Universitat de les Illes Balears

escola politècnica superior

Trabajo Final de Grado

INGENIERÍA TÉCNICA EN TELECOMUNICACIONES,
ESPECIALIDAD TELEMÁTICA

Smart Bell

RODRIGO ALAN VERA BAUTISTA

Tutor
Dr. Bartomeu Alorda Ladaria

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, 1 de febrero de 2018

ÍNDICE GENERAL

Índice general i

Acrónimos iii

Lista de Figuras iv

Lista de Códigos vii

Lista de Tablas viii

Resumen ix

Agradecimientos xi

1 Introducción y Objetivos 1
- 1.1 Introducción . . . 1
  - 1.1.1 Motivación . . . 2
- 1.2 Objetivos . . . 3
  - 1.2.1 Objetivo principal . . . 3
  - 1.2.2 Objetivos específicos . . . 3
  - 1.2.3 Tareas . . . 4

2 Esquema general 5
- 2.1 Hardware . . . 5
- 2.2 Software . . . 6

3 Hadware SmartBell 7
- 3.1 Placas con microcontrolador . . . 7
  - 3.1.1 Arduino Uno rv3 . . . 8
- 3.2 Comunicación sin cables . . . 24
  - 3.2.1 Módulo WiFly RN-171 . . . 24

4 Aplicación SmartBell 29
- 4.1 IDE - Framework . . . 29
  - 4.1.1 Aplicación Android . . . 30

5 Resultados 37
- 5.1 Configuración de la red WiFi . . . 39

5.2 Conexión con el timbre . . . 39
5.3 Conexión con la aplicación Android . . . 40
5.3.1 Historial y configuración de notificaciones . . . 41
5.4 Notificaciones . . . 42
5.5 Conexión desde Internet con el SmartBell . . . 44

6 Conclusiones **49**
6.1 Objetivos . . . 49
6.2 Valoración personal . . . 49
6.3 Líneas de trabajo futuro . . . 50

A Códigos del Proyecto **51**
A.1 Códigos de Arduino . . . 51
A.1.1 Lectura y escritura de memoria EEPROM . . . 51
A.1.2 Tono de notificación auditiva . . . 53
A.1.3 Conexion con red WiFi . . . 54
A.1.4 Interacción con solicitudes de la aplicación Android . . . 54
A.2 Métodos importantes de la aplicación SmartBell . . . 56
A.2.1 onCreate() de *MainActivity* . . . 56
A.2.2 crearTimer() de *MainActivity* . . . 57
A.2.3 estadoTimbre() de *MainActivity* . . . 58
A.2.4 setSERVERIP() de *TCPClient* . . . 61
A.2.5 setTimeOut() de *TCPClient* . . . 61
A.2.6 run() de *TCPClient* . . . 61
A.2.7 stopClient() de *TCPClient* . . . 62
A.2.8 onCreateView() de *InicioFragment* . . . 63
A.2.9 conectar() de *InicioFragment* . . . 64
A.2.10 mostrarDesconectar() de *InicioFragment* . . . 65
A.2.11 mostrarConectado() de *InicioFragment* . . . 65
A.2.12 onCreateView() de *HistorialFragment* . . . 66
A.2.13 agregarLista() de *HistorialFragment* . . . 67
A.2.14 limpiarLista() de *HistorialFragment* . . . 67
A.2.15 onCreateView() de *ConfiguracionFragment* . . . 67
A.2.16 configuracionActual() de *ConfiguracionFragment* . . . 69
A.2.17 activoDesactivo() de *ConfiguracionFragment* . . . 70
A.2.18 cambiarConfig() de *ConfiguracionFragment* . . . 70
A.2.19 noConectado() de *ConfiguracionFragment* . . . 71
A.2.20 AndroidManifest.xml . . . 71

Bibliografía **73**

ACRÓNIMOS

SB SmartBell

IP Internet Protocol

SSID Service Set Identifier

DHCP Dynamic Host Configuration Protocol

LAN Local Area Network

WAN Wide Area Network

UDP User Datagram Protocol

AC Alternating Current

DC Direct current

Vin Voltaje de entrada

GND Ground

UART Universal Asynchronous Receiver-Transmitter

RX Receptor de datos

TX Transmisor de datos

SBC Single Board Computer

CPU Central Processing Unit

GPU Graphics Processing Unit

PWM Pulse-Width Modulation

TCP Transport Control Protocol

LISTA DE FIGURAS

1.1 Ring [12] . . . 2
1.2 Nucli Smart Lock [13] . . . 3

2.1 Esquema del Hardware . . . 5
2.2 Esquema del Software . . . 6

3.1 Arduino Uno rv3 [1] . . . 8
3.2 Esquema del Arduino [2] . . . 8
3.3 Puente de diodos [3] . . . 10
3.4 Diagrama de flujo del funcionamiento de SB . . . 13
3.5 Monitor Serie de Arduino IDE . . . 13
3.6 Diagrama de flujo de la comunicación con el *Monitor Serie* . . . 14
3.7 Diagrama de flujo del menú de conexión WiFi . . . 15
3.8 Menú de configuración de la conexión WiFi . . . 16
3.9 Conexión con la red WiFi . . . 16
3.10 Configuración de *DHCP* . . . 17
3.11 Implementación de la configuración WiFi . . . 17
3.12 Diagrama de flujo del menú de notificaciones . . . 18
3.13 Menú de configuración de las notificaciones . . . 19
3.14 Configuración de notificación luminosa . . . 19
3.15 Transformador HLK-PM01 [7] . . . 20
3.16 Hex Buffer MC14050B . . . 20
3.17 Pines de MC14050B [8] . . . 20
3.18 Diagrama de flujo de las notificaciones . . . 21
3.19 Esquema del funcionamiento de las notificaciones . . . 22
3.20 Buzzer *AST-030c0mr-r* [9] . . . 22
3.21 *Relé KY-019* [10] . . . 23
3.22 Conexión de bombilla LED con Arduino . . . 23
3.23 WiFly RN-171 [4] . . . 25
3.24 Arduino Wireless SD Shield [5] . . . 26
3.25 Modificación de Pines RX y TX de Shield [6] . . . 26
3.26 Diagrama de flujo de la comunicación Socket . . . 27

4.1 Información enviada por *Ionic*, en comunicación Socket . . . 30
4.2 Información enviada por *Android*, en comunicación Socket . . . 30
4.3 Diagrama de flujo de la aplicación Android . . . 31
4.4 *Fragments* de la aplicación Android . . . 32

4.5 Fragment de inicio 33
4.6 Fragment de historial 34
4.7 Fragment de inicio 35

5.1 Circuito montado en Protoboard 37
5.2 Circuitos impresos 38
5.3 Circuitos montados y conectados 38
5.4 Conexión del SB con el ordenador 39
5.5 Conexión con el timbre 39
5.6 Alimentación del SB 40
5.7 Conexión de aplicación Android con SB 40
5.8 Conexión exitosa de aplicación Android con SB 41
5.9 Historial de las veces que llamaron al timbre 41
5.10 Configuración de las notificaciones del SB 42
5.11 LEDs de notificaciones del SB 42
5.12 Notificaciones centralizadas del SB 43
5.13 Notificación del SB al SmartPhone 44
5.14 Obtención de la IP de un Router doméstico 45
5.15 Conexión con el Router desde un navegador web 45
5.16 WAN IP del Router 46
5.17 Abrir el puerto 666 en el Router 46
5.18 Vídeo explicativo alojado en Youtube 47

Lista de Códigos

A.1 Lectura en EEPROM 51
A.2 Escritura en EEPROM 52
A.3 Código del tono [21] 53
A.4 Conectar con red WiFi 54
A.5 Mensajes de solicitudes de la aplicación Android 54
A.6 Método onCreate de la clase *MainActivity* 56
A.7 Método crearTimer de la clase *MainActivity* 57
A.8 Método estadoTimbre de la clase *MainActivity* 58
A.9 Método setSERVERIP de la clase *TCPClient* 61
A.10 Método setTimeOut de la clase *TCPClient* 61
A.11 Método run de la clase *TCPClient* 61
A.12 Método stopClient de la clase *TCPClient* 62
A.13 Método onCreateView de la clase *InicioFragment* 63
A.14 Método conectar de la clase *InicioFragment* 64
A.15 Método mostrarDesconectar de la clase *InicioFragment* 65
A.16 Método mostrarConectado de la clase *InicioFragment* 65
A.17 Método onCreateView de la clase *HistorialFragment* 66
A.18 Método agregarLista de la clase *HistorialFragment* 67
A.19 Método limpiarLista de la clase *HistorialFragment* 67
A.20 Método onCreateView de la clase *ConfiguracionFragment* 67
A.21 Método configuracionActual de la clase *ConfiguracionFragment* 69
A.22 Método activoDesactivo de la clase *ConfiguracionFragment* 70
A.23 Método cambiarConfig de la clase *ConfiguracionFragment* 70
A.24 Método noConectado de la clase *ConfiguracionFragment* 71
A.25 Contenido del *AndroidManifest.xml* 71

LISTA DE TABLAS

3.1 Uso de memoria EEPROM 11
3.2 Funcionamiento de MC14050B [8] 21
3.3 LEDs informativos módulo WiFly [11] 25

A.1 Estructura del Byte **estadoSB** 52

RESUMEN

El proyecto se basa en experimentar una conexión entre dispositivos electrónicos con un móvil, mediante una aplicación. Para poder gestionar los eventos de dichos dispositivos electrónicos, mejorando así la calidad de vida en el hogar (Domótica).

El dispositivo gestionado en este proyecto es un timbre mediante una aplicación Android. De esta forma, SmartBell (SB) el *timbre inteligente* desarrollado sobre *Arduino*, notifica a la aplicación móvil cuando alguien toca el timbre, también se puede activar/-desactivar las notificaciones desde la aplicación. La conexión entre SB y la aplicación se realiza por WiFi mediante una comunicación Socket.

SB aporta un control del ruido que genera un timbre convencional. También da un soporte a las personas con dificultades auditivas, ya que incorpora una notificación luminosa, la cual consiste en encender una bombilla LED. SB agrega una notificación inteligente, puesto que notifica a la aplicación móvil cuando alguien llama a la puerta y se almacena en un historial para un mejor control del timbre.

AGRADECIMIENTOS

Este proyecto va dedicado a la memoria de mis abuelos Jacoba y Sebastian, quienes siempre me alentaron a estudiar y a seguir adelante. Su gran sueño siempre fue verme egresar de la universidad. También agradecer a mi madre querida Marina, quien siempre me guía y apoya. Especial mención a mi novia Katherine quien siempre fue un sustento a lo largo de esta carrera. No quisiera olvidarme de mi familia y mis amigos, pero no puedo nombrarlos a todos. En definitiva gracias a mis amigos, familia y profesores que estuvieron a mi lado.

CAPÍTULO 1

INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

Si se tiene un bebé y está durmiendo, lo último que se desea es que alguien toque el timbre, porque el bebé se despertará, pero tampoco se quiere dejar de saber cuándo alguien llama a la puerta. Puede ocurrir algo parecido si algún familiar que trabaja por las noches y duerme durante el día (tampoco se desea que se levante por el ruido del timbre).

¿Y qué sucede con las personas con dificultades auditivas?, En estos casos también deben saber cuándo alguien llama a la puerta. O simplemente se está en el jardín y no se escucha cuando alguien llama a la puerta.

SmartBell (SB), el timbre inteligente solventa estos problemas. Desde éste, conectándose a la red WiFi de casa, se podrá recibir las notificaciones si alguien llama al timbre. Puede notificar de forma *Auditiva*, *Luminosa* e *Inalámbrica*.

Estas tres formas de notificación pueden funcionar de forma simultánea o combinada. Para activar o desactivar las notificaciones se podrán hacer desde un SmartPhone o cualquier dispositivo Android, que tenga la aplicación instalada y configurada correctamente, o conectándote a SB con un ordenador mediante un cable USB, donde también se configura la conexión WiFi.

El proyecto se ha desarrollado sobre Arduino, ya que ofrece muchas facilidades, tanto a nivel de componentes, librerías, como en foros en Internet, usando un módulo WiFi para la conexión a la red y otros elementos más comunes como LEDs, resistencias, etcétera; para el funcionamiento del timbre. Todo esto se verá de forma más detallada en el **capítulo: 3**.

Durante el desarrollo del proyecto se fueron realizando diferentes pruebas para estudiar las características y el funcionamiento del SB.

En este documento se expondrá y explicará, en primer lugar, un esquema general, tanto del Hardware como Software de este proyecto. Éste ayudará a entender de una manera rápida el modo de funcionamiento del SB.

Después del esquema del proyecto, se explicarán las decisiones de diseño tanto del Hardware como del Software, donde se detallarán los motivos por los que se ha decidido utilizar los dispositivos y tecnologías ya mencionadas, realizando comparaciones con otros dispositivos y tecnologías similares que se podrían haber utilizado. También se describirá de una forma detallada y clara cada una de las tecnologías utilizadas.

A continuación, se mostrarán y explicarán los resultados obtenidos en cada una de las posibles situaciones en las que podría trabajar el SB. Finalmente, se expondrán futuras mejoras del SB y también se mostrarán las conclusiones a las que se han llegado una vez concluido el proyecto.

1.1.1 Motivación

Esta idea nació a como fruto de la actividad laboral del alumno como repartidor de pizzas. En este oficio, algunos clientes pedían que no se tocara el timbre y se les avisase a través de una llamada perdida al teléfono móvil, ya que éste permite tenerlo en silencio, ya sea porque tenían al bebé durmiendo o cualquier otro motivo que les importase no ser molestados. De allí surgió SB, el timbre inteligente.

Conforme se desarrollaba la idea del funcionamiento y los sistemas que podrían utilizar, también se pensó en aquellas personas con dificultades auditivas, que debido a su problema no podían atender a aquellos que tocaran el timbre de casa. Es por ello por lo que se decidió utilizar algún otro tipo de aviso que no fuera sonoro, como la notificación luminosa (que consistirá en el encendido de una bombilla LED).

El proyecto se basa en los conocimientos obtenidos a lo largo de la carrera, además de conocimientos adquiridos de forma autodidacta sobre algunas tecnologías utilizadas, como es el caso de Arduino.

Durante el desarrollo de la idea de SB, se buscaron dispositivos similares en el mercado, algunos de ellos son:

- ***Ring***[12] Es un timbre que cuenta con una cámara, micrófono, altavoz y detector de movimientos, desde el cual se puede interactuar con la persona que llama al timbre, también se puede abrir la puerta desde la aplicación.

Figura 1.1: Ring [12]

- ***Nucli Smart Lock***[13] Este timbre cuenta con los mismos elementos y funcionalidades que el anterior, además de un lector de huellas, para abrir la puerta.

Figura 1.2: Nucli Smart Lock [13]

Ambos dispositivos cuentan con servicio de notificaciones al dispositivo móvil, pero con algunas funciones adicionales como interactuar y ver quien llama a la puerta, o abrir la puerta, entre otros. Sin embargo, estos dispositivos son muy caros (por encima de los 200 €). La idea de este proyecto es hacer un timbre accesible económicamente y poder ser desarrollado por cualquier persona, además de añadir la opción de notificación para las personas con dificultades aditivas, pero las opciones adicionales con las que cuentan estos dispositivos comerciales son posibles opciones por desarrollar sobre este mismo proyecto, o también llamadas posibles mejoras.

1.2 Objetivos

1.2.1 Objetivo principal

Facilitar la gestión de un timbre para poder ser notificado y controlar dichas notificaciones desde un dispositivo móvil.

1.2.2 Objetivos específicos

Comunicar (mediante una red WiFi) el módulo Arduino y un dispositivo móvil con sistema operativo Android, para poder notificar y configurar las notificaciones del timbre.

Primero se tendrá que poder conectar el SmartBell (SB) a la red WiFi de casa.

Una vez conectado a la red WiFi, las notificaciones que se pueden activar/desactivar son:

- *Auditiva*: El timbre sonará, como un timbre cualquiera.
- *Luminosa*: Se encenderá una bombilla LED.
- *Inalámbrica*: El timbre enviará una notificación a los dispositivos de la red que tengan su aplicación instalada y configurada. A diferencia de las otras dos notificaciones, ésta se activa y desactiva de forma local en cada dispositivo móvil.

Por ejemplo, cuando alguien active o llame al timbre, éste notificará con las opciones activadas en ese momento. Es decir, si están activadas las notificaciones Auditiva y Luminosa, sonará el timbre y se encenderá la bombilla LED.

1.2.3 Tareas

Para poder completar el proyecto se ha dividido en una serie de tareas:

- **Estudiar y analizar el funcionamiento de los diferentes tipos de timbre del mercado**: Para poder tomar decisiones de diseño en el desarrollo del circuito a montar en Arduino.
- **Comparar los diferentes tipos de placas Arduino**: Para poder elegir y justificar motivo por el que se elige una placa *Arduino UNO r3* y no otra placa.
- **Estudiar el funcionamiento del módulo WiFly RN-171**: Para poder establecer la comunicación entre Arduino y la aplicación Android, mediante la conexión a la red WiFi. Se ha tenido que aprender y entender la configuración del módulo WiFi por línea de comandos, para la posterior integración con el código de Arduino mediante el uso de librerías.
- **Establecer una conexión entre Android y Arduino a través de un Socket**: Una vez que SB esté conectado a la red WiFi se tiene que poder conectar con la aplicación Android para el intercambio de información. Para ello se utilizará una conexión *Socket* entre el SB y la aplicación Android.
- **Desarrollo de una aplicación Android**: Para interpretar y configurar la información intercambiada entre el SB y Android, se ha desarrollado una aplicación en *Android Studio*, desde la cual se podrá conectar con el SB, recibir las alertas cuando alguien llama al timbre, tener un historial de todas las veces que llamaron al timbre y poder configurar las notificaciones.

CAPÍTULO 2

ESQUEMA GENERAL

Para poder entender de una manera rápida y sencilla la relación entre todos los dispositivos utilizados durante este proyecto, se presentan dos esquemas: uno representa el Hardware y el otro el Software utilizado.

2.1 Hardware

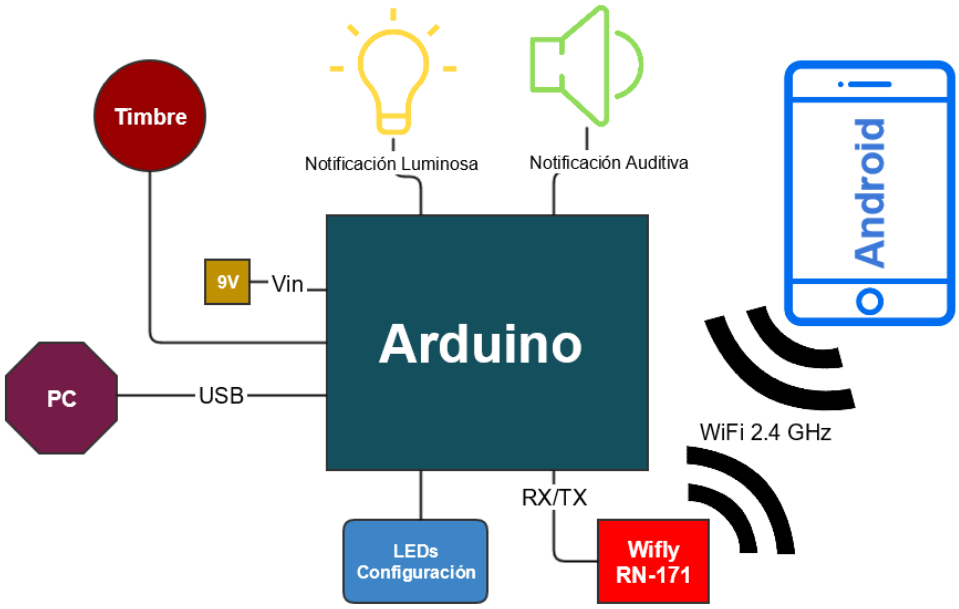


Figura 2.1: Esquema del Hardware

Como se puede observar en la **figura 2.1**, el SmartBell (SB) cuenta con:

- **Arduino** que gestiona todas las funcionalidades.
- **Conexión USB** para conectarse con un PC. Desde allí se podrá configurar la conexión a la Red WiFi y las notificaciones del SB.
- **Jack de alimentación** para alimentar el SB con una fuente de 9V.
- **Pines de conexión para el timbre** donde se conecta el pulsador del timbre y así saber si alguien llama al timbre. De este modo, SB activa la bombilla LED, el audio, o la notificación inalámbrica, según como esté configurado.
- **WiFly RN-171** para la interacción entre Arduino y Android. Para ello se utiliza una comunicación *Universal Asynchronous Receiver-Transmitter (UART)*, por un *Puerto Serial* virtual mediante los pines 8 y 9, Receptor de datos (RX) y Transmisor de datos (TX) respectivamente.
- **Bombilla** para la notificación luminosa.
- **Altavoz** para la notificación auditiva.
- **LEDs Configuración** indican qué tipo de notificación se encuentran activadas.

