
1.1.- DESCRIPCIÓN DEL PROYECTO.....	5
1.2.- VISIÓN DEL DOCUMENTO .....	5
2.- Requisitos para la utilización de la aplicación.....	6
3.- Instalación de la aplicación .....	7
4.- Manejo de la aplicación .....	8
5.- Manejo del intercomunicador .....	21
6.- Manejo del panel.....	28

# Índice de figuras

Figura 3. 2.- Icono de la aplicación para la comunicación BUSing® .....	7
Figura 4. 1.- Pantalla de registro de la aplicación BUSing® .....	8
Figura 4. 2.- Pantalla principal de la aplicación BUSing® .....	9
Figura 4. 3.- Pantalla de iluminación de la aplicación BUSing® .....	9
Figura 4. 4.- Ejemplo de pantalla de iluminación con luces encendidas.....	10
Figura 4. 5.- Pantalla de persianas de la aplicación BUSing® .....	11
Figura 4. 6.- Pantalla de clima de la aplicación BUSing® .....	11
Figura 4. 7.- Pantalla del Menú de la aplicación BUSing® .....	12
Figura 4. 8.- Petición de contraseña para acceso a lectura y envío de telegramas. ....	12
Figura 4. 9.- Pantalla de lectura y envío de telegramas de la aplicación BUSing® .....	13
Figura 4. 10.- Ejemplo de lectura y envío de telegramas. ....	14
Figura 4. 11.- Pantalla de seguridad de la aplicación BUSing® .....	15
Figura 4. 12.- Pantalla de configuración de la aplicación BUSing® .....	15
Figura 4. 13.- Pantalla de configuración de fecha y hora de la aplicación BUSing® .....	16
Figura 4. 14.- Pantalla de idioma de la aplicación BUSing® .....	16
Figura 4. 15.- Pantalla de apariencia de la aplicación BUSing® .....	17
Figura 4. 16.- Ejemplo de selección de imagen mediante pulsación larga. ....	17
Figura 4. 17.- Pantalla de registro de la aplicación Videoportero.....	18
Figura 4. 18.- Pantalla principal de la aplicación Videoportero. ....	19
Figura 4. 19.- Ejemplo de recepción de cámara.....	19
Figura 4. 20.- Ejemplo de recepción de llamada.....	20
Figura 5. 1.- Pantalla principal del servidor.....	21
Figura 5. 2.- Pantalla del sistema del intercomunicador.....	22
Figura 5. 3.- Pantalla de registro de usuario propio del Intercomunicador.....	23
Figura 5. 4.- Pantalla de estado de los servicios del Intercomunicador.....	23
Figura 5. 5.- Pantalla de directorio de usuario de intercomunicador. ....	24
Figura 5. 6.- Pantalla de parametrización de la llamada del intercomunicador. ....	24
Figura 5. 7.- Pantalla de parametrización de la puerta del intercomunicador. ....	25
Figura 5. 8.- Pantalla de parametrización de interruptores del intercomunicador. ....	26
Figura 5. 9.- Pantalla de parametrización de botones del intercomunicador. ....	26
Figura 5. 10.- Pantalla de eventos del intercomunicador. ....	27
Figura 6. 1.- Panel de pruebas.....	28
Figura 6. 2.- Pantalla de la Smart Touch del panel.....	29
Figura 6. 3.- Pantalla de ajustes de la pantalla del panel.....	29
Figura 6. 4.- Pantalla de la PPL-4 del panel. ....	29

# 1.- Introducción

## 1.1.- DESCRIPCIÓN DEL PROYECTO

<b>Título</b>	Desarrollo de servicio SIP para videoportero
<b>Tutor:</b>	Antonio Robles Álvarez
<b>Autor:</b>	Laura Rico Álvarez
<b>Titulación:</b>	Máster en Automatización e Informática Industrial
<b>Fecha:</b>	Enero de 2019
<b>Financiación:</b>	INGENIUM, Ingeniería y Domótica, S.L.

## 1.2.- VISIÓN DEL DOCUMENTO

En el presente documento, se pretende explicar al usuario el manejo total de la aplicación, así como del servidor y del panel utilizados para las pruebas.



## 2.- Requisitos para la utilización de la aplicación

Esta aplicación está diseñada para la comunicación intercomunicador – pantalla y también para el control del sistema domótico BUSing®. Esto implica que es necesario poseer un sistema domótico BUSing®, del estilo al del panel de pruebas mencionado en otros documentos, además de una conexión WiFi a la que esté conectada la pantalla para poder comunicarse con el servidor en el que está registrado el panel. Dicha red WiFi tendrá que ser la misma que le llegue por cable Ethernet al intercomunicador.

Otro requisito, dándose el caso actual en el que aún no existe esa pantalla, se necesitará un Smartphone conectado a esa misma red WiFi.

### 3.- Instalación de la aplicación

En el caso de contar con la pantalla que, actualmente, se está desarrollando en INGENIUM. S.L, la instalación de la aplicación, o aplicaciones en este caso, vendría dada de fábrica y al encender esta sólo habría que introducir las credenciales apropiadas dependiendo de la instalación que se considere.

En cambio, en el caso actual, no contando aun con esa pantalla, la aplicación se instalará en un Smartphone por medio del su apk, ejecutándolo desde el gestor de archivos del teléfono. Los iconos que aparecerán son los siguientes:



Figura 3. 1.- Icono de la aplicación para la comunicación BUSing®.



Figura 3. 2.- Icono de la aplicación para la comunicación con el intercomunicador.

## 4.- Manejo de la aplicación

Para el manejo y control de la aplicación, aunque puedan diferenciarse dos aplicaciones, se contará a partir de ahora como una ya que internamente están comunicadas, de manera que solamente hará falta lanzar la de comunicación BUSing® para poder empezar.

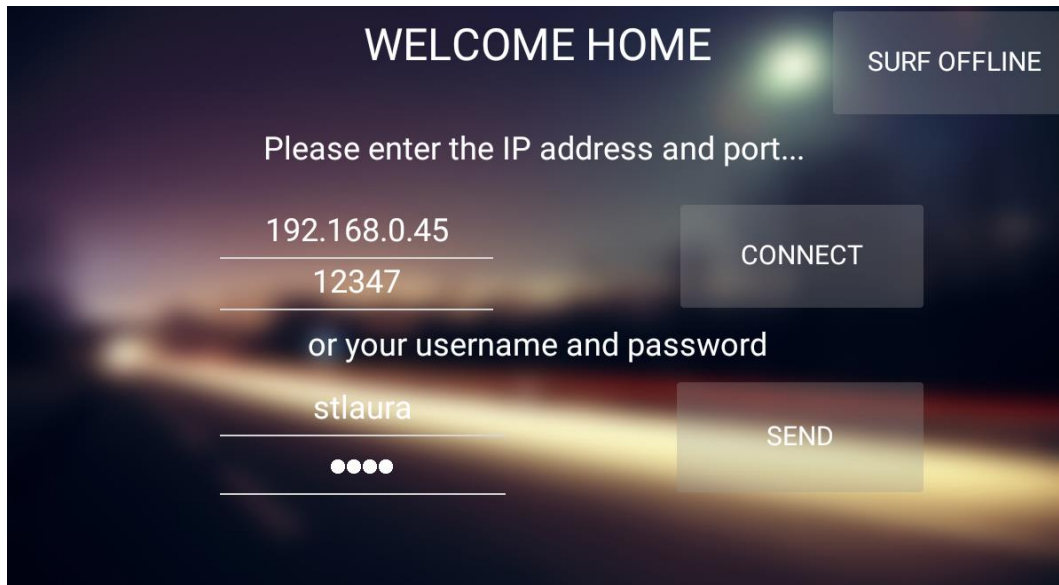


Figura 4. 1.- Pantalla de registro de la aplicación BUSing®.

Una vez ejecutada la aplicación BUSing®, la imagen que aparecerá en pantalla será la que se puede ver en la Figura 4.1. Se puede observar que existen 2 maneras de conectarse a la red, bien sea por medio de la dirección IP y el puerto de la Smart Touch que se comunica con el panel, pulsando el botón *CONNECT*, o a través de unas credenciales previamente registradas en el servidor de INGENIUM S.L, pulsando el botón *SEND*. En este caso, a modo de interacción para la presentación, existe una tercera manera de acceder a la aplicación, pulsando el botón *SURF OFFLINE*, que sirve para poder navegar por la aplicación sin necesidad de estar conectado a la red, aunque no se podrá interactuar con los dispositivos, es meramente visual.

Pulsado alguno de dichos botones se llega a la siguiente pantalla, que es la pantalla principal de la aplicación, Figura 4.2, en la que se pueden ver 3 apartados, *Illumination*, *Blinds* y *Climate*, para poder interactuar con cada tipo de dispositivos de la casa. En la parte superior, además de la fecha y la hora, también se observa un icono formado por tres rayas horizontales, que servirá para acceder al menú.



Figura 4. 2.- Pantalla principal de la aplicación BUSing®.

Accediendo al apartado *Illumination*, aparece la pantalla de la Figura 4.3, en la que se muestran las diferentes luminarias existentes y a que estancia pertenecen, a excepción de la de regulación, que al ser solamente esa no se especifica su estancia.

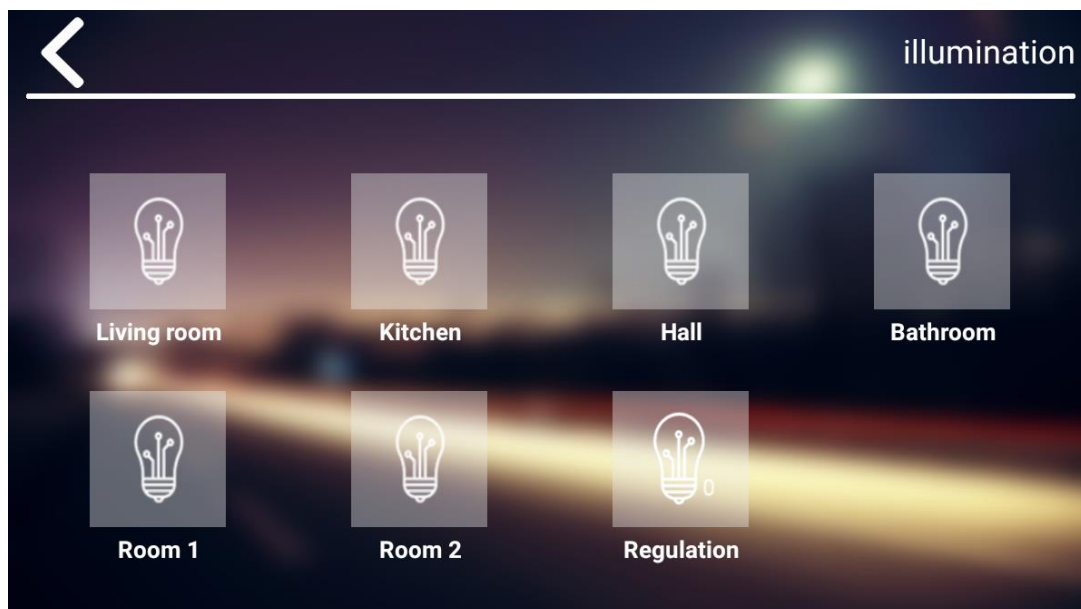


Figura 4. 3.- Pantalla de iluminación de la aplicación BUSing®.

Para el encendido y apagado de las luces fijas, bastará con hacer click sobre cada una de ellas, mientras que la luz de regulación tiene dos modos de uso. El primero, mediante pulsación corta, actúa igual que las fijas, mientras que si se realiza una pulsación

larga, al lado de la imagen aparecerá una etiqueta con el porcentaje al que se encuentra la intensidad de la bombilla y se podrá deslizar el dedo hacia arriba o hacia abajo para ajustarla.

