UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

"DISEÑO E IMPLEMENTACIÓN DE UNA APP DE MONITORIZACIÓN DE TAQUILLAS A TRAVÉS DEL MÓDULO ESP32 Y LAS TECNOLOGÍAS WIFI, BLE Y RFID"

TRABAJO FIN DE GRADO

Junio - 2023

AUTOR: José Antonio Sánchez Sánchez

DIRECTOR: Miguel Onofre Martínez Rach

RESUMEN

En la actualidad, *Internet de las cosas (IoT)* ha generado un gran impacto significativo en la sociedad debido al aumento de posibilidades destinadas a la facilitación y mejora de la vida de las personas y de las grandes empresas. De igual modo, IoT ofrece mejoras en diferentes ámbitos tales como el mundo de la salud, industria, agricultura, entre otros.

Atendiendo a las numerosas ventajas que ofrece, cabe destacar el beneficio que IoT proporciona en relación con el comercio en línea. En los últimos años, el comercio electrónico ha cobrado un gran protagonismo, especialmente en respuesta a la pandemia mundial de 2019. A raíz de este problema, las empresas de repartición de pedidos han comenzado a utilizar tecnologías como IoT. Una de las mejoras impartidas por muchas empresas son las taquillas inteligentes o *Pack Home* las cuales permiten tener el control total de los productos almacenados en su interior obteniendo así numerosos beneficios tanto para la empresa como para el consumidor.

El objetivo principal de este proyecto es la monitorización de un prototipo compuesto por dos taquillas a través de tecnologías como Wifi, bluetooth de baja energía (BLE) e identificación por radiofrecuencia (RFID). Esto se logra mediante la creación de una aplicación Android que se encarga de gestionar y controlar dicho prototipo. Para ello, se ha utilizado un módulo ESP32 que permite establecer la conexión entre la aplicación Android y las tecnologías mencionadas. El desarrollo de la aplicación se ha llevado a cabo mediante el programa Android Studio, haciendo uso del lenguaje de programación JAVA.

Palabras clave: IoT, Taquilla inteligente, ESP32, Wifi, BLE, RFID.

ABSTRACT

Currently, Internet of Things (IoT) has generated a significant impact on society due to the increase in possibilities aimed at facilitating and improving the lives of individuals and large companies. Similarly, IoT offers improvements in various fields such as healthcare, industry, agriculture, among others.

Given the numerous advantages it offers, it is worth highlighting the benefit that the Internet of Things (IoT) provides in relation to online commerce. In recent years, e-commerce has gained great prominence, especially in response to the global pandemic of 2019. As a result of this problem, delivery companies have started to use technologies such as the Internet of Things. One of the improvements implemented by many companies are intelligent lockers or Pack Home, which allow total control of the products stored inside, obtaining numerous benefits for both the company and the consumer.

The main objective of this project is the monitoring of a prototype composed of two lockers using technologies such as Wifi, Bluetooth low energy (BLE) and radio frecuency identification (RFID). This is achieved by creating an Android application that manages and controls the said prototype. To accomplish this, an ESP32 module has been used to establish the connection between the Android application and the mentioned technologies. The application development has been carried out using Android Studio program, using the java programming language.

Keywords: IoT, Smart locker, ESP32, Wifi, BLE, RFID.

AGRADECIMIENTOS

Este proyecto de fin de grado es el fruto de varios meses de dedicación y aprendizaje. Por ello, me gustaría agradecer a varias personas que sin ellas, nada de esto habría sido posible.

En primer lugar, agradecer a todos los profesores del Grado en Ingeniería de Tecnologías de Telecomunicación por los conocimientos transmitidos, sin ellos no habría sido posible realizar este proyecto. En especial a mi tutor, Miguel Ángel Martínez Rach, por haberme brindado la oportunidad de realizar este proyecto con él y por su orientación y disponibilidad siempre que la he necesitado.

También, me gustaría agradecer a mis compañeros del grado, con los que he compartido largas horas de estudio y he formado una amistad para toda la vida.

Por último, agradecer a mi familia, por el esfuerzo y sacrificio que han realizado para ayudarme en toda esta etapa. También agradecer a mi pareja, que ha estado apoyándome en todo momento. Estoy enormemente agradecido de teneros a mi lado.

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Internet de las cosas (IoT) es una infraestructura global que permite la ejecución de servicios avanzados mediante la interconexión física y/o virtual de cosas mediante tecnologías de la información y comunicación que existen actualmente. Hace referencia a cualquier dispositivo físico (Televisión, instrumentos, vehículos…) susceptible a conectarse a internet a través de la integración con software, circuitos, sensores etc permitiendo el intercambio de datos entre ambos.

IoT ha supuesto una mejora en la calidad de vida de las personas en los últimos años proporcionando hoy en día la interconectividad de múltiples dispositivos electrónicos. A raíz de la pandemia de COVID-19, IoT ha podido obtener un papel destacable. Gracias a ello, el trabajo, la formación, las relaciones sociales, las formas de entretenimiento etc. han podido realizarse de manera remota a través de los distintos dispositivos electrónicos. Sin embargo, uno de los sectores que más crecimiento ha experimentado ha sido el comercio online. La evolución de la venta por internet ha crecido exponencialmente y continúa haciéndolo año tras año.

Los nuevos avances tecnológicos en los últimos años han supuesto que las grandes empresas se adapten a ellos con el fin de ofrecer mejores servicios. Entre ellas se encuentran las empresas de reparto de pedidos. Como consecuencia del auge de las compras por internet y los cambios de vida de la sociedad actual, la cual nos aleja de la vivienda durante horas, se refleja uno de los principales problemas asociados a la compra online siendo la recepción de las compras en el domicilio. La ausencia de los usuarios a la hora de la recepción provoca costos extras que no asume el comprador.

A raíz de este problema, surgió la idea de *PackHome,* que consiste en taquillas o casilleros inteligentes controlados a través de un smartphone, con el objetivo de facilitar la entrega de paquetes a pesar de no haber nadie para recogerlo. Además, existen varias alternativas para la apertura de estas taquillas en caso de no disponer de un teléfono móvil, por ejemplo, a través de códigos numéricos, lectores de código QR y RFID, etc.

2. OBJETIVOS

El principal objetivo de este proyecto es la monitorización y control de varias taquillas electrónicas mediante el uso de un dispositivo móvil, ya sea a través de las tecnologías Bluetooth de baja energía (BLE) o Wifi, llevándose a cabo gracias a la creación de una aplicación Android, desarrollada a partir del lenguaje Java. A través de dicha aplicación podremos controlar tantas taquillas como sean necesarias. También contará con la posibilidad de poder abrir la taquilla a través de la tecnología RFID, en el caso de no disponer de un teléfono móvil.

Como objetivos secundarios se incluyen:

- Utilización de la herramienta Android Studio para la creación de la aplicación.
- Programación del diseño de la aplicación a través del lenguaje JAVA.
- Programación del módulo ESP32 a través del entorno de desarrollo Arduido IDE.
- Integración de los dispositivos hardware para el desarrollo del proyecto.

3. ESTRUCTURA DE LA MEMORIA

En este apartado se definen los diferentes capítulos donde se expone la realización del proyecto.

- Introducción: Se detalla la definición y descripción del proyecto y sus objetivos principales y secundarios.
- Estado del arte: Se desarrollan las tecnologías y proyectos similares existentes en la actualidad.
- Arquitectura del proyecto: Se detallan los materiales a utilizar, así como sus elementos hardware y software.
- Análisis del proyecto: Se especifican los requisitos y los casos de uso del proyecto.
- Implementación: Se explica el desarrollo y la metodología del proyecto.
- Resultados: Se analiza el cumplimiento de los objetivos del proyecto.
- Conclusiones y líneas futuras: Se revisan y recopilan las ideas y resultados obtenidos en el proyecto, así como posibles futuras mejoras.
- Bibliografía: Se citan las referencias utilizadas para el desarrollo del proyecto.

4. ESTADO DEL ARTE

En este apartado se pretende realizar un estudio sobre el uso de taquillas inteligentes en España. Se analizará el funcionamiento y las tecnologías que se encuentran actualmente disponibles para el desarrollo, tanto de las taquillas como el de la aplicación Android empleada.

4.1. Estado Actual

Tras la pandemia, el auge de las compras por internet ha aumentado de manera notable, esto junto a la falta de tiempo, la diversificación de horarios y por la ausencia de usuarios a la hora de entregar los pedidos en sus casas, ha provocado la aparición de nuevas empresas que combinan el uso de las taquillas con los smartphones. Estas empresas se encargan de colocar taquillas inteligentes en sitios estratégicos de paso con el fin de cubrir toda la ciudad. Las taquillas funcionan a modo de buzón, facilitando la entrega de paquetes en puntos de recogida con el objetivo de poder entregar el paquete, aunque no esté el propietario. Estas empresas han desarrollado estas tecnologías tanto para facilitar la entrega de los paquetes como para evitar la dependencia por parte del comprador de estar en casa en el momento de la entrega.

Ilustración 1. Amazonlocker

Gracias a la evolución de la electrónica embebida hoy en día existen varias alternativas de desbloqueo de cerraduras. Los más empleados en la actualidad son los lectores de código QR, los códigos numéricos y los lectores RFID. Éstos últimos permiten un mayor control de las taquillas ya que permiten abrirla y cerrarla a distancia, enviar datos

mediante conexión Wireless y realizar bloqueos a partir de teléfonos móviles, tarjetas, pulseras, llaveros, etc.

También existen cerraduras que emplean lectores de huella dactilar con el fin de aumentar la seguridad y cerraduras controladas a través de las tecnologías WIFI y bluetooth.

4.2. Cerraduras en la actualidad

4.2.1. Cerraduras de código numérico

Este tipo de cerraduras permiten ser programadas con el fin de crear un código de manera aleatoria cada vez que se utilicen, de esta forma impide que los usuarios que recuerden la contraseña tengan de nuevo acceso a estas.

La mayoría de las empresas que emplean este tipo de cerraduras envían un SMS al usuario con el código de desbloqueo de su correspondiente taquilla.

Ilustración 2. Cerradura de código numérico

4.2.2. Cerraduras mediante código electrónico QR

Este tipo de cerraduras contienen un lector de códigos QR. Estos códigos son combinaciones de barras y cuadros que acompañan a un producto o a una unidad de consumo para que pueda ser leído y descifrado a través de un lector. Además, permiten

almacenar códigos numéricos, alfabéticos, símbolos, datos binarios, etc. Es la evolución de los códigos de barras.

La principal ventaja de este tipo de sistema es que se trata de una tecnología de bajo coste, segura y muy fiable, ya que, aunque el código o el lector estuvieran sucios o rotos, podría seguir funcionando perfectamente ya que tan solo leyendo un 30% del código, la información se puede leer de forma precisa.

Suelen emplearse en lugares como estaciones de tren, metro, aeropuertos, etc donde se utilizan como tarjetas de embarque en la cual un lector se encarga de identificarlo.

Ilustración 3. Cerradura con código QR

4.2.3. Cerraduras con lectores RFID

Estos dispositivos funcionan mediante la identificación por radiofrecuencias. Para su funcionamiento se requiere un lector, que se encarga de reconocer e identificar la señal y un aparato que realice la función de generador de señales, ya sea un dispositivo móvil, un reloj inteligente u otro elemento que realice la misma función. Estos aparatos pueden emplear distintas tecnologías como Wireless o NFC, explicadas con detalle más adelante.

Ilustración 4. Cerradura con lectura RFID

4.2.4. Lector de huella dactilar

Las cerraduras con huella dactilar son cada vez más utilizadas debido a su facilidad de uso, a la alta seguridad que proporciona y a su precio económico. Estos lectores realizan una imagen de la huella dactilar y la compara con las que se encuentran almacenadas, teniendo en cuenta el patrón, formas, crestas y valles.