2.2 Software

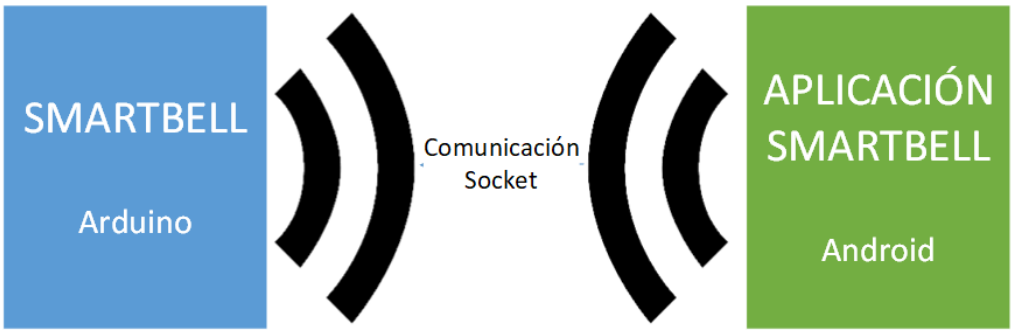


Figura 2.2: Esquema del Software

Como se puede ver en la **figura 2.2**, el *SmartBell (SB)* (desarrollado en *Arduino*) y la *Aplicación SmartBell* (desarrollado en *Android Studio*) se comunican con una comunicación *Socket*, mediante la cual se intercambian información como estados y configuración de las notificaciones.

- **SB**, gestiona las notificaciones, es decir, las activa/desactiva y las guarda en memoria. También notifica cuando alguien toca el timbre.
- **Aplicación SmartBell**, se conecta con el *SB*, para poder modificar las notificaciones y ser notificado cuando alguien llama a la puerta y almacenarlo en el historial.

CAPÍTULO 3

HADWARE SMARTBELL

Para este proyecto se utilizaron un conjunto de dispositivos y tecnologías, ya mencionadas. Ahora se pretende justificar el uso de cada una de éstas y no otras. Para ello, se explicará cada una de las distintas posibilidades a utilizar y se justificará él motivo por el que se utilizaron estas tecnologías o dispositivos.

3.1 Placas con microcontrolador

Para la creación del circuito y el control de las notificaciones del SmartBell (SB) se necesita una placa con un microcontrolador, el cual permite una mayor integración con el Hardware extra que se le agregue y un buen soporte en el desarrollo del código. Existen dos grandes opciones:

- *RASPBERRY PI*: *Es un computador de Single Board Computer (SBC) de bajo coste. El diseño incluye un System-on-a-chip Broadcom BCM2835, que contiene una Central Processing Unit (CPU) ARM1176JZF-S a 700 MHz (el firmware incluye unos modos Turbo para que el usuario pueda hacerle overclock de hasta 1 GHz sin perder la garantía), una Graphics Processing Unit (GPU) VideoCore IV, y 512 MB de memoria RAM*[14]. Con un coste medio de 30 €.

- *ARDUINO*: entre la gran diversidad de modelos, destacan los siguientes:

  1. *MEGA*, es una *placa de microcontrolador basado en ATmega1280. Cuenta con 54 pines de entrada/salida (14 de los cuales se pueden usar como salida Pulse-Width Modulation (PWM)) 16 entradas analógicas, 4UARTs (puertos serie) y un oscilador de cristal 16Mhz*[15]. Con un coste de 35 €.
  2. *LEONARDO*, es una *placa de microcontrolador basado en ATmega32u4. Cuenta con 20 pines de entrada/salida (7 de los cuales se pueden usar como salida PWM y 12 entradas analógicas) y un oscilador de cristal 16Mhz*[16]. Con un coste de 22 €.

3. *UNO rv3*, es una *placa de microcontrolador basado en ATmega328P. Cuenta con 14 pines de entrada/salida (6 de los cuales se pueden usar como salida PWM) 6 entradas analógicas y un oscilador de cristal 16Mhz*[1]. Con un coste de 20 €.

   El microcontrolador no está soldado a la placa, lo cual permite un cambio rápido y económico, si se estropea el microcontrolador

De todas estas posibilidades, se escogió la opción de **Arduino UNO rv3**, porque no hace falta más recursos en cuanto a Hardware, además que tiene un gran soporte en Internet (librerías), y es más económico.

3.1.1 Arduino Uno rv3

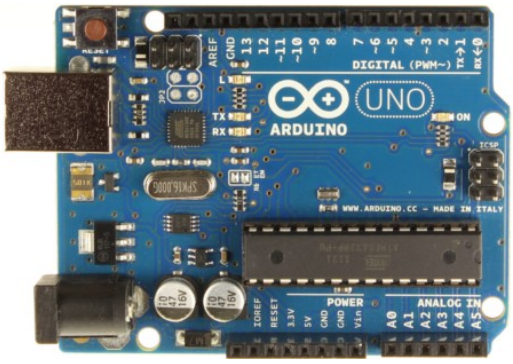


Figura 3.1: Arduino Uno rv3 [1]

En la **figura 3.1** se puede observar la placa *Arduino Uno rv3*. A continuación se explicará con un esquema las parte a destacar del Arduino UNO rv3:

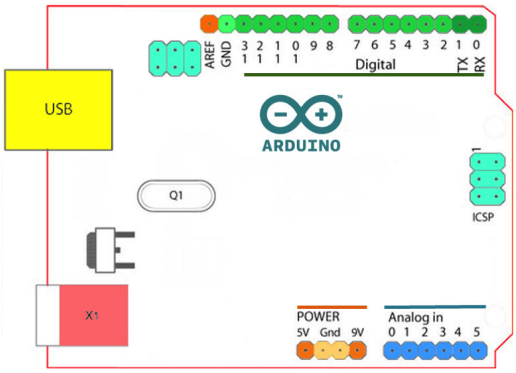


Figura 3.2: Esquema del Arduino [2]

- **Pin analógico de Referencia (pin naranja)** No utilizados para el proyecto.
- **Tierra digital (verde claro)** No utilizados para el proyecto.

- **Pines digitales 2-13 (verde)** Utilizados para las notificaciones, LEDs informativos, entrada de control del timbre y el puerto serie virtual (*Serial Virtual*).

- **Pines digitales 0-1 para comunicación serie In/Out - TX/RX (verde oscuro)** Utilizados para la comunicación serie (*Serial*) y así poder interactuar con SB mediante un cable USB.

- **Conector de programación ICSP (In-circuit Serial Programmer) (turquesa)** No utilizados para el proyecto.

- **Pines analógicos 0-5 (celeste)** No utilizados para el proyecto.

- **Pines de Alimentación y Tierra (Alimentación (POWER) o *Voltaje de entrada (Vin)*: naranja, Tierra o Ground (GND): naranja claro)** Utilizados para alimentar SB, *Relé KY-019* y *Hex BufferMC14050B*.

- **Conector jack 21mm para alimentación externa (9-12V *DC*) - X1 (rosa)** No utilizados para el proyecto.

- **Conector USB (para comunicar el Arduino con el ordenador) (amarillo)** Utilizado para configurar las notificaciones y la conexión WiFi.

3.1.1.1 Alimentación

Arduino UNO puede ser alimentado a través de la conexión USB, por una fuente de alimentación externa mediante el jack de carga o a través de los pines *Vin* y *GND*. El origen de la alimentación es seleccionado automáticamente por la placa.

La placa puede trabajar con una alimentación externa de 6 a 20V. Si se utiliza una alimentación inferior a 7V, la alimentación digital de 5V puede ser inestable. Por otro lado, si se utiliza más de 12V, el regulador de voltaje puede sobrecalentarse y dañar la placa. Por lo tanto, la alimentación recomendada debe estar entre 7 y 12V.

Como se ha comentado, el SB tiene una notificación luminosa mediante una bombilla LED, la cual funciona con una alimentación de 9 a 14V. Por ello se decidió utilizar una alimentación de 9V, el cual alimenta a la bombilla y luego al Arduino, mediante los pines *Vin* y *GND*.

Se tomó esta decisión porque, si se alimentaba mediante el jack de carga del propio Arduino y después se alimentaba a la bombilla por los pines *Vin* y *GND*, el voltaje de salida de esos pines era de 7V aproximadamente, lo cual no era suficiente para encender a la bombilla.

Como el hecho de tener bien polarizados la entrada de los pines *Vin* y *GND* es crítico (ya que una mala polarización estropearía la placa Arduino), se decidió utilizar un ***puente de diodos*** entre la alimentación externa y los pines *Vin* y *GND*, garantizando así una polarización correcta en los pines de *Vin* y *GND* de Arduino.

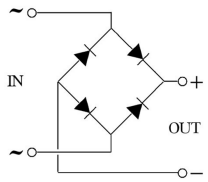


Figura 3.3: Puente de diodos [3]

Se puede observar el puente de diodos en la **figura 3.3**. Si bien se utiliza normalmente para convertir corriente alterna (Alternating Current (AC)) en corriente continua (Direct current (DC)), en este caso se utilizará para que, sea cual sea la posición de los polos de entrada, siempre se tenga polarizada de una misma forma los polos de salida y así conectarlos a los pines *Vin* y *GND* de Arduino de forma adecuada.

3.1.1.2 Memoria

El *Arduino Uno rv3* cuenta con el ATmega328, que tiene 32 KB (con 0,5 KB utilizados para el cargador de arranque). También tiene 2 KB de SRAM y 1 KB de memoria **EEPROM** (que puede leer y escribir con la librería ***EEPROM.h***).

En la memoria EEPROM se almacenará la información que no se desea perder al reiniciar el SB. Estos pueden ser los datos de conexión a la red WiFi: Service Set Identifier (SSID), contraseña e Internet Protocol (IP). Cada vez que el SB tenga una nueva configuración (activar/desactivar la notificación auditiva o luminosa), esta configuración se debe guardar también.

1 KB de memoria EEPROM																	
Reservados para	eepromTimbre			eepromWifi						eepromSsid				eepromClave			
Nº de Byte	0	1	...	30	31	32	33	34	...	50	51	...	99	100	101	...	149
Tipo de dato	boolean	boolean		boolean	int	int	int	int		char	char		char	char	char		char
Utilizados para	Notificación Auditiva	Notificación Luminosa	...	Dynamic Host Configuration Protocol (DHCP)	1º octeto de IP	2º octeto de IP	3º octeto de IP	4º octeto de IP	...	1º letra de SSID	2º letra de SSID	...	50º letra de SSID	1º letra de la contraseña	2º letra de la contraseña	...	50º letra de la contraseña

Cuadro 3.1: Uso de memoria EEPROM

En la **tabla 3.1** se puede observar la organización del uso de la memoria EEPROM.

- Los Bytes 0, 1 y 30 son booleanos (boolean), es decir solo pueden ser verdadero o falso (activado o desactivado).

- Los Bytes 31, 32, 33 y 34 son enteros (int), donde se guardan los 4 octetos de una dirección IPv4.

- Del Byte 50 al 149 son caracteres (char), donde se guardan los caracteres de la SSID y la contraseña de la red WiFi.

Como ya se comentó, para leer y escribir en la memoria EEPROM, se utilizará la librería *EEPROM.h*, como se puede ver en el **apéndice A.1.1**.

3.1.1.3 Programación del circuito SB

La plataforma Arduino utiliza su propio IDE, que se puede descargar de manera gratuita en su página web (https://www.arduino.cc/).

El entorno de desarrollo integrado Arduino - Arduino o software (IDE) - contiene un editor de texto para escribir código, un área de mensajes, una consola de texto, una barra de herramientas con botones para las funciones comunes y una serie de menús. Se conecta al hardware Arduino y Genuino para cargar programas y comunicarse con ellos. [20] Está basado en el lenguaje de C, usando las operaciones: aritméticas, de comparación (==, !=, <, >, <=, >=), operadores booleanos, etc. También tiene las estructuras de control, como: condicionales (if, else, switch case), bucles (for, while, do while) y saltos (break, continue, return, goto).

Los tipos de datos con los que trabaja son: void, boolean, char, unisgned char, byte, int, unsigned int, word, long, unsigned long, float, double, string, array.

Su estructura de programación es mediante funciones, teniendo dos Funciones esenciales que son:

- **setup()** Esta función inicia la carga del programa en la placa. Donde se inicializan las variables, indican el modo de los pines, inicializan las librerías, etc. Esta función se ejecutará una vez, y solo después de cada encendido o reinicio del SB.

- **loop()** Esta función se ejecuta después de la función *setup()*. Ejecuta un bucle infinito, permitiendo al programa cambiar y responder según el estado de sus entradas.

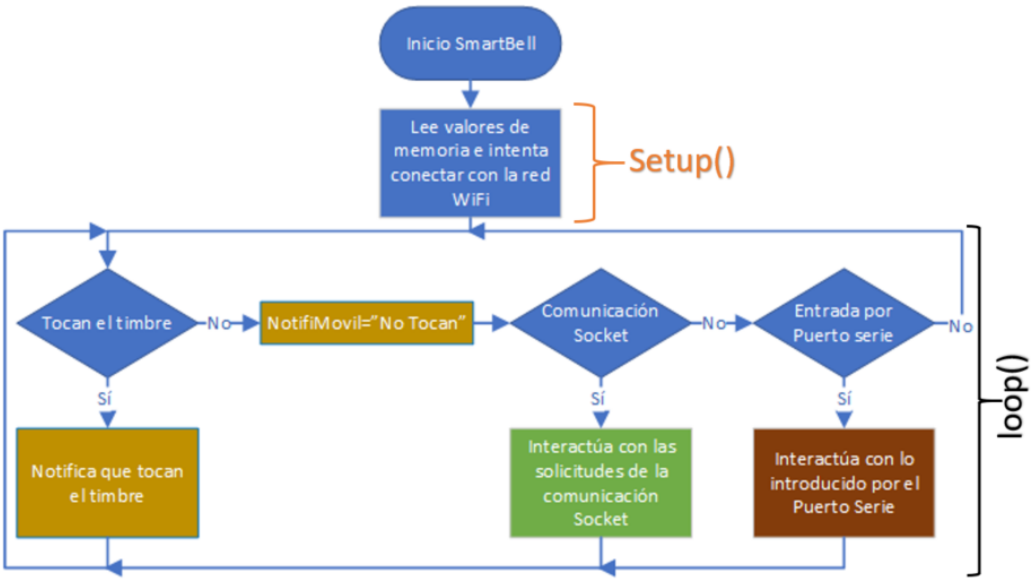


Figura 3.4: Diagrama de flujo del funcionamiento de SB

En la **figura 3.4** se puede observar que primero lee los valores de memoria e intentar conectarse con la red WiFi (**setup()**). Finalmente hace tres comprobaciones (si tocan el timbre, si existe una comunicación Socket y si existe alguna entrada por el Puerto Serie) de forma infinita (**loop()**).

3.1.1.4 Monitor Serie

Arduino IDE proporciona una herramienta que permite enviar y visualizar los datos que se manejan a través del puerto Serie. Dicha herramienta se conoce como *Monitor Serie* y se puede encontrar en el menú de *Herramientas/Monitor Serie*. Es la forma más simple que existe para establecer la comunicación serial con Arduino.

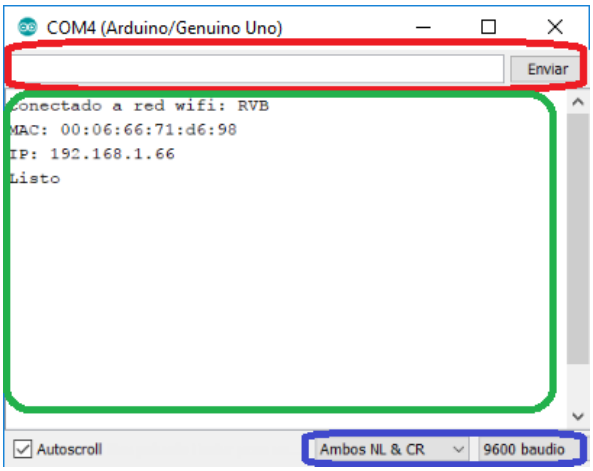


Figura 3.5: Monitor Serie de Arduino IDE

En la **figura 3.5** se puede observar 3 partes diferenciadas de la ventana del *Monitor Serie*:

- **Enviar mensaje** Es el recuadro rojo. Donde se pone el texto que se desea enviar al SB, para que este lo procese y realice la acción programada.
- **Recibir mensaje** Es el recuadro verde. Donde se puede observar los mensajes enviados por el SB. En este caso muestra la configuración de la conexión con la red WiFi.
- **Configuración puerto serie** Es el recuadro azul. Donde se configura la comunicación con el puerto serie. En el primer desplegable se elige cómo se mostrarán los mensajes, que pueden ser: NL(NewLine, nueva linea) y/o CR (CarriageReturn, retorno de carro). En el segundo desplegable se elige la velocidad (en baudios) para la transmisión de datos en el puerto serie, esta velocidad viene determinada en la función **setup()**, con el comando *Serial.begin(velocidad)*.

Cada vez que se abre el *Monitor Serie*, SB se reinicia, ejecutándose primero la función **setup()** y finalmente en bucle la función **loop()**.

Con SB, al abrir el *Monitor Serie*, primero intenta conectar a la red WiFi (con los datos de la configuración WiFi guardados en la memoria EEPROM) y muestra por pantalla (*Monitor Serie*) el estado de la conexión WiFi, como se pudo observar en la **figura 3.5**

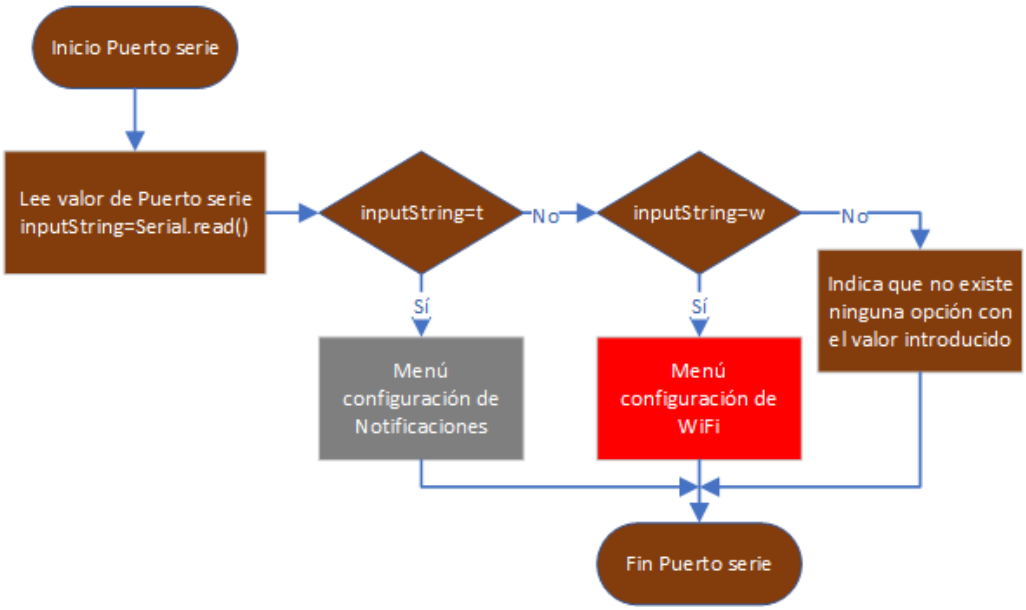


Figura 3.6: Diagrama de flujo de la comunicación con el *Monitor Serie*

Se puede observar en la **figura 3.6**, que dependiendo del valor ingresado por el *Monitor Serie* se ingresara a la configuración de las notificaciones o de la conexión WiFi.

A partir de este punto se pueden acceder a dos menús:

1. **Menú de Configuración del WiFi**

   Para acceder a este menú, se debe enviar la letra **w** por el *Monitor Serie*. En este menú se podrá: conectar con una red WiFi y configurar el servicio *DHCP*.