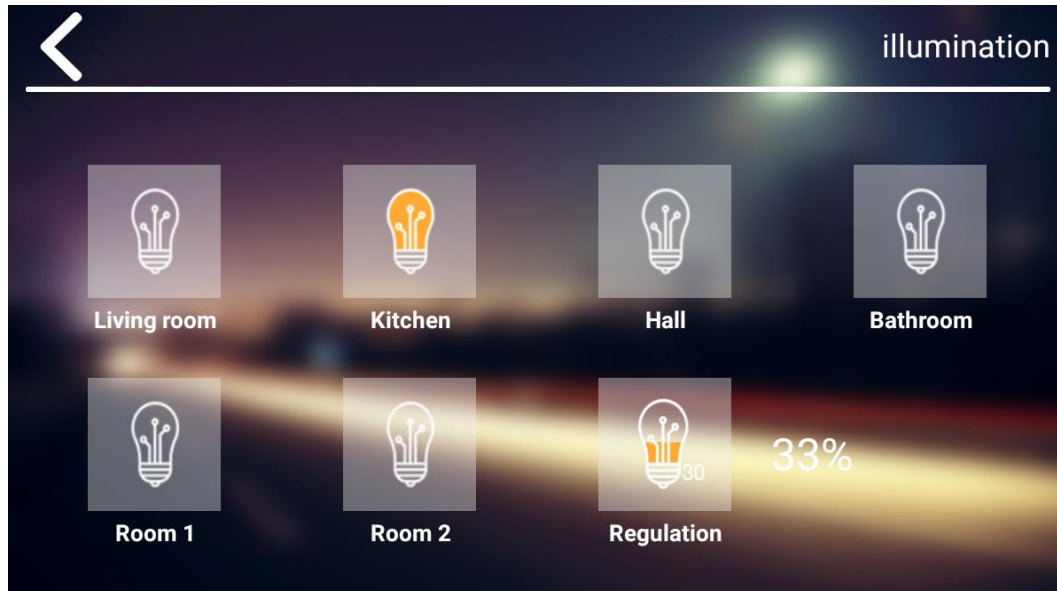


Figura 4. 4.- Ejemplo de pantalla de iluminación con luces encendidas.

En la parte superior de la imagen, aparece una etiqueta con el nombre del subapartado en el que se encuentra y una flecha en la esquina izquierda para volver a la pantalla principal.

En el apartado de *Blinds (Persianas)* se puede encontrar algo parecido a la iluminación, aunque en este caso solo existe una persiana programada, que pertenece al salón. Su modo de uso es exactamente igual que al de la luz regulada. Mediante pulsación corta la persiana va hasta sus extremos, mientras que, por pulsación larga, aparecerá el porcentaje de altura al que se encuentra, y este podrá ser regulado deslizando el dedo hacia arriba o hacia abajo. El ejemplo de esta pantalla puede verse en la Figura 4.5.



Figura 4. 5.- Pantalla de persianas de la aplicación BUSing®.

La pantalla perteneciente a *Climate (Clima)*, cuenta también con un solo dispositivo, ya que en la vivienda existe solamente un climatizador, incorporado en la SmartTouch. En cuanto al uso de este mediante la aplicación, se podrá encender o apagar por pulsación corta y mostrará la temperatura ambiente constantemente.

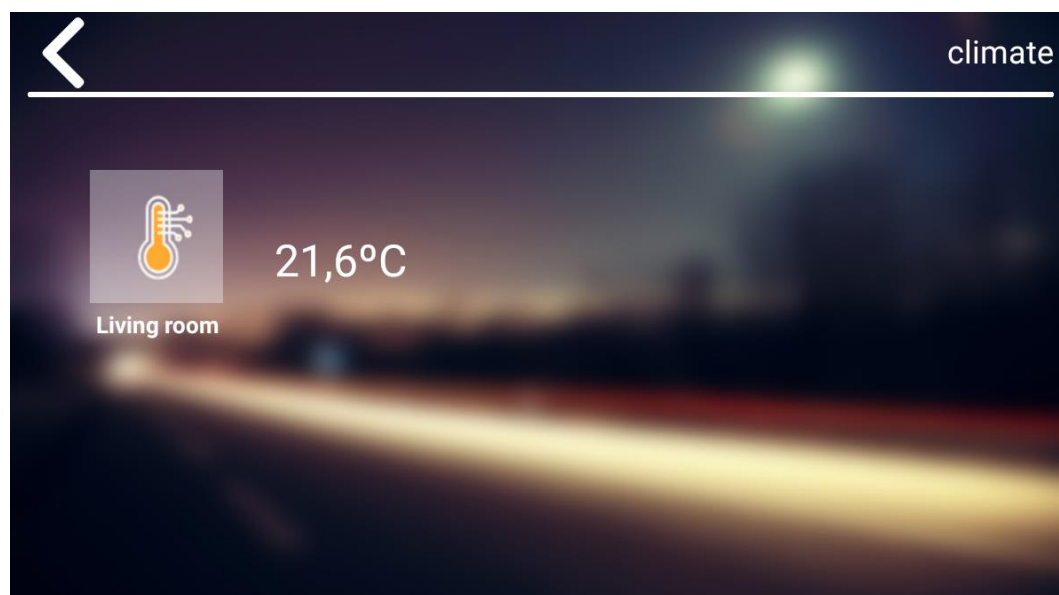


Figura 4. 6.- Pantalla de clima de la aplicación BUSing®.

Vistos los apartados relacionados con la pantalla principal, se pasa a la pantalla del menú, Figura 4.7, en la que se encuentran otros 5 apartados. También se observa en

la parte superior, una etiqueta a la izquierda, que dice en que pantalla te encuentras, y una X a la derecha para retornar a la pantalla principal.

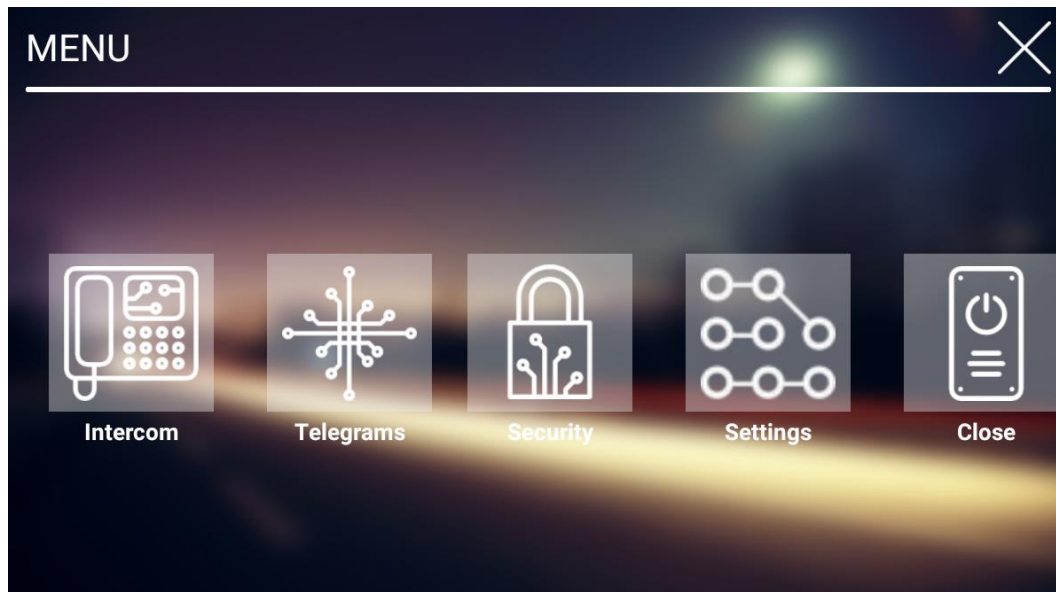


Figura 4. 7.- Pantalla del Menú de la aplicación BUSing®.

En este caso, no se va a seguir el orden visto en la imagen y se va a dejar la explicación del *Intercom* (*Intercomunicador*) para el final, ya que aunque se ha mencionado que se verá como parte de la aplicación, hay que recordar que no comparten programación.

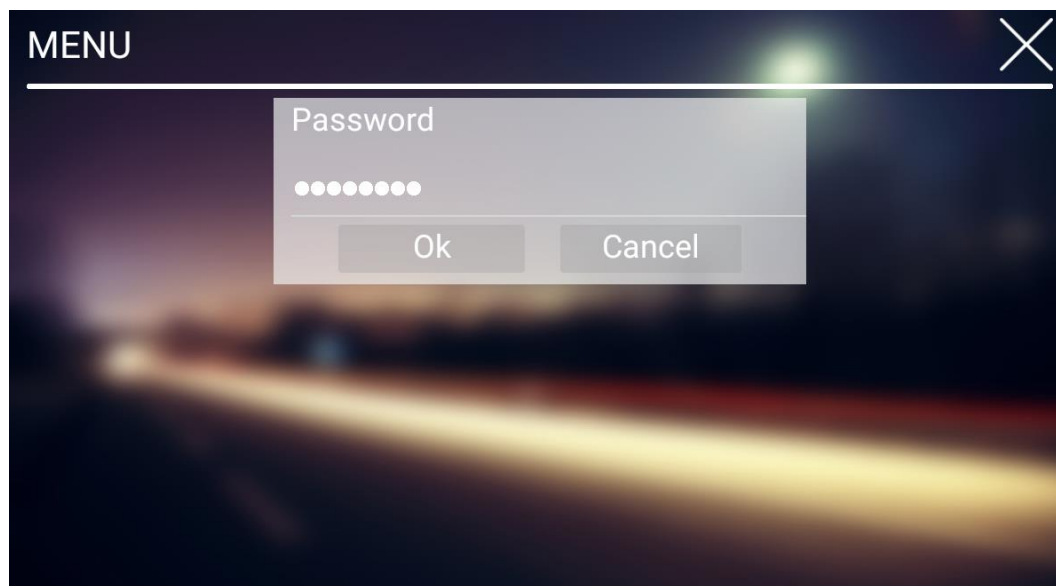


Figura 4. 8.- Petición de contraseña para acceso a lectura y envío de telegramas.



Por lo tanto, accediendo primeramente al apartado *Telegrams (Telegramas)*, aparece la Figura 4.8, en la que se pide por pantalla una contraseña, indicando que no es un sitio habitual en el que suelen entrar los usuarios, estaría más visto para el uso de algún técnico que tuviera que realizar algún ajuste sobre la instalación, por lo que solo este conocería la contraseña.

Dando por hecho que se conoce la contraseña, se llega a la pantalla de la Figura 4.9, en la que se puede apreciar un cambio importante en referencia a las demás pantallas vistas hasta ahora, y es que el fondo ya no es un paisaje difuminado, si no un fondo en tonos grises. Esto es debido a que, como se ha mencionado antes, no es una pantalla en la que el usuario vaya a navegar normalmente, es más técnica.



Figura 4. 9.- Pantalla de lectura y envío de telegramas de la aplicación BUSing®.

A la vista de la Figura 4.9, se observa que existen una serie de apartados a rellenar que son *Source (Fuente)*, *Destiny (Destino)*, *Command (Comando)*, *Data 1 (Dato 1)* y *Data 2 (Dato 2)*, que como se ha visto en los documentos **Memoria** y **Manual de programador**, son los diferentes campos que componen un telegrama. En ellos se escribe el telegrama de la acción que se quiera realizar sobre el bus, y a través del botón *SEND* se envían a este mismo.

En la zona blanca que aparece en la pantalla se van imprimiendo tanto los telegramas que van llegando como los que se van enviando, diferenciándose por unas flechas, < - para los que se envían y - > para los que se reciben. Pulsando el botón



*DELETE* se limpia la pantalla. En la Figura 4.10 aparece un ejemplo de la pantalla recibiendo y enviando telegramas.