Existen dos tipos de lectores, los ópticos y los de capacitancia:

- Los lectores ópticos emplean luces LED que iluminan cada zona creando superficies iluminadas y otras oscuras de las crestas y los valles de la huella con el fin de obtener la imagen de esta.
- Los lectores de capacitancia realizan mediciones de voltaje en diferentes zonas del soporte de lectura aprovechando la conductividad de la piel.

Ilustración 5. Lector huella digital

4.3.Uso de taquillas inteligentes en la actualidad

El aumento de las compras por internet ha provocado la aparición de diferentes empresas cuya función es la distribución de taquillas inteligentes a modo de buzón. Algunas de ellas son:

4.3.1 Citibox

Citibox se controla a través de su propia aplicación móvil. Es una aplicación muy intuitiva ya que en ella se encuentra las instrucciones necesarias del proceso a seguir a la hora de recoger un pedido. En el momento que el repartidor deposita el paquete en una de las taquillas de esta empresa, el usuario recibe un código numérico con el cual poder desbloquear la taquilla en el momento de la recogida del pedido.

Ilustración 6. Empresa Citibox

4.3.2. Mayordomo

Mayordomo es una empresa que se encarga de la instalación de taquillas inteligentes a modo de buzón solamente en oficinas. La apertura de las taquillas se lleva a cabo a través de un código y presentan una forma de pago mediante tarjeta de crédito, previamente registrada en su aplicación.

4.3.3. Pudo

Pudo se gestiona a través de una página web debido a que no disponen de aplicación móvil. A pesar de ello, el mecanismo de la apertura de la taquilla es el mismo que en las anteriores empresas, es decir, a partir de un código numérico.

El principal inconveniente de esta empresa es la forma de controlar las taquillas ya que la gestión se realiza a partir de una pantalla central. Este sistema puede generar colas en el momento que varias personas coincidan a la hora de recoger el pedido no siendo lo más conveniente.

Ilustración 7. Empresa Pudo

En la actualidad existen muchas formas de abrir taquillas electrónicas, en muchos centros de deporte y gimnasios se utilizan pulseras, tarjetas electrónicas o llaveros para abrirlas. Sin embargo, estos sitios los usuarios que tienen acceso a estas taquillas son socios y las usan frecuentemente. El principal objetivo de este proyecto es satisfacer las necesidades tanto de los usuarios frecuentes como de los que no lo son.

Este problema demuestra la necesidad de la creación de una aplicación debido a que un 80% de la población tiene a su alcance un teléfono móvil.

5. ARQUITECTURA DEL PROYECTO

5.1. Arquitectura Hardware

5.1.1. ESP32

ESP32 es un SoC creado por la compañía china *Espressif Systems* y fabricado por TSMC. Gracias a su capacidad de conexión tanto de Bluetooth como de Wifi a un bajo coste energético y económico, está considerado como uno de los mejores microcontroladores del mercado para su uso en IoT. Se trata de una solución para microcontroladores que no dispongan de conectividad ya que los diferentes modelos de ESP32 pueden ser un puente de acceso a la red o soluciones IoT. Gracias a la gran potencia y rendimiento de radiofrecuencia, mostrando robustez, versatilidad y fiabilidad en una amplia variedad de aplicaciones y escenarios de potencia, el ESP32 es un componente esencial en este proyecto.

Este nuevo SoC es una evolución del ESP8266, fabricado por la misma empresa la cual ha añadido nuevas funcionalidades y mejora del rendimiento general. Algunas de sus diferencias más destacadas son la adición de un núcleo, una conexión Wifi más rápida, mayor número de pines de entrada/salida y una compatibilidad con Bluetooth 4.2 y Bluetooth de baja energía. A pesar de ello, su función principal continúa siendo la misma, procesar la información que emiten los diferentes sensores conectados a él y transmitirla hacia los mismos, además de enviarla mediante Wifi a los diferentes servidores Web que se encargan de almacenarla.

Atendiendo a las ventajas que ofrece el ESP32, este módulo destaca por su precio competitivo, su bajo consumo de energía, comunicación inalámbrica Wifi y Bluetooth, la presencia de numerosos canales de comunicaciones (Ethernet, USB, micro USB) y la proporcionalidad de una seguridad criptográfica por hardware.
La estructura del ESP32 se compone de chips, módulos y placas de desarrollo. Atendiendo a los chips se encuentran 4 tipos diferentes:

Ordering code	(D) Core	(0/2) Embedded flash	(WD)Connection	(Q6)Package
ESP32-DOWDQ6	Dual Core	No embedded flash	Wi-Fi b/g/n + BT/BLE Dual Mode	QFN 6*6
ESP32-D0WD	Dual Core	No embedded flash	Wi-Fi b/g/n + BT/BLE Dual Mode	QFN 5*5
ESP32-D2WD	Dual Core	16-Mbit embedded flash (40 MHz)	Wi-Fi b/g/n + BT/BLE Dual Mode	QFN 5*5
ESP32-S0WD	Dual Core	No embedded flash	Wi-Fi b/g/n + BT/BLE Dual Mode	QFN 5*5

Tabla 1. Chips de ESP32

Por otro lado, *Espressif Systems* ha desarrollado diferentes módulos donde cada uno de ellos contiene un chip integrado, un cristal de 40 Mhz, una memoria flash y una antena dependiendo del modelo. En la tabla 2 se encuentran los diferentes módulos desarrollados:

Módulo	Chip	Flash	RAM	Antena
ESP32-WROOM-32	ESP32-DOWDQ6	4 MB		MIFA
ESP32-WROOM-32D	ESP32-D0WD	4 MB		MIFA
ESP32-WROOM-32U	ESP32-D0WD	4 MB		U.FL
ESP32-SOLO-1	ESP32-SOWD	4 MB		MIFA
ESP32-WROVER	ESP32-DOWDQ6	4 MB	4 MB	MIFA
ESP32WROVER-I	ESP32-DOWDQ6	4 MB	4 MB	U.FL

Tabla 2. Módulos de ESP32

Por último, el ESP32 se monta en multitud de placas de desarrollo compatibles con la IDE de Arduino, facilitando su programación. La tarjeta de desarrollo suele integrar al módulo en un PCB permitiendo la conexión serie/USB, alimentación USB, botones de *booty resety* y pines soldados a la placa.

PINEADO ESP32

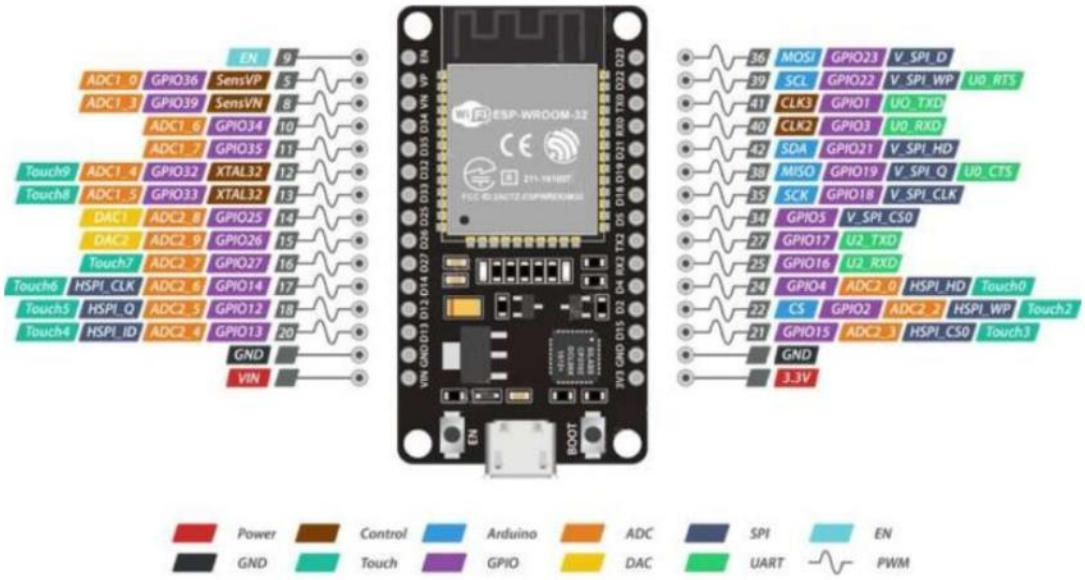

Ilustración 8. Pineado ESP32

En la presente tarjeta de desarrollo, es importante prestar especial atención a ciertos pines, cuya descripción se detalla a continuación.

- Pines para el ADC: No todos los pines están conectados a las ADC internos. Únicamente los siguientes pines pueden usarse: 0, 2, 4, 12, 13, 14, 15, 25, 26, 27, 32, 33, 34, 35, 36 y 39.
- Pines GPIO6 hasta GPIO11: Estos pines están conectados al flash SPI interna del chip ESP-WROOM-32, por lo que no se aconseje su uso, ya que su utilización podría fallar al cargar el código en el dispositivo.
- Pines GPIO21 y GPIO22: Estos pines están reservados para la comunicación I2C.
- Pines GPIO5(CS), GPIO18(CLK), GPIO19(MISO), GPIO23(MOSI): Estos pines son los predeterminados para la comunicación mediante SPI.
- Pines GPIO25 y GPIO26: Estos pines pueden emplearse para el DAC.
- Pines GPIO34 y GPIO39: Estos pines solamente permiten la entrada de señales y no pueden generar pulsos PWM ni actuar como salidas GPIO.

CARACTERÍSTICAS ESP32

Procesador

ESP32 contiene uno o dos microprocesadores Xtensa LX6 de 32 bits de bajo consumo cuyas características son:

- Solución *UltraLowPower*: El ESP32 cuenta con dos microprocesadores de bajo consumo Tensilica Xtensa de 32 bits LX6. Además, cuenta con un co-procesador de ultra bajo consumo que es utilizado para realizar conversiones analógico-digital y otras operaciones mientras el dispositivo se encuentra funcionando en el bajo consumo permitiendo un consumo muy bajo del SoC. La salida del amplificador de potencia también es ajustable lo que supone una mejora en la comunicación, velocidad de datos y consumo de energía.

- Solución de integración completa: ESP32 es una solución altamente integrada para aplicaciones IoT de Wi-Fi y Bluetooth, con alrededor de 20 componentes externos. Dentro de sus componentes esenciales se encuentra un interruptor de antena, un balun RF, un amplificador de potencia y de recepción de bajo ruido, filtros y módulos de administración de energía totalmente integrados en el mismo chip.

- Admite modo dual Wi-Fi y Bluetooth: Este módulo integra Wi-fi y Bluetooth. Mientras la conectividad Wi-fi permite una amplia gama de conexiones de comunicación y conexión directa a Internet a través de un enrutador, Bluetooth permite a los usuarios conectarse a los teléfonos móviles o transmitir baliza Bluetooth LE para facilitar la detección de señales.

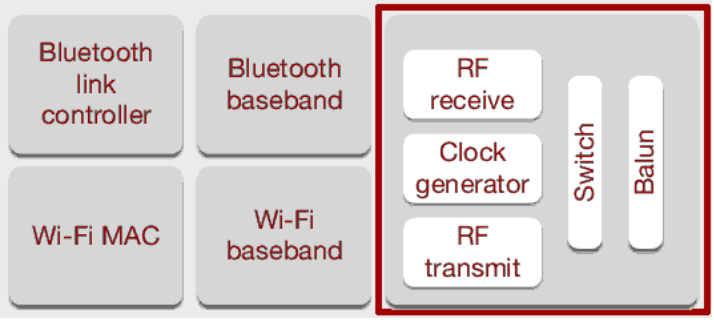

Ilustración 9. Componentes ESP32

El ESP32 hace de interfaz entre la aplicación Android y el usuario, siendo uno de los elementos más importantes de este proyecto. A pesar de la gran variedad de modelos existentes del ESP32, en este proyecto se ha escogido el modelo WROOM-32.