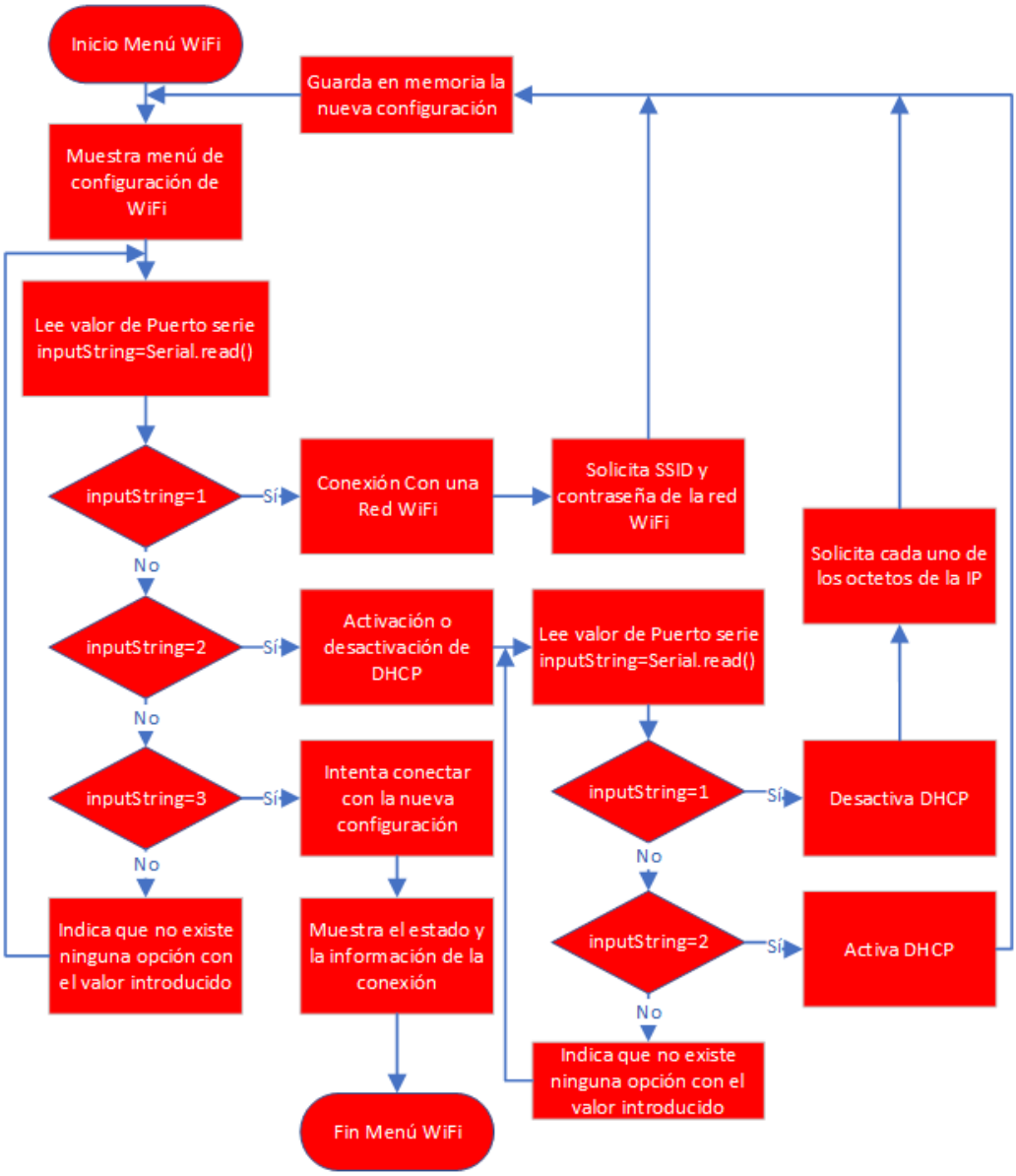


Figura 3.7: Diagrama de flujo del menú de conexión WiFi

En la **figura 3.7** se puede observar el funcionamiento de SB para la configuración de la conexión WiFi, que dependiendo de lo ingresado va configurando diferentes parámetros, como la SSID y la contraseña de la red o el estado del servicio DHCP.

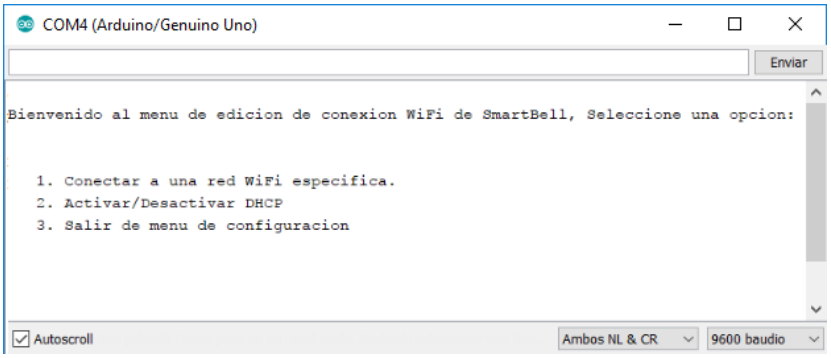


Figura 3.8: Menú de configuración de la conexión WiFi

Se puede observar el menú de la configuración del WiFi en la **figura 3.8**. Para seleccionar una opción se debe enviar el número de lo que se quiera configurar.

- **Conectar con una red WiFi**

  Para conectar con una red WiFi, se debe enviar **1**, por el *Monitor Serie*.

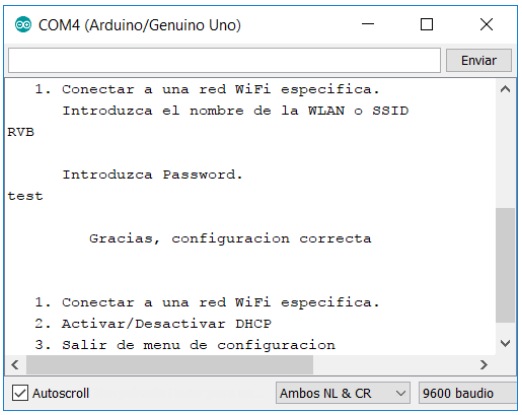


Figura 3.9: Conexión con la red WiFi

Se puede observar en la **figura 3.9**, que después de seccionar la opción **1. Conectar a una red WiFi específica**, el SB solicita la SSID y la contraseña. Una vez introducidos estos datos, confirma por pantalla que se configuró correctamente y se vuelve al menú principal de la *Configuración del WiFi*

- **DHCP**

  Para configurar el servicio *DHCP*, se debe enviar **2**, por el *Monitor Serie*.

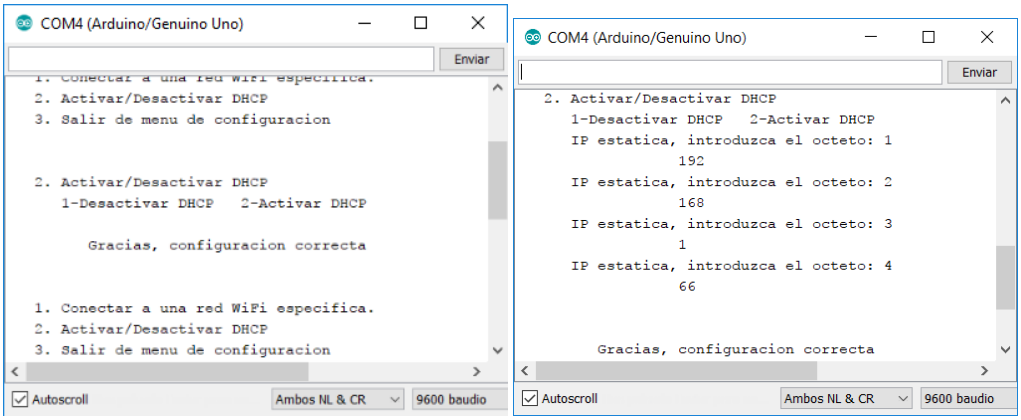


Figura 3.10: Configuración de *DHCP*

En la **figura 3.10**, se puede observar en la imagen de la izquierda, la activación del servicio de *DHCP*, enviando la *opción 2* (confirmando por pantalla que se configuró correctamente y volviendo al menú principal de la *Configuración del WiFi*).

Por otra parte en la imagen de la derecha se muestra los mensajes a intercambiar si se desea desactivar el servicio *DHCP*, donde después de enviar la *opción 1* de desactivar el servicio *DHCP*, SB solicita uno a uno los 4 octetos de la dirección **IPv4** que se quiera establecer al sistema. Cuando se introducen los cuatro octetos correctamente, confirma por pantalla que se configuró correctamente y se vuelve al menú principal de la *Configuración del WiFi*

- **Implementación de la configuración**

  Para guardar la configuración en la memoria EEPROM e implementar la nueva *Configuración del WiFi*, se debe enviar el comando **3**, por el *Monitor Serie*.

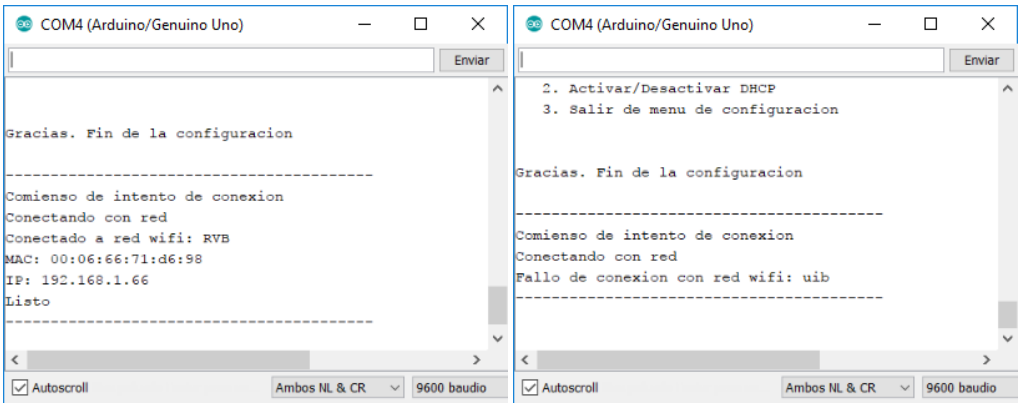


Figura 3.11: Implementación de la configuración WiFi

Después de salir de la configuración del WiFi, se Muestra por pantalla el intento de conexión, como se puede observar en la **figura 3.11**. En la imagen

de la derecha, se puede ver el caso de asociación correcta con la red WiFi, además de los datos de la conexión. En la imagen de la izquierda, un intento de conexión fallido. Esto puede deberse a que la SSID y/o contraseña no son los correctos.

2. **Menú de Configuración de las notificaciones**

   Para acceder a este menú, se debe enviar la letra **t** por el *Monitor Serie*. En este menú se podrá activar o desactivar cualquiera de los 2 tipos de notificaciones centralizadas (es decir, las que se guardan en la memoria del SB), que son la auditiva y la luminosa. La configuración de la notificación inalámbrica se gestiona de forma local en cada dispositivo móvil

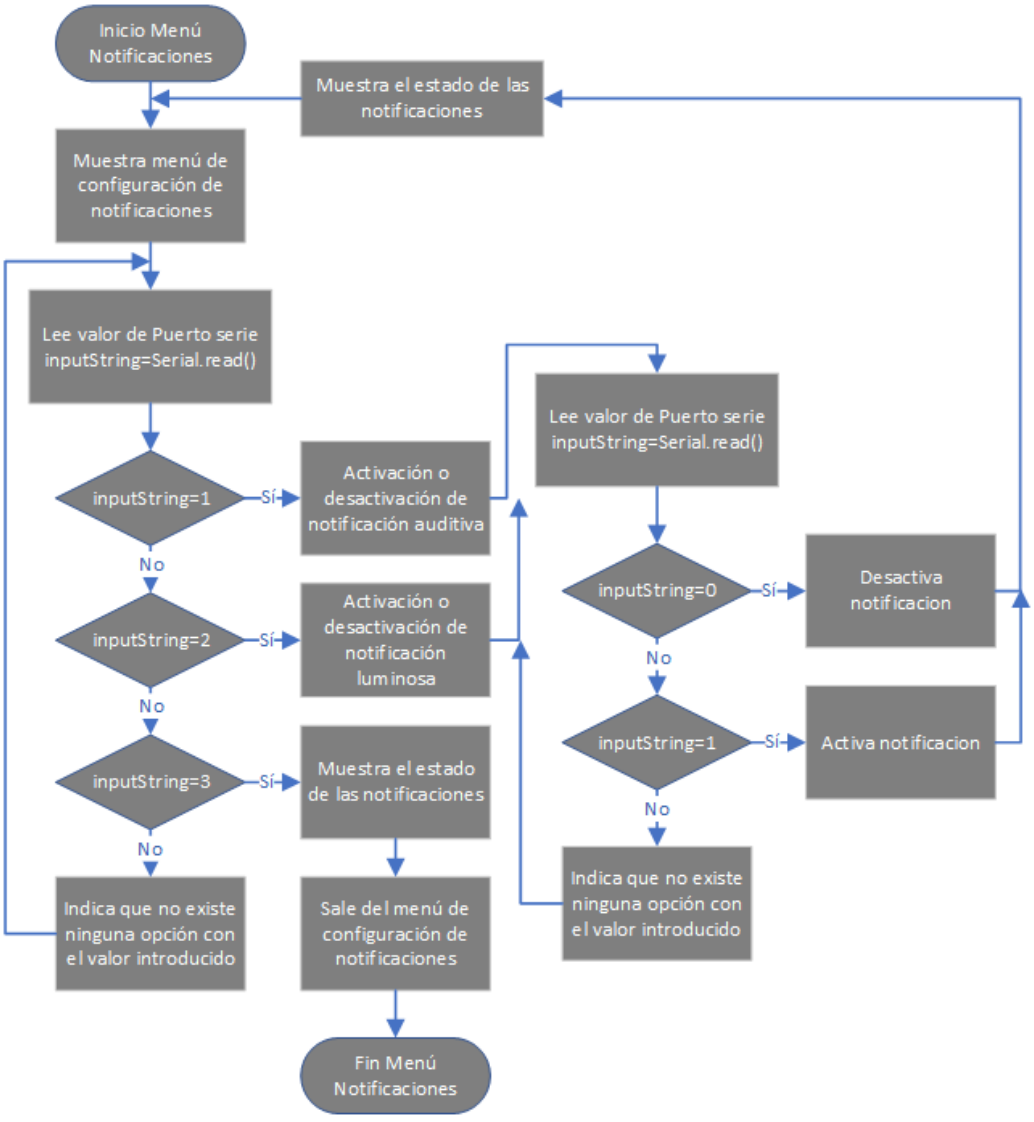


Figura 3.12: Diagrama de flujo del menú de notificaciones

En la **figura 3.12** se puede observar el funcionamiento de SB para la configuración de las notificaciones, que dependiendo de lo ingresado se activa o desactiva la notificación seleccionada.

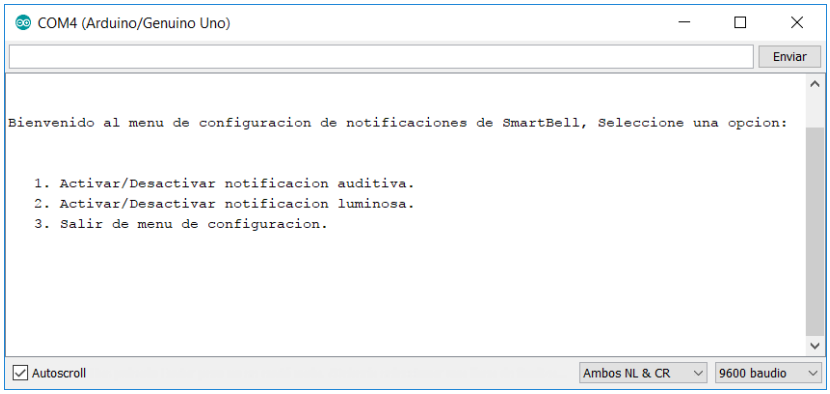


Figura 3.13: Menú de configuración de las notificaciones

Se puede observar el menú de la configuración de las notificaciones en la **figura 3.13**. Para seleccionar una opción se debe enviar el número de la notificación que se quiera configurar. Por ejemplo, para modificar la notificación luminosa, se debe enviar **2** por el *Monitor Serie*.

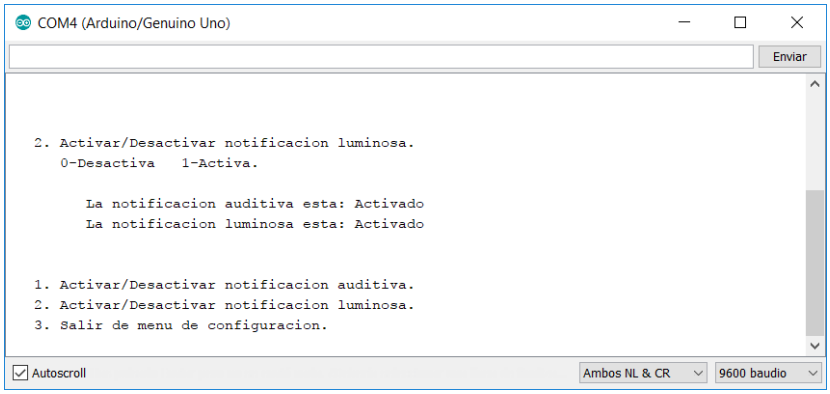


Figura 3.14: Configuración de notificación luminosa

En la **figura 3.14** se puede observar que salen 2 opciones: activar (opción 1) y desactivar (opción 0). Después de seleccionar una opción (en el ejemplo la opción 1), muestra el estado de todas las notificaciones. Finalmente vuelve al menú principal de *Configuración de las notificaciones*.

Para salir de esta configuración desde el menú principal de *Configuración de las notificaciones*, se elige la opción 3, saliendo así de este menú y mostrando el mensaje: **Gracias. Fin de la configuración** por pantalla.

3.1.1.5 Conexión con el timbre de casa

No todos los timbres son iguales. Pese a ello, podemos diferenciarlos en: aquellos que usan *Corriente Continua DC* (normalmente son los que se usan en comunidad, o los que tiene un portero automático) y los que usan *Corriente Alterna AC* (los que solamente cuentan con un pulsador y el timbre).

Los timbres que usan corriente alterna funcionan con 110 o 220 voltios según la región donde se utiliza el SB. Pero como Arduino solo trabaja con corriente continua. Para ello se utilizará el transformador ***HLK-PM01***, que transforma corriente alterna de *100-240V AC* a continua de *5V DC*.

Figura 3.15: Transformador HLK-PM01 [7]

Con el uso del transformador, se tiene una entrada de 5V en continua. Y si la entrada es de un timbre comunitario o con portero automático, ambas están en continua, pero ésta puede oscilar de los 5V a 24V.

Figura 3.16: Hex Buffer MC14050B

Como Arduino soporta una entrada de hasta 5V, se utiliza el *Hex Buffer MC14050B*, el cual independientemente del voltaje de entrada lo adapta al voltaje de alimentación del Buffer.

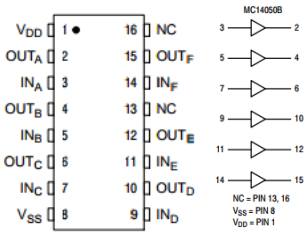


Figura 3.17: Pines de MC14050B [8]

En la **figura 3.17** se puede ver que el Buffer cuenta con 16 pines, de los cuales el 13 y 16 no se usan. Además, destacan los pines 1 y 8 (*VDD* y *VSS*, respectivamente), que son la alimentación del Buffer. En *VDD* se pondrá 5V y en *VSS* *GND*. También se puede ver que cada pin de entrada con su respectivo pin de salida es una puerta lógica *YES*.

Voltaje de Entrada	Símbolo	VDD (V_{DC})	Min	Max	Unidad
Nivel 0	V_{IL}	5	-	1.5	V_{DC}
		10	-	3	
		15	-	4	
Nivel 1	V_{IH}	5	3.5	-	V_{DC}
		10	7	-	
		15	11	-	

Cuadro 3.2: Funcionamiento de MC14050B [8]

Se puede ver en la **tabla 3.2** que dependiendo del voltaje de *VDD*, se considera de forma diferente los voltajes de entrada.

Por ejemplo, si *VDD* vale 5V, se considerará una entrada de *nivel 0*, si el voltaje de entrada vale como mucho 1.5V, por lo cual la salida será de *nivel 0* (es decir, 0V). Se considera una entrada de *nivel 1*, si la entrada tiene un voltaje mínimo de 3.5V, por lo cual la salida tendrá un *nivel 1* (es decir, 5V).

De esta forma se asegura que independientemente del tipo de timbre al que se conecte, cuando se llame al timbre al SB siempre le llegaran 5V.

3.1.1.6 Notificaciones

SB tiene 3 LEDs, dos para cada tipo de notificación que se configuran de forma centralizada (auditiva y luminosa), los cuales indican que tipo o tipos de notificaciones están activadas, mientras que ultimo LED se enciende cuando alguien llama a la puerta.