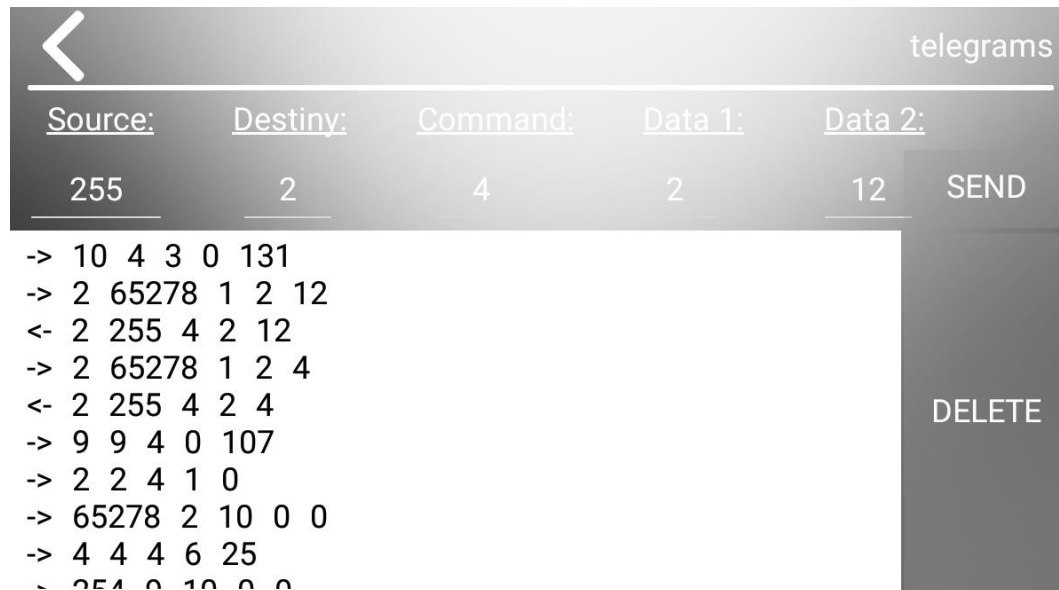


Figura 4. 10.- Ejemplo de lectura y envío de telegramas.

Además, en la parte superior, como las demás pantallas, también aparece una etiqueta a la derecha que muestra en qué pantalla se encuentra uno en ese mismo instante y con la flecha de la izquierda se vuelve a la pantalla anterior, la del menú.

El siguiente apartado del que consta el Menú es *Security (Seguridad)*, Figura 4.11, y en el se puede ver 2 subapartados que son *Alarms (Alarmas)* y *Passwords (Contraseñas)*, aunque aún no cuentan con implementación, ya que, en este caso, las pruebas no se han realizado con ninguna alarma, pero se podría dar el caso de que alguna instalación lo tuviera.

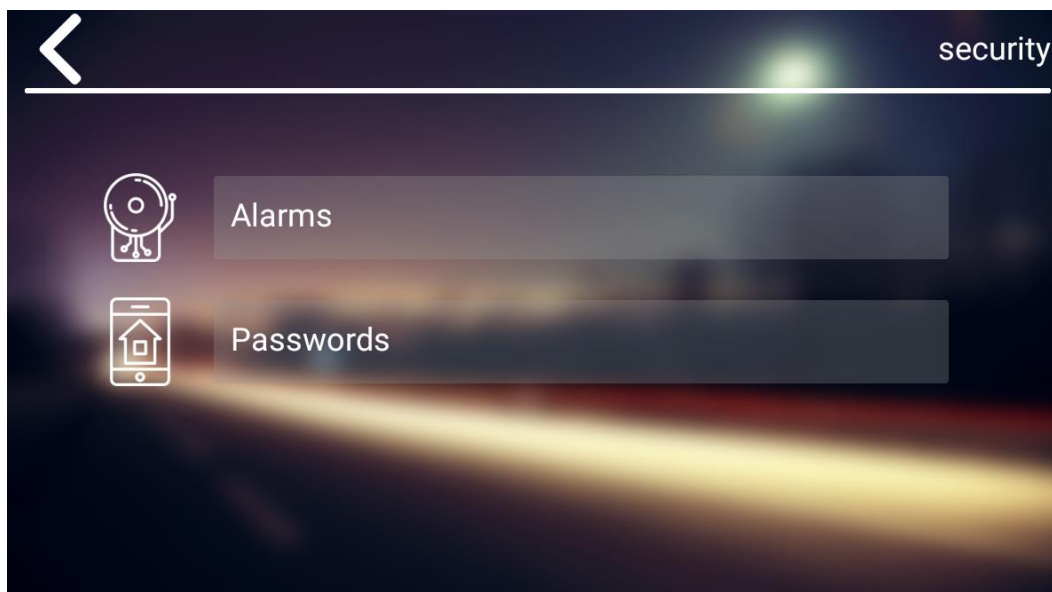


Figura 4. 11.- Pantalla de seguridad de la aplicación BUSing®.

El siguiente apartado, *Settings (Configuración)*, cuenta, en este caso, con 3 sub-apartados, que son *Setting date/time (Configuración de fecha y hora)*, *Language (Idioma)* y *Appearance (Apariencia)*, mostrados en la Figura 4.12.

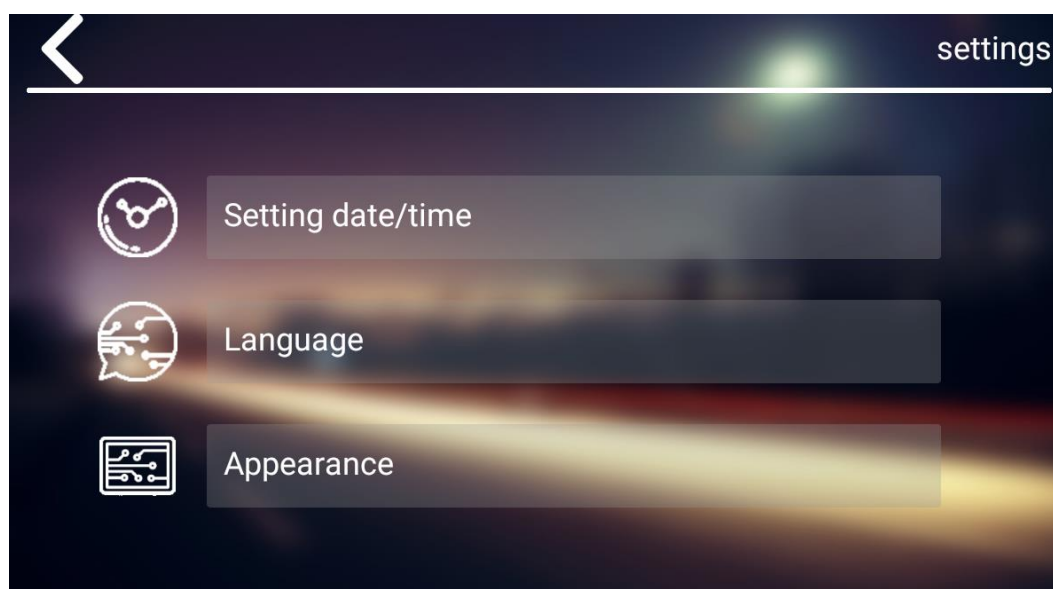


Figura 4. 12.- Pantalla de configuración de la aplicación BUSing®.

Dentro del sub-apartado de configuración de fecha y hora se tienen dos opciones independientes para cambiar el formato de la vista de estas, Figura 4.13. A través de las flechas se puede ir visualizando que formato tomarán tanto fecha como hora.

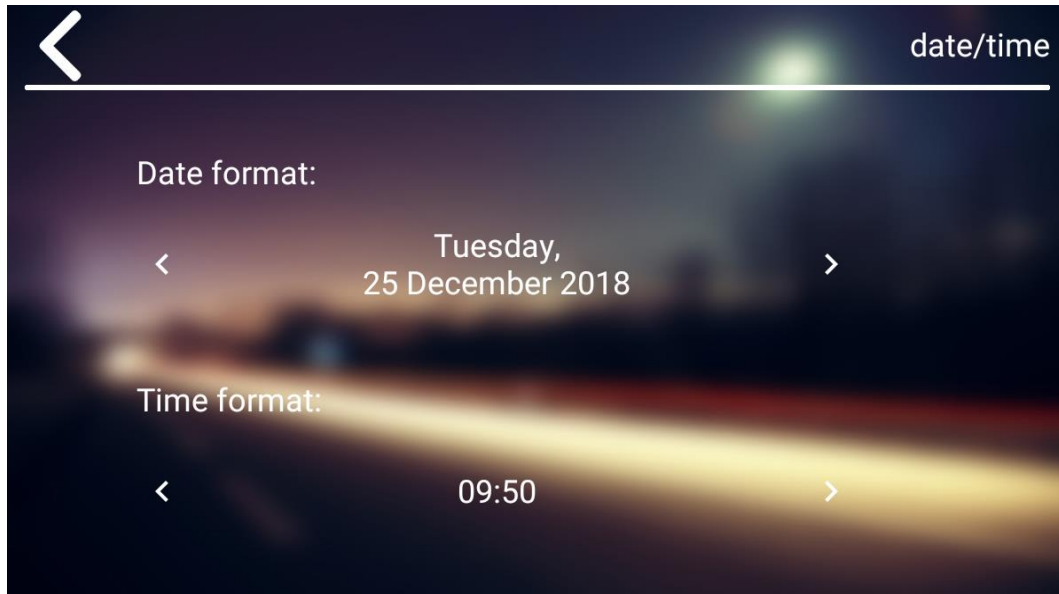


Figura 4. 13.- Pantalla de configuración de fecha y hora de la aplicación BUSing®.

El segundo de los sub-apartados de la configuración es, como se ha visto antes, el idioma, en él existe la opción de poner el idioma de la aplicación en Inglés, Español o Francés, Figura 4.14.



Figura 4. 14.- Pantalla de idioma de la aplicación BUSing®.

En cuanto al tercer sub-apartado, apariencia, cuenta con la posibilidad de cambiar la imagen de fondo, dentro de unas imágenes dadas, 4 en este caso.



Figura 4. 15.- Pantalla de apariencia de la aplicación BUSing®.

Con una pulsación corta se cambia el fondo directamente, mientras que con una pulsación larga aparece una imagen más grande, de la imagen pulsada, en el centro para poder visualizarla mejor. Si se quiere seleccionar se realizará una pulsación corta en la imagen central, si no, la pulsación corta se realizará fuera de esta.

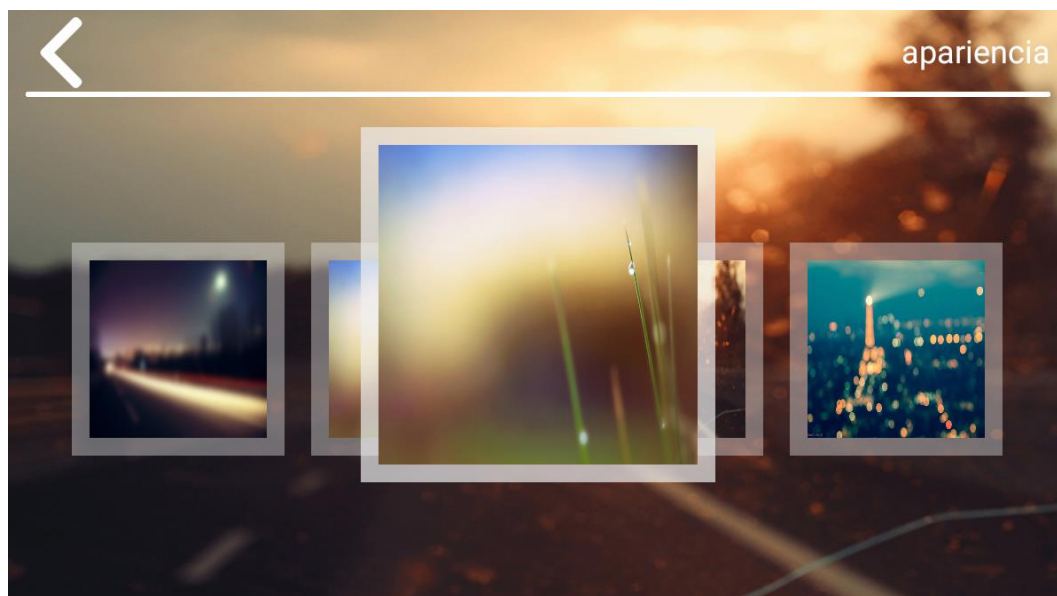


Figura 4. 16.- Ejemplo de selección de imagen mediante pulsación larga.