Memoria

El ESP32 contiene una gran cantidad de memorias internas:

- Memoria SRAM de 520KiB. Se utiliza para datos e instrucciones.
- Memoria ROM de 448 KiB. Se emplea para funciones de núcleo y boot.
- Memoria RTC de 8 KiB. Existen dos tipos: Fast SRAM empleada en el almacenamiento de datos y Slow SRAM utilizada por el coprocesador para acceder en el momento depp-sleep.
- Memoria flash integrada de 0 a 4 MiB, dependiendo del tamaño.

Timers

Contiene cuatro temporizadores integrados de 64 bits. Se pueden configurar de forma ascendente o descendente y pueden utilizarse como interrupción por flanco o nivel.

Watchdog

En cada núcleo se encuentra un *Main Watchdog Timer* (MWDT) y en el RTC un RTC *Watchdog Timer* (RWDT). Se emplean en caso de un fallo imprevisto con el fin de recuperar al dispositivo.

Relojes del sistema

El módulo ESP32 cuenta con varios tipos de relojes integrados. En la CPU contiene un reloj cuya fuente es por defecto un reloj de cristal de 40Mhz que se conecta a un PLL para generar una frecuencia de 160 Mhz. Otro tipo de reloj es el de tiempo real (RTC) cuya fuente puede ser un oscilador RC de 150Khz o de 8Mhz.

Radiofrecuencia

El dispositivo contiene los siguientes bloques de radiofrecuencia:

- Un receptor de 2.4 GHz que se encarga de demodular la señal de radiofrecuencia al dominio digital a través de dos ADCs de alta resolución. El chip contiene filtros

de RF, control automático de ganancia, cancelación de continua y filtros de banda base con el fin de adaptarse a las distintas condiciones que presenta la señal.

- Un transmisor de 2.4 GHz que modula la señal que se encuentra en banda base a señal RF a 2.4 GHz, saliendo después por la antena con la ayuda de un amplificador de potencia basado en CMOS. Permite anular las imperfecciones generadas por el ruido a partir de calibraciones integradas.

- Un generador de reloj que genera una señal cuadrada de 2.4 GHz para transmisión y recepción de WIFI con los componentes integrados y calibración automática.

Seguridad

En el caso de conexión vía Wifi, el módulo permite estándares de seguridad de 802.11 como WPA/WPA2, WAPI y WFA. También permite encriptar la memoria flash del módulo y aceleración criptográfica hardware como AES (*Advanced Encryption Stardard*), SHA-2 (*Secure Hash Algorithm* 2), RSA, ECC (*Elliptic Curve Cryptography*) y RNG (*Random Number Generator*).

Modos de operación

Este SoC permite estar en diferentes modelos de operación dependiendo de la energía empleada y pueden ser:

- Activo: el chip permite recibir, escuchar y transmitir, es decir, se encuentra todo encendido.
- *Modem-sleep*: el WiFi y el bluetooth se encuentran inhabilitados. En cambio, la CPU está operativa y el reloj es configurable.

- *Light-sleep*: La CPU solamente se encuentra habilitada ante un evento que lo necesite. La memoria y los periféricos RTC están habilitados junto al coprocesador ULP de bajo consumo.

- *Deep-sleep*: Solamente se encuentran habilitados la memoria y los periféricos RTC. Los datos de conexión ya sean de Wifi o bluetooth se registran en la memoria RTC.

- Hibernación: El RTC Timer o un evento creado por los pines GPIO pueden habilitar al chip que se encuentra en modo hibernación.

MODO BLE

El dispositivo utilizado en este proyecto dispone de comunicaciones inalámbricas vía Bluetooth. Existen dos estándares, el Bluetooth clásico y el Bluetooth de bajo consumo. En este proyecto se utiliza el modo BLE.

Bluetooth Low Energy, creado por Bluetooth *Special Interest Group*, es una tecnología diseñada para la transmisión de pequeñas cantidades de datos con un modo de consumo de energía ultra-bajo de manera que garantiza un menor tiempo de establecimiento de conexión.

Atendiendo a su arquitectura, el módulo BLE contiene una pila de protocolos en referencia a la capa OSI orientada a conexiones sencillas, es decir, a dispositivos dependientes de batería o pila. Dicha pila se divide en tres partes básicas: Aplicación, Host y Controlador.

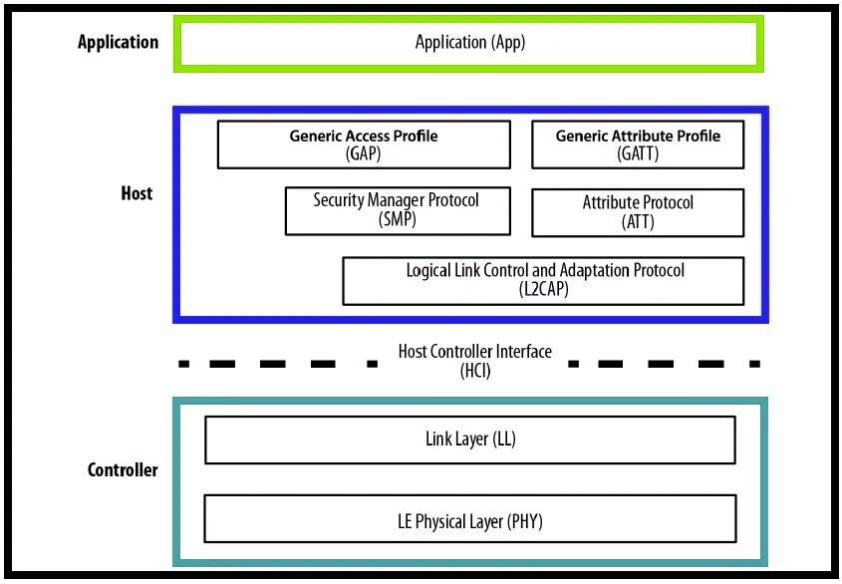

Ilustración 10. Capas del módulo BLE.

Capa de aplicación

La capa de aplicación se encuentra encima de la pila y contiene la interfaz de usuario, la lógica y la estructura general de la aplicación desarrollada para dispositivos compatibles con BLE.

Capa Host

La capa host es la que se encarga de administrar cómo dos o más dispositivos se comunican entre ellos. Está compuesta por varias subcapas:

1. Capa de Protocolo de Control y Adaptación del Enlace Lógico (L2CAP)

Es la encargada de transmitir paquetes con o sin orientación a la conexión a sus capas superiores. Para ello multiplexa los protocolos de las capas superiores y los encapsula en el formato estándar del paquete de BLE. Por lo tanto, los paquetes de datos provenientes de la aplicación son enviados a través de paquetes L2CAP. Otra de sus funciones consiste en la fragmentación y recombinación, es decir, los paquetes de la aplicación que suponen datagramas de gran cantidad de bytes son fragmentados correctamente adaptándose a la unidad máxima de transferencia de BLE (27 bytes).

Por otro lado, esta capa se encarga de dos protocolos superiores: El protocolo de Atributos (ATT) y el protocolo de Gestión de seguridad (SMP).

2. Protocolo de Atributo (ATT)

Este protocolo tiene como función definir como un servidor expone sus datos a un cliente y como se estructuran esos datos dentro del servidor. Los datos se estructuran como atributos y cada atributo puede adquirir dos roles, cliente, servidor o ambos. Por una parte, el rol de servidor se encarga de recibir peticiones desde el cliente y enviar respuestas. Además, almacena todos los datos, organizados en atributos. Por último, el rol del cliente se encarga de interactuar con el servidor con el fin de leer los datos expuestos por el mismo o controlar su comportamiento. Envía comandos y solicitudes al servidor y recibe una respuesta de este.

Cada atributo viene identificado por un manejador de atributos llamado *handler*. Se trata de un identificador de 16 bits presente en cada uno de los atributos en el servidor. Además, contiene un identificador único universal (UUID) que puede ser de 16 o 128 bits y permite

conocer la naturaleza del atributo. También presenta un conjunto de permisos de escritura y lectura los cuales establecen algunas reglas de interacción sobre el valor del atributo y, por último, contiene un valor, es decir, posee la información acerca del atributo.
Cada atributo contiene un valor que contiene los datos que el servidor necesita y una serie de permisos que determinan si un valor se puede leer o escribir y los niveles de seguridad requeridos para cada operación.

3. Protocolo de Gestión de Seguridad (SMP)

Este protocolo contiene una serie de algoritmos de seguridad que permite que la pila de protocolos de Bluetooth genere, intercambie y almacene claves de seguridad. SMP presenta dos funciones principalmente:

- Función de iniciador: Corresponde al maestro y a la central del perfil de acceso genérico.

- Función de respondedor: Corresponde al esclavo de la capa de enlace y al periférico del perfil de acceso genérico.

Por otro lado, este protocolo participa en numerosos procedimientos como:

- Emparejamiento: Se genera una clave de cifrado común y temporal para una conexión cifrada segura. Es una clave de uso único, es decir, no podrá ser utilizada en conexiones posteriores.
- Vinculación: Generación e intercambio de claves de seguridad permanentes creando una conexión permanente entre dos dispositivos con el fin de llevar a cabo conexiones seguras en conexiones posteriores.

- Cifrado de restablecimiento: Las claves de seguridad se almacenan en ambos dispositivos para usarlas en posteriores conexiones.

4. Perfil Genérico de Atributo (GATT)

GATT está basado en el Protocolo de Atributos y define los mecanismos con los cuales los dispositivos pueden comunicarse e intercambiar datos entre aplicaciones mediante el uso de servicios y características.

Un servicio consiste en una o más características formadas por atributos que presentan diferentes funciones: valor, descripción, configuración etc. es decir, definen la conducta de un dispositivo. Los datos del servidor se encapsulan en servicios que definen la conducta de un dispositivo. En BLE se encuentran dos tipos de servicios, el primario, haciendo referencia a la función principal del dispositivo, y el servicio secundario cuya relevancia está sujeta a la participación de otro servicio. Además, contiene perfiles encargados de definir el comportamiento tanto del cliente como del servidor como por ejemplo las características de los servicios, las conexiones y requisitos de seguridad. Por otro lado, se encargan de la implementación de los servicios y características solo en el modo esclavo.

GATT asume los mismos roles que el Protocolo de Atributos, cliente y servidor. En este caso, los roles se determinan para la transacción.

5. Perfil de Acceso Genérico (GAP)

GAP se encarga de controlar la detección de dispositivos, la conexión, difusión o sistemas de seguridad con el fin de que se lleve a cabo el intercambio de datos entre diferentes dispositivos. Además, se encarga de la interacción de los diferentes roles que un dispositivo puede adoptar:

- Difusor (*Broadcaster*): Este rol se encarga de enviar periódicamente mensajes de anuncio con datos utilizando el rol de *Advertiser* de la capa de enlace. Este rol no permite que se establezca una conexión.
- Observador: Se encarga de escuchar los datos contenidos en los mensajes de anuncio utilizando el rol de *Scanning* de la capa de enlace. Además, busca otros dispositivos, pero tampoco establece una conexión.

- Central: Corresponde con el rol de Máster de la capa de enlace. Se encarga de escuchar paquetes de anuncio de otros dispositivos e iniciar una conexión.

- Periférico: Corresponde con el rol Slave de la capa de enlace. Utiliza los paquetes de anuncio para permitir al rol central encontrarle y establecer una conexión.

Dentro del Perfil de Acceso Genérico se encuentran diferentes modos. Un modo es un estado que el dispositivo puede adoptar con el fin de lograr un objetivo determinado por lo que puede ser temporal. Algunos de los modos pueden ser: Transmisión, detectabilidad, conectividad, unión…

6. Capa Interfaz de Control del Host (HCI)

Esta capa se encarga de comunicar la capa Host con la Capa del Controlador. Transporta los comandos entre las dos capas. Define como se intercambian los comandos, eventos, paquetes de datos etc.