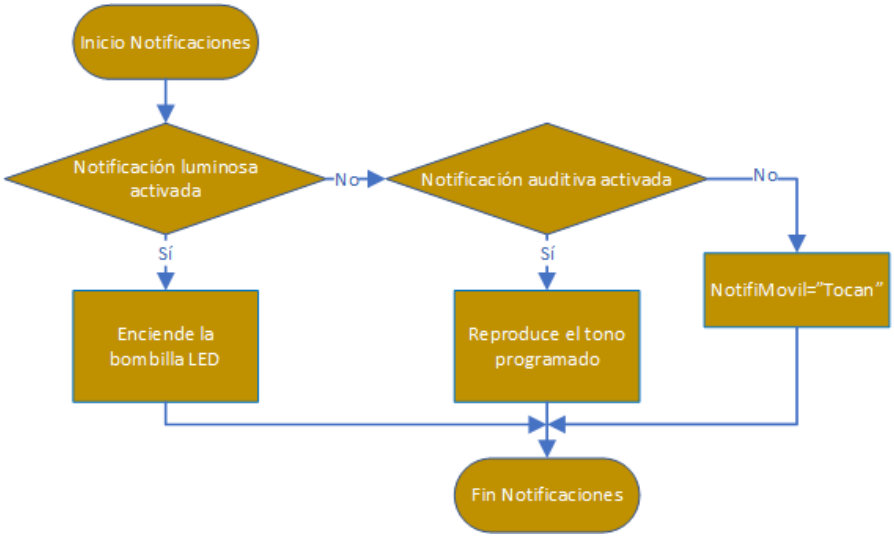


Figura 3.18: Diagrama de flujo de las notificaciones

En la **figura 3.18** se puede observar el comportamiento de SB para las notificaciones. Dependiendo del estado de la notificación enciende la bombilla o reproduce el tono del timbre y cambia el estado de *NotifMovil* a **Tocan** para enviarlo por el módulo WiFly.

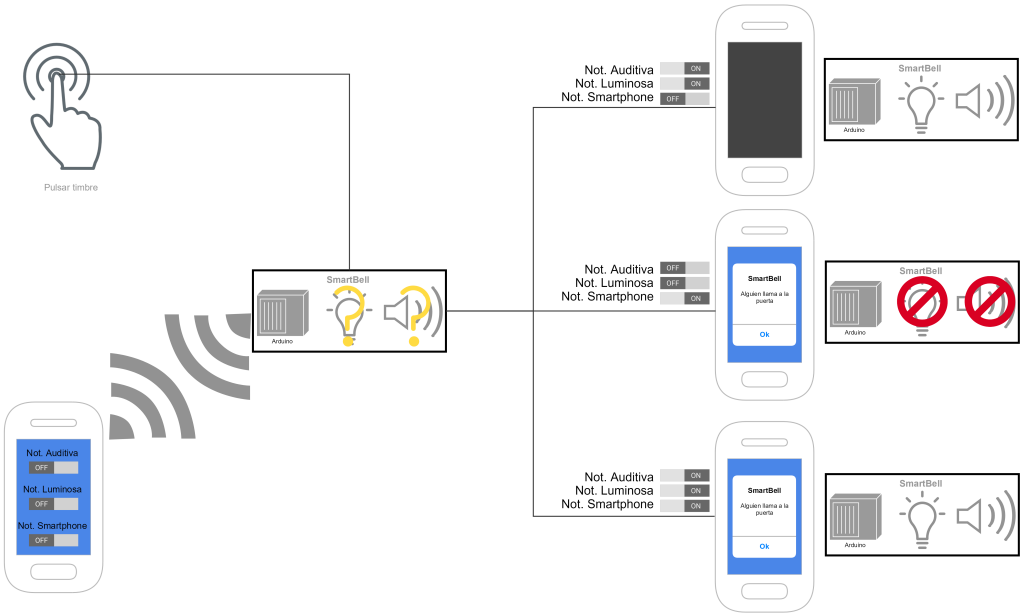


Figura 3.19: Esquema del funcionamiento de las notificaciones

En la **figura 3.19**, se puede ver un esquema del funcionamiento de las notificaciones. Cada una de las notificaciones centralizadas tiene un tiempo de ejecución, ya sea reproduciendo el tono auditivo o encendiendo por unos segundos la bombilla LED. Debido a que la programación en Arduino es secuencial, las solicitudes desde los dispositivos móviles sobre el estado del timbre se rechazarán. Para solventar este problema, se decidió hacer una comprobación de las solicitudes al módulo WiFly de forma más seguida. Esto implica que dentro de cada notificación se comprueba las solicitudes WiFi tantas veces como sean necesarias.

Para los distintos tipos de notificaciones, se utilizarán diferentes dispositivos. Por ello, se explicará el funcionamiento de cada uno de ellos:

1. **Auditiva**

   Para esta notificación se utiliza un buzzer o altavoz, el *AST-030c0mr-r*, el cual funciona con 5V. Este se conectará al pin 6 de Arduino.

Figura 3.20: Buzzer *AST-030c0mr-r* [9]

Para que el altavoz no tenga un simple sonido, se utiliza parte de un código que simula una melodía (**apéndice A.1.2**)

2. **Luminosa**

   Para esta notificación se utiliza una bombilla LED, la cual está conectada a la alimentación de SB de 9V, y esté se activa mediante el *Relé KY-019*.

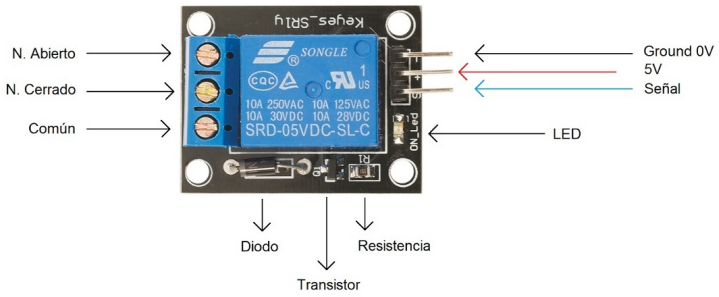


Figura 3.21: *Relé KY-019* [10]

En la **figura 3.21**, se puede ver el relé utilizado en el proyecto. Los pines ***GND*** y ***5V*** se conectan al Arduino en sus respectivos pines *GND* y *5V*. El pin ***Señal***, se conecta al pin 7 de Arduino, que es el que activa o desactiva el relé.

Se conectará uno de los polos de alimentación al ***Común*** del relé y ***N. Abierto*** a la bombilla LED. Esto implica que el circuito se encontrará abierto hasta que se active el pin ***Señal*** del relé con una señal de *nivel alto* (5V).

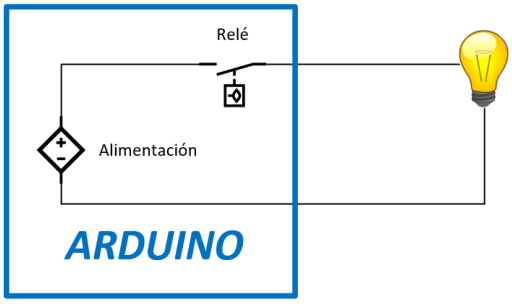


Figura 3.22: Conexión de bombilla LED con Arduino

En la **figura 3.22** se puede observar el esquema de conexión de la bombilla con Arduino mediante el relé. Este *relé* funciona solo si esta activada la *notificación luminosa*, de modo que se accionará cuando toquen al timbre.

3. **Inalámbrica**

   Esta notificación responde a las solicitudes de la aplicación Android: si llaman al timbre envía **Tocan**, en caso contrario envía **No tocan**.

Se envía una de estas dos respuestas porque este tipo de notificación se realiza mediante solicitudes de la aplicación Android, dichas solicitudes se hacen de forma periódica.

Se decidió que se notifique de esta forma debido a que el lenguaje de Arduino, (que es ***C***) no permite concurrencia. De esta forma no se mantiene la aplicación bloqueada en una acción.

También se instaló un LED para esta notificación. Éste se utiliza de forma informativa. Por ejemplo, si todas las notificaciones centralizadas están desactivadas, este LED se encenderá cuando toquen el timbre y se envié la notificación inalámbrica.

3.2 Comunicación sin cables

Es necesario una comunicación entre el SB y la aplicación a desarrollar, para el intercambio de información, así como las notificaciones y sus configuraciones. Existen una gran variedad, como:

- *NFC (Near field communication)*: *es una tecnología de comunicación inalámbrica, de corto alcance y alta frecuencia que permite el intercambio de datos entre dispositivos. Funciona en la banda de los 13.56 MHz, su tasa de transferencia puede alcanzar los 424 kbps y un alcance máximo de 20 centímetros*[17].

- *BLUETOOTH*: *es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2.4 GHz. Con un alcance de 10 metros, sin ningún obstáculo por en medio y permite tasas de transferencias desde 25 Mbps hasta 32 Mbps*[18]. No permite una escalabilidad de la señal.

- *WIFI (802.11g)*: es un *mecanismo de conexión de dispositivos electrónicos de forma inalámbrica, que utiliza la banda de 2,4 GHz, su tasa de transferencia puede alcanzar los 54 Mbps y un alcance de 100 metros*[19]. permite una gran escalabilidad del alcance de la señal, mediante el uso de repetidores.

De las opciones ya citadas, se escogió la de **WIFI (802.11g)**, porque es la que más se utiliza, tiene mayor alcance, mayor tasa de transmisión y permite una mayor escalabilidad.

3.2.1 Módulo WiFly RN-171

Como ya se explicó anteriormente, se utilizará una comunicación WiFi para el intercambio de información entre la aplicación Android y el SB. Como la placa Arduino UNO no tiene integrada ninguna antena WiFi, se decidió utilizar el módulo WiFly RN-171.

Figura 3.23: WiFly RN-171 [4]

La serie RN-171-XV de Microchip son módulos wifi con certificación IEEE 802.11b/g que incorporan un procesador SPARC de 32 bits, pila Transport Control Protocol (TCP)/IP, reloj de tiempo real, criptoacelerador, unidad de administración de potencia y una interfaz de sensor analógico [4].

El módulo cuenta con tres LEDs, a través de los cuales informa de su estado. Estos LEDs son verde, amarillo y rojo. Sus estados pueden ser: apagado, encendido constante, parpadeo rápido y parpadeo lento. En la siguiente tabla se muestra el significado de cada uno de los estados:

Estado	LED Rojo	LED Amarillo	LED Verde
Encendido constante	-	-	Conectado por TCP
Parpadeo rápido	No asociado	RX/TX transferencia de datos	-
Parpadeo lento	Asociado y sin Internet	-	Dirección IP correcta
Apagado	Asociado y con Internet	-	-

Cuadro 3.3: LEDs informativos módulo WiFly [11]

3.2.1.1 Integración con Arduino

Para una integración correcta con Arduino, es necesario un *ARDUINO WIRELESS SD SHIELD*.

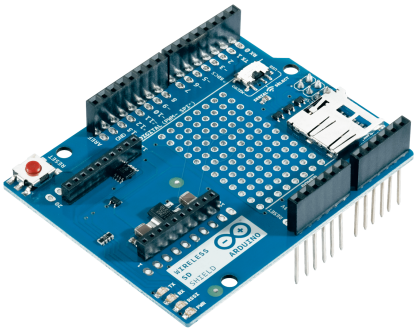


Figura 3.24: Arduino Wireless SD Shield [5]

Esta Shield permite la conexión y comunicación entre el Arduino y el módulo WiFly.

La configuración del módulo WiFly RN-171 se hace mediante comandos en lenguaje ASCII, usando las conexiones UART RX y TX, que se asocian a los pines 1 y 2 de Arduino respectivamente. Dichos pines ya se utilizan para la configuración del mismo Arduino, por lo cual deben estar libres de la placa de Arduino.

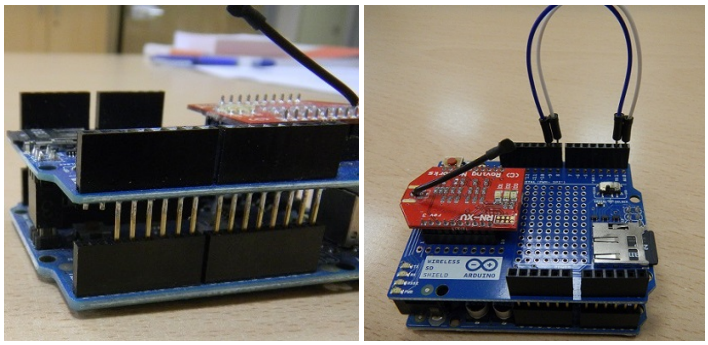


Figura 3.25: Modificación de Pines RX y TX de Shield [6]

Como se puede observar en la **figura 3.25**, se cortarán los pines 1 y 2 de la **Arduino Wireless SD Shield** y se hará un puente entre los pines 8 y 9 con los pines 1 y 2 (respectivamente), pudiendo usar así un puerto serie virtual en los pines 8 y 9, desde los cuales se configurara el módulo WiFly.

Como ya se comentó la configuración del módulo se hace mediante comandos ASCII. Pero para una configuración más eficiente, se utilizará la librería ***WiFlyHQ.h***.

En las **figuras 3.4 y 3.7** se pudo observar que uno de los procesos de SB es intenta conectar con la red WiFi. La forma de establecer una conexión con la red WiFi se describe en el **apéndice A.1.3**

3.2.1.2 Intercambio de información con la aplicación Android

SB intercambiará información con la aplicación Android. Esta información estará formada por las respuestas a las solicitudes de la aplicación Android. Dichas solicitudes pueden ser:

- *Estado de configuración de SB:* Esta solicitud hace referencia al estado de los distintos tipos de notificaciones, si están activadas o desactivadas. El SB enviará el Byte **estadoSB**, anteriormente mencionado. Con ese único Byte, se informa de la configuración de todas las notificaciones.

- *Activar o desactivar notificación:* Desde la aplicación Android se puede activar o desactivar las notificaciones del SB. Sin embargo, cualquier cambio en la configuración de las notificaciones se hace de forma individual. Por ejemplo, si se quiere activar la notificación auditiva se envía una solicitud pero si se quiere desactivar la misma notificación se envía otro tipo de solicitud, teniendo así 4 tipos de solicitudes para la configuración de las notificaciones auditiva y luminosa, 2 por cada tipo de notificación(activar y desactivar). La configuración de la notificación inalámbrica es propia de cada dispositivo móvil, es decir, un dispositivo puede tener la notificación inalámbrica activada y otro desactivada.

- *Estado del SB:* Esta solicitud es básicamente para saber si alguien toca el timbre o no. SB responde *Tocan* o *No tocan*, según su estad en ese momento.

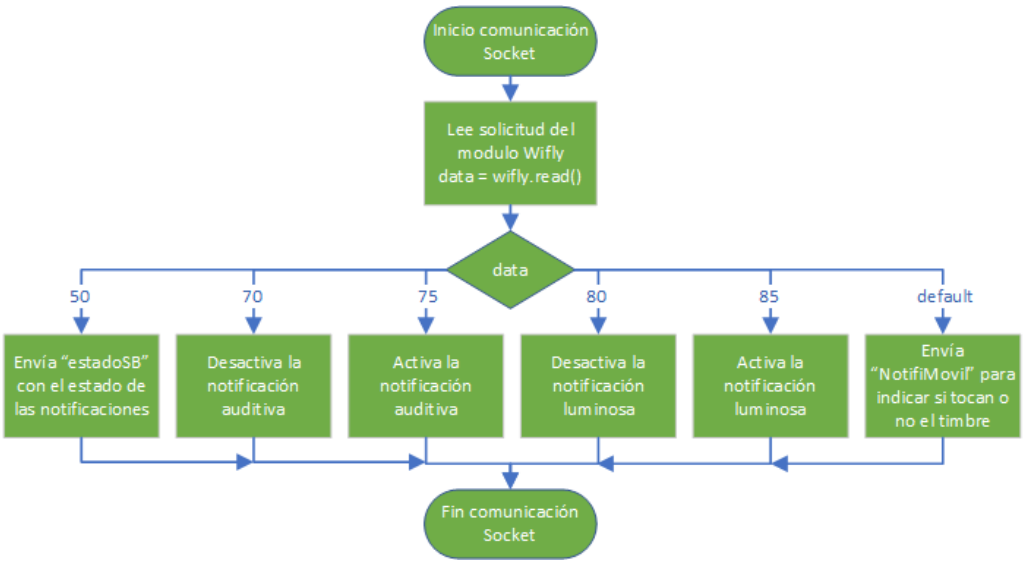


Figura 3.26: Diagrama de flujo de la comunicación Socket

En la **figura 3.26** se puede observar el funcionamiento de SB, según el valor recibido por la comunicación Socket realiza una acción u otra (**apéndice A.1.4**):

- **50:** Esta solicitud hace referencia al *Estado de configuración de SB*, donde se responderá con el Byte **estadoSB**.

- **70 y 80:** Estas solicitudes hacen a la referencia a la desactivación de las notificaciones auditiva y luminosa, respectivamente.

- **75 y 85:** Estas solicitudes hacen referencia a la activación de las notificaciones auditiva y luminosa, respectivamente. Tanto en la activación o desactivación

de cualquiera de las notificaciones, después de configurar la notificación y almacenarla en la memoria EEPROM, se envía una respuesta ***ok*** a la aplicación Android

- **default:** Si la solicitud no fue ninguna de las anteriores, sea cual sea el mensaje de la solicitud, realizará esta acción, donde el SB envía su estado, es decir, si tocan o no el timbre.

CAPÍTULO 4

APLICACIÓN SMARTBELL

Para este proyecto se utiliza el *IDE* de *Android Studio*. Ahora se pretende justificar el uso de esta *IDE* y no otras. Para ello, se explicará cada una de las distintas posibilidades a utilizar y se justificará él motivo por el que se utilizó estas *IDE*.

4.1 IDE - Framework

Para el desarrollo de la aplicación se necesita un IDE o framework, el cual podrá limitar al desarrollo para una sola plataforma. Las más comunes son:

- *Xcode*: es un IDE para las plataformas macOS, iOS, watchOS y tvOS. Pero es exclusivo de macOS, lo cual limita mucho el desarrollo. se programa con el lenguaje de programación Swift.

- *Android Studio*: es un IDE para la plataforma de Android y su lenguaje de programación es Java. Está disponible para Windows, macOS y Linux. Y es el que se trató con más detalle en el transcurso de la carrera.

- *IONIC*: es un framework híbrido, para las plataformas Andoid, IOS, Windows y BlackBerry. Se programa con terminologías web, como: CSS, HTML5 y Sass. Se puede desarrollar desde Windows, macOS y Linux.

La primera opción elegida fue **IONIC**, por su gran variedad de plataformas soportadas con un único código. Pero a la hora de desarrollar la aplicación, se vio que al generar la comunicación Socket, la información se enviaba con cabeceras.

Figura 4.1: Información enviada por *Ionic*, en comunicación Socket

Como se puede observar en la **figura 4.1**, antes de enviar el mensaje deseado, se envían cabeceras. Esto se debe a que Ionic está basado en lenguaje web. Filtrar la información necesaria de las cabeceras colapsaba el módulo WiFly y no permitía ningún otro tipo de comunicación hasta que este terminase.

De este modo, se decidió utilizar **Android Studio**, ya que en la comunicación socket no se envía ninguna cabecera y se puede desarrollar en más plataformas. Además, el alumno está más familiarizado con este IDE.

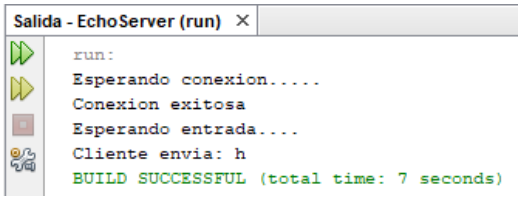


Figura 4.2: Información enviada por *Android*, en comunicación Socket

En la **figura 4.2** se puede observar que Android solo envía el mensaje ***h*** sin ninguna cabecera previa.

4.1.1 Aplicación Android

Las aplicaciones Android se componen de Activities. *Una Activity es un componente de la aplicación que contiene una pantalla con la que los usuarios pueden interactuar para realizar una acción, como marcar un número telefónico, tomar una foto, enviar un correo electrónico o ver un mapa. A cada actividad se le asigna una ventana en la que se puede dibujar su interfaz de usuario. Una aplicación generalmente consiste en múltiples actividades vinculadas de forma flexible entre sí. Normalmente, una actividad en una aplicación se específica como la actividad "principal"que se presenta al usuario cuando este inicia la aplicación por primera vez. Cada actividad puede a su vez iniciar otra actividad para poder realizar diferentes acciones* [22].