Volviendo a la pantalla del Menú, queda un solo apartado, sin contar el Intercomunicador, y sirve en este caso únicamente para la aplicación en el móvil, ya que cierra la aplicación y se desconecta del servicio, caso que en la pantalla no se contemplaría.

En cuanto al Intercomunicador, la primera vez que se pulse su botón, aparecerá la pantalla de la Figura 4.17, en la que es necesario introducir la dirección IP del servidor al que se quiere conectar en este caso, el intercomunicador físico, la extensión que se va a utilizar y la contraseña para realizar el registro. Para acceder se pulsará el botón *REGISTER*.

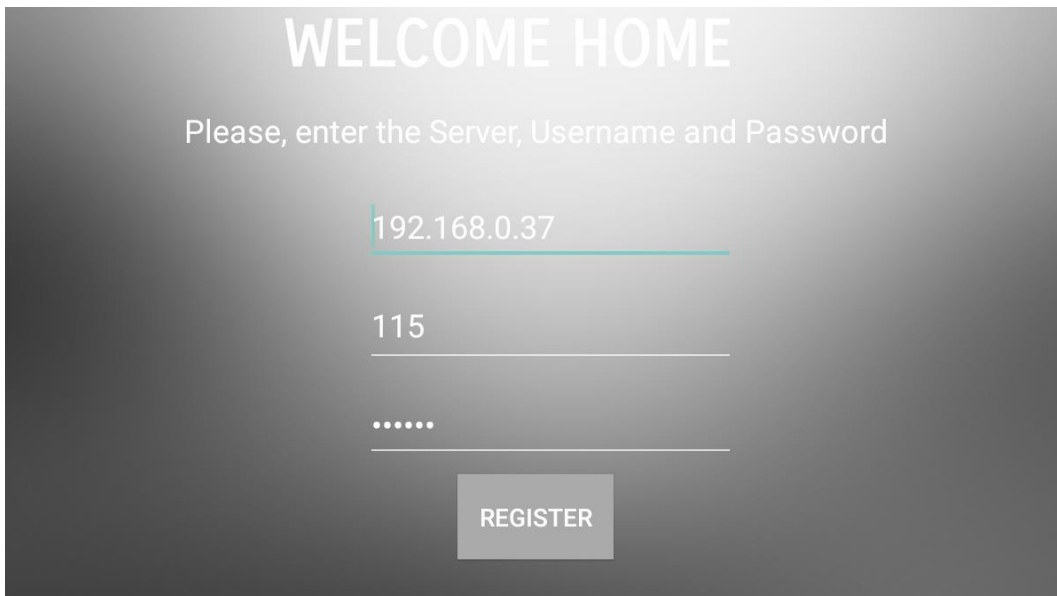


Figura 4. 17.- Pantalla de registro de la aplicación Videoporter.

Se puede apreciar que el fondo de esta pantalla y la siguiente es igual al fondo de los telegramas ya que son pantallas más técnicas, como se ha dicho antes.

Una vez realizado el registro, la siguiente pantalla mostrará un aspecto similar a las de los apartados anteriores, ya que cuenta con un encabezado en el cual hay una flecha, para volver a la pantalla del Menú de la aplicación BUSing®, y una X para cerrar la conexión, sólo para la aplicación móvil.

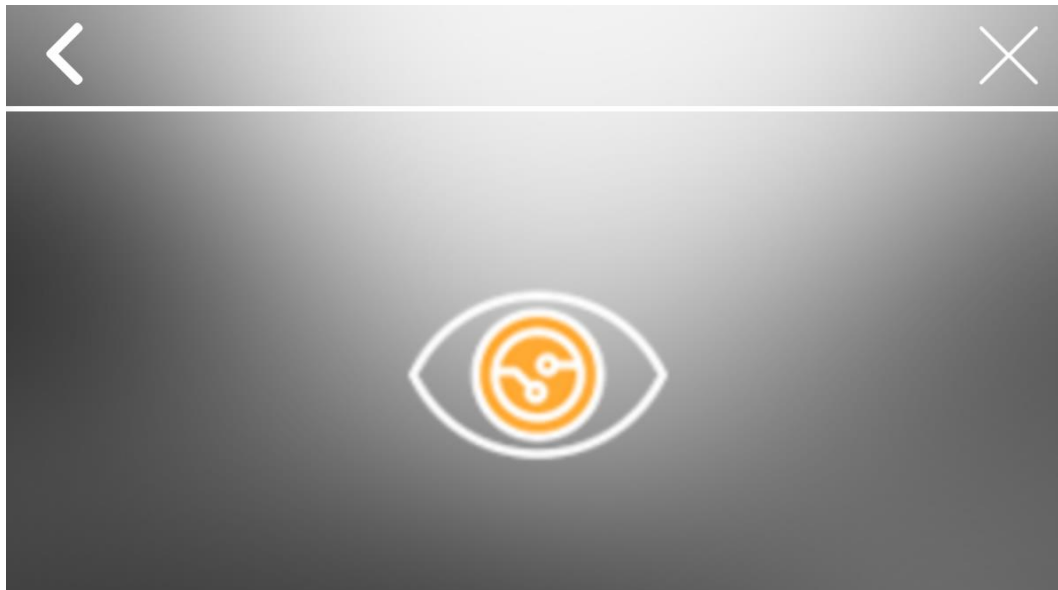


Figura 4. 18.- Pantalla principal de la aplicación Videoportero.

En el centro de la pantalla aparece un botón con forma de ojo, que al pulsarlo accede al contenido de la cámara en tiempo real, pudiendo interactuar para abrir la puerta, botón de la llave, activar el audio, botón del micrófono, y colgar, botón rojo.

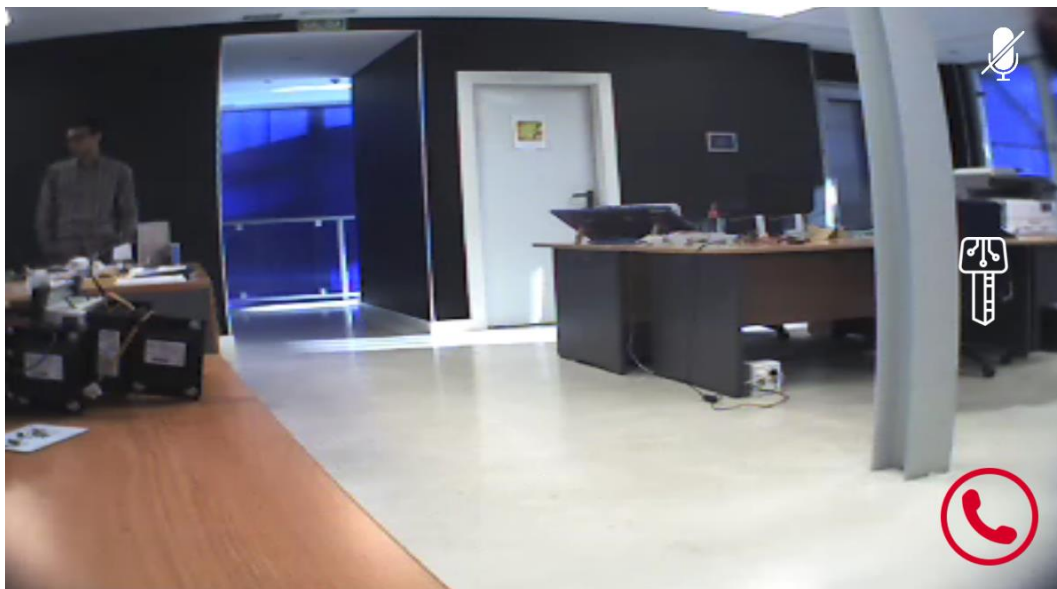


Figura 4. 19.- Ejemplo de recepción de cámara.

El botón de colgar hará el mismo efecto que el de la flecha de la pantalla principal, retornará la aplicación a la pantalla de Menú de BUSing® y cuando se le vuelva a dar al botón del Intercomunicador, se accederá directamente a la pantalla principal, sin pasar por la pantalla de registro.



En cuanto a la recepción de llamadas, tanto si se está en la aplicación BUSing®, en cualquiera de sus pantallas o apartados, o en la pantalla del Intercomunicador, aparecerá una pantalla emergente que ocupará todo el espacio, en el que se mostrarán las imágenes captadas por la cámara, se podrá realizar todas las acciones dichas anteriormente además de descolgar la llamada en el botón verde, Figura 4.20.



Figura 4. 20.- Ejemplo de recepción de llamada.

## 5.- Manejo del intercomunicador

En cuanto a la conexión y manejo del intercomunicador, aunque no se hayan diseñado sus especificaciones, por ser un producto comprado, si ha sido necesario aprender su uso como usuario, aunque no en profundidad. En este apartado se comentarán los aspectos más importantes.

Primeramente, en cuanto a la conexión del intercomunicador físico, este ha de ser previamente conectado a un cable de red, RJ45, y conectado también a tensión a 12V. Una vez enchufado, se encenderá él solo, y será necesario realizar una pulsación de 5 segundos para que se conecte a la red y configure internamente su dirección IP.

Debido a que es el modelo más simple y no cuenta con pantalla, la única manera de conocer la dirección que ha adquirido es a través de un software externo, en este caso se ha utilizado Advanced IP Scanner, que escanea todas las direcciones IP utilizadas dentro de la red en la que se encuentra.

Una vez escaneadas las direcciones y encontrado cual pertenece al intercomunicador, se abre una pestaña del navegador y en el se escribe dicha dirección. Esto llevará a una página remota en la que pedirá unas credenciales para crearse una cuenta que se use como administrador.



Figura 5. 1.- Pantalla principal del servidor.



Una vez dentro, Figura 5.1, lo primero que hay que hacer es ir al apartado de *Sistema*, para cambiar el modo DHCP en el que se inicia siempre el servidor, ya que si no si alguna vez hay un fallo en la conexión, cuando este se reinicie lo más seguro es que no obtenga la misma dirección que tenía y haya que volver a registrar todos los dispositivos.