Capa del controlador

1. Capa física

La capa física es la que se encuentra en el nivel más bajo en la pila de protocolos de BLE. Su función se basa en el envío y recepción de datos mediante el uso de radiofrecuencias. Opera en banda ISM de licencia libre de 2.400 GHx a 2.4835 GHz dividiéndose en 40 canales de radio de 2MHz de ancho y numerándose del 0 al 39. Existen dos tipos de canales: de anuncio, cuya función consiste en el descubrimiento de dispositivos, transmisión de mensajes de *broadcast* y configuración de conexiones, y, por otro lado, los canales de datos utilizados para las comunicaciones bidireccionales entre dispositivos conectados.

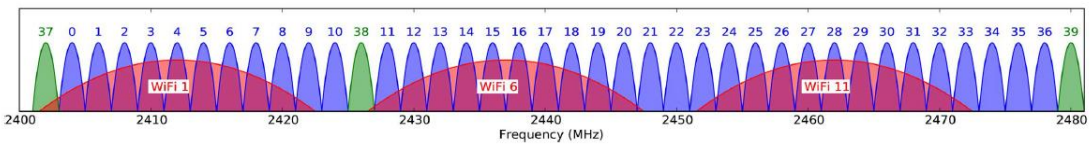

Ilustración 11.Canales módulo BLE

2. Capa de enlace

Se encuentra por encima de la capa física interactuando directamente con ella. Se encarga de la negociación y establecimiento de conexión y selecciona la frecuencia de trasferencia de datos, es decir, es responsable de cumplir todos los requisitos de tiempo definidos. De este modo, logra una conexión fiable y eficiente entre dos dispositivos.

Su funcionamiento viene definido por su máquina de estado y sus roles diferenciados. BLE presenta una estructura asimétrica entre maestro y esclavo requiriéndose más recursos para actuar como maestro. Dentro de su máquina de estados se diferencian los siguientes:

- *Standby*: El dispositivo ni transmite ni recibe pudiendo acceder al mismo desde cualquier otro estado.

- *Advertising*: El dispositivo envía mensajes de anuncio en los canales correspondientes con el fin de establecer una conexión. Este estado también permite escuchar cualquier solicitud de los paquetes desde el servicio central. Además, este estado afecta directamente al consumo de energía y potencia.

- *Scanning*: El dispositivo comienza la búsqueda de periféricos que se encuentran en el estado *Advertising* en su área de alcance. Existen dos sub-estados a partir de este, el escaneo pasivo que recibe paquetes de anuncio provenientes de dispositivos que se encuentran en su sector, y el escaneo activo que recibe y envía peticiones de escaneo a cada dispositivo.

- *Initiating*: Se reciben los paquetes del dispositivo *Advertising* y responde iniciando una conexión.

- *Connection*: Se trata del estado final tras pasar los anteriores estados. Se establece el enlace de trasmisión de dato entre dos dispositivos. En este estado se diferencian dos roles, *Master* y *Slave.* Por un lado, si el dispositivo llega al estado *Connection* desde el estado *Initiating,* el dispositivo estará actuando como *Master*. En cambio, si se llega al estado *Connection* desde el estado *Advertising,* el dispositivo se comportará como *Slave.*

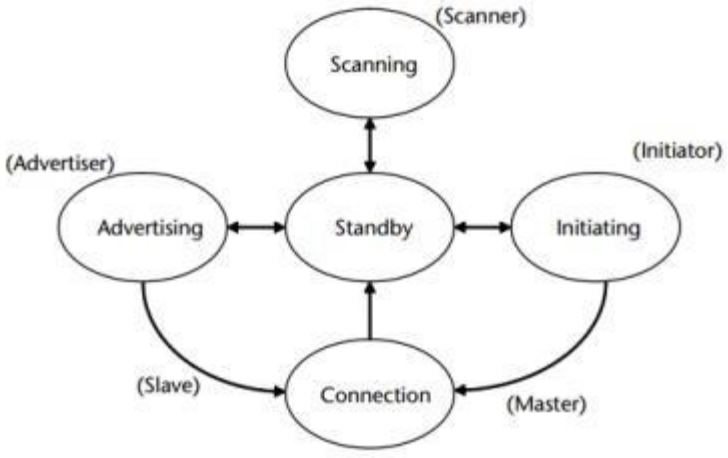

Ilustración 11. Modelo de máquina de estados de la capa de enlace

MODO WIFI

El dispositivo ESP32 contiene un módulo Wifi con tecnología 802.11b/g/n a una velocidad máxima de 150 Mbit/s con una frecuencia de 2,4 Ghz. Dispone de varios modos de seguridad como WPA o WPA2.

Para la comunicación de datos es necesario un conjunto de protocolos denominado TCP (*Transmission Control Protocol*) e IP (*Internet Protocol*). Todos ellos son el soporte de un conjunto de aplicaciones y servicios de aplicación.

El protocolo TCP está dirigido a una conexión fiable y orientado a un flujo de bytes, es decir, el cliente y el servidor deben establecer una comunicación para poder intercambiar datos. Esto permite regular el flujo de bytes enviados y su correcta recepción, retrasmitiendo los bytes recibidos incorrectamente o perdidos. Además, realiza una fase de desconexión del receptor antes de finalizar la comunicación.

Las redes IP son necesarias para el funcionamiento de los protocolos. Se trata de un protocolo no fiable y no orientado a la conexión encargado del transporte de los datos por Internet. Están construidas por un esquema de capas (layers) donde cada una de ellas es responsable de las diferentes partes de la comunicación. Por lo tanto, la familia de protocolos TC/IP consiste en una combinación de cuatro capas según el modelo OSI:

- La capa de enlace (*link layer*): Contiene los mecanismos que permiten el envío y recepción de información a través de la red conectada.

- La capa de red (*network layer*): Su función consiste en mover los paquetes de información a través de las diferentes redes hasta llegar a su destino. En esta se encuentra el protocolo IP.

- La capa de transporte (*Transport layer*): Proporciona un flujo de datos entre dos ordenadores (TCP/UDP).

- La capa de aplicaciones (*Applications layer*): Maneja los detalles relacionados con las aplicaciones que utiliza el usuario.

La conexión Wifi es proporcionada por un punto de acceso que actúa como un centro para uno o más dispositivos y cada dispositivo que se conecta a la red se denomina estación. Los modos de conexión Wifi que permite ESP32 pueden ser:

- Modo estación (STA): El ESP32 se conecta a una red, es decir, actúa como una estación. A través de la dirección IP se configura un punto de entrada que da acceso a los servicios ofrecidos por el dispositivo. Es necesario estar conectado al router para controlar el ESP32.

- Modo Punto de Acceso (*Soft AP*): En este modo el ESP32 actúa como un punto de acceso para la red por lo que no es necesario estar conectado a un router para controlar el ESP32. El dispositivo crea su propia red local pudiendo configurar el nombre de red, identificador y contraseña.

- Modo combinado (*Soft AP + STA*): El dispositivo actúa como punto de acceso pudiendo estar conectado a otra red a la vez como estación.

5.2.2. Módulo relé de dos canales

Un relé es un dispositivo electromagnético cuya función es la apertura o cierre de un circuito eléctrico cuando se cumple ciertas condiciones que influyen sobre él. Su funcionamiento se basa en la excitación de una bobina la cual produce un campo electromagnético permitiendo que el contacto del relé, que normalmente está abierto, se cierre y permita el paso de la corriente por el circuito.

En el presente proyecto se utiliza un módulo con 2 relés debido al uso de dos pestillos controlados de forma independiente. Este dispositivo funciona con una tensión de 5V y se compone de 3 salidas distintas, una de ellas llamada *Normally open* actúa como un circuito abierto mientras que no se active el relé, es decir, no permitirá el paso de la corriente impidiendo el funcionamiento del pestillo. Por el contrario, la otra salida llamada *Normally closed* mantendrá el circuito cerrado permitiendo el paso de la corriente. Por último, se encuentra la salida común la cual se utiliza para la corriente que se desea controlar, es decir, el voltaje de la red.

Ilustración 12. Módulo relé.

5.2.3. Protoboard

La placa protoboard es una herramienta que se compone de múltiples agujeros que nos permiten insertar fácilmente componentes electrónicos para hacer un circuito electrónico. Además, permite conectar los componentes eléctricos y sus cables entre sí.

La placa protoboard tiene tres partes fáciles de identificar: El canal central, las pistas y los buses.

Por un lado, se encuentra el canal central cuya función consiste en la separación de las zonas de conexión superior e inferior de la placa de manera que, cuando se conecten los circuitos integrados en la placa protoboard, los pines de ambos lados de dicho circuito se queden aislados.

Por otro lado, los buses están ubicados en los costados de la placa y, generalmente, se usan para conectar la tierra del circuito y el voltaje de este.

Por último, el resto de los orificios pertenecen a las pistas. Estas se encuentran separadas por filas de orificios conectados eléctricamente entre sí, es decir, cada fila tiene conexión entre sí, y cada columna es independiente eléctricamente con las demás columnas.

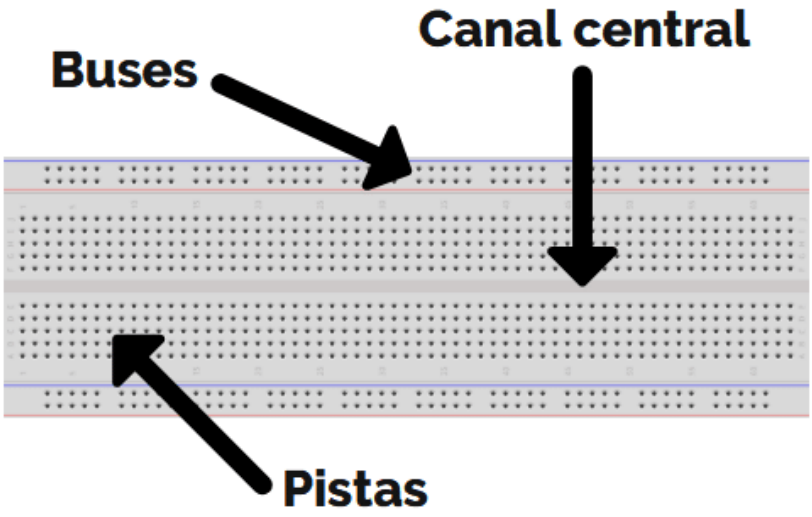

Ilustración 13. Placa Protoboard

5.2.4. Tarjeta y lector RFID

La tecnología RFID *Radio Frecuency IDentification* es un sistema de identificación automática de información contenida en etiquetas electrónicas. Uno de los aspectos más destacables de esta tecnología en el funcionamiento mediante identificación por radiofrecuencias sin necesidad de que exista un contacto físico o visual entre el lector y las etiquetas, aunque en muchos casos se exige una cierta proximidad. Además, es necesario el uso de un teléfono móvil, tarjeta, pulsera u otro elemento que genere la señal.

En un sistema RFID, el elemento que se quiere identificar se etiqueta con un pequeño chip de silicio unido a una antena de radiofrecuencia llamado *tag* o etiqueta. Este puede ser identificado y comunicarse mediante ondas de radiofrecuencia gracias a un dispositivo transmisor/receptor que en este proyecto es la tarjeta RFID. Su funcionamiento consiste en el envío de una señal continua por parte del receptor dentro de un radio de alcance en concreto. Cuando, en este caso, la tarjeta RFID entra en contacto con la señal, se envía información que el lector interpreta según esté programado.

Dentro de los componentes de la tecnología RFID encontramos:

- Etiquetas RFID: Sus componentes son el chip, el cual almacena la información y ejecuta los comandos específicos, la antena que absorbe las ondas de radio y

difunde la información contenida en el chip, y el sustrato siendo este el material que mantiene el chip y la antena juntos.

- Lector: Encargado de emitir y recibir las señales de radio a través de las antenas que llevan acopladas. Son los responsables de la trasmisión de información entre las etiquetas y el sistema central, que en este caso es el ESP32.

Ilustración 14. Lector y tarjetas RFID

5.2.5. Cables de Interconexión

Se han utilizado diferentes cables de interconexión, tanto macho-macho como hembra-macho para poder interconectar cada uno de los elementos empleados en el proyecto.