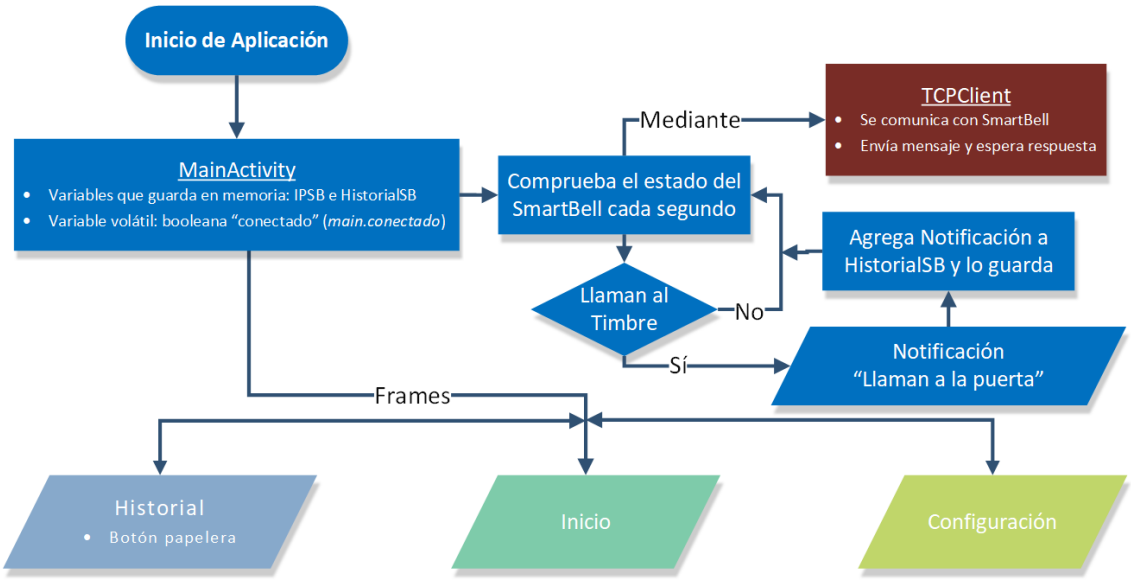


Figura 4.3: Diagrama de flujo de la aplicación Android

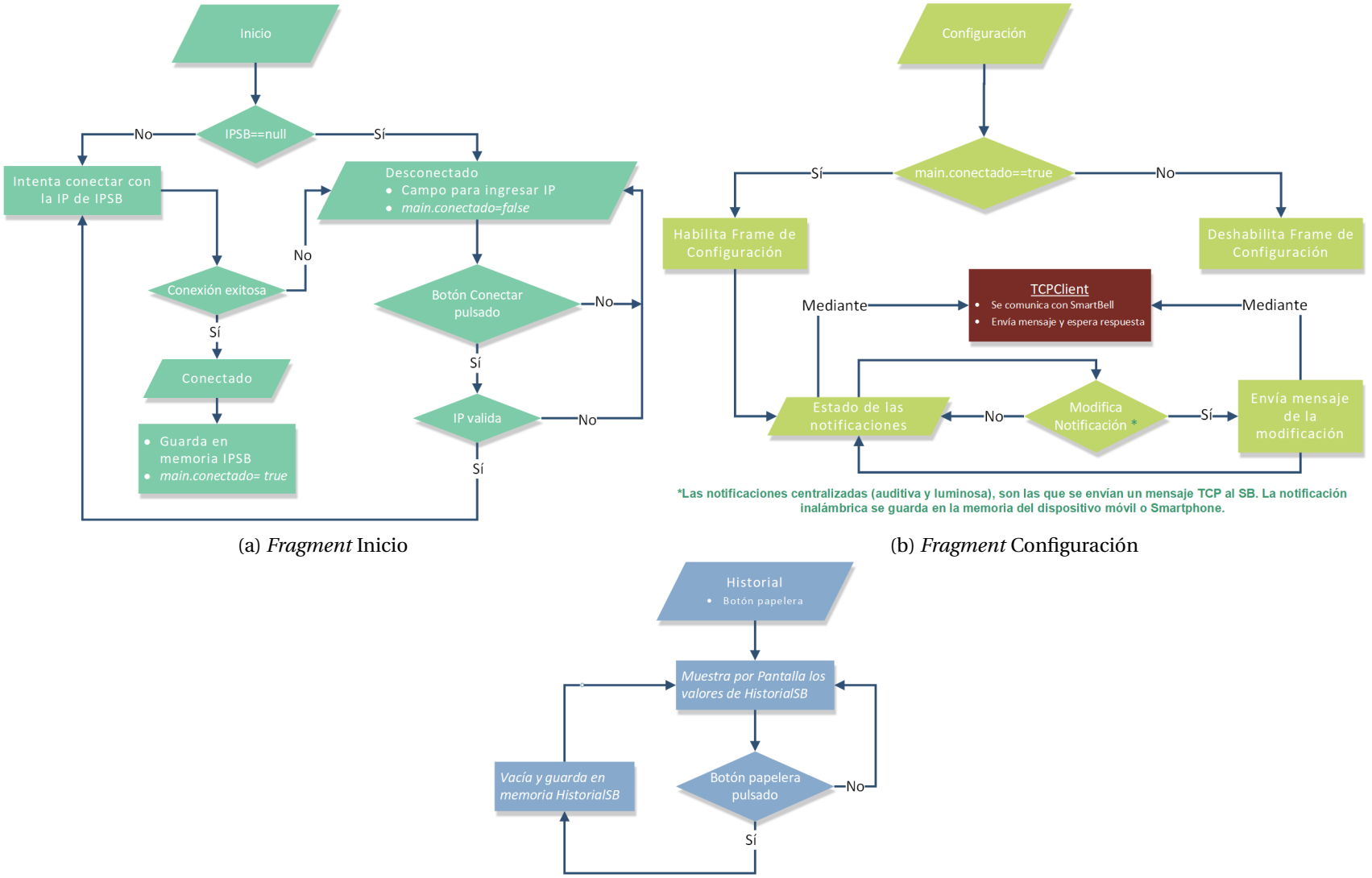
En la **figura 4.3** se puede observar que la aplicación tiene 3 *Frames* o vistas (Historial, Inicio y Configuración), además de la *MainActivity* que comprueba de forma periódica si llaman al timbre. También guarda las variables como la IP, el historial o el estado de la conexión.

Para este proyecto, la aplicación Android desarrollada se llama **SmartBell**, el cual está compuesto por 2 *Activities*, como se puede observar en la **figura 4.3**:

- **MainActivity:** Es la *Activity* principal, la que se ejecuta cuando se inicia la aplicación. Desde esta *Activity*, se almacena información en la memoria del dispositivo móvil, así como el historial de las veces que llamaron al timbre, la dirección IP del SmartBell (SB) y el estado de la notificación inalámbrica. También es desde donde se comprueba de forma periódica el estado del timbre, lanzando una alerta por pantalla si llaman al timbre (**apéndice A.2.3**). Esta *Activity* está compuesta por una interfaz de 3 pestañas, donde cada una de estas pestañas se le denomina *Fragment*.

- **TCPClient:** Este *Activity* se extiende de un *Thread*, es decir, es una tarea en segundo plano. Esta *Activity* es muy importante, ya que es la que se utiliza para conectarse con el SB, enviando y recibiendo mensajes (**apéndice A.2.6**).

Un Fragment representa un comportamiento o una parte de la interfaz de usuario en una Activity. Puedes combinar múltiples fragmentos en una sola actividad para crear una IU multipanel y volver a usar un fragmento en múltiples actividades. Puedes pensar en un fragmento como una sección modular de una actividad que tiene su ciclo de vida propio, recibe sus propios eventos de entrada y que puedes agregar o quitar mientras la actividad se esté ejecutando (algo así como una "subactividad"que se puede volver a usar en diferentes actividades) [23]. La aplicación está compuesta por los siguientes *Fragments*:



(a) *Fragment* Inicio

(b) *Fragment* Configuración

(c) *Fragment* Historial

Figura 4.4: *Fragments* de la aplicación Android

- **Inicio Fragment:** Desde este *Fragment* se ve el estado de la conexión entre la aplicación y el SB, y es el que se ve al iniciar la aplicación. Como se puede observar en la **figura 4.4 (a)**, cuando se accede a este *Fragment*, la aplicación comprobará si tiene almacenada en memoria una dirección IP válida e intentará conectarse con el sistema SB (**apéndice A.2.9**), también intentará conectarse con el sistema al pulsar el botón *Conectar con SmartBell* mostrando uno de los dos resultados mostrados en la siguiente imagen.

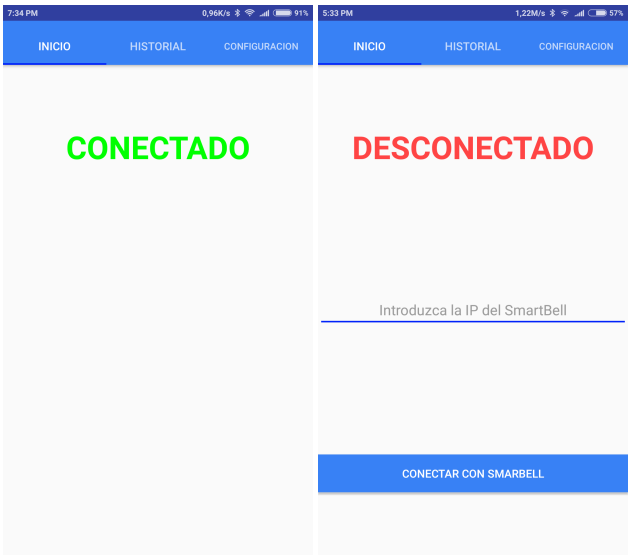


Figura 4.5: Fragment de inicio

En la **figura 4.5** se puede apreciar los posibles casos en los que se puede encontrar este *Fragment*. Por un lado, existe el caso en el que la aplicación se encuentra conectada con SB (en la imagen de la izquierda). Por otra parte, el caso que se puede apreciar es aquel en que la aplicación no esté conectada con SB, de modo que se proporcionará la opción de ingresar la dirección IP del SB para poder conectarse.

- **Historial Fragment:** En este *Fragment* se puede observar el historial de las veces que se ha tocado el timbre. En la **figura 4.4 (c)** se puede ver que este *Fragment* muestra el historial de la veces que llamaron al timbre, y que al pulsar el botón de la *Papelera* se vacía el historial (**apéndice A.2.14**).

Figura 4.6: Fragment de historial

En la **figura 4.6** se puede ver el historial de la aplicación SB.

- **Configuracion Fragment:** Desde este *Fragment* se puede activar o desactivar cualquiera de los 3 tipos de notificaciones del SB. En la **figura 4.4 (b)** se puede observar que al acceder a este *Fragment*, la aplicación envía una solicitud del estado de las notificaciones centralizadas (auditiva y luminosa) al SB y consulta el estado de la notificación inalámbrica en la memoria del dispositivo móvil, mostrando por pantalla el estado de cada una de las notificaciones del SB (**apéndice A.2.16**). Si se modifica una de las notificación centralizadas (**apéndice A.2.18**), la aplicación envía el código de solicitud de la modificación (estos códigos de solicitud ya han sido explicados en el **punto 3.2.1.2**). Después de enviar el código de solicitud, espera una respuesta de **OK**, indicando que el cambio se ha realizado de forma exitosa. Si se modifica el estado de la notificación inalámbrica, éste lo modifica en la memoria del dispositivo móvil.

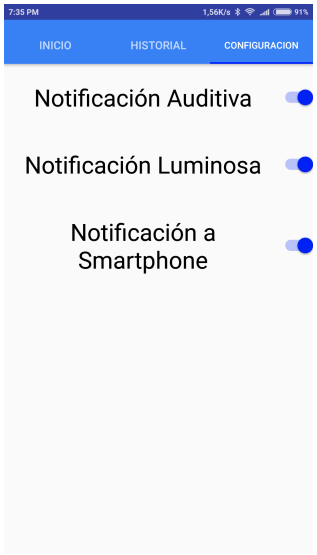


Figura 4.7: Fragment de inicio

Como se puede ver en la **figura 4.7**, desde este *Fragment* se puede ver el estado de las notificaciones y modificarlas.

Uno de los archivos importantes de la aplicación Android es **AndroidManifest.xml** (**apéndice A.2.20**). *El archivo Manifest proporciona información esencial sobre la aplicación al sistema Android, información que el sistema debe tener para poder ejecutar el código de la app. Entre otras cosas, el archivo de Manifest hace lo siguiente:*

- *Nombra el paquete de Java para la aplicación. El nombre del paquete sirve como un identificador único para la aplicación.*
- *Describe los componentes de la aplicación, como las actividades, los servicios, los receptores de mensajes y los proveedores de contenido que la integran. También nombra las clases que implementa cada uno de los componentes y publica sus capacidades, como los mensajes Intent con los que pueden funcionar. Estas declaraciones notifican al sistema Android los componentes y las condiciones para el lanzamiento.*
- *Determina los procesos que alojan a los componentes de la aplicación.*
- *Declara los permisos debe tener la aplicación para acceder a las partes protegidas de una API e interactuar con otras aplicaciones. También declara los permisos que otros deben tener para interactuar con los componentes de la aplicación.*
- *Enumera las clases Instrumentation que proporcionan un perfil y otra información mientras la aplicación se ejecuta. Estas declaraciones están en el manifiesto solo mientras la aplicación se desarrolla y se quitan antes de la publicación de esta.*
- *Declara el nivel mínimo de Android API que requiere la aplicación.*
- *Enumera las bibliotecas con las que debe estar vinculada la aplicación* [24].

CAPÍTULO 5

RESULTADOS

En este apartado se explicará de manera detallada la lógica del proyecto, con el fin de conseguir una mejor comprensión del resultado.

Todos los elementos utilizados para el proyecto se desarrollaron en una *Protoboard*, ya que permite un gran abanico de cambios mientras se desarrollan y se hacen las pruebas.

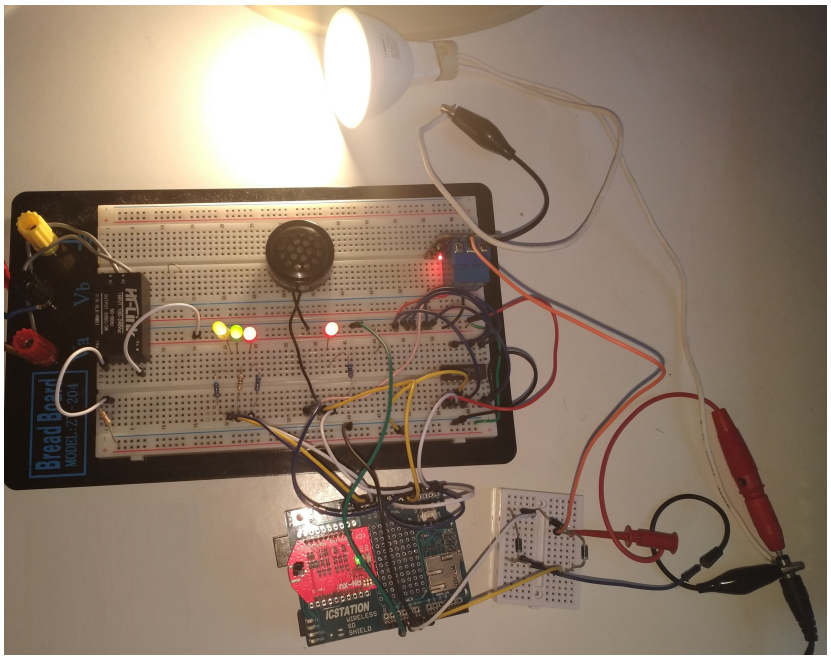


Figura 5.1: Circuito montado en Protoboard

Como se puede ver en la **figura 5.1**, el circuito final montado en la *Protoboard*

Una vez se tuvo el circuito final con todas las funcionalidades, para una mejor integración de todos los elementos utilizados, se realizaron dos circuitos impresos.

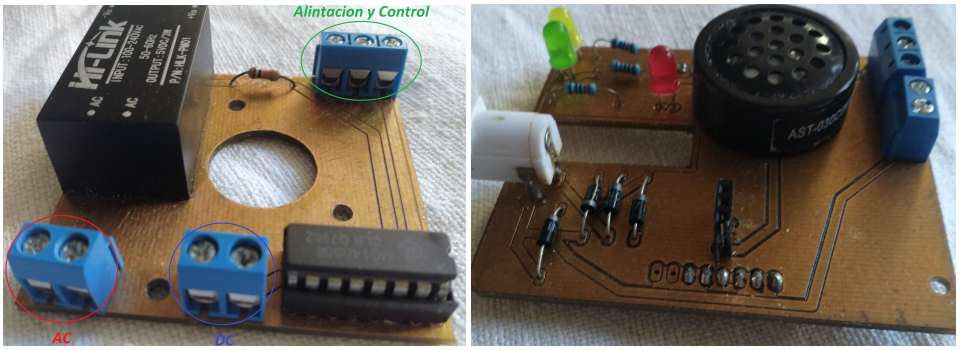


Figura 5.2: Circuitos impresos

En la **figura 5.2** se puede apreciar los dos circuitos impresos. En la imagen de la izquierda, aparecen los elementos tales como los dos tipos de entrada del timbre(AC y DC), el transformador (que convierte la corriente AC a DC), el *Hex Buffer* y los pines de *Alimentación y Control* (el pin de control indica si tocaron el timbre o no, 5V y GND). En la imagen de la derecha están los elementos restantes, como la entrada de alimentación, el puente de diodos, el altavoz, 2 LEDs (uno amarillo y otro verde que indican si están activadas las notificaciones luminosa y auditiva, respectivamente), 1 LED rojo (que indica si llaman al timbre independientemente de la configuración de las notificaciones anteriores) y los pines que se encargan alimentar al otro circuito impreso (circuito de la izquierda) y otro par de pines para el *Relé* (que se conecta con la bombilla LED).

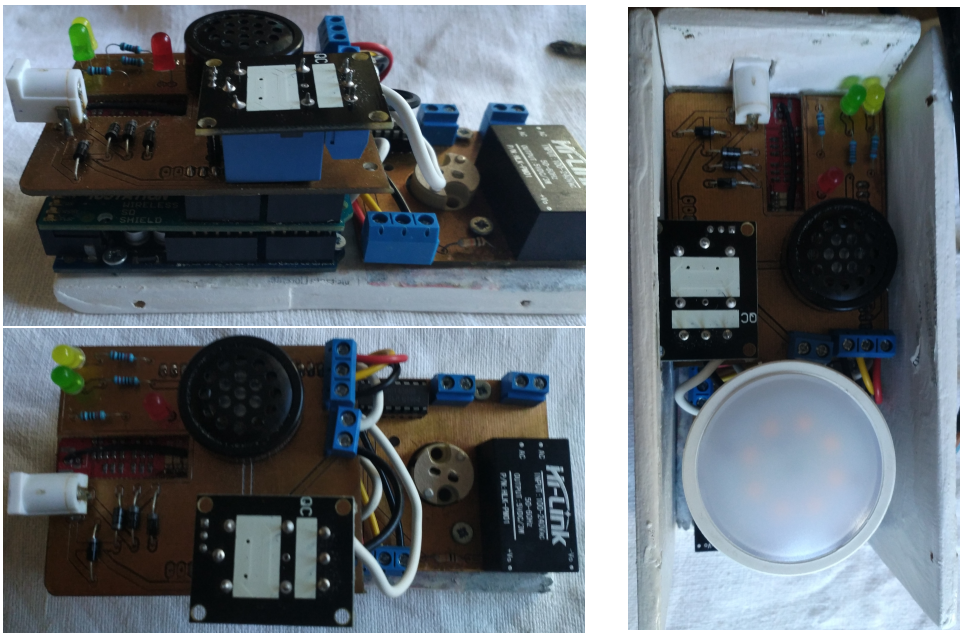


Figura 5.3: Circuitos montados y conectados

En la **figura 5.3** se puede observar todo el conjunto de circuitos montados y conectados para su funcionamiento.

Para una mejor compresión del funcionamiento del SB, se explicará cada parte del funcionamiento del proyecto.

5.1 Configuración de la red WiFi

Para el funcionamiento del SB, es necesario que se conecte a la red WiFi de casa. Para ello, se ha de conectar el SB con el ordenador mediante el puerto USB.

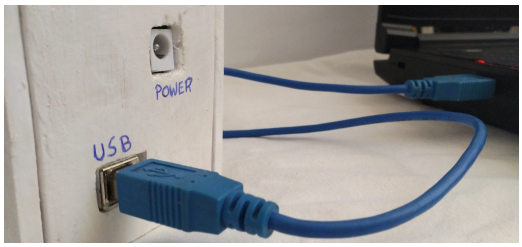


Figura 5.4: Conexión del SB con el ordenador

Una vez conectado el SB con el ordenador (como muestra la **figura 5.4**) se abre el *IDE de Arduino* en el ordenador, para ejecutar el *Monitor Serie*. Desde donde se interactúa con el SB, para configurar la conexión WiFi o las notificaciones (como ya se explicó en el **punto 3.1.1.4**). Cabe recordar que al abrir el *Monitor Serie*, se muestra el estado de la conexión con la red WiFi y los detalles de dicha conexión, como la dirección IP.