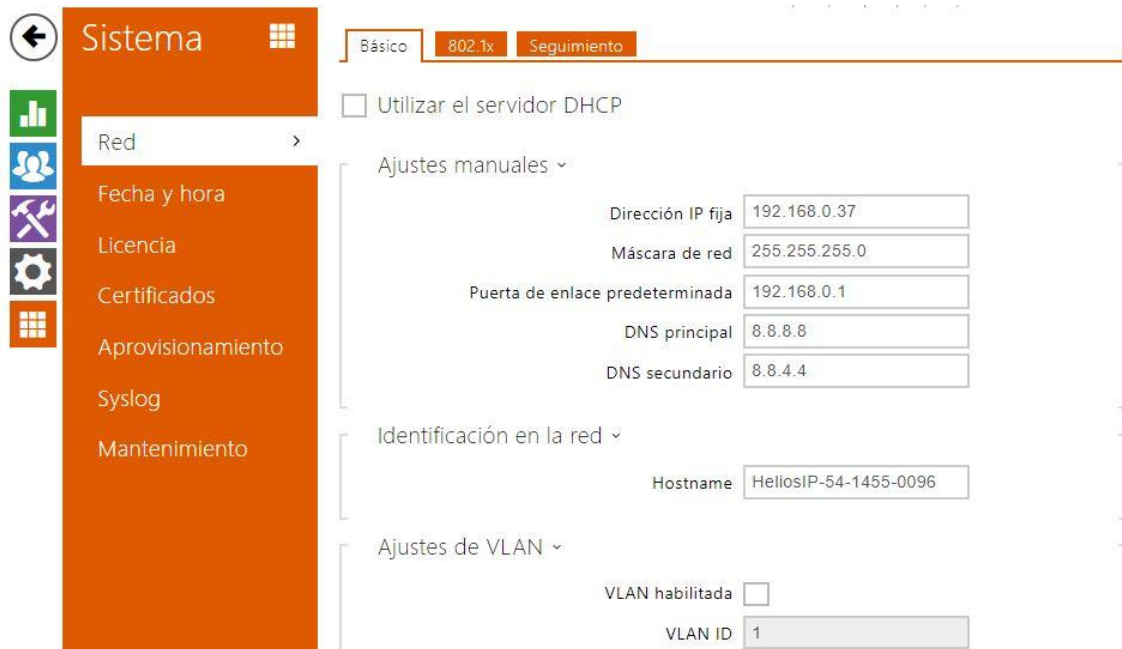


Figura 5. 2.- Pantalla del sistema del intercomunicador.

En la Figura 5.2 se tiene las opciones para darle una dirección IP fija y a su vez también asignarle máscara de red, puerta de enlace y DNS.

Una vez configurada la dirección, ya pueden configurarse las características particulares que se quiere que tenga el intercomunicador.

El intercomunicador, posee dos cuentas de usuario para poder funcionar como un “teléfono” que llame a la pantalla, pero para ello es necesario registrar dichos usuarios. En este caso, solo hará falta utilizar uno, ya que solo poseemos un botón con el que llamar.

Para ello habrá que ir a la pestaña *Servicios*, una vez ahí a *Teléfono* y dentro de esta a *SIP1*, Figura 5.3. Ahí se le dará un nombre, una extensión o ID y el dominio al que pertenece, que es la dirección IP del servidor, así como indicar el proxy y el registrar sobre el que se va a registrar el cliente, al igual que un usuario normal.



Figura 5. 3.- Pantalla de registro de usuario propio del Intercomunicador.

En la pestaña de *Estado*, y dentro de ella en *Servicios*, puede verse un breve resumen tanto del registro del propio servidor como de los registros de los usuarios internos que este posee, Figura 5.4.

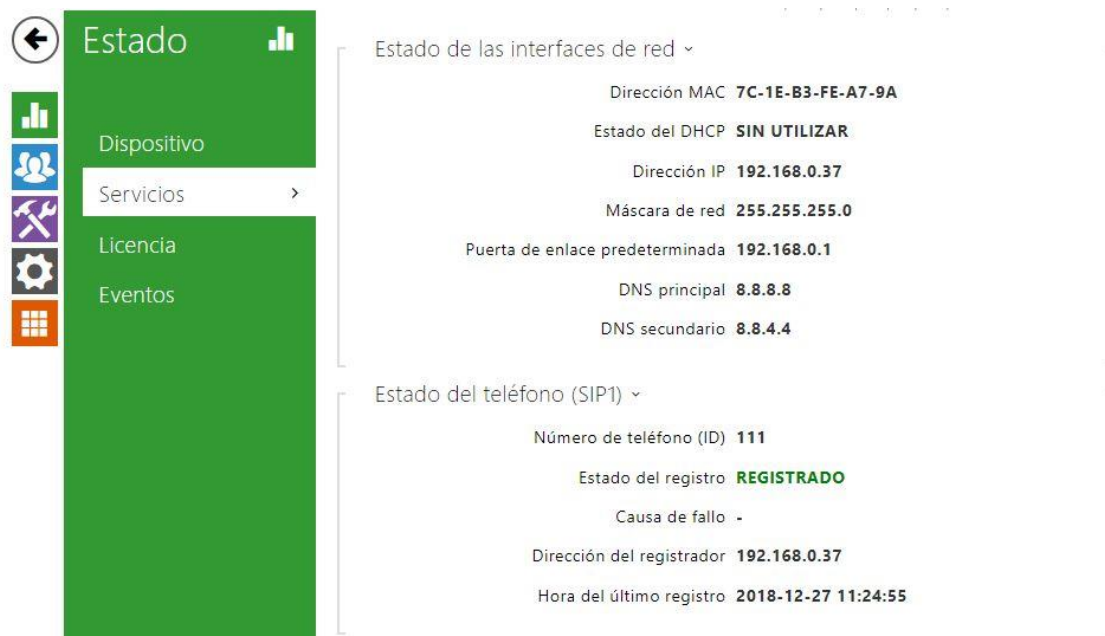


Figura 5. 4.- Pantalla de estado de los servicios del Intercomunicador.

Una vez registrado el usuario interno, hay que crear las cuentas para poder registrar a los usuarios externos, como es en este caso la pantalla. Para ello hay que ir al apartado *Directorio-Usuarios* y ahí ir agregando tantas extensiones como se quiera o necesite.

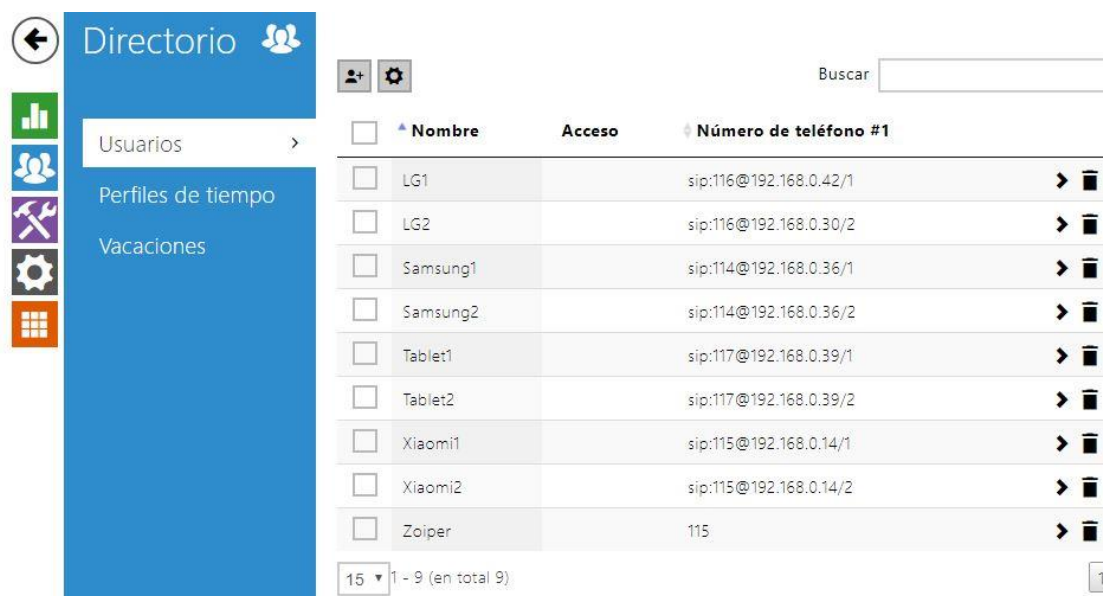


Figura 5. 5.- Pantalla de directorio de usuario de intercomunicador.

Teniendo ahora los usuarios configurados, se pasará a configurar como se quiere que sea la llamada, pudiendo parametrizarse el tiempo límite de esta, la aceptación de la llamada... Para ello hay que ir al apartado *Servicios-Teléfono-Llamadas*, Figura 5.6.

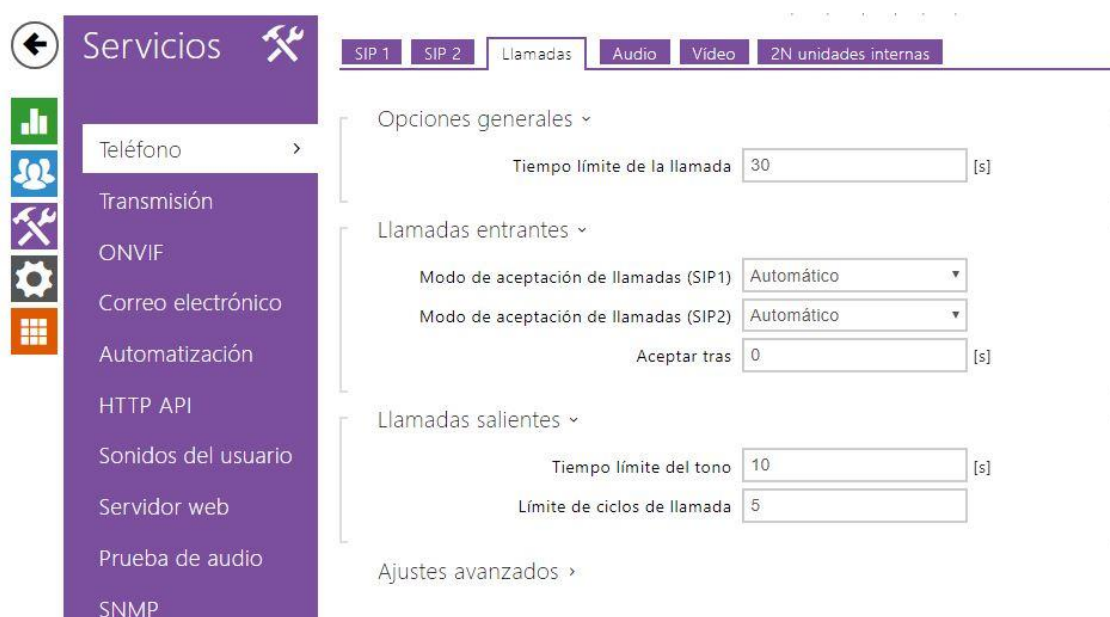


Figura 5. 6.- Pantalla de parametrización de la llamada del intercomunicador.

En cuanto al hardware a configurar, en este caso, al ser un intercomunicador que cuenta con solamente un botón, será necesario configurar la acción de ese botón, la puerta y el interruptor al que está asociada.

Para ello, empezando por la puerta, se irá al apartado *Hardware-Puerta*, y se le asignará el interruptor 1, el cual se configurará a continuación. El modo de entrada será sin invertir, para que al mandar el código se de tensión y se abra y no se contará con ningún sensor, por lo que el tiempo máximo de apertura no es condicionante.