Ilustración 15. Cables de Interconexión Macho-Macho y Hembra-Macho

5.2.6. Pestillo eléctrico solenoide 12V

La cerradura escogida en el proyecto se trata de un pestillo eléctrico solenoide de 12V que contiene una bobina de alambre de cobre junto a una armadura en el medio. Cuando

la bobina recibe energía, esta armadura comienza a moverse provocando la actuación del solenoide que permite la apertura y cierre de la cerradura.

Ilustración 16. Pestillo eléctrico solenoide

5.2.7. Fuente de alimentación 12V

Se ha utilizado una fuente de alimentación de 12 voltios para proporcionar corriente a cada uno de los pestillos eléctricos. A esta se le ha añadido un conector hembra (+ -) con el fin de poder interconectar la fuente de alimentación a los pestillos eléctricos a través de cables de puente. En esta interconexión también se encuentra conectado el módulo relé ya que es el responsable de abrir y cerrar la corriente cuando se envíe la orden.

Ilustración 17. Fuente de alimentación del pestillo eléctrico y su adaptador

5.2.8. Cable de alimentación micro USB

La alimentación del ESP32 se ha llevado a cabo a través de un cable micro USB que va conectado al ordenador. Además, permite la compilación del código creado en el entorno de desarrollo (IDE) en dicho microcontrolador.

Ilustración 18. Cable micro USB

5.3. Arquitectura Software

5.3.1. Lenguajes de programación

Hoy en día los lenguajes que se utilizan para desarrollar aplicaciones Android suelen ser Java o Kotlin, ya que estos lenguajes permiten a los programadores utilizar una serie de funciones que facilitan tanto el uso del hardware como los recursos del sistema. Esto no quiere decir que no existan otros lenguajes que permitan desarrollar aplicaciones ya que disponiendo de un compilador compatible con Linux y el hardware del dispositivo que se esté utilizando se puede crear una aplicación Android.

JAVA

Java es uno de los lenguajes de programación más conocidos que se han utilizado en el desarrollo de aplicaciones Android, también ha cooperado en el desarrollo de sistemas operativos, software empresarial y un gran abanico de aplicaciones. Es uno de los primeros lenguajes orientado a objetos, esto quiere decir, que su código se organiza a partir de clases y objetos, en vez de funciones y comandos. Estos rasgos permiten que este lenguaje presente las siguientes características:

- Simple
- Orientado a objetos
- Tipado estáticamente
- Distribuido
- Interpretado
- Robusto
- Seguro

- Arquitectura neutral
- Multihilo
- Recolector de basura (*Garbage Collector*)
- Portable
- Alto rendimiento
- Dinámico

Presenta una gran cantidad de características, pero las más importantes que han provocado el gran interés a los distintos fabricantes son:

- Robustez: Java realiza verificaciones con el fin de buscar plataformas tanto en tiempo de compilación como de ejecución. Este lenguaje obliga a la declaración explícita de los métodos en busca de disminuir en un 50% los errores típicos que se producen en los lenguajes de programación como C y C++. También se encarga de liberar la memoria, reduciendo las preocupaciones del programador.

- Independiente de plataforma: La mayoría de los lenguajes de programación requieren la necesidad de recompilar el código fuente cada vez que se cambia de plataforma. Java utiliza una máquina virtual para cada sistema con el fin de que el mismo código escrito se pueda ejecutar en diferentes sistemas operativos, sin la necesidad de modificarlo o recompilarlo de nuevo, permitiendo que los archivos generados se compilen una vez y se puedan ejecutar en **cualquier plataforma.**

- Multitarea: Esta característica depende en gran parte al sistema operativo en el cual se ejecuten. Esta propiedad permite ejecutar varios procesos ligeros o hilos de ejecución de forma concurrente.

Ilustración 19. Logo de lenguaje de programación Java

La elección de este lenguaje de programación a la hora de programar la aplicación del proyecto se debe a que este ha sido uno de los más utilizados en la carrera del estudiante.

KOTLIN

Kotlin es un lenguaje de programación estático de código abierto que permite tanto la programación funcional como la orientada a objetos. Se trata de un sistema que provee una sintaxis y conceptos similares a los de otros lenguajes, como C# y Java, entre muchos otros. Su objetivo es eliminar el código repetitivo, reduciendo de forma considerable los posibles errores. Kotlin es el lenguaje de programación predilecto para la mayoría de los programadores de aplicaciones Android. Esto se debe a las características y ventajas que se muestran a continuación:

- Multiplataforma: Su lenguaje se ejecuta en JVM. Esto nos permite desarrollar y compartir los proyectos en cualquier plataforma.

- Interoperable con Java: Este lenguaje permite interactuar con Java sin ningún problema. Además, aborda varios problemas de java y presenta mayor seguridad, esto hace que se convierta en el lenguaje favorito de muchos desarrolladores.

- Funcionalidad y orientación hacia objetos: permite trabajar con ambas, simplificando las tareas y ahorrando mucho tiempo a los desarrolladores.

- Corrutinas: son patrones de diseño que se utilizan en Android con el fin de reducir el código que se ejecuta de forma asíncrona.

Ilustración 20. Logo de lenguaje de programación Kotlin

En líneas futuras, se prevé la migración a Kotlin debido a su gran cantidad de ventajas:

A continuación, se detallan las ventajas y desventajas que presentan los diferentes lenguajes:

VENTAJAS	
JAVA	• Es utilizado por muchos programadores • Presenta una gran comunidad y facilidad de conseguir ayuda mediante *StackOverflow* • Contiene miles de librerías • Documentación y recursos para el aprendizaje • Código repetitivo y específico que facilita el aprendizaje
KOTLIN	○ Simple, conciso y flexible ○ Facilidad de aprendizaje ○ Apoyo completo de Google ○ Interoperabilidad con Java ○ Favorito para los desarrolladores y programadores de Android ○ Funciones de alto orden y procedimientos que Java no dispone

Tabla 3. Ventajas de lenguajes de programación Java y Kotlin

DESVENTAJAS	
JAVA	• Lenguaje interpretado, su rendimiento en la ejecución de programas es menor • Programas compilados en código nativo, solo permite su ejecución con máquina virtual • Debido a que está orientado a objetos, es aconsejable no implementarlo con desarrolladores principiantes • Evolución lenta.
KOTLIN	○ Coexistencia de patrones débiles ○ Tamaño extra de ejecución de programas ○ Código tedioso de leer e interpretar al inicio ○ Pequeña comunidad de soporte debido a su reciente aparición

Tabla 4. Desventajas de lenguajes de programación Java y Kotlin

HTML

HTML es el componente más básico de una página web, con él se define tanto el significado como la estructura de éstas. Es un lenguaje muy sencillo que permite definir hipertexto, es decir, texto presentado de forma estructurada con vínculos o enlaces que nos conducen a distintos documentos o fuentes de información relacionadas que presentan inserciones multimedia (imágenes, sonido, vídeos, etc.)

Este lenguaje permite a los desarrolladores la utilización de herramientas como:

- Publicación de documentos en línea con encabezados, listas, fotos, etc.
- Recopilación de información en línea mediante vínculos de hipertexto.
- Diseño de formularios con el fin de realizar transacciones a partir de servicios remotos.
- Añadir hojas de cálculo, sonidos, vídeos, etc.

La estructura básica de un documento HTML consta siempre de estas tres partes:

1. Una línea que muestra información de la versión del HTML.
2. Una sección de la cabecera del documento delimitada por el elemento HEAD.
3. El cuerpo en el que se encuentra el contenido que se mostrará desde el navegador, delimitada por el elemento BODY.

Ilustración 21. Estructura de HTML

C++

El lenguaje de programación C++ se basa en el lenguaje C y lo extiende con características adicionales con el fin de proporcionar programación estructurada de alto nivel y soporte para la programación orientada a objetos.

La razón por la cual C++ se utiliza en una amplia variedad de herramientas y proyectos se debe a la gran cantidad de ventajas que ofrece. En primer lugar, este lenguaje destaca por su eficiencia y rendimiento. Esto permite desarrollar aplicaciones que requieren un uso intensivo de recursos, como juegos, sistemas embebidos, etc.

Otra ventaja clave de C++ es que proporciona soporte para la programación orientada a objetos. Esta ventaja permite organizar el código en clases y objetos con el objetivo de reutilizar código y desarrollar aplicaciones modulares y escalables.

Además, permite acceso directo a componentes de bajo nivel del sistema, como la memoria y los dispositivos hardware. Esto es útil cuando se necesita un control preciso sobre el funcionamiento del sistema y se requiere interacción directa con los periféricos, como en el caso de los microcontroladores utilizados en Arduino.

También ha influido en el desarrollo de muchos otros lenguajes de programación, y su uso es amplio en diversas áreas, como la programación de sistemas operativos, aplicaciones web, software embebido, inteligencia artificial, etc. Es por ello, que dicho lenguaje se ha empleado en el proyecto, en concreto, en el entorno de desarrollo Arduino IDE debido a que el lenguaje que utiliza esta herramienta está basado en C++ y ha sido uno de los más utilizados en el grado.

JAVASCRIPT

JavaScript es un lenguaje de programación de alto nivel, interpretado y orientado a objetos que se utiliza principalmente en el desarrollo web con el fin de agregar interactividad a las páginas web y ejecutar scripts en el lado del cliente.

Este lenguaje permite crear elementos interactivos en páginas web como formularios dinámicos, botones desplegables, efectos visuales, etc. Además, puede responder a eventos como clics de mouse, pulsaciones de teclas y acciones del usuario. Gracias a esta

característica, se ha implementado un fragmento de este lenguaje en el diseño del servidor web, con el objetivo de poder interactuar con los botones definidos en dicha página web.

Otras funcionalidades que presenta este lenguaje son:

- Manipulación del DOM: permite realizar cambios en tiempo real relacionados con la apariencia y contenido de la página.
- Validación de formularios: permite validar datos ingresados en formularios web antes de enviarlos al servidor.
- Comunicaciones con el servidor: permite realizar solicitudes HTTP asíncronas para enviar y recibir datos del servidor sin tener que recargar la página con el objetivo de crear aplicaciones web dinámicas y actualizaciones en tiempo real.

SQL

SQL es un lenguaje de programación diseñado para gestionar y manipular bases de datos. El objetivo principal de este lenguaje es permitir a los desarrolladores y administradores realizar consultas y operaciones en una base de datos de manera eficiente y efectiva. Esto se consigue a través de sentencias SQL, las cuales permiten acceder a los datos almacenados en una base de datos, manipularlos, modificarlos y gestionar su estructura. Algunas de estas sentencias SQL incluyen:

- SELECT: permite recuperar datos de una o varias tablas.
- INSERT: permite insertar nuevos registros en una tabla.
- UPDATE: permite modificar los registros existentes en una tabla.
- DELETE: permite eliminar registros de una tabla.
- CREATE: permite crear nuevas tablas o vistas.

SQL es compatible con la mayoría de los sistemas de gestión de bases de datos como SQLite, MySQL, Oracle, PostgreSQL, entre otros. En este proyecto, se ha empleado el sistema SQLite, aunque hay algunas diferencias a la hora de la implementación, el núcleo del lenguaje y la mayoría de las funcionalidades son comunes entre ellos.

5.3.2. Bases de datos locales

Una base de datos es un conjunto de información o datos estructurados que se encuentran organizados, los cuales se almacenan de forma electrónica en un sistema informático. Estas bases de datos se encuentran controladas por un sistema de gestión de bases de datos (DBMS). La recopilación de datos, el DBMS y las aplicaciones asociadas a ellos reciben el nombre de bases de datos.

Los datos se suelen agrupar en estructuras de filas y columnas con el objetivo de facilitar el procesamiento y la consulta de datos.

La mayoría de las bases de datos emplean un lenguaje de consulta estructurado (SQL) para consultar, manipular, controlar el acceso y definir datos.