Es aconsejable deshabilitar el servicio DHCP, para establecer una IP fija al SB. Si se activa el servicio DHCP el SB puede tener una IP diferente cada cierto tiempo, lo cual puede ser un tanto molesto, ya que se deberá averigua la dirección IP cada vez que ésta cambie.

Una vez configurada la conexión WiFi, ya se puede desconectar el SB del ordenador.

5.2 Conexión con el timbre

Como ya se comentó anteriormente, en el mercado existen dos tipos de timbres: los que se usan con porteros automáticos (los cuales utilizan DC) y los que son solo pulsador y timbre (los cuales usan AC).

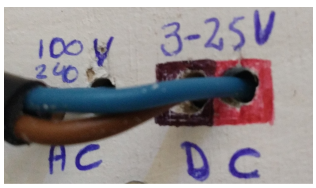


Figura 5.5: Conexión con el timbre

Una vez identificado el tipo de timbre que se tenga, se conecta al SB respetando la polaridad en el caso de DC, como se puede observar en la en la **figura 5.5**.

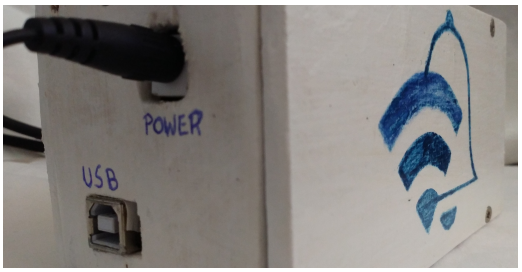


Figura 5.6: Alimentación del SB

Finalmente, se conecta la alimentación del SB, en este caso con una fuente de 9V y 0.6A, como muestra en la **figura 5.6**.

5.3 Conexión con la aplicación Android

Una vez instalada la aplicación Android, se accede a la aplicación.

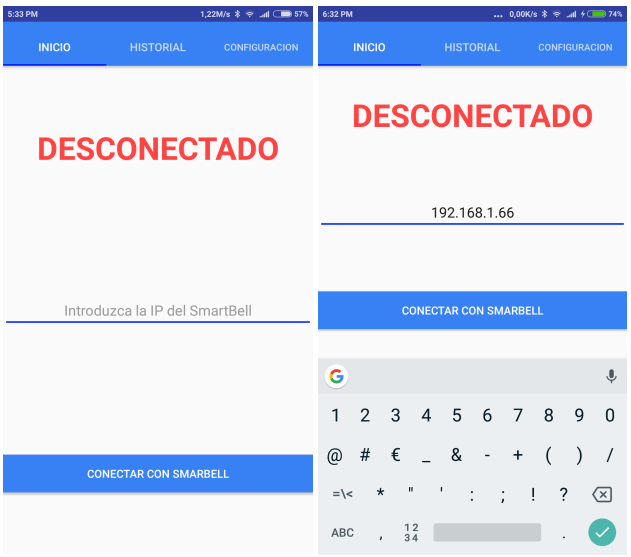


Figura 5.7: Conexión de aplicación Android con SB

Al ser la primera vez, no se tendrá ninguna IP almacenada en memoria, y se mostrará la pantalla de desconectado, donde se ingresará la IP del SB. Como se puede ver en la **figura 5.7**

Para conectarse se pulsa el botón *CONECTAR CON SMARTBELL*. Si la conexión es exitosa se mostrará la pantalla de conexión exitosa. Como se puede observar en la siguiente imagen.

Figura 5.8: Conexión exitosa de aplicación Android con SB

Cada vez que se abra la aplicación y el SB esté conectado a la red WiFi con la misma IP, se mostrará en pantalla el contenido de la **figura 5.8**. Esto indicara que está conectado en el SB. Si la conexión se pierde, se mostrará por pantalla el mensaje **DESCONECTADO**, como se puede apreciar en la **figura 5.7**

5.3.1 Historial y configuración de notificaciones

Desde la aplicación se puede acceder al Historial, donde se muestran los detalles de fecha, hora y las veces que ha sido pulsado el timbre.

Figura 5.9: Historial de las veces que llamaron al timbre

En la **figura 5.9**, se puede ver un ejemplo del historial. Si se desea borrar el historial, se debe pulsar el botón azul con el icono de una papelera.

Desde la aplicación, también se puede acceder a la configuración de las notificaciones.

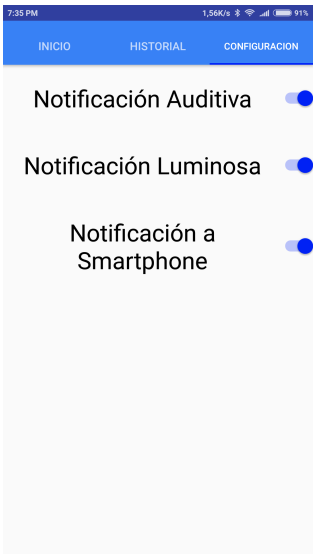


Figura 5.10: Configuración de las notificaciones del SB

Como se puede observar en la **figura 5.10**, se tiene las tres notificaciones habitadas. Desde este punto, se puede activar o desactivar cada una de las notificaciones. Cabe recalcar que las notificaciones auditiva y luminosa son centralizadas, es decir afectan directamente al SB, mientras que la notificación inalámbrica es local, de modo que solo afecta al dispositivo móvil desde donde se ejecuta la aplicación.

5.4 Notificaciones

Como ya se comentó existen tres tipos de notificaciones.

Figura 5.11: LEDs de notificaciones del SB

En la **figura 5.11**, se puede observar que existen dos LEDS (amarillo y verde) los cuales informan del estado de las notificaciones luminosa y auditiva, respectivamente. Si la notificación está activada el LED correspondiente estará encendido. En la figura también se pueden ver otros dos LEDs, uno rojo con la etiqueta **NOT** (el cual se enciende cuando llaman al timbre, estén o no activadas las notificaciones centralizadas), y otro LED con la etiqueta **CON** (el cual indica el estado del módulo WiFly, el significado de cada color ya fue explicado en la **tabla 3.3**). En la imagen también se puede ver la bombilla LED para la notificación luminosa y los orificios para la salida del audio para la notificación auditiva.

Como ya se explicó anteriormente, solo se notificará por los tipos de notificaciones activadas, cuando alguien toque el timbre.

A partir de este punto se va a suponer que las tres notificaciones están activadas, tanto las dos centralizadas como la inalámbrica.

Figura 5.12: Notificaciones centralizadas del SB

Como se puede observar en la **figura 5.12**, la bombilla LED se encuentra encendida debido a que la notificación luminosa está activada. También se reproduce el tono, porque la notificación auditiva también está activada. De igual manera, se puede ver que el LED **NOT** está activado, se debe recordar que, independientemente de que las notificaciones centralizadas estén activadas o no, este LED **NOT** se enciende cada vez que se toca el timbre, indicando que se está notificando de forma inalámbrica a los dispositivos conectados al SB.

Los dispositivos móviles que estén conectados al SB y tengan la notificación inalámbrica activada, recibirán dicha notificación.

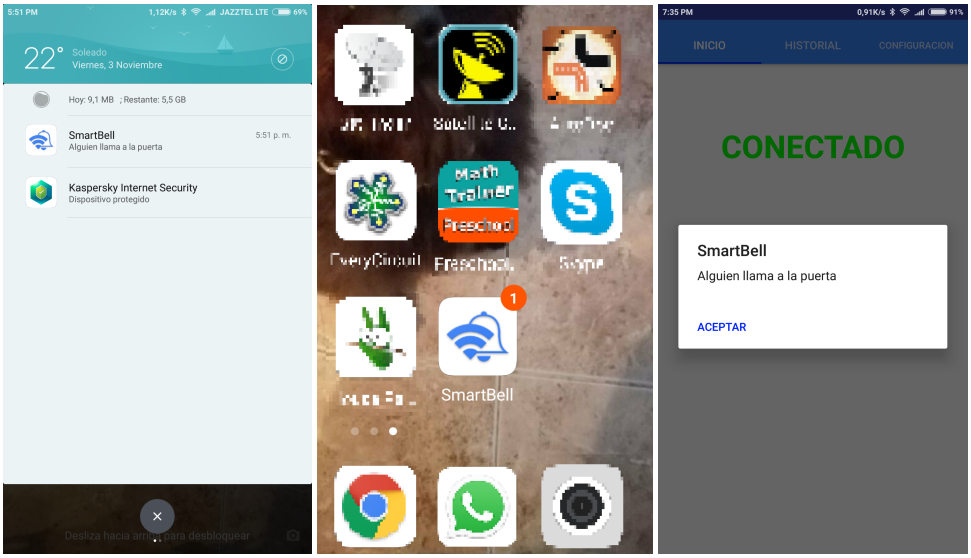


Figura 5.13: Notificación del SB al SmartPhone

En la **figura 5.13** se puede observar la notificación del SB en la barra de notificaciones, en el icono de la aplicación de la pantalla de inicio, y en forma de mensaje emergente en el caso de que se tenga la aplicación abierta.

5.5 Conexión desde Internet con el SmartBell

Puede suceder que se quiera conectar, configurar y recibir las notificaciones del SB fuera de nuestra red local (Local Area Network (LAN)), es decir, desde *Internet* (Wide Area Network (WAN)).

Para ello, se tendrá que saber la **WAN IP** del Router de casa. También se deberá abrir el puerto *666* en el Router para que se pueda conectar la aplicación con el SB. Finalmente, el Router deberá redireccionar las consultas a ese puerto hacia la IP de SB.

Para acceder a un Router doméstico, primero se debe saber su LAN IP. Para ello desde cualquier ordenador conectado al Router. Se debe abrir un terminal e insertar el siguiente comando:

- **ipconfig:** Para ordenadores con sistema operativo *Windows*.

- **ifconfig:** Para ordenadores con sistema operativo *Linux* y *MacOS*.

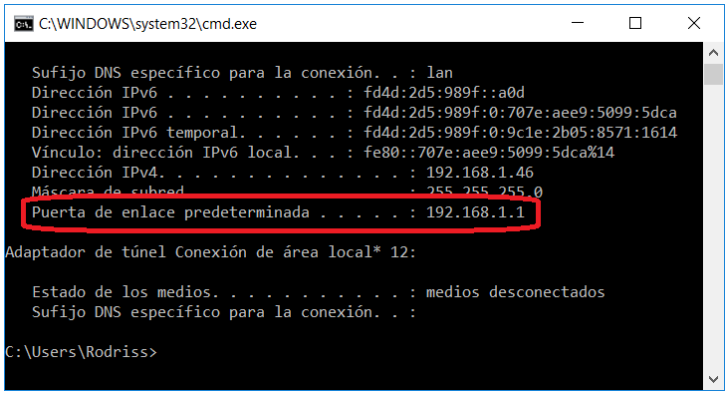


Figura 5.14: Obtención de la IP de un Router doméstico

Una vez ejecutado el comando, se mostrará toda la información de las conexiones y de los puertos de red del ordenador. De toda esa información, la IP del Router es la del **Puerta de enlace Predeterminado** o **Gateway**, como se puede observar en la **figura 5.14**.

Desde cualquier navegador web, se debe acceder a la dirección IP del Router.

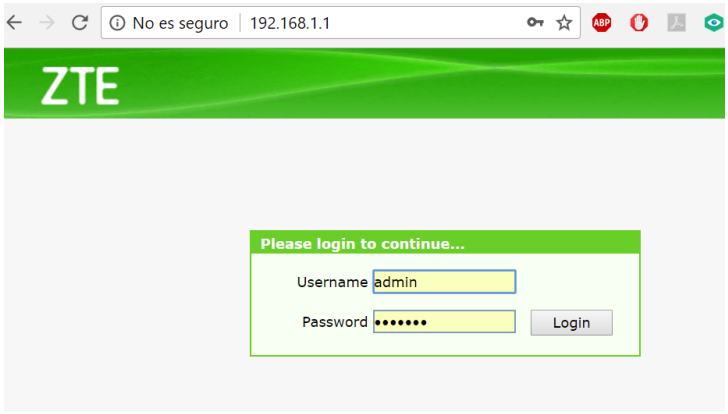


Figura 5.15: Conexión con el Router desde un navegador web

Como se puede ver en la **figura 5.15**, el Router solicitará un *usuario* y *contraseña*. Estos datos suelen estar situados en la parte posterior del mismo Router. Si no fuese así, deberá solicitarlo a su proveedor de servicio de Internet.

Una vez dentro del Router, se podrá averiguar la **WAN IP**.

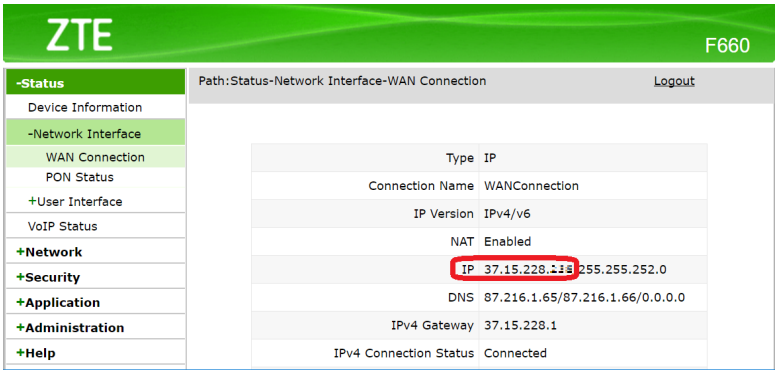


Figura 5.16: WAN IP del Router

Como existen muchos modelos de Routers, la forma de obtener la **WAN IP**, puede variar. Sin embargo, ésta suele estar en el apartado **WAN Connection** o **WAN Status**. Como se puede ver en la **figura 5.16**.

Una vez obtenida la dirección IP, ésta será la que se deberá ingresar en la aplicación Android. Pero en este punto todavía no se podrá establecer la conexión, ya que no está abierto el puerto *666*.

La forma de abrir un puerto en los Routers varía mucho entre un modelo u otro. Por eso es aconsejable se busque en el manual del Router, la forma de abrir un puerto.

En este punto se mostrará mostrará una forma de abrir un puerto, para un Router *ZTE* modelo *F660*.

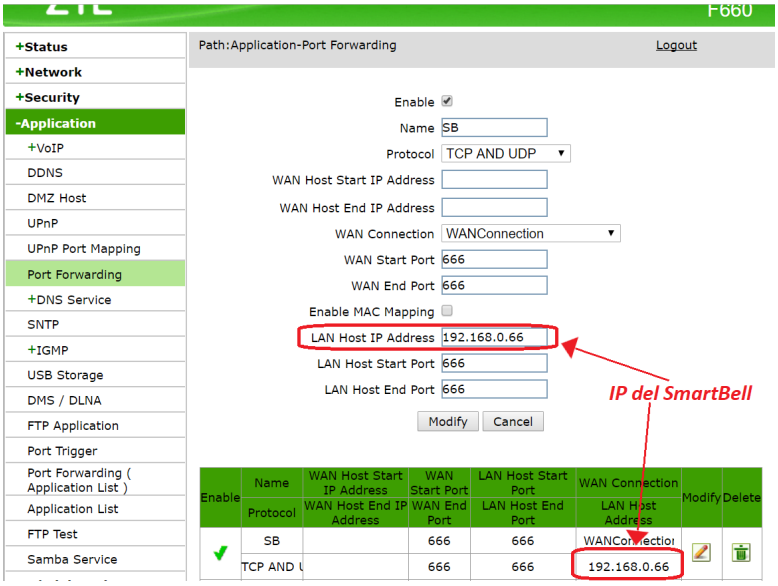


Figura 5.17: Abrir el puerto 666 en el Router

Primero se deberá agregar el puerto en *Port Forwarding*, indicando los puertos que se abren para la *WAN* y para la *LAN*, que en este caso es el puerto *666*. Los protocolos

deberán ser *TCP y User Datagram Protocol (UDP)*. Finalmente, se ingresa un nombre, la IP de SB, y se guarda la configuración. Este proceso puede verse en la **figura 5.17**.

Llegados a este punto, ahora sí se podrá conectar con el SB desde *Internet*, utilizando la **WAN IP** del Router en la aplicación Android, por lo que todas las funciones de la aplicación son funcionales.

Para una mejor comprensión, se realizó y se subió a Youtube un vídeo. En este vídeo se explica de una forma detallada el funcionamiento del SB.

Figura 5.18: Vídeo explicativo alojado en Youtube

CAPÍTULO 6

CONCLUSIONES

6.1 Objetivos

El objetivo principal del proyecto era facilitar la gestión de un timbre para poder ser notificado y controlar dichas notificaciones desde un dispositivo móvil.

Como se ha visto en este documento, este objetivo ha sido completado satisfactoriamente. Se han podido unir todas las dispositivos y tecnologías planteados para la creación del sistema del SmartBell (SB), obteniendo un correcto funcionamiento del conjunto.

Por lo tanto, se puede concluir que todos los objetivos específicos que se listarán a continuación han sido cumplidos:

1. Comunicar (mediante una red WiFi) el módulo Arduino y un dispositivo móvil con sistema operativo Android, para poder ser notificado y configurar las notificaciones del timbre.

2. Las notificaciones que se pueden activar/desactivar son:

   - *Auditiva*: El timbre sonará, como un timbre cualquiera.
   - *Luminosa*: Se encenderá una bombilla LED.
   - *Inalámbrica*: El timbre enviará una notificación a los dispositivos de la red que tengan su aplicación instalada y configurada. A diferencia de las otras dos notificaciones, ésta se activa y desactiva de forma local en cada dispositivo móvil.

6.2 Valoración personal

En el desarrollo de este proyecto, el alumno se ha tenido que enfrentar a diferentes plataformas y tecnologías, así como distintos lenguajes de programación, algunos de

ellos nuevos para él. El hecho de no limitarse a un lenguaje de programación conocido por el alumno para realizar el proyecto se convirtió en una motivación.

Realizar este proyecto le ha permitido comprobar el alcance que puede tener la placa Arduino, utilizando algunas de las tantas las posibilidades de configuración y programación que éste proporciona.

Además, aunque ya tenía unas nociones básicas de desarrollo de aplicaciones con Android (adquiridas en el transcurso de la carrera), realizar este proyecto le ha ayudado a entender mejor su estructura, programación y su versatilidad, cosa que valora mucho por la importancia que tienen hoy en día las aplicaciones móviles.

Finalmente, haber unido todas estas tecnologías, intentando que esto no se convierta en una complicación para el resultado final del proyecto, hizo comprender mejor el mundo de las telecomunicaciones y la informática, ya que, de una forma u otra, siempre será posible la combinación de diferentes tecnologías mediante algún protocolo de comunicación. Esto permite a los ingenieros crear proyectos de todo tipo, utilizando nuevas tecnologías, adaptarse y trabajar con ellos sin ningún problema.

6.3 Líneas de trabajo futuro

A continuación, se presentan algunas mejoras que se podrían realizar al proyecto, en cualquiera de los aspectos del sistema o en el funcionamiento:

- Realizar la aplicación, para otras plataformas como iOS, Windows Phone, o incluso para sistemas operativos de Windows o Mac, siguiendo la estructura de los mensajes que se intercambien entre la aplicación y el SB.
- Mejorar la aplicación Android para que se adapten perfectamente a todo tipo de pantallas, sin tener que depender de la resolución que éstas utilicen.
- Partiendo de los dispositivos comerciales que están en el mercado y la gran posibilidad de configuración y programación de Arduino, se podrían agregar características al SB, como:

    a) Poder abrir la puerta desde la aplicación Android.

    b) Mediante el uso de una cámara. Poder ver quien llama al timbre.

    c) Mediante el uso de un lector de huellas abrir la puerta.

Por otro lado, ya que el sistema se basa en la comunicación entre Arduino y una aplicación Android, tomando como base las clases creadas en las diferentes plataformas (ya detalladas en este documento), sería posible realizar cualquier proyecto que necesitara una comunicación entre Arduino y una aplicación Android.

Por lo tanto, este proyecto abre la puerta a la imaginación, para cualquiera que quiera crear nuevos proyectos.

APÉNDICE A

CÓDIGOS DEL PROYECTO

A.1 Códigos de Arduino

A continuación, se enumerarán y explicarán partes del código de SmartBell (SB)

A.1.1 Lectura y escritura de memoria EEPROM

```
/**
 * Se cargan las variables de la configuración del timbre
 *  guardados en memoria.
 */
void SmartBell::cargarConfTimbre()
{
  audio = EEPROM.read(eepromTimbre);
  bitWrite (estadoSB, eepromTimbre, audio);
  eepromTimbre ++;
  flash = EEPROM.read(eepromTimbre);
  bitWrite (estadoSB, eepromTimbre, flash);
  eepromTimbre = 0;

  digitalWrite (ledNotFlash, flash);
  digitalWrite (ledNotAudio, audio);
}
```

Código A.1: Lectura en EEPROM

Como se puede observar en el **código A.1** se lee de la memoria EEPROM la configuración de las notificaciones del timbre.