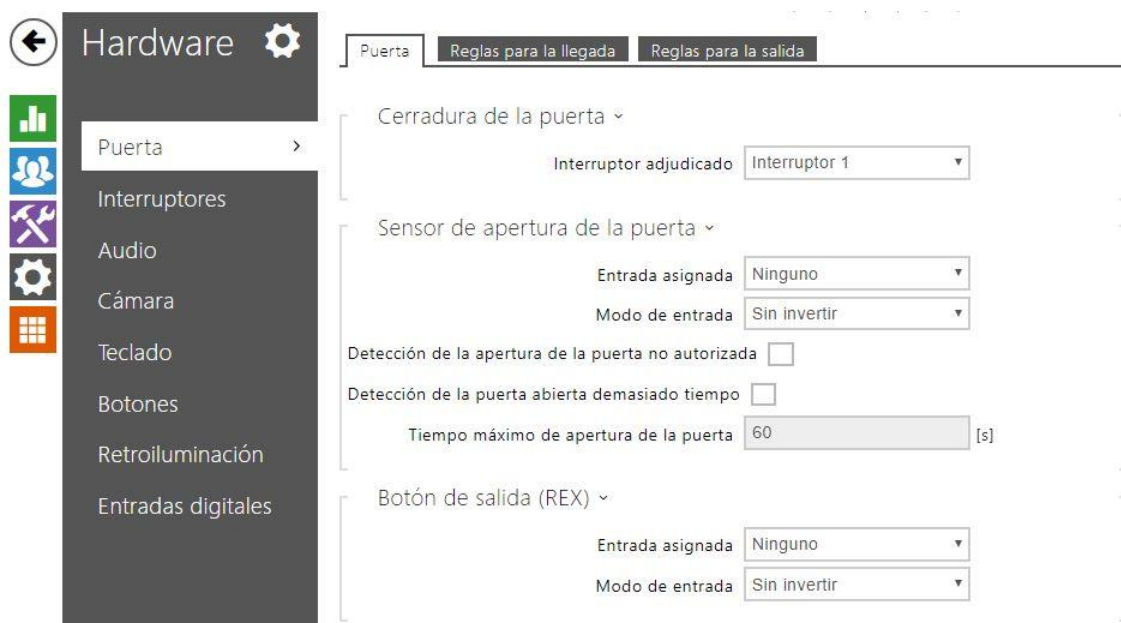


Figura 5. 7.- Pantalla de parametrización de la puerta del intercomunicador.

El interruptor, Figura 5.8, será monoestable, lo que quiere decir que con la pulsación se le dará tensión el tiempo que se le asigne, si fuera monoestable, al pulsar el botón empezaría a darle tensión para que quedara abierta hasta que se volviera a pulsar y entonces se cerrara.

La salida será por relé y desde los dispositivos que sean habrá que enviar un código DTMF para que el intercomunicador envíe la tensión al relé. En este caso el código será 101, pero puede ser cualquier código que este previamente guardado.

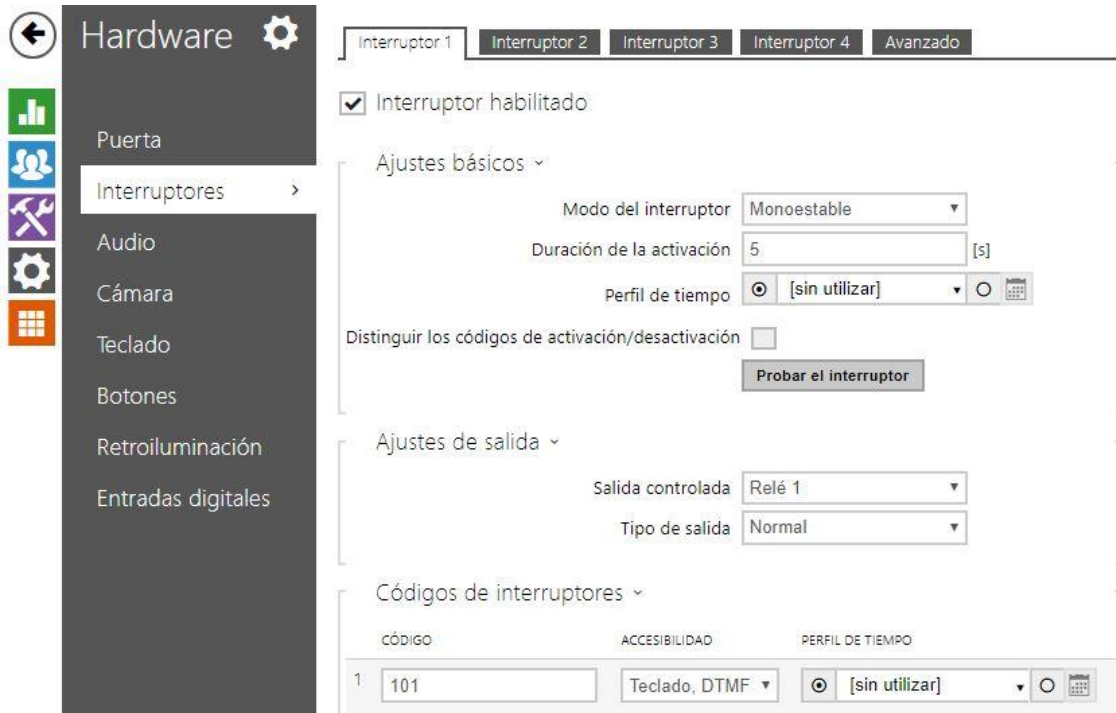


Figura 5. 8.- Pantalla de parametrización de interruptores del intercomunicador.

El botón físico del intercomunicador sirve para realizar llamada directa al usuario que se programe. En el caso de que existieran mas botones o incluso teclado numérico, las posibilidades de llamada serían más amplias.

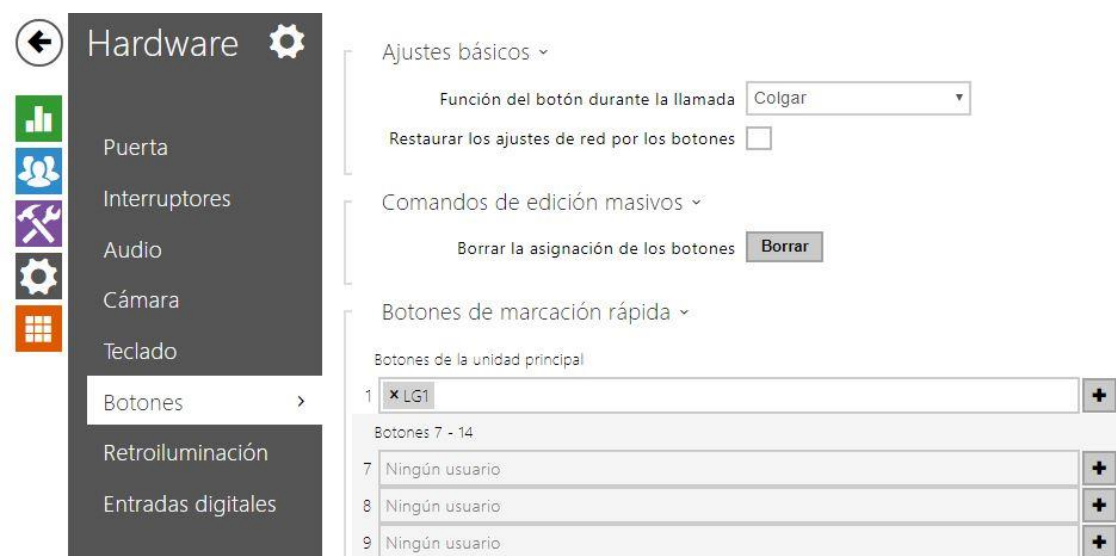
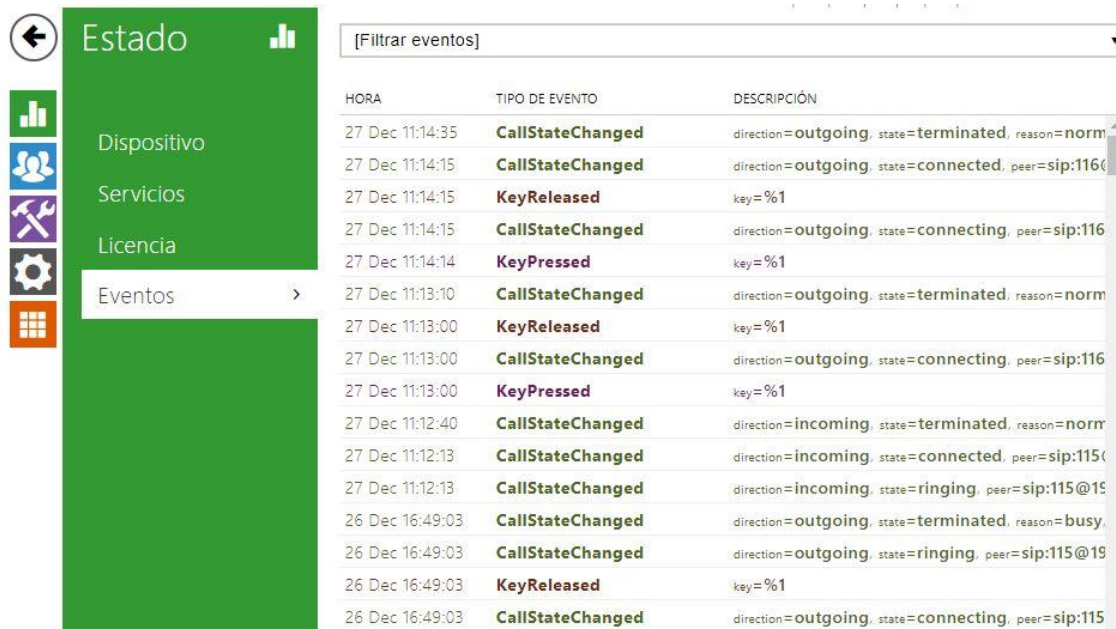


Figura 5. 9.- Pantalla de parametrización de botones del intercomunicador.

Para finalizar con esta parte, dentro del apartado Estado-Eventos, se encuentra un registro de todos los eventos ocurridos en el intercomunicador, siendo llamadas entrantes, llamadas salientes, pulsación del botón, apertura del relé...



HORA	TIPO DE EVENTO	DESCRIPCIÓN
27 Dec 11:14:35	<b>CallStateChanged</b>	direction=outgoing, state=terminated, reason=norm
27 Dec 11:14:15	<b>CallStateChanged</b>	direction=outgoing, state=connected, peer=sip:116
27 Dec 11:14:15	<b>KeyReleased</b>	key=%1
27 Dec 11:14:15	<b>CallStateChanged</b>	direction=outgoing, state=connecting, peer=sip:116
27 Dec 11:14:14	<b>KeyPressed</b>	key=%1
27 Dec 11:13:10	<b>CallStateChanged</b>	direction=outgoing, state=terminated, reason=norm
27 Dec 11:13:00	<b>KeyReleased</b>	key=%1
27 Dec 11:13:00	<b>CallStateChanged</b>	direction=outgoing, state=connecting, peer=sip:116
27 Dec 11:13:00	<b>KeyPressed</b>	key=%1
27 Dec 11:12:40	<b>CallStateChanged</b>	direction=incoming, state=terminated, reason=norm
27 Dec 11:12:13	<b>CallStateChanged</b>	direction=incoming, state=connected, peer=sip:115
27 Dec 11:12:13	<b>CallStateChanged</b>	direction=incoming, state=ringing, peer=sip:115@15
26 Dec 16:49:03	<b>CallStateChanged</b>	direction=outgoing, state=terminated, reason=busy,
26 Dec 16:49:03	<b>CallStateChanged</b>	direction=outgoing, state=ringing, peer=sip:115@19
26 Dec 16:49:03	<b>KeyReleased</b>	key=%1
26 Dec 16:49:03	<b>CallStateChanged</b>	direction=outgoing, state=connecting, peer=sip:115

Figura 5. 10.- Pantalla de eventos del intercomunicador.



## 6.- Manejo del panel

Para completar la funcionalidad de la aplicación puede ser importante saber cómo funciona el panel de pruebas, para el caso de que se quieran realizar comprobaciones.