Ilustración 22. Estructura de una base de datos

5.3.3. Entornos de desarrollo

Un entorno de desarrollo en el mundo del software y la tecnología hace referencia al conjunto de procedimientos y herramientas que se emplean con el objetivo de desarrollar un código fuente o un programa. Usualmente se denomina bajo las siglas IDE que es el editor de código integrado que contiene herramientas tanto de construcción como de depuración. Normalmente cuentan con funcionalidades avanzadas que permiten autocompletar el código con el fin de facilitar trabajo al desarrollador.

Los entornos de desarrollo cuentan normalmente con tres niveles de servidores, clasificados como desarrollo, montaje y producción, denominándose en su conjunto como DSP.

- Servidor de desarrollo: El desarrollador realiza pruebas en el código para comprobar que la aplicación funciona correctamente con dicho código. Una vez que la comprobación se ha realizado de forma correcta se desplaza hacia el siguiente servidor.

- Servidor de Integración: La aplicación se prueba en el servidor de ensayo para asegurarse de que no se produzca ningún fallo en el servidor real. La aplicación una vez aprobada, se mueve hacia el servidor de producción.

- Servidor de producción: Una vez se ha realizado la aprobación, la aplicación forma parte de este servidor.

A la hora de llevar a cabo el desarrollo de una aplicación Android, debemos destacar Android Studio como entorno de desarrollo recomendado por Google.

ANDROID STUDIO

Android Studio es un entorno de desarrollo de apps Android creado por Google. Cuenta con un editor de códigos y herramientas para desarrolladores IntelliJ que son necesarias para crear dichas apps. Android Studio presenta una gran cantidad de funciones con el fin de aumentar la productividad a la hora de desarrollar una aplicación. Las más importantes son:

- Compilación flexible basado en *Gradle*.
- Compatible con C++ y NDK.
- Herramientas de identificación de problemas (rendimiento, compatibilidad, …).
- Emulador repleto de funciones y de gran velocidad.
- Ayuda al compilar funciones de apps debido a su gran cantidad de plantillas de código.

Este entorno de desarrollo es uno de los estudiados en el grado y es el que se ha utilizado para diseñar la aplicación del proyecto.

Existen otros tipos de entornos de desarrollo especializados en la programación de microcontroladores como por ejemplo Arduino Ide**.**

ARDUINO IDE

El IDE de Arduino es un entorno de programación que se utiliza para compilar e interpretar código con el fin de desarrollar proyectos en microcontroladores. Sus funciones principales son:

- Soporte multiplataforma de Arduino
- Muestra memoria flash y SRAM ocupada por un sketch
- Carga de sketch vía red (wifi o ethernet)

Estos entornos de desarrollo son algunos de los estudiados en la carrera y son los que se han utilizado para llevar a cabo el proyecto.

Ilustración 23. Logo de Arduino IDE

6. ANÁLISIS DEL PROYECTO

En este proyecto se aborda la creación y desarrollo de un prototipo de taquilla inteligente formado por un sistema compuesto por 2 pestillos eléctricos y un microprocesador como unidad central, que permita realizar la apertura de ambas taquillas a través de una aplicación Android o a través de una tarjeta RFID.

La aplicación móvil se encarga de enviar los datos correspondientes de cada una de las taquillas al microprocesador, a través de la utilización de la tecnología inalámbrica, específicamente BLE y WIFI. Además, se ha implementado una base de datos dentro de la aplicación, la cual tiene como finalidad llevar un registro detallado de todas las aperturas efectuadas.

6.1. Requisitos

El sistema Hardware está formado por un módulo ESP32, dos pestillos eléctricos de solenoide, un módulo relé compuesto por dos canales, un lector RFID, cables macho – hembra de diferentes longitudes y una protoboard. El principal objetivo de este sistema radica en la apertura de cada una de las taquillas mediante la respuesta que transmite la aplicación móvil o la tarjeta RFID al módulo ESP32, permitiendo éste varias alternativas a la hora de abrir las taquillas.

En relación con los requisitos *Software,* es necesario obtener una conexión BLE con el fin de consolidar la conexión entre el ESP32 y el dispositivo móvil y poder establecer un intercambio de información entre ambos. Además, este proyecto también ofrece la posibilidad de realizar una conexión mediante tecnología WIFI ofrecida por el módulo ESP32 o, en el caso de no disponer de un dispositivo móvil, a través de la tecnología RFID. En el caso de realizarse la conexión a través de la tecnología WIFI, el envío de datos se realizará mediante un servidor web.

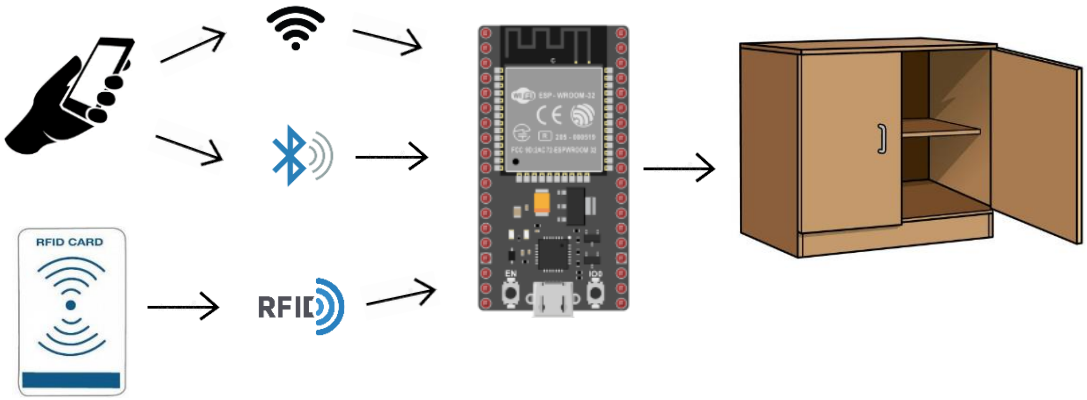

Ilustración 24. Proceso de intercambio de datos con el ESP32

6.2. Análisis de los casos de uso

A partir de los diferentes requisitos definidos en los puntos anteriores, se diferencian los casos de uso del sistema según el tipo de acceso:

6.2.1. Acceso a través de BLE

La primera alternativa para llevar a cabo la monitorización de las taquillas es mediante la tecnología BLE, debido a su bajo consumo se ha llevado a cabo un mayor desarrollo de código en comparación con otras tecnologías, con el fin de proporcionar al usuario una mayor cantidad de información y funcionalidades. A continuación, se detallan cada uno de los pasos para poder llevar a cabo dicha monitorización:

Búsqueda dispositivos BLE

Para establecer una conexión inalámbrica a través del bluetooth de baja energía (BLE), el usuario debe realizar un escaneo de los dispositivos BLE disponibles, detectar el bluetooth del ESP32 que se ha configurado con el nombre “UMH ESP32 JASS” y realizar el emparejamiento de la conexión para poder llevar a cabo el intercambio de información.

Control taquillas BLE

Una vez realizado el proceso de escaneo BLE y establecida la conexión con el bluetooth del ESP32, el usuario debe redirigirse a la siguiente pantalla para monitorizar las taquillas a través de un botón de apertura configurado en la parte inferior de la vista. Cada taquilla

tiene asignado un botón específico para su apertura. De esta manera, el usuario puede llevar a cabo la apertura de cada taquilla de forma individual y precisa.

Consultar base de datos BLE

En caso de que el usuario haya monitorizado alguna de las taquillas, existe la posibilidad de comprobar en qué momento sea realizado cada apertura y a qué taquilla corresponde. Para ello, se almacena un registro de aperturas que permite consultar dicha información en cualquier momento.
Asimismo, se ofrece la opción de eliminar todos los registros, en caso de que el usuario lo desee, a través de un botón situado en la parte inferior de la vista.

6.2.2. Acceso a través de Wifi

La segunda alternativa que se emplea para el control de las taquillas es la tecnología Wifi. Ésta representa la última opción para monitorizar las taquillas a través de la aplicación móvil. A continuación, se explican cada una de las vistas utilizadas en esta tecnología:

Búsqueda redes WIFI

Para llevar a cabo la conexión a través de la tecnología Wifi, el usuario debe realizar un escaneo de las redes Wifi que se encuentren disponibles y establecer la conexión con la red definida en el entorno de desarrollo configurada con el nombre "UMH_JASS_2022''. Para establecer la conexión es necesario introducir la contraseña definida en la vista de la aplicación.

Control taquillas WIFI

Una vez se han completado cada uno de los procesos anteriores, el usuario tiene la posibilidad de monitorizar las taquillas a través del servidor web definido en la vista siguiente de la aplicación. En dicho servidor se muestran, en la parte central, los respectivos botones que se encargan de la apertura de ambas taquillas.

6.2.3. Acceso a través de RFID

La tercera alternativa utilizada en la monitorización de las taquillas es la tecnología RFID. Es la única tecnología empleada en el proyecto que no es necesario el uso de una

aplicación móvil. A continuación, se describen cada uno de los procesos empleados para poder llevar a cabo el control de ambas taquillas:

Activación tarjetas RFID

Con el fin de monitorizar las taquillas a través de esta tecnología, el usuario tiene que registrar el código de la tarjeta en el módulo ESP32. Para realizar dicho proceso, el usuario debe arrastrar la tarjeta RFID por el lector correspondiente con el fin de obtener el código asociado de la misma, éste aparece en la ventana COM4. Una vez se muestre, hay que añadirlo dentro del código de programación que se compila en el ESP32 para llevar a cabo dicha acción.

Control taquillas RFID

Una vez completado el proceso anterior, el usuario está habilitado para controlar las taquillas utilizando la tarjeta RFID. Para abrir la taquilla, es necesario deslizar la tarjeta RFID por el lector ubicado en uno de los extremos de las taquillas. Se dispone de dos tarjetas RFID, una asignada a cada taquilla, con el fin de poder llevar a cabo la apertura de la taquilla correspondiente.

7. IMPLEMENTACIÓN

En este apartado se explica con detalle el trabajo desarrollado para la realización de este proyecto, se exponen las soluciones adoptadas a nivel hardware y software.

7.1. Implementación componentes hardware y software

Los componentes hardware hacen referencia a las conexiones empleadas en los componentes físicos utilizados en el proyecto, es decir, el módulo ESP32, el módulo relé, el pestillo eléctrico, etc.

En cambio, los componentes software engloban a los componentes lógicos que intervienen en el proyecto, es decir, la programación en el IDE Arduino y la empleada en el desarrollo de la aplicación Android.