A continuación, se explicará el comando utilizado para la lectura de la memoria EEPROM:

variableX = EEPROM.read(posiciónMemoria);

Byte estadoSB								
bit	7	6	5	4	3	2	1	0
Notificación	-	-	-	-	-	-	Luminosa	Auditiva

Cuadro A.1: Estructura del Byte **estadoSB**

- *variableX:* es la variable donde se guardará la información leída de la memoria EEPROM.
- *EEPROM.read:* es el comando en sí, que lee una posición de la memoria.
- *posiciónMemoria:* es un entero qué india que posición de memoria se leerá.

Cabe destacar el uso del comando ***bitWrite(x, n, b);***. Aunque no se utilice para leer de la memoria EEPROM. Este comando ayuda a enviar en un solo Byte toda la configuración de las notificaciones a la aplicación Android (este Byte se denominará **estadoSB**), mediante el módulo WiFly. Así se optimiza la cantidad de mensajes que se intercambian entre el SB y la aplicación Android.

- *bitWrite:* este comando escribe 1 ó 0, en uno de los ocho bits posibles de un Byte.
- *x:* Byte a modificar.
- *n:* número del bit a modificar. Siendo el bit menos significativo el 0, y el más significativo el 7 (hacia la izquierda)
- *b:* valor a escribir, 1 ó 0.

```
/**
 * Activa o desactiva una notificacion, segun el booleano
 *  que recibe como parametro y lo guarda en memoria
 * @param estado - booleano a comprobar
 */
void SmartBell::conf(boolean estado)
{
  switch (eepromTimbre) {
    case 0:
      bitWrite (estadoSB, eepromTimbre, estado);
      audio = estado;
      EEPROM.write(eepromTimbre, audio);
      break;

    case 1:
      bitWrite (estadoSB, eepromTimbre, estado);
      flash = estado;
      EEPROM.write(eepromTimbre, flash);
      break;
  }
}
```

Código A.2: Escritura en EEPROM

En el **código A.2** se puede observar parte del código donde se guarda en la memoria EEPROM el estado de las notificaciones.

Ahora se explicará el comando utilizado para escribir en la memoria EEPROM:

EEPROM.write(posiciónMemoria, valorX);

- *EEPROM.write:* es el comando en sí, que escribe un valor en una posición de la memoria.
- *posiciónMemoria:* es un entero que india que posición de memoria donde se escribirá la información.
- *valorX:* es la información a escribir en la memoria EEPROM.

Cabe destacar, que antes de guardar la información de la conexión WiFi en la memoria EEPROM, es mejor limpiar o reiniciar esa posición o posiciones de memoria.

Toda esta información se guarda en la memoria EEPROM cada vez que se modifica algún parámetro de la conexión WiFi o se modifica la configuración de las notificaciones. Ademas, se lee al encender o reiniciar el SB.

A.1.2 Tono de notificación auditiva

```
/////Variables de Tono Underworld melody
int underworld_melody[] = {262, 523, 220, 440, 233, 466};

/**
 * Tono que se reproduce por el altavoz, en la notificación auditiva
 *  El tono es de Mario Bross -Underworld melody-
 */
void SmartBell::tono()
{
  for (int thisNote = 0; thisNote < 6; thisNote++) {
    // to calculate the note duration, take one second
    // divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000 / 12;
    tone(NotAudio, underworld_melody[thisNote], noteDuration);

    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(NotAudio);
    solicitudWifly();
  }
}
```

Código A.3: Código del tono [21]

Este **código A.3** simula parte del tono de *Mario Bros, Underwolrd* [21].

También se puede observar que cada vez que se reproduce una nota se consulta el estado de las solicitudes al módulo WiFly, mediante la función *solicitudWifly* (que se detallara en el **apéndice A.1.4**).

A.1.3 Conexion con red WiFi

```
/* Nos conectamos a la red WiFi, si aun no estamos conectados */
if (!wifly.isAssociated()) {
  /* Configuramos el WiFly para conectar a la red WiFi */
  wifly.setSSID(mySSID);
  wifly.setPassphrase(myPassword);
  if (dhcp) {
    wifly.enableDHCP();
  } else {
    wifly.disableDHCP();
    wifly.setIP(myipConca);
  }
  wifly.save();
  if (wifly.join()) {
    Serial.print(F("Conectado a red wifi: "));
    Serial.println(ssid);
    WifiInfo();
  } else {
    Serial.print(F("Fallo de conexion con red wifi: "));
    Serial.println(ssid);
  }
} else {
  Serial.print(F("Conectado a red wifi: "));
  Serial.println(wifly.getSSID(buf, sizeof(buf)));
  WifiInfo();
}
```

Código A.4: Conectar con red WiFi

En el **código A.4** se puede observar parte del código, donde se comprueba si el SB está conectado a una red WiFi. Si no es así, se configura la conexión a la red WiFi, con la SSID y contraseña correspondientes. Luego comprueba si esta activada o desactivada la función DHCP (la que proporciona una dirección IP de forma automática, por parte del Router). Si se encuentra desactivada, configura la dirección IP establecida por el usuario. Finalmente, se conecta a la red WiFi con los parámetros ya establecidos anteriormente y comprueba si la conexión ha sido exitosa o no, mostrando por por pantalla (*Monitor Serie* de la IDE de Arduino) el respectivo mensaje informativo.

Este código se utiliza en la función **setup()**, es decir, cuando se inicia o reinicia el SB. Cuando se ingresa datos nuevos en la configuración WiFi, el método es prácticamente el mismo, con la excepción de que no comprueba si está conectado a la red desde al inicio.

A.1.4 Interacción con solicitudes de la aplicación Android

```
/**
 * Escucha si hay solicitudes en el módulo Wifly
 * Lee el valor data, que puede valer:
 * @data 50 - estado de las configuraciones de las notificaciones.
 * @data 70 - desactiva la notificacion auditiva.
 * @data 75 - activa la notificacion auditiva.
 * @data 80 - desactiva la notificacion luminosa.
 * @data 85 - activa la notificacion luminosa.
```

```
 * @data default - estado del timbre, si llaman o no al timbre
 */
void SmartBell::solicitudWifly(){
    if (wifly.available() > 0) {
    data = wifly.read();
    switch (data) {
      case 50:
        wifly.println(estadoSB);
        wifly.flush();
        wifly.close();
        break;
      case 70:
        eepromTimbre = 0;
        digitalWrite (ledNotAudio, LOW);
        actDes = false;
        conf(actDes);
        wifly.println("ok");
        wifly.flush();
        wifly.close();
        break;
      case 75:
        eepromTimbre = 0;
        digitalWrite (ledNotAudio, HIGH);
        actDes = true;
        conf(actDes);
        wifly.println("ok");
        wifly.flush();
        wifly.close();
        break;
      case 80:
        eepromTimbre = 1;
        digitalWrite (ledNotFlash, LOW);
        actDes = false;
        conf(actDes);
        wifly.println("ok");
        wifly.flush();
        wifly.close();
        break;
      case 85:
        eepromTimbre = 1;
        digitalWrite (ledNotFlash, HIGH);
        actDes = true;
        conf(actDes);
        wifly.println("ok");
        wifly.flush();
        wifly.close();
        break;
      default:
        wifly.println(notificacionWEB);
        wifly.flush();
        wifly.close();
        break;
    }
  }
}
```

Código A.5: Mensajes de solicitudes de la aplicación Android

Se puede observar en el **código A.5** el tratamiento de las distintas solicitudes por parte de la aplicación Android. Mediante la función *solicitudWifly*

Primero se analiza con **if (wifly.available() >0)** si hay solicitudes en el módulo WiFly. Si fuese así, con **data = wifly.read()** se almacena esa solicitud en una variable llamada *data*.

Dependiendo del tipo de solicitud, se realizará una acción y se responderá a la solicitud. Las solicitudes son la representación numérica de los valores en *ASCII*. Esto se debe a que el módulo WiFly lee la representación numérica de los caracteres *ASCII* en los mensajes recibidos. Las solicitudes pueden ser:

- **50 (2 en *ASCII*):** Esta solicitud hace referencia al *Estado de configuración de SB*, donde se responderá con el Byte **estadoSB**.
- **70 y 80 (F, P en *ASCII* respectivamente):** Estas solicitudes hacen a la referencia a la desactivación de las notificaciones auditiva y luminosa, respectivamente.
- **75 y 85 (K, U en *ASCII* respectivamente):** Estas solicitudes hacen referencia a la activación de las notificaciones auditiva y luminosa, respectivamente. Tanto en la activación o desactivación de cualquiera de las notificaciones, después de configurar la notificación y almacenarla en la memoria EEPROM, se envía una respuesta ***ok*** a la aplicación Android
- **default:** Si la solicitud no fue ninguna de las anteriores, sea cual sea el mensaje de la solicitud, realizará esta acción, donde el SB envía su estado, es decir, si tocan o no el timbre.

A.2 Métodos importantes de la aplicación SmartBell

A continuación, se enumerarán y explicarán los métodos más importantes de la aplicación:

A.2.1 onCreate() de *MainActivity*

Este método es el principal de la clase *MainActivity*, ya que es el que primero se ejecuta al abrir la aplicación.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Create the adapter that will return a fragment for each of
        the three
    // primary sections of the activity.
    mSectionsPagerAdapter = new SectionsPagerAdapter(
        getSupportFragmentManager());

    // Set up the ViewPager with the sections adapter.
    mViewPager = (ViewPager) findViewById(R.id.container);
    mViewPager.setAdapter(mSectionsPagerAdapter);
```

```
    TabLayout tabLayout = (TabLayout) findViewById(R.id.tabs);
    tabLayout.setupWithViewPager(mViewPager);

    //Se recoge de memoria la IP guardada
    preferences = getSharedPreferences(SB, 0);
    IPSB = preferences.getString(IP, "");

    //Se crea la alerta que se mostrara si llaman al timbre
    mcontext = this;
    conectado = false;
    builder = new AlertDialog.Builder(this);
    builder.setMessage("Alguien llama a la puerta")
            .setTitle("SmartBell")
            .setCancelable(false)
            .setNeutralButton("Aceptar",
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog,
                            int id) {
                            dialog.cancel();
                        }
                    });
    alert = builder.create();

    //Se crea el Timer que se ejecuta cada segundo y sin retardo
    crearTimer(0, 1000);

    //Se crean los Fragments
    inicioFragment = new InicioFragment();
    historialFragment = new HistorialFragment();
    configuracionFragment = new ConfiguracionFragment();
}
```

Código A.6: Método onCreate de la clase *MainActivity*

Como se puede observar en el **código A.6**, primero se crea y configura un *Adapter*, el cual sirve para mostrar una de las tres pestañas que tiene la aplicación (*Fragments*). Luego lee de memoria con (*SharedPreferences*) la IP guardada para conectarse con el SB. También se crea la alerta, que se mostrará si llaman al timbre. Esto se hace con un *AlertDialog*.

Después se crea *Timer*, llamando al método *crearTimer()*. Como esta es la *Activity* principal, se ejecuta al abrir la aplicación y es la que siempre está en ejecución, es desde este punto que se ejecuta el *Timer* cada 1000 milisegundos y sin retardo. Finalmente crea cada uno de los *Fragments* de la aplicación.

A.2.2 crearTimer() de *MainActivity*

```
/**
 * Crea un Timer que se ejecuta de un tiempo de retraso establecido,
     Y se jecuta cada "x" tiempo, que es el periodo
 *
 * @param retraso Tiempo en milisegundos, espera ese tiempo antes de
     ejecutar el Timer
 * @param periodo Tiempo en milisegundos, que se va sejecuta la
    consulta del estado del SB
```

```
 */
private void crearTimer(long retraso, long periodo) {
    timer = new Timer();
    task = new TimerTask() {
        @Override
        public void run() {
            handler.post(new Runnable() {
                public void run() {
                    try {

                        //Crea un objeto TCPClient()
                        sbTcpClient = new TCPClient();

                        //Si esta conectado con el SmartBell ejecuta
                             estadoTimbre()
                        if (conectado) {
                            estadoTimbre();
                        }
                    } catch (Exception e) {
                        Log.e("error", e.getMessage());
                    }
                }
            });
        }
    };
    timer.schedule(task, retraso, periodo);
}
```

Código A.7: Método crearTimer de la clase *MainActivity*

En el **código A.7** se puede observar como se crea *Timer*, donde se declaran las acciones que se realizarán periódicamente. En este caso creara el objeto *TCPClient*, que sirve para comunicarse con el SB. Si se está conectado con el SB, ejecuta el método *estadoTimbre()*, con el cual se consulta el estado del timbre (es decir, saber si llaman o no a la puerta, este método será detallado más adelante).

Este método recibe por parámetro el periodo de tiempo que indica cada cuanto tiempo se debe ejecutar la tarea programada, además del retraso que será el tiempo que se debe esperar antes de ejecutar las instrucciones por primera vez, ambos expresados en milisegundos.

A.2.3 estadoTimbre() de *MainActivity*

```
/**
 * Comprueba el estado del SB, mediante una conexión TCP. Si recibe
     la notificación, que llaman a la puerta agrega la información
     al historial. Si la notificación inalámbrica esta activada,
     genera y muestra la notificación.
 */
public void estadoTimbre() {
    sbTcpClient.setSERVERIP(IPSB);
    sbTcpClient.start();

    //Intenta conectar con el SB.
    try {
```

```
            sbTcpClient.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        while (sbTcpClient.isAlive()) {
        }
        estado = sbTcpClient.getEstado();
        if (estado != "OK") {
            sbTcpClient.interrupt();
            sbTcpClient.start();
            try {
                sbTcpClient.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            while (sbTcpClient.isAlive()) {
            }
            estado = sbTcpClient.getEstado();
        }

        //Si la conexión fue exitosa, analiza la respuesta.
        if (estado == "OK") {
            mensajeTimbre = sbTcpClient.getServerMessage();

            //Si llaman a la puerta.
            if (mensajeTimbre.equals("Tocan")) {

                //Se agrega la notificación al historial.
                date = new Date();
                hourdateFormat = new SimpleDateFormat("dd/MM/yyyy -- HH:
                    mm:ss");
                historialFragment.agregarLista(hourdateFormat.format(
                    date));
                this.addItems(hourdateFormat.format(date));

                //Si la notificación inalámbrica esta activada.
                if (isNotifMovil()) {

                    //Se genera una notificación Auditiva
                    try {
                        Uri notification = RingtoneManager.getDefaultUri
                            (RingtoneManager.TYPE_NOTIFICATION);
                        Ringtone r = RingtoneManager.getRingtone(
                            getApplicationContext(), notification);
                        r.play();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }

                    //Se muestra la Notificación emergente, si la
                        aplicación esta abierta
                    alert.show();

                    //Se crea notificación para la barra de
                        notificaciones
                    mBuilder = new NotificationCompat.Builder(this)
                            .setSmallIcon(R.drawable.ic_logo)
```

```
                        .setContentTitle("SmartBell")
                        .setContentText("Alguien llama a la puerta")
                            ;
                Intent resultIntent = new Intent(this, MainActivity.
                    class);
                PendingIntent resultPendingIntent = PendingIntent.
                    getActivity(
                        this,
                        0,
                        resultIntent,
                        PendingIntent.FLAG_UPDATE_CURRENT
                );
                mBuilder.setContentIntent(resultPendingIntent);
                NotificationManager mNotifyMgr =
                        (NotificationManager) getSystemService(
                            NOTIFICATION_SERVICE);
                mNotifyMgr.notify(mNotificationId, mBuilder.build())
                    ;
            }

            //Se cancela el actual Timer y se crea uno nuevo, que se
                 ejecuta cada segundo y con un retardo de 4 segundos
            timer.cancel();
            crearTimer(4000, 1000);
        }
    } else {
        conectado = false;
        inicioFragment.mostrarDesconectar(estado);
        configuracionFragment.noConectado(estado);
    }
    sbTcpClient.interrupt();
    sbTcpClient = null;
}
```

Código A.8: Método estadoTimbre de la clase *MainActivity*

Como se puede observar en el **código A.8**, primero intenta conectar con el SB mediante *TCPClient*. Cabe destacar que se intenta conectar dos veces, para evitar que el SB rechace la conexión (en el caso en que el SB esté respondiendo consultas de otros dispositivos móviles).

Si se puede conectar con el SB, comprueba el estado del timbre, es decir, si llaman o no a la puerta. Si llaman a la puerta, recibirá el mensaje **Tocan** y agregará la notificación al historial con la información de fecha y hora de la notificación. Después, si la notificación inalámbrica está habilitada, crea una alerta auditiva y si la aplicación está abierta y no en segundo plano, muestra una alerta en una ventana emergente. También genera una notificación para la barra de notificaciones de Android.

Seguidamente, cancela el *Timer* actual y vuelve a crear uno nuevo con un retardo de 4 segundos y un periodo de 1 segundo. Esto se hace por si alguien llama a la puerta varias veces seguidas, proporcionando así un margen de 4 segundos para volver a consultar el estado del timbre.

Finalmente se cierra la conexión con el SB.

A.2.4 setSERVERIP() de *TCPClient*

```
/**
 * Modifica la IP del SB, al cual se conectará la aplicación.
 *
 * @param IP IP del SB
 */
public void setSERVERIP(String IP) {
    SERVERIP = IP;
    try {
        serverAddr = InetAddress.getByName(SERVERIP);
        estado = "OK";
    } catch (NetworkOnMainThreadException e) {
        Log.e("IP", "IP: Error", e);
        estado = "No es una IP valida";
    } catch (UnknownHostException e) {
        Log.e("Otro", "OTRO: Error", e);
        estado = e.toString();
    }
}
```

Código A.9: Método setSERVERIP de la clase *TCPClient*

En el **código A.9** se puede observar que este método sirve para cambiar la IP para la conexión con el SB. Comprueba si el valor ingresado es una IP valida, y si no fuese así, enviaría un mensaje **No es una IP valida**.

A.2.5 setTimeOut() de *TCPClient*

```
/**
 * Modifica el tiempo de espera de respuesta del SB.
 *
 * @param tiempo Tiempo de espera en milisegundos
 */
public void setTimeOut(int tiempo) {
    timeOut = tiempo;
}
```

Código A.10: Método setTimeOut de la clase *TCPClient*

Como se puede observar en el **código A.10**, este método es utilizado para establecer el tiempo de espera de la conexión con el SB. Si la conexión no es exitosa en este periodo de tiempo, se da por fallida dicha conexión.

Esto se debe a que la primera conexión con una IP no guardada en memoria puede necesitar un tiempo de espera más largo, que si se tratase de una IP ya conocida y almacenada en memoria.

A.2.6 run() de *TCPClient*

```
/**
 * Se ejecuta cuando se llama al método start.
 */
```

```
public void run() {
    try {

        //Crea umn socket para conectar con el SmartBell
        sockaddr = new InetSocketAddress(serverAddr, SERVERPORT);
        socket = new Socket();
        socket.connect(sockaddr, timeOut);
        try {

            //Out enviara los mensajes al SmartBell
            out = new PrintWriter(new BufferedWriter(new
                OutputStreamWriter(socket.getOutputStream())), true)
                ;

            //In recibe los mensajes del SmartBell
            in = new BufferedReader(new InputStreamReader(socket.
                getInputStream()));
            out.println(mensaje);
            serverMessage = in.readLine();
            estado = "OK";
        } catch (Exception e) {
            estado = e.toString();
            Log.e("TCP", "S: Error", e);
        } finally {

            //Siempre se debe cerrar el socket
            stopClient();
        }
    } catch (ConnectException e) {
        estado = "No se puede conectar con el SB, Compruebe la IP";
        Log.e("TCP", "No se puede conectar con el SB", e);
    } catch (IOException e) {
        estado = "No se puede conectar con el SB, Compruebe la IP";
        Log.e("TCP", "C: Error", e);
    }
}
```

Código A.11: Método run de la clase *TCPClient*

Este método es el más importante, puesto que es el que se ejecuta cuando se lo invoca, ya que esta clase se extiende de un *Thread* y por lo tanto se ejecutará en segundo plano. Como se puede ver **código A.11**, primero se crea la conexión mediante un *socket*, luego envía el mensaje y guarda la respuesta, para finalmente cerrar la conexión.

A.2.7 stopClient() de *TCPClient*

```
/**
 * Limpia y cierra el socket.
 */
public void stopClient() {
    out.flush();
    out.close();
    try {
        in.close();
    } catch (IOException e) {
```

```
            e.printStackTrace();
        }
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Código A.12: Método stopClient de la clase *TCPClient*

En el **código A.12** se puede observar que este método vacía y cierra el flujo de entrada y salida de información, para finalmente cerrar la conexión. Este método se utiliza para garantizar que la conexión esté cerrada, y así otros usuarios pueden conectarse al SB.