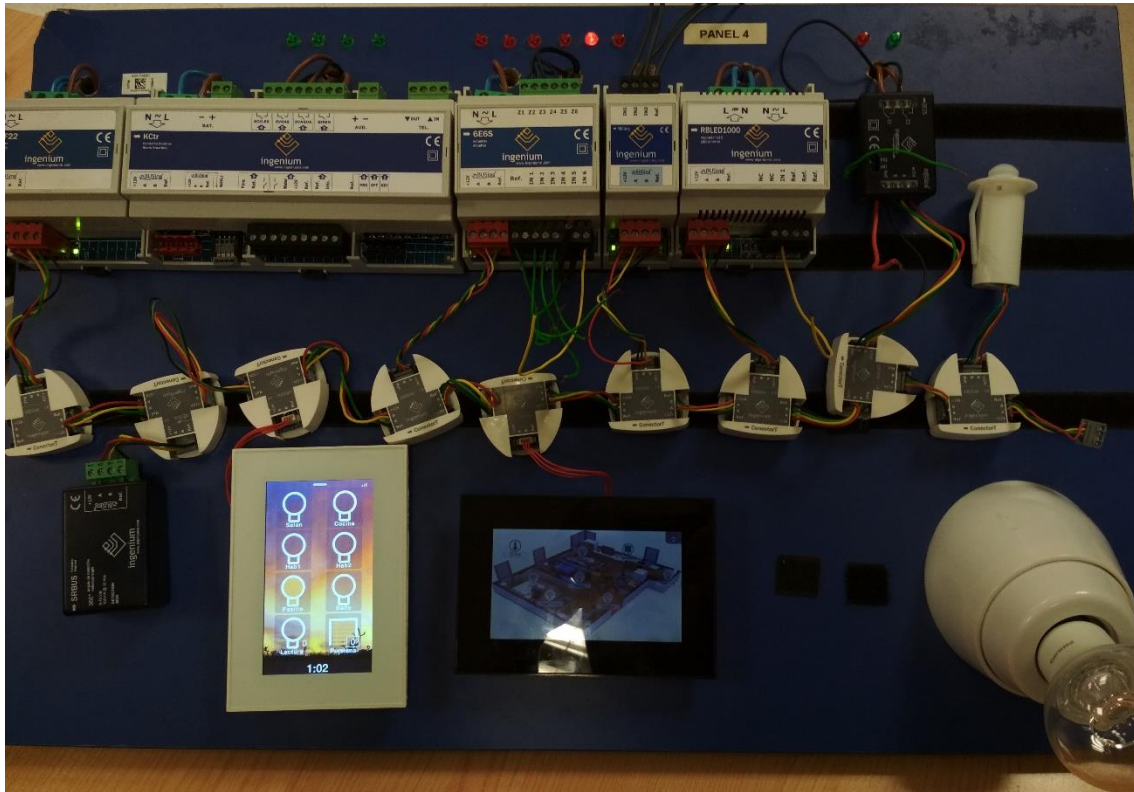


Figura 6. 1.- Panel de pruebas.

El panel de pruebas de la Figura 6.1, está compuesto por, en la parte de arriba de izquierda a derecha, una fuente de alimentación, una KCtr para el control de las alarmas, un actuador 6 entradas – 6 salidas para el control de las luces fijas, un MECing para la posibilidad de programar escenas, un RBLED1000 conectado a la bombilla de regulación, un actuador 2 entradas – 2 salidas para la persiana y un sensor de presencia. En la parte central hay 9 conectores T para la conexión del bus, y en la parte inferior, también de izquierda a derecha, hay un SRBUS o sonda de temperatura, una Smart Touch que posee un termostato, una PPL-4 y la bombilla de regulación. Además, también se cuenta con 12 pequeñas bombillas fijas encima de los elementos superiores del panel.

Para las pruebas realizadas se han despreciado la KCtr, el sensor de presencia, la sonda de temperatura y 4 de las 12 bombillas fijas, las 4 verdes pertenecientes a la KCtr.

En cuanto al manejo que hay que conocer para interactuar con el panel, será el de la Smart Touch y la PPL-4, desde las cuales podrá controlarse en todo momento la activación o desactivación de los componentes para ver si la aplicación móvil recibe y envía bien los datos.

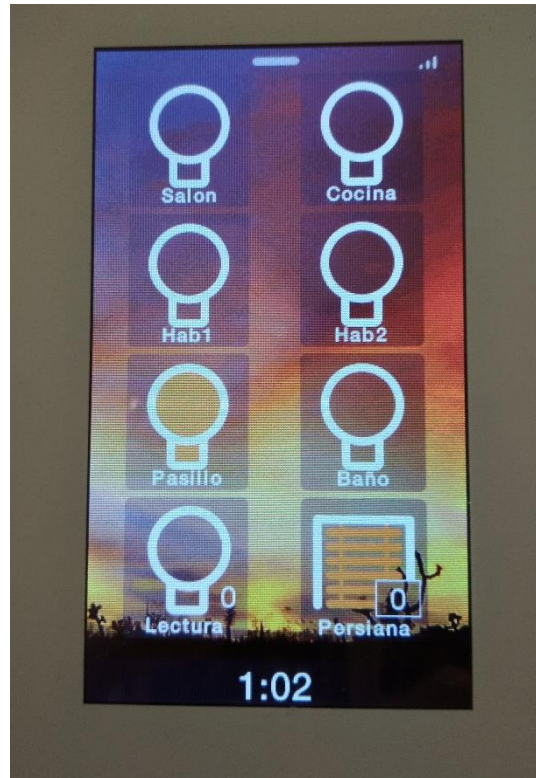


Figura 6. 2.- Pantalla de la Smart Touch del panel.

Al igual que en la aplicación del móvil desarrollada en estos documentos, a través de las pulsaciones cortas se encienden o apagan las luces fijas. En el caso de la bombilla de regulación y la persiana, con la pulsación corta se abre una pantalla nueva donde se muestra su porcentaje y ahí si es necesario deslizar el dedo hacia arriba o hacia abajo para su regulación.

También cuenta con una pantalla en la que se puede configurar aspectos del tiempo, la apariencia, la seguridad, el idioma... así como ajustar su configuración WiFi y su registro en la nube, Figura 6.3



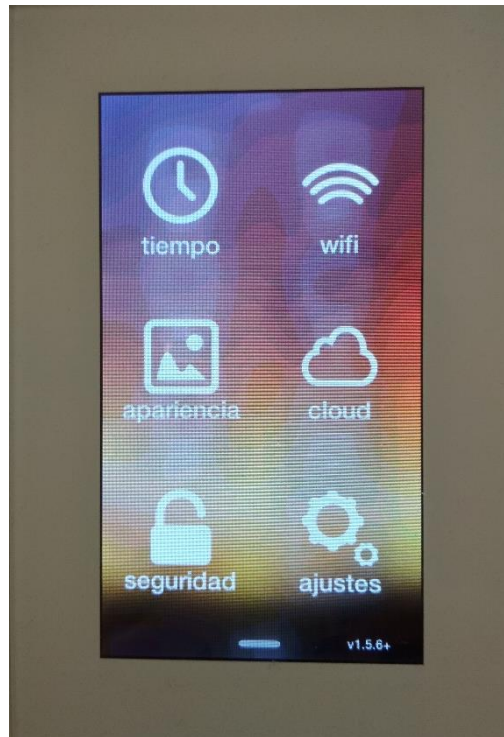


Figura 6. 3.- Pantalla de ajustes de la pantalla del panel.

Para finalizar, está también la PPL-4 que también servirá para visualizar, además de las bombillas o persiana, el estado y temperatura del clima. Su uso es igual al de la SmartTouch.

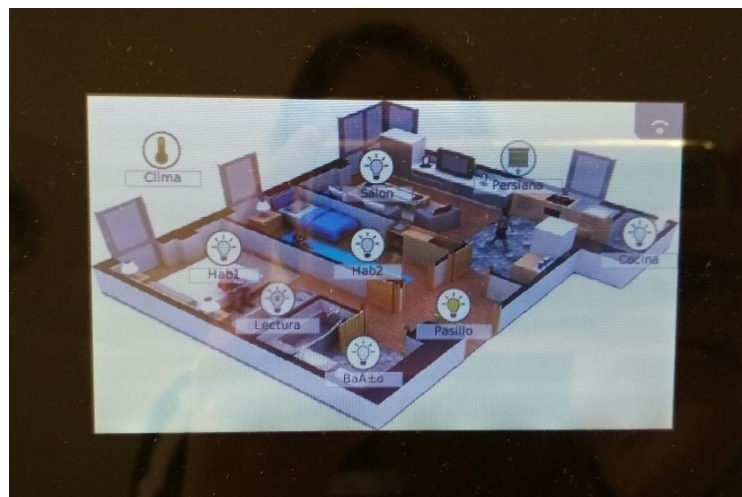


Figura 6. 4.- Pantalla de la PPL-4 del panel.



Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

LAURA RICO ÁLVAREZ

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

## **DESARROLLO DE SERVICIO SIP PARA VIDEOPORTERO**

Presupuesto

ENERO 2019



# Índice general

1.	Introducción.....	5
1.1.	DESCRIPCIÓN DEL PROYECTO .....	5
1.2.	VISIÓN DEL DOCUMENTO.....	5
2.	Recursos hardware.....	6
3.	Recursos software.....	7
4.	Recursos humanos.....	8
5.	Presupuesto final .....	9
5.1.	PRESUPUESTOS PARCIALES .....	9
5.2.	PRESUPUESTO DE EJECUCIÓN.....	9

# Índice de tablas

Tabla 2. 1.- Cuadro de precios de recursos hardware.....	6
Tabla 3. 1.- Cuadro de precios de recursos software.....	7
Tabla 4. 1.- Cuadro de precios de recursos humanos.....	8
Tabla 5. 1.- Cuadro de presupuestos parciales.....	9
Tabla 5. 2.- Cuadro de presupuestos de ejecución.....	9

# 1. Introducción

## 1.1. DESCRIPCIÓN DEL PROYECTO

<b>Título</b>	Desarrollo de servicio SIP para videoportero
<b>Tutor:</b>	Antonio Robles Álvarez
<b>Autor:</b>	Laura Rico Álvarez
<b>Titulación:</b>	Máster en Automatización e Informática Industrial
<b>Fecha:</b>	Enero de 2019
<b>Financiación:</b>	INGENIUM, Ingeniería y Domótica, S.L.

## 1.2. VISIÓN DEL DOCUMENTO

El presente documento da a conocer el coste detallado del proyecto...

## 2. Recursos hardware

Escuela Politécnica de Ingeniería de Gijón					
ID UNIDAD	DESCRIPCIÓN	MEDICIÓN	UNIDADES	PRECIO UNITARIO (€)	PRECIO TOTAL (€)
HW01	Ordenador tipo PC	Uds.	1	6000,00	600,00
HW02	Smartphone	Uds.	1	300,00	300,00
HW03	Intercomunicador	Uds.	1	570,00	570,00
HW04	Actuador 2E2S	Uds.	1	70,00	70,00
HW05	Actuador 6E6S	Uds.	1	195,00	195,00
HW06	Regulador RBLED1000	Uds.	1	120,00	120,00
HW07	Smart Touch	Uds.	1	600,00	600,00
HW08	PPL-4	Uds.	1	500,00	500,00
HW09	MECing	Uds.	1	54,00	54,00
HW10	Conector T	Uds.	7	20,00	140,00
HW11	Fuente	Uds.	2	68,00	136,00
HW12	Bombilla LED	Uds.	1	5,00	5,00
HW13	Bombilla	Uds.	8	0,50	4,00
HW14	Router	Uds.	1	70,00	70,00
<b>PRECIO TOTAL HARDWARE</b>					<b>3,364</b>

Tabla 2. 1.- Cuadro de precios de recursos hardware.

### 3. Recursos software

Escuela Politécnica de Ingeniería de Gijón					
ID UNIDAD	DESCRIPCIÓN	MEDICIÓN	UNIDADES	PRECIO UNITARIO (€)	PRECIO TOTAL (€)
SW01	Microsoft Windows 10	Uds.	1	135,00	135,00
SW02	Oracle VirtualBox	Uds.	1	0	0
SW03	RAD Studio	Uds.	1	4122,50	4122,50
SW04	Android Studio	Uds.	1	0	0
SW05	Asterisk	Uds.	1	0	0
SW06	Elastix	Uds.	1	0	0
SW07	SiDe	Uds.	1	550,00	550,0
<b>PRECIO TOTAL SOFTWARE</b>					<b>4807,50</b>

Tabla 3. 1.- Cuadro de precios de recursos software.