7.1.1. Conexiones de los componentes Hardware

Apertura a través de aplicación Móvil

La siguiente tabla proporciona una lista de los diferentes pines y sus correspondientes conexiones que se han utilizado para establecer la conexión entre el módulo relé y el módulo ESP32:

Cableado módulo relé y ESP32	
Pin de alimentación (VCC)	V5
Pin de datos 1 (DATA)	G12
Pin de datos 2 (DATA)	G13
Pin de tierra (GND)	GND

Tabla 5. Pines del relé empleados para el módulo ESP32

En la tabla que se muestra a continuación, se especifican los pines y las conexiones necesarias para llevar a cabo una conexión adecuada entre el pestillo eléctrico, el módulo relé y la fuente de alimentación:

Cableado pestillo eléctrico, módulo relé y fuente alimentación		
Pines Módulo Relé	Pines Pestillo Eléctrico	Pines Adaptador VCC
COM	-	Negativo
NO	Negativo	-
-	Positivo	Positivo

Tabla 6. Módulo relé, pestillo eléctrico y fuente de alimentación

En la siguiente imagen se expone una vista clara de cómo deben estar conectados los diferentes componentes electrónicos para que el sistema funcione correctamente:

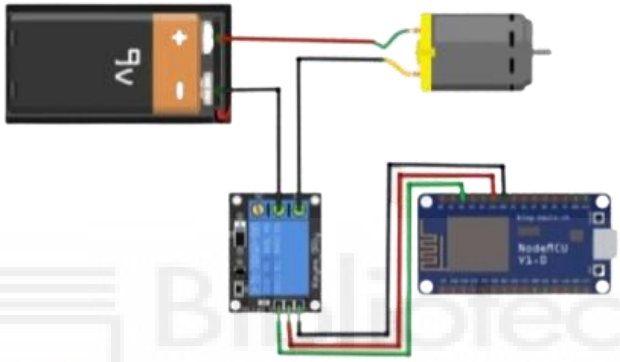

Ilustración 25. Cableado eléctrico del proyecto a través del programa fritzing

Apertura a través de la tarjeta RFID

En la tabla que se muestra a continuación, se detalla el esquema de conexiones que se ha utilizado para establecer la conexión entre el módulo ESP32 y el lector RFID:

Cableado del lector RFID con el ESP32	
Pines Tarjeta RFID	Pines ESP - 32
Sda	G34
Sck	G35
Mosi	G36
Miso	G36
Rq	-
Pin de tierra (GND)	GND
RST	G39
Pin de alimentación (VCC)	3.3 V

Tabla 7. Pines de la tarjeta RFID empleados para el módulo ESP32

Seguidamente, se aprecia una imagen del diseño completo de cada una de las conexiones empleadas entre el lector RFID y el módulo ESP32:

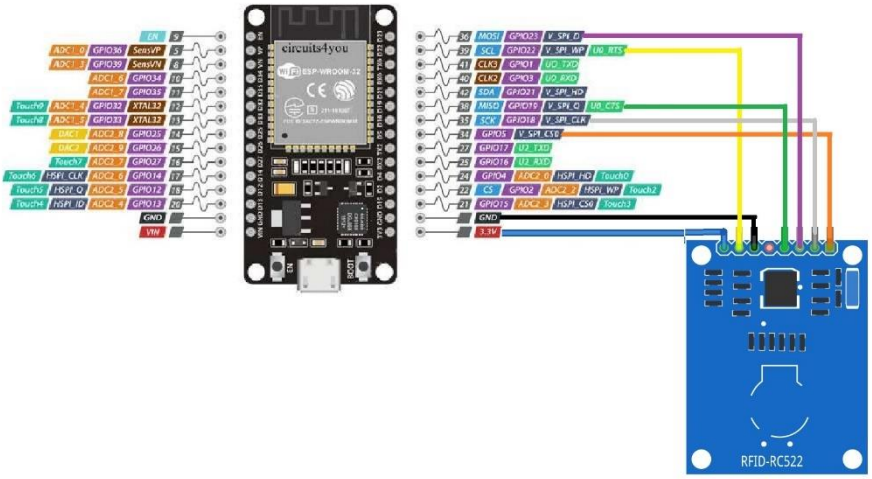

Ilustración 26. Cableado tarjeta RFID con el ESP32

Debido al elevado número de conexiones necesarias para establecer la conexión con el lector RFID, se ha tenido que añadir una protoboard adicional para llevar a cabo dichas conexiones. A continuación, se muestra una imagen completa del proyecto, donde se muestran cada una de las conexiones mencionadas anteriormente.

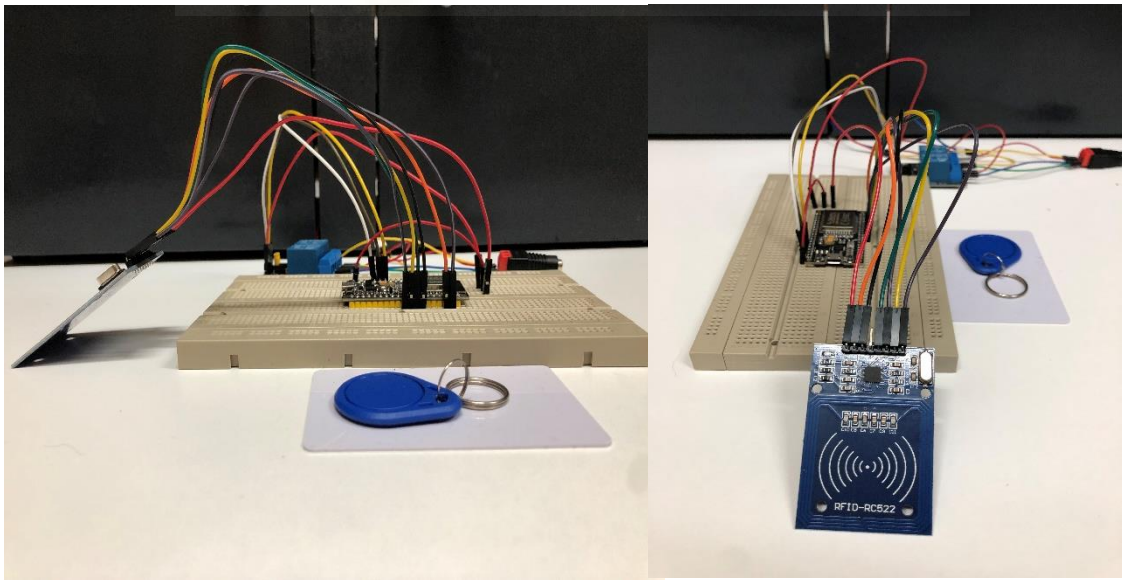

Ilustración 27. Conexiones del Proyecto completo

7.1.2. Programación módulo ESP32 en los tres modos

MODO WIFI

En este apartado se explica la programación empleada para el correcto funcionamiento del módulo ESP32 a partir de la tecnología inalámbrica Wifi.
En las primeras líneas de código de programación, en el entorno de desarrollo Arduino IDE, hay que importar las librerías necesarias para poder establecer una correcta comunicación con el módulo ESP32. En este caso, dado que se emplea la tecnología Wifi, es necesario contar con sus librerías correspondientes:

```
#include "WiFi.h"
#include "ESPAsyncWebServer.h"
```

Ilustración 28. Librerías Wifi

A continuación, se definen los parámetros iniciales del relé, incluyendo el estado de encendido o apagado, el número de canales de relé empleados, los pines asignados a cada uno de ellos y una variable que corresponde con el valor actual del relé:

```
#define RELAY_NO    true

#define NUM_RELAYS  2

int relayGPIOs[NUM_RELAYS] = {12, 13};
int relayValue;
```

Ilustración 29. Definición de características de los relés

Más adelante, se especifica tanto el nombre como la contraseña que se asignan a la red para que el ESP32 pueda funcionar en modo *Access Point* (AP). Esta información garantiza una seguridad a la hora de establecer la conexión con los dispositivos móviles:

```
const char* ssid = "UMH_JASS_TFG";
const char* password = "JASS_TFG_2022";
```

Ilustración 30. Definición de la red de acceso

Después, se definen dos parámetros llamados “relay” y “state” con el propósito de controlar individualmente cada relé y obtener información sobre su estado actual en el servidor web:

```
const char* PARAM_INPUT_1 = "relay";
const char* PARAM_INPUT_2 = "state";
```

Ilustración 31. Creación de state y relay

Seguidamente se establece el número de puerto empleado por el servidor web para poder llevar a cabo la comunicación entre el servidor web y la aplicación móvil. En este caso, se asigna el valor predeterminado 80 ya que es el puerto destinado para comunicaciones web a través del protocolo HTTP:

```
AsyncWebServer server(80);
```

Ilustración 32. Definición del puerto del servidor web

Una vez definidos las librerías y los parámetros, se llevan a cabo dos configuraciones que aparecen en cualquier código de Arduino:

Por un lado, se encuentra la configuración inicial denominada “setup()”. Se trata de la primera función que se ejecuta dentro del programa Arduino. En esta sección, se establecen las funciones que el microcontrolador llevará a cabo durante la ejecución, siendo ejecutadas una sola vez.

El propósito de este bloque es definir la configuración y las condiciones iniciales necesarias para el correcto funcionamiento del programa.

Aquí se define la comunicación serial y se establece la velocidad de transmisión de la comunicación, normalmente se establece un valor de 9600 bits por segundo (baudios), sim embargo, al emplear el módulo ESP32 se han definido 115200 baudios.

```
Serial.begin(115200);
```

Ilustración 33. Definición de los bits por segundo

También se realiza la configuración de los pines de entrada/salida. En este caso, se asignan cada uno de los pines de los relés como pines de salida. Después, se configuran

las condiciones para la activación y desactivación de cada relé en función del estado en el que se encuentren. Si se encuentran en el estado inicial se desactivarán con el comando "HIGH" y si se encuentran desactivados se activarán con el comando "LOW":

```
for(int i=1; i<=NUM_RELAYS; i++){
  pinMode(relayGPIOs[i-1], OUTPUT);
  if(RELAY_NO){
    digitalWrite(relayGPIOs[i-1], HIGH);

  }
  else{
    digitalWrite(relayGPIOs[i-1], LOW);
  }
}
```

Ilustración 34. Configuración de la activación de los relés

Al establecer estas condiciones, se asegura que los relés funcionen de acuerdo con los requerimientos del proyecto.

Después se realiza la configuración del nombre y la contraseña definidas anteriormente en el modo *Access Point*:

```
WiFi.softAP(ssid, password);
IPAddress ip = WiFi.softAPIP();

Serial.print("Nombre de mi red esp32: ");
Serial.println(ssid);
Serial.print("La IP es: ");
Serial.println(ip);
```

Ilustración 35. Configuración de la red de acceso

La dirección IP asignada a la red es la 192.168.4.1. Para acceder a esta dirección IP a través del navegador web es necesario estar conectado en la red definida para el modo Access point. Una vez establecida la conexión, solamente hay que introducir la dirección en el navegador web para acceder a la interfaz donde se controlará la apertura de las taquillas.

Después de realizar el paso anterior, se lleva a cabo la instancia de la web previamente definida:

```
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/html", index_html, processor);
});

server.on("/update", HTTP_GET, [] (AsyncWebServerRequest *request)
```

Ilustración 36. Instancia de la web

Mas adelante, se almacena la respuesta de la taquilla en dos parámetros definidos anteriormente, los cuales corresponden al identificador de los relés y al estado actual en el que se encuentran.

```
if (request->hasParam(PARAM_INPUT_1) & request->hasParam(PARAM_INPUT_2))
  inputMessage = request->getParam(PARAM_INPUT_1)->value();
  //inputParam = PARAM_INPUT_1;
  inputMessage2 = request->getParam(PARAM_INPUT_2)->value();
```

Ilustración 37. Respuesta de la taquilla

Dichos parámetros son los encargados de la apertura y cierre de las taquillas, en función del estado en el que se encuentre cada relé.

```
if(RELAY_NO){
  Serial.print("Abriendo pestillo");
  relayValue = !inputMessage2.toInt();
  Serial.print(" Pestillo Abierto ");
  Serial.println("NO " + inputMessage + " " + inputMessage2 + " " + relayValue);
  digitalWrite(relayGPIOs[inputMessage.toInt()-1], relayValue);
  delay(2000);
  Serial.print("Cerrando pestillo");
  digitalWrite(relayGPIOs[inputMessage.toInt()-1], HIGH);
  Serial.print(" Pestillo Cerrado ");
  Serial.println("NO " + inputMessage + " " + !inputMessage2 + " " + !relayValue);

}
else{
  Serial.print("NC ");
  digitalWrite(relayGPIOs[inputMessage.toInt()-1], inputMessage2.toInt());
}
```

Ilustración 38. Función de los parámetros Param Imput 1 y Param Imput 2

En este fragmento se distinguen dos partes. Por un lado, si el estado de los pestillos es "*RELAY_NO"*, es decir, se encuentran cerrados, el valor asignado al relé cambia y se lleva a cabo la apertura de la taquilla. Una vez transcurridos 2 segundos, el pestillo vuelve a su posición inicial a través del comando "HIGH".

```
if(RELAY_NO){
  Serial.print("Abriendo pestillo");
  relayValue = !inputMessage2.toInt();
  Serial.print(" Pestillo Abierto ");
  Serial.println("NO " + inputMessage + " " + inputMessage2 + " " + relayValue);
  digitalWrite(relayGPIOs[inputMessage.toInt()-1], relayValue);
  delay(2000);
  Serial.print("Cerrando pestillo");
  digitalWrite(relayGPIOs[inputMessage.toInt()-1], HIGH);
  Serial.print(" Pestillo Cerrado ");
  Serial.println("NO " + inputMessage + " " + !inputMessage2 + " " + !relayValue);

}
```