A.2.8 onCreateView() de *InicioFragment*

```
@Override
public View onCreateView(LayoutInflater inflater, final ViewGroup
    container, Bundle savedInstanceState) {

    //Elementos del Fragment
    View view = inflater.inflate(R.layout.fragment_inicio, container
        , false);
    IP = (EditText) view.findViewById(R.id.editTextIP);
    conectado = (TextView) view.findViewById(R.id.textViewConectado)
        ;
    botonConectar = (Button) view.findViewById(R.id.buttonConectar);

    //Se crea los objetos necesarios para la conexión con el
        SmartBell
    IPSB = MainActivity.getIPSB();

    //Si está conectado con el SB muestra la imagen de conectado.
        Caso contrario intenta conectarse
    if (MainActivity.isConectado()) {
        mostrarConectado();
    } else {

        //Si existe una IP guardada en memoria, intenta conectar con
            el SmartBell
        if (IPSB != "") {
            conectar(IPSB, 500);
        } else {
            mostrarDesconectar("Ninguna IP guardada en memoria");
        }
    }

    //Si se pulsa el botón conectar, intenta conectar con el
        SmartBell
    botonConectar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            IPSB = String.valueOf(IP.getText());
            conectar(IPSB, 5000);
```

```
        }
    });
    return view;
}
```

Código A.13: Método onCreateView de la clase *InicioFragment*

Como se puede ver en el **código A.13**, primero se hace referencia a los elementos que se ven en el *Fragment*. Después, mediante el *MainActivity*, se recupera la IP y se comprueba si se está conectado al SB.

Si el *MainActivity* indica que no está conectado al SB y existe una IP guardada en la memoria, se intenta conectar con el SB. En caso de que la conexión sea exitosa, se muestra el mensaje de **CONECTADO**. En caso contrario, muestra por pantalla el estado **DESCONECTADO** y habilita el campo de texto para introducir la dirección IP del SB, con el botón para intentar la conexión.

En este método también se declara la acción a realizar si se pulsa el botón *Conectar*. Dicha acción consiste en intentar conectar con la IP ingresada en el campo de texto.

A.2.9 conectar() de *InicioFragment*

```
/**
 * Intenta conectarse con el SB, con la IP que recibe como parámetro
     . También recibe como parámetro el tiempo de espera de
     respuesta. Si se tiene guardado en memoria una IP ese tiempo de
      espera es medio segundo. Si es la primera vez que se conectara
      con una IP, el tiempo de espera es de hasta 5 segundos.
 *
 * @param IPSB    IP del SB
 * @param timeOut Tiempo de espera de la conexión.
 */
private void conectar(String IPSB, int timeOut) {

    //Se modifica la IP y se comprueba si el formato de la IP es
        válida.
    sbTcpClient = new TCPClient();
    sbTcpClient.setSERVERIP(IPSB);
    sbTcpClient.setTimeOut(timeOut);
    estado = sbTcpClient.getEstado();

    //Si la IP tiene un formato correcto, intenta conectar con el
        SmartBell.
    if (estado == "OK") {
        sbTcpClient.start();
        while (sbTcpClient.isAlive()) {
        }
        estado = sbTcpClient.getEstado();
        if (estado == "OK") {
            MainActivity.setIPSB(IPSB);
            mostrarConectado();
            MainActivity.setConectado(true);
        } else {
            Toast.makeText(getContext(), estado, Toast.LENGTH_SHORT)
                .show();
        }
```

```
            sbTcpClient = null;
        } else {
            mostrarDesconectar(estado);
        }
}
```

Código A.14: Método conectar de la clase *InicioFragment*

Este método es el que se utiliza para intentar conectar con el SB. Se puede observar en el **código A.14**. Primero se asigna la IP a *TCPClient*, el cual comprueba si la IP tiene el formato correcto. Si es así, se modifica el estado a OK.

Si la IP tiene el formato correcto, intenta conectarse con el SB. Si la conexión es exitosa, llama al método **mostrarConectado()**. Finalmente modifica la variable *conectado* a TRUE del *MainActivity*. En caso contrario, llama al método **mostrarDesconectar()**.

A.2.10 mostrarDesconectar() de *InicioFragment*

```
/**
 * Muestra la pantalla de desconectado, donde aparece un campo para
     ingresar la IP y el botón de conectar. También se muestra un
     mensaje emergente, del motivo por el cual esta desconectado del
      SB.
 *
 * @param estadoActual Mensaje por el cual se está desconectado del
     SB
 */
public void mostrarDesconectar(String estadoActual) {
    conectado.setText("DESCONECTADO");
    conectado.setTextColor(Color.RED);
    IP.setVisibility(View.VISIBLE);
    botonConectar.setVisibility(View.VISIBLE);
    Toast.makeText(
        getContext(),
        estadoActual,
        Toast.LENGTH_SHORT).show();

}
```

Código A.15: Método mostrarDesconectar de la clase *InicioFragment*

En el **código A.15** se puede observar como este método recibe un mensaje por parámetro, el cual indica el motivo de la desconexión. Después se modifica el estado en pantalla a DESCONECTADO y hace visible el campo de texto y el botón para conectarse.

Los mensajes que puede mostrar mediante una notificación emergente son: **No se puede conectar con el SB. Compruebe la IP** y si la IP no tiene el formato correcto el mensaje seria **No es una IP valida**.

A.2.11 mostrarConectado() de *InicioFragment*

```
/**
 * Muestra la pantalla de conectado.
 */
```

```
private void mostrarConectado() {
    conectado.setText("CONECTADO");
    conectado.setTextColor(Color.GREEN);
    IP.setVisibility(View.INVISIBLE);
    botonConectar.setVisibility(View.INVISIBLE);
}
```

Código A.16: Método mostrarConectado de la clase *InicioFragment*

En el **código A.16** se puede observar como este método modifica el estado en pantalla a CONECTADO y oculta el campo de texto y el botón para conectarse.

A.2.12 onCreateView() de *HistorialFragment*

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState) {

    //Elementos del Fragment.
    View view = inflater.inflate(R.layout.fragment_historial,
        container, false);
    listView = (ListView) view.findViewById(R.id.lista);

    //Se lee el historial de memoria.
    items = MainActivity.getItems();

    //Crea Adapter y lo une con el listView.
    adapter = new ArrayAdapter<String>(getContext(), R.layout.
        list_item, items);
    listView.setAdapter(adapter);

    //Botón de borrar historial.
    botonBorrar = (FloatingActionButton) view.findViewById(R.id.
        boton_borrar);
    botonBorrar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            limpiarLista();
        }
    });
    listView.setSelection(listView.getCount() - 1);
    return view;
}
```

Código A.17: Método onCreateView de la clase *HistorialFragment*

Como se puede ver en el **código A.17**, primero se hace referencia a los elementos que se ven en el *Fragment*. Para después leer el historial de la memoria. También se debe crear un *adapter* y un *listView*, para poder ver por pantalla una lista con todo el historial y poder agregar o eliminar elementos.

Finalmente, también se crea el botón flotante de borrar historial, declarando la acción a realizar al hacer clic en dicho botón.

A.2.13 agregarLista() de *HistorialFragment*

```
/**
 * Agrega un elemento a la lista del historial, cuando alguien llama
      a la puerta.
 *
 * @param elemento Elemento a agregar al historial
 */
public void agregarLista(String elemento) {

    //Se agrega el elemento al historial.
    items.add(elemento);
    adapter = new ArrayAdapter<String>(getContext(), R.layout.
        list_item, items);

    //Une el adapter con el listView.
    listView.setAdapter(adapter);
    listView.setSelection(listView.getCount() - 1);
}
```

Código A.18: Método agregarLista de la clase *HistorialFragment*

En el **código A.18** se puede observar que este método agrega un elemento al historial y actualiza la vista del *Fragment*.

A.2.14 limpiarLista() de *HistorialFragment*

```
/**
 * Elimina todos los elementos del historial.
 */
private void limpiarLista() {

    //Se vacía el historial.
    MainActivity.limpiarItrems();
    items.clear();
    adapter = new ArrayAdapter<String>(getContext(), R.layout.
        list_item, items);

    //Une el adapter con el listView.
    listView.setAdapter(adapter);
}
```

Código A.19: Método limpiarLista de la clase *HistorialFragment*

Como se puede ver en el **código A.19**, este método limpia el historial de la memoria del dispositivo móvil. También limpia y actualiza la vista del *Fragment*.

A.2.15 onCreateView() de *ConfiguracionFragment*

```
@Override
public View onCreateView(LayoutInflater inflater, final ViewGroup
    container, Bundle savedInstanceState) {
```

```
    //Elementos del Fragment
    View view = inflater.inflate(R.layout.fragment_configuracion,
        container, false);
    audio = (Switch) view.findViewById(R.id.switchAudio);
    luz = (Switch) view.findViewById(R.id.switchLuz);
    movil = (Switch) view.findViewById(R.id.switchMovil);

    //Se crea los objetos necesarios para la conexión con el
        SmartBell
    IPSB = MainActivity.getIPSB();

    //Se declara las acciones a realizar si se cambia el estado de
        los Switches
    audio.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            if (audio.isChecked()) {
                cambiarConfig("K");
                configuracionActual();
            } else {
                cambiarConfig("F");
                configuracionActual();
            }
        }
    });
    luz.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            if (luz.isChecked()) {
                cambiarConfig("U");
                configuracionActual();
            } else {
                cambiarConfig("P");
                configuracionActual();
            }
        }
    });
    movil.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            MainActivity.setNotifMovil(movil.isChecked());
        }
    });

    //Si el MainActivity está conectado, consulta el estado actual
        de las notificaciones
    if (MainActivity.isConectado()) {
        configuracionActual();
    } else {
        audio.setEnabled(false);
        luz.setEnabled(false);
        movil.setEnabled(false);
    }
    return view;
}
```

Código A.20: Método onCreateView de la clase *ConfiguracionFragment*

Este es el método donde se crea la vista del *Fragment*. Como se puede ver en el **código A.20**, primero se declaran los elementos de la vista. Luego se crean los objetos necesarios para la conexión con el SB.

Si la *MainActivity* indica que está conectado al SB, consulta el estado de las notificaciones. En caso contrario, desactiva los *Switches* del *Fragment*.

También se declaran las acciones a realizar si se cambia la configuración de las notificaciones. En este punto, se puede observar cómo se mandan los mensajes de cambio de estado de una notificación centralizada (auditiva y luminosa), ya comentados en el **punto 3.2.1.2**. Por otro lado el estado de la notificación inalámbrica se cambia y se guarda en memoria mediante el método *setNotifMovil()*.

A.2.16 configuracionActual() de *ConfiguracionFragment*

```
/**
 * Consulta el estado de las notificaciones. Y los muestra en la
     aplicación.
 */
private void configuracionActual() {
    sbTcpClient = new TCPClient();
    sbTcpClient.setSERVERIP(IPSB);
    sbTcpClient.sendMessage(consulta);
    sbTcpClient.start();
    try {
        sbTcpClient.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    while (sbTcpClient.isAlive()) {
    }
    estado = sbTcpClient.getEstado();
    if (estado == "OK") {
        estadoSB = Integer.parseInt(sbTcpClient.getServerMessage());
        configuracionSB = (byte) estadoSB;
        bitSB = Integer.toBinaryString(configuracionSB);
        audio.setChecked(activoDesactivo(configuracionSB & 1));
        audio.setEnabled(true);
        luz.setChecked(activoDesactivo(configuracionSB & 2));
        luz.setEnabled(true);
        movil.setChecked(MainActivity.isNotifMovil());
        movil.setEnabled(true);
    } else {
        MainActivity.setConectado(false);
        noConectado(estado);
    }
    sbTcpClient = null;
}
```

Código A.21: Método configuracionActual de la clase *ConfiguracionFragment*

Este método se utiliza para saber el estado actual de las notificaciones del SB, enviando el mensaje **consulta** que vale **2** (que es el que se utiliza para consultar el estado de las notificaciones). En el **código A.21** se puede puede apreciar que primero se intenta conectar con el SB.

El SB responderá con un Byte denominado **estadoSB**. Como se comentó anteriormente en este Byte se incluirá el estado de todas las configuraciones centralizadas, siendo la posición de menor peso (0) el bit de más a la derecha y el de mayor peso (7) el más la izquierda. En el bit situado en la posición 0 está la configuración de la notificación auditiva, y en el 1 la notificación luminosa (**tabla A.1**). Finalmente, la notificación inalámbrica se lee de la memoria del SmartPhone.

Si no se recibe una respuesta exitosa del SB, cambia el estado de la conexión a FALSE del *MainActivity* y llama al método **noConectado()**.

A.2.17 activoDesactivo() de *ConfiguracionFragment*

```
/**
 * Devuelve falso si recibe un 0, caso contrario verdadero. Para
     valorar el estado de las notificaciones, a partir de la lectura
      de un bit del byte que envía el SB.
 *
 * @param est Valor de un bit, del Byte enviado por el SB
 * @return Falso, si vale 0. Verdadero, caso contrario
 */
private boolean activoDesactivo(int est) {
    if (est == 0) return false;
    else return true;
}
```

Código A.22: Método activoDesactivo de la clase *ConfiguracionFragment*

En el **código A.22** se puede observar que este método recibe un número y devuelve un booleano, comprueba si el número es un 0 o no. Si lo es, devuelve FALSE, en caso contrario devuelve TRUE. Este método se utiliza para que a partir del valor de un bit saber el estado de la notificación auditiva o luminosa.

A.2.18 cambiarConfig() de *ConfiguracionFragment*

```
/**
 * Modifica el estado de una notificación, a partir del mensaje que
     se envía al SB.
 *
 * @param mensaje Mensaje para modificar una notificación
 */
private void cambiarConfig(String mensaje) {
    sbTcpClient = new TCPClient();
    sbTcpClient.setSERVERIP(IPSB);
    sbTcpClient.sendMessage(mensaje);
    sbTcpClient.start();
    try {
        sbTcpClient.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    while (sbTcpClient.isAlive()) {
    }
    estado = sbTcpClient.getEstado();
```

```
    if (estado != "OK") {
        MainActivity.setConectado(false);
        noConectado(estado);
    }
    sbTcpClient = null;
}
```

Código A.23: Método cambiarConfig de la clase *ConfiguracionFragment*

Como se puede observar en el **código A.23**, este método recibe un *String* con el mensaje de configuración a enviar al SB, intenta conectar con el SB y se envía el mensaje de configuración. Si la operación falla, cambia el estado de la conexión a FALSE del *MainActivity* y llama al método **noConectado()**.

A.2.19 noConectado() de *ConfiguracionFragment*

```
/**
 * Si no existe o se rompe la conexión con el SB, se deshabilitan
     los elementos de la pestaña. Y se muestra un mensaje emergente
     con el motivo de la desconexión.
 *
 * @param mensajeError Motivo de la desconexión
 */
public void noConectado(String mensajeError) {
    audio.setEnabled(false);
    luz.setEnabled(false);
    movil.setEnabled(false);
    Toast.makeText(getContext(),
            mensajeError, Toast.LENGTH_SHORT).show();
}
```

Código A.24: Método noConectado de la clase *ConfiguracionFragment*

En el **código A.24** se puede observar que este método recibe como parámetro un mensaje, donde se indica el motivo por el que la conexión ha fallado. También desactiva los elementos del *Fragment* y muestra un mensaje emergente.

A.2.20 AndroidManifest.xml

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

    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="SmartBell"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
```

```
            android:name=".MainActivity"
            android:label="SmartBell"
            android:screenOrientation="portrait"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Código A.25: Contenido del *AndroidManifest.xml*

Se puede observar este fichero en el **código A.25**. En él aparecen los permisos que se le debe dar a la aplicación. En este caso, se proporcionará el siguiente permiso: **INTERNET**, el cual permite el acceso a Internet para la comunicación y el intercambio de información con el SB. También se asigna el icono y el nombre de la aplicación.

BIBLIOGRAFÍA

[1] Arduino CC, "Arduino UNO Rev3," 2014. [Online]. Available: https://store.arduino.cc/arduino-uno-rev3 (document), 3, 3.1

[2] Arduino y solo Arduino and M. Domínguez, "Arduino UNO esquemático." [Online]. Available: https://soloarduino.blogspot.com.es/2014/ (document), 3.2

[3] Blogspot, "Puente de diodos." [Online]. Available: http://3.bp.blogspot.com/-_hfkhrfbGjU/T8lYDzSvfzI/AAAAAAAAAJ0/V86uzpmMmXs/s1600/Puente+de+diodos+simbolo.png (document), 3.3

[4] RS, "WiFly RN-171 | Microchip." [Online]. Available: http://es.rs-online.com/web/p/modulos-wlan/7694144/ (document), 3.23, 3.2.1

[5] Cdn-reichelt, "ARDUINO WIRELESS SD SHIELD." [Online]. Available: https://cdn-reichelt.de/bilder/web/xxl_ws/A300/FRANZIS_65188-10.png (document), 3.24

[6] ISM Solar and I. Martínez Marchena, "La guía definitiva de comunicaciones Wifi con el Arduino Wireless SD Shield y WiFly RN-XV." [Online]. Available: http://www.ismsolar.com/blog/la-guia-definitiva-de-comunicaciones-wifi-con-el-arduino-wireless-sd-shield-y-wifly-rn-xv (document), 3.25

[7] Hi-Link, "Fuente de poder HLK-PM01." [Online]. Available: http://www.hlktech.net/product_detail.php?ProId=54 (document), 3.15

[8] ON Semiconductor, "MC14049B, MC14050B Hex Buffer." [Online]. Available: https://www.onsemi.com/pub/Collateral/MC14049B-D.PDF (document), 3.17, 3.2

[9] Digi-Key Electronics, "Buzzer AST-030c0mr-r." [Online]. Available: https://www.digikey.es/product-detail/es/pui-audio-inc/AST-030C0MR-R/668-1138-ND/1464877 (document), 3.20

[10] Arduino y solo Arduino and Protoboard, "Módulo Relé 5V DC - Keyes KY019 - Parte 1." [Online]. Available: https://soloarduino.blogspot.com.es/2014/01/modulo-keyes-rele-sr1y.html (document), 3.21

[11] Roving Networks, "RN-171-XV 802.11 b/g Wireless LAN Module," 2012. [Online]. Available: http://docs-europe.electrocomponents.com/webdocs/1385/0900766b81385c4d.pdf (document), 3.3

[12] Ring, "Ring Video Doorbell for Your Smartphone." [Online]. Available: https://ring.com/ (document), 1.1.1, 1.1

[13] Indiegogo, "Nucli Smart Lock." [Online]. Available: https://www.indiegogo.com/projects/the-westinghouse-nucli-smart-lock/ (document), 1.1.1, 1.2

[14] Wikipedia, "Raspberry Pi." [Online]. Available: https://es.wikipedia.org/wiki/Raspberry_Pi 3.1

[15] Arduino CC, "Arduino Mega," 2013. [Online]. Available: https://store.arduino.cc/arduino-mega-2560-rev3 1

[16] ——, "Arduino Leonardo," pp. 1–5, 2014. [Online]. Available: http://arduino.cc/en/Main/arduinoBoardLeonardo 2

[17] J. Penalva and Xataka, "NFC: qué es, cómo funciona en el móvil y para qué sirve," 2011. [Online]. Available: https://www.xataka.com/moviles/nfc-que-es-y-para-que-sirve 3.2

[18] U. Méndez and 330ohms, "Bluetooth. Clases y versiones desde v1.0 hasta v5.0." [Online]. Available: https://www.330ohms.com/blogs/blog/que-es-el-bluetooth 3.2

[19] CCM, "Introducción a Wi-Fi (802.11 o WiFi)." [Online]. Available: http://es.ccm.net/contents/789-introduccion-a-wi-fi-802-11-o-wifi 3.2

[20] Arduino CC, "Arduino Software (IDE)." [Online]. Available: https://www.arduino.cc/en/Guide/Environment 3.1.1.3

[21] A. ecda and El cajón de Arduino, "Haciendo Sonidos con Arduino." [Online]. Available: http://elcajondeardu.blogspot.com.es/2014/01/tutorial-haciendo-sonidos-con-ardu.html (document), 26, A.1.2

[22] Android Developers, "Actividades." [Online]. Available: https://developer.android.com/guide/components/activities.html 4.1.1

[23] ——, "Fragmentos." [Online]. Available: https://developer.android.com/guide/components/fragments.html 4.1.1

[24] ——, "Manifiesto de la app." [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro.html 4.1.1