## 4. Recursos humanos

Escuela Politécnica de Ingeniería de Gijón					
ID UNIDAD	DESCRIPCIÓN	MEDICIÓN	UNIDADES	PRECIO UNITARIO (€)	PRECIO TOTAL (€)
HU01	Análisis y diseño software	Horas	294	22,00	6468,00
HU02	Desarrollo software	Horas	376	16,00	6016,00
<b>PRECIO TOTAL RECURSOS HUMANOS</b>					<b>12484,00</b>

Tabla 4. 1.- Cuadro de precios de recursos humanos.

# 5. Presupuesto final

## 5.1. PRESUPUESTOS PARCIALES

Escuela Politécnica de Ingeniería de Gijón	
PRESUPUESTO	IMPORTE (€)
Recursos hardware	3364,00
Recursos software	4807,50
Recursos humanos	12484,00

Tabla 5. 1.- Cuadro de presupuestos parciales.

## 5.2. PRESUPUESTO DE EJECUCIÓN

Escuela Politécnica de Ingeniería de Gijón	
CONCEPTO	IMPORTE (€)
Presupuesto de ejecución material	20655,50
Beneficio industrial (6%)	1239,33
Costes generales (15%)	3098,33
Total sin I.V.A	24993,16
I.V.A	5248,56
Total	30241,72

Tabla 5. 2.- Cuadro de presupuestos de ejecución.

El presupuesto asciende a la cantidad de **TREINTA MIL DOSCIENTOS CUARENTA Y UN EUROS CON SETENTA Y DOS CÉNTIMOS.**



Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

LAURA RICO ÁLVAREZ

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

## **DESARROLLO DE SERVICIO SIP PARA VIDEOPORTERO**

Pruebas

ENERO 2019



# Índice general

1.- Introducción .....	5
1.1.- DESCRIPCIÓN DEL PROYECTO.....	5
1.2.- VISIÓN DEL DOCUMENTO .....	5
2.- Condiciones de diseño .....	6
2.1.- MÓDULO DE CONEXIÓN BUSING® .....	6
2.2.- MÓDULO DE LECTURA .....	6
2.3.- MÓDULO DE CONEXIÓN AL INTERCOMUNICADOR .....	7
2.4.- MÓDULO DE LLAMADAS.....	7
3.- Diseño de pruebas .....	8
3.1.- CASOS DE PRUEBA PARA EL MÓDULO DE CONEXIÓN BUSING® .....	8
3.2.- CASOS DE PRUEBA PARA EL MÓDULO DE LECTURA .....	9
3.3.- CASOS DE PRUEBA PARA EL MÓDULO DE CONEXIÓN AL INTERCOMUNICADOR.....	10
3.4.- CASOS DE PRUEBA PARA EL MÓDULO DE LLAMADAS .....	11

# Índice de tablas

Tabla 3. 1.- Caso de prueba P001.....	8
Tabla 3. 2.- Caso de prueba P002.....	8
Tabla 3. 3.- Caso de prueba P003.....	9
Tabla 3. 4.- Caso de prueba P004.....	9
Tabla 3. 5.- Caso de prueba P005.....	10
Tabla 3. 6.- Caso de prueba P006.....	10
Tabla 3. 7.- Caso de prueba P007.....	11
Tabla 3. 8.- Caso de pruebas P008. ....	11

# 1.- Introducción

## 1.1.- DESCRIPCIÓN DEL PROYECTO

<b>Título</b>	Desarrollo de servicio SIP para videoportero
<b>Tutor:</b>	Antonio Robles Álvarez
<b>Autor:</b>	Laura Rico Álvarez
<b>Titulación:</b>	Máster en Automatización e Informática Industrial
<b>Fecha:</b>	Enero de 2019
<b>Financiación:</b>	INGENIUM, Ingeniería y Domótica, S.L.

## 1.2.- VISIÓN DEL DOCUMENTO

Este documento expone el diseño y los resultados de las pruebas realizadas al software desarrollado.

## 2.- Condiciones de diseño

Dentro de la aplicación, se pueden diferenciar distintos módulos que realizan funciones específicas. Dichas funciones han de ser verificadas para garantizar el correcto funcionamiento de estas.

### 2.1.- MÓDULO DE CONEXIÓN BUSING®

Como primer módulo puede entenderse la conexión de la aplicación al servidor o pantalla del panel de pruebas.

Dependiendo del tipo de conexión debe cumplir diferentes requisitos.

Para la conexión a la pantalla del panel de pruebas:

- Asegurarse de que la conexión con la pantalla se ha realizado con éxito.

Para la conexión a través del servidor:

- Asegurarse de que la conexión al servidor se ha realizado con éxito.
- Leer el ID enviado por el servidor si las credenciales son correctas.
- Asegurarse de que la conexión con la pantalla se ha realizado con éxito.

En ambas conexiones, ante un fallo en la conexión se avisará al usuario por pantalla.

### 2.2.- MÓDULO DE LECTURA

El segundo de los módulos de la aplicación es el de lectura del bus.

- Comprobar la conexión a la pantalla.
- Verificar que hay elementos que leer en el buffer.
- Leer los elementos del buffer.
- Separarlos según cada componente.
- Simular la lectura del telegrama.



### **2.3.- MÓDULO DE CONEXIÓN AL INTERCOMUNICADOR**

Este es uno de los módulos más importantes de la aplicación, ya que es el encargado de conectar la pantalla al servidor del intercomunicador.

- Realizar la carga de librerías PJSIP.
- Configurar los elementos de inicio.
- Creación de la cuenta SIP de usuario.
- Registrar las credenciales en el servidor.
- Comprobar el registro.

### **2.4.- MÓDULO DE LLAMADAS**

Por último, el módulo de gestión de llamadas será el que permita la comunicación entre el intercomunicador y el usuario de la aplicación.

- Permanecer activo constantemente.
- Dirigir las llamadas al servidor.
- Asegurarse de que en la pantalla se muestre siempre la imagen captada por el intercomunicador.

## 3.- Diseño de pruebas

Una vez explicador los diferentes módulos que componen la aplicación, es el momento de explicar las diferentes pruebas que se han realizado para comprobar su correcto funcionamiento.

Las pruebas que se han realizado han sido siempre con los elementos comentados en los demás documentos que forman el proyecto.

### 3.1.- CASOS DE PRUEBA PARA EL MÓDULO DE CONEXIÓN BUSING®

Función: Conexión directa a la pantalla del panel de pruebas
Caso de prueba: P001
Valores introducidos: <ul style="list-style-type: none"> <li>• IP: 192.168.0.45</li> <li>• Puerto: 12347</li> </ul>
Salida esperada: La aplicación ha de crear un cliente con el puerto y la dirección IP indicadas y conectarse.
Resultado obtenido: Conexión directa a la pantalla.
Medidas correctivas: Ninguna.

Tabla 3. 1.- Caso de prueba P001.

Función: Conexión a la pantalla a través del servidor.
Caso de prueba: P002
Valores introducidos: <ul style="list-style-type: none"> <li>• Usuario: stlaura</li> <li>• Contraseña: 1234</li> </ul>
Salida esperada: La aplicación ha de conectarse primero al servidor a través del puerto correcto y pasarle las credenciales, al ser correctas, este enviará un ID de verificación para que la aplicación se pueda conectar a la pantalla a través del servidor.
Resultado obtenido: La aplicación se conecta primero al servidor correctamente y a continuación hace lo mismo con la pantalla.
Medidas correctivas: Ninguna.

Tabla 3. 2.- Caso de prueba P002.

Función: Pérdida de conexión
Caso de prueba: P003
Valores introducidos: <ul style="list-style-type: none"><li>• IP: 192.168.0.45</li><li>• Puerto: 12347</li></ul>
Salida esperada: Al perder la conexión, por ejemplo, de WiFi la aplicación perderá la señal y por lo tanto se desconectará del servidor o de la pantalla.
Resultado obtenido: La aplicación muestra por pantalla que ha ocurrido un error de conexión.
Medidas correctivas: Ninguna.

Tabla 3. 3.- Caso de prueba P003.

### 3.2.- CASOS DE PRUEBA PARA EL MÓDULO DE LECTURA

Función: Lectura del bus
Caso de prueba: P004
Valores introducidos: Ninguno
Salida esperada: La aplicación ha de comprobar que está conectada a la pantalla y que hay datos de lectura en el bus, entonces leerá dichos datos y separará los elementos del buffer dependiendo de la posición que ocupan.
Resultado obtenido: La aplicación realiza la comprobación de la conexión y espera a que haya algo que leer en el buffer, entonces dependiendo de su posición los separa según direcciones, comando o datos.
Medidas correctivas: Ninguna.

Tabla 3. 4.- Caso de prueba P004.

### 3.3.- CASOS DE PRUEBA PARA EL MÓDULO DE CONEXIÓN AL INTERCOMUNICADOR

Función: Conexión al intercomunicador.
Caso de prueba: P005
Valores introducidos: <ul style="list-style-type: none"> <li>• IP: 192.168.0.37</li> <li>• Extensión: 115</li> <li>• Contraseña: 068799</li> </ul>
Salida esperada: La aplicación se conectará a esa dirección IP, que pertenece al servidor/intercomunicador y a través de la extensión y la contraseña realizará un registro en este, que hará una relación entre la dirección IP de la aplicación y la extensión que se ha indicado.
Resultado obtenido: La aplicación se registra en el intercomunicador, pero este no realiza la relación entre la dirección IP de la aplicación y la extensión que se le ha indicado.
Medidas correctivas: Para llamar, durante la fase de pruebas con este intercomunicador, habrá que indicarle, además de la extensión habrá que marcar la dirección IP de la aplicación.

Tabla 3. 5.- Caso de prueba P005.

Función: Conexión errónea al intercomunicador.
Caso de prueba: P006
Valores introducidos: <ul style="list-style-type: none"> <li>• IP: 192.168.0.35</li> <li>• Extensión: 115</li> <li>• Contraseña: 068799</li> </ul>
Salida esperada: La aplicación intentará conectarse a un servidor erróneo y la conexión será fallida.
Resultado obtenido: Aparece por pantalla que la conexión ha fallado.
Medidas correctivas: Ninguna.

Tabla 3. 6.- Caso de prueba P006.

### 3.4.- CASOS DE PRUEBA PARA EL MÓDULO DE LLAMADAS

Función: Llamada desde el intercomunicador a la aplicación.
Caso de prueba: P007
Valores introducidos: <ul style="list-style-type: none"><li>• IP: 192.168.0.15</li><li>• Extensión: 115</li></ul>
Salida esperada: El intercomunicador dirigirá la llamada al móvil en el que se encuentra instalada la aplicación y en la aplicación aparecerá una pantalla emergente con las imágenes captadas por el intercomunicador y la opción de aceptar o rechazar la llamada.
Resultado obtenido: El servidor dirige la llamada al móvil en el que está instalada la aplicación y en este aparece la pantalla espera.
Medidas correctivas: Ninguna.

Tabla 3. 7.- Caso de prueba P007.

Función: Llamada de la aplicación al intercomunicador.
Caso de prueba: P006
Valores introducidos: <ul style="list-style-type: none"><li>• IP: 192.168.0.37</li></ul>
Salida esperada: La aplicación llamará al servidor directamente y la llamada será instantáneamente conectada.
Resultado obtenido: Comienza la llamada y aparecen en pantalla las imágenes captadas por el intercomunicador.
Medidas correctivas: Ninguna.

Tabla 3. 8.- Caso de pruebas P008.