Ilustración 39. Configuración de la apertura y cierre de la taquilla

Por otro lado, en el caso de que los pestillos no se encuentren en el estado "RELAY_NO*"*, se mantiene el valor asignado al relé, a la espera de que cambie de estado.

```
else{
  Serial.print("NC ");
  digitalWrite(relayGPIOs[inputMessage.toInt()-1], inputMessage2.toInt());
}
```

Ilustración 40. Configuración del estado de la taquilla inicial

Finalmente, se inicia el servidor web:

```
server.begin();
```

Ilustración 41.
Inicio del servidor web

La otra parte del código se denomina "loop()", es la parte del código que se ejecutará de forma continua, es decir, actúa como un bucle mientras el ESP32 no se apague. En la tecnología wifi, no se lleva a cabo ninguna configuración:

```
void loop() {

}
```

Ilustración 42.
Configuración Loop

A continuación, se muestra el código completo empleado en la tecnología WIFI.

```
//Aplicacion TFG CON UN MÓDULO RELÉ DE DOS CANALES

// Import required libraries

#include "WiFi.h"
#include "ESPAsyncWebServer.h"

// Set to true to define Relay as Normally Open (NO)
#define RELAY_NO    true

// Set number of relays
#define NUM_RELAYS  2

// Assign each GPIO to a relay
int relayGPIOs[NUM_RELAYS] = {12, 13};
int relayValue;

// Replace with your network credentials
const char* ssid = "UMH_JASS_TFG";
const char* password = "JASS_TFG_2022";

const char* PARAM_INPUT_1 = "relay";
const char* PARAM_INPUT_2 = "state";

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

void setup(){
  // Serial port for debugging purposes
  Serial.begin(115200);

  for(int i=1; i<=NUM_RELAYS; i++){
    pinMode(relayGPIOs[i-1], OUTPUT);
    if(RELAY_NO){
      digitalWrite(relayGPIOs[i-1], HIGH);

    }
    else{
      digitalWrite(relayGPIOs[i-1], LOW);
    }
  }

  // Connect to Wi-Fi
   WiFi.softAP(ssid, password);
  IPAddress ip = WiFi.softAPIP();

  Serial.print("Nombre de mi red esp32: ");
  Serial.println(ssid);
  Serial.print("La IP es: ");
  Serial.println(ip);

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html, processor);
  });

  // Send a GET request to <ESP_IP>/update?relay=<inputMessage>&state=<inputMessage2>
  server.on("/update", HTTP_GET, [] (AsyncWebServerRequest *request) {
```

```
  String inputMessage;
  String inputParam;
  String inputMessage2;
  String inputParam2;
  // GET input1 value on <ESP_IP>/update?relay=<inputMessage>
  if (request->hasParam(PARAM_INPUT_1) & request->hasParam(PARAM_INPUT_2)) {
    inputMessage = request->getParam(PARAM_INPUT_1)->value();
    //inputParam = PARAM_INPUT_1;
    inputMessage2 = request->getParam(PARAM_INPUT_2)->value();
    //inputParam2 = PARAM_INPUT_2;
    if(RELAY_NO){
      Serial.print("Abriendo pestillo");
      relayValue = !inputMessage2.toInt();
      Serial.print(" Pestillo Abierto ");
      Serial.println("NO " + inputMessage + " " + inputMessage2 + " " + relayValue);
      digitalWrite(relayGPIOs[inputMessage.toInt()-1], relayValue);
      delay(2000);
      Serial.print("Cerrando pestillo");
      digitalWrite(relayGPIOs[inputMessage.toInt()-1], HIGH);
      Serial.print(" Pestillo Cerrado ");
      Serial.println("NO " + inputMessage + " " + !inputMessage2 + " " + !relayValue);

    }
    else{
      Serial.print("NC ");
      digitalWrite(relayGPIOs[inputMessage.toInt()-1], inputMessage2.toInt());
    }
  }
  else {
    inputMessage = "No message sent";
    inputParam = "none";
  }
    //Serial.println("-" + inputMessage + inputMessage2);
    request->send(200, "text/plain", "OK");
  });
  // Start server
  server.begin();
}

void loop() {

}
```

Ilustración 43. Configuración completa del modo Wifi

En este apartado también se lleva a cabo el diseño del servidor web definido en el módulo ESP32. Este fragmento de código se encontraría en el archivo de código fuente (sketch) relacionado con la configuración de la tecnología wifi, antes de la sección correspondiente al método “void setup()”:

```
<!DOCTYPE HTML><html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>
    html {font-family: Arial; display: inline-block; text-align: center;}
    h2 {font-size: 3.0rem;}
    p {font-size: 3.0rem;}
    body {max-width: 600px; margin:0px auto; padding-bottom: 25px;}
    .switch {position: relative; display: inline-block; width: 120px; height: 68px}
    .switch input {display: none}
    .slider {position: absolute; top: 0; left: 0; right: 0; bottom: 0;
    background-color: #ccc; border-radius: 34px}
    .slider:before {position: absolute; content: ""; height: 52px; width: 52px; left: 8px;
    bottom: 8px; background-color: #fff; -webkit-transition: .4s; transition: .4s;
    border-radius: 68px}
    input:checked+.slider {background-color: #2196F3}
    input:checked+.slider:before {-webkit-transform: translateX(52px);
    -ms-transform: translateX(52px); transform: translateX(52px)}
  </style>
</head>
<body>
  <h2>ESP Web Server</h2>
  %BUTTONPLACEHOLDER%
<script>function toggleCheckbox(element) {
  var xhr = new XMLHttpRequest();
  if(element.checked){ xhr.open("GET", "/update?relay="+element.id+"&state=1", true); }
  else { xhr.open("GET", "/update?relay="+element.id+"&state=0", true); }
  xhr.send();
}</script>
</body>
</html>
)rawliteral";
String processor(const String& var){
  Serial.println(var);
  if(var == "BUTTONPLACEHOLDER"){
    String buttons ="";
    for(int i=1; i<=NUM_RELAYS; i++){
      String relayStateValue = relayState(i);
      buttons+= "<h4>Taquilla #" + String(i) + " - GPIO " + relayGPIOs[i-1] + "</h4>
      <label class=\"switch\">


       <input type=\"checkbox\" onchange=\"toggleCheckbox(this)\"
       id=\"" + String(i) + "\" "+ relayStateValue +">
       <span class=\"slider\"></span></label>";
     }
     return buttons;
   }
   return String();
 }
```

Ilustración 44. Diseño del servidor Web

MODO RFID

En el siguiente apartado se explica la programación empleada para el funcionamiento del módulo ESP32 a partir de la tecnología inalámbrica RFID.

Al inicio del código de programación en el entorno de desarrollo Arduino IDE, se incluyen las librerías necesarias para asegurar un correcto funcionamiento del proyecto. En este caso, dado que se utiliza la tecnología RFID, se requieren las siguientes librerías:

```
#include <SPI.h>
#include <MFRC522.h>
```

Ilustración 45. Librerías RFID

Seguidamente, se definen y configuran dos pines específicos de la tarjeta RFID:

```
#if defined(ESP32)
  #define SS_PIN 5
  #define RST_PIN 22
#endif
```

Ilustración 46. Pines de tarjeta RFID

Una vez definidos los pines de la tarjeta, se instancia la clase, se definen los pines correspondientes al módulo relé y se registran cada uno de los códigos de las tarjetas RFID que se emplean en el proyecto:

```
MFRC522 rfid(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;
byte nuidPICC[4];
const byte relayPin1 = 12;
const byte relayPin2 = 13;
//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
String DatoHex;
const String UserReg_1 = "3A74B881";
const String UserReg_2 = "D7EFC9B4";
//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Ilustración 47. Definición de las características de la tarjeta RFID

Después de realizar el paso anterior, se llevan a cabo las mismas configuraciones empleadas en la tecnología anterior. Éstas se encuentran en la mayoría de los códigos de Arduino:

Por un lado, se encuentra la función “void setup()” ya mencionada anteriormente. Dentro de esta función se lleva a cabo:

- El inicio del bus SPI y la librería MFRC522:

```
SPI.begin(); // Init SPI bus
rfid.PCD_Init(); // Init MFRC522
Serial.println();
```

Ilustración 48. Inicio de SPI y librería MFRC522

- A continuación, se definen los pines de cada uno de los relés como pines de salida y se almacena el valor de la tarjeta leída en la variable denominada “DatoHex”:

```
pinMode(relayPin1, OUTPUT);
pinMode(relayPin2, OUTPUT);
DatoHex = printHex(key.keyByte, MFRC522::MF_KEY_SIZE);
```

Ilustración 49. Definición de los pines de salida

Por otro lado, se encuentra la configuración “void loop()”. Esta configuración, debido a que va a estar ejecutándose de forma continua, se llevan a cabo una serie de acciones que son necesarias para garantizar que solo las tarjetas autorizadas del tipo adecuado puedan tener acceso a la apertura de las taquillas:

1. Definición del tipo de tarjeta que se utiliza para realizar la apertura de la taquilla:

```
Serial.print(F("PICC type: "));
MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);
Serial.println(rfid.PICC_GetTypeName(piccType));
```

Ilustración 50. Definición de la tarjeta para la apertura

2. Comprobación de que la tarjeta introducida es del mismo tipo que la tarjeta definida:

```
if (piccType != MFRC522::PICC_TYPE_MIFARE_MINI && piccType !=
MFRC522::PICC_TYPE_MIFARE_1K && piccType != MFRC522::PICC_TYPE_MIFARE_4K)
{
  Serial.println("Su Tarjeta no es del tipo MIFARE Classic.");
  return;
}
```

Ilustración 51. Comprobación de la tarjeta introducida

3. Almacenamiento del valor de la tarjeta ingresada en la variable ‘’DatoHex’’ y realización de una comparación para corroborar si la tarjeta ingresada se encuentra registrada o no:

```
DatoHex = printHex(rfid.uid.uidByte, rfid.uid.size);
Serial.print("Codigo Tarjeta: "); Serial.println(DatoHex);
   if(UserReg_1 == DatoHex){
        Serial.println("USUARIO 1 - PUEDE INGRESAR");
        digitalWrite(relayPin1, LOW);
        delay(2000);
        digitalWrite(relayPin1, HIGH);
   }
  else if(UserReg_2 == DatoHex){
      Serial.println("USUARIO 2 - PUEDE INGRESAR");
      digitalWrite(relayPin2, LOW);
      delay(2000);
      digitalWrite(relayPin2, HIGH);
  }
else
{
 Serial.println("NO ESTA REGISTRADO - PROHIBIDO EL INGRESO");
 digitalWrite(relayPin1, LOW);
 digitalWrite(relayPin2, LOW);
}
Serial.println();
```

Ilustración 52. Configuración de la apertura y cierre de la taquilla a través de RFID

A continuación, se muestra el código completo empleado en la tecnología RFID:

```
#include <Arduino.h>
#include <SPI.h>
#include <MFRC522.h>

#if defined(ESP32)
  #define SS_PIN 5
  #define RST_PIN 22
#endif

MFRC522 rfid(SS_PIN, RST_PIN); // Instance of the class
MFRC522::MIFARE_Key key;
// Init array that will store new NUID
byte nuidPICC[4];
const byte relayPin1 = 12;
const byte relayPin2 = 13;
//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
String DatoHex;
const String UserReg_1 = "3A74B881";
const String UserReg_2 = "D7EFC9B4";
const String UserReg_3 = "7762C83B";
//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
pinMode(relayPin1, OUTPUT);
pinMode(relayPin2, OUTPUT);
DatoHex = printHex(key.keyByte, MFRC522::MF_KEY_SIZE);
Serial.println();
Serial.println();
Serial.println("Iniciando el Programa");

void setup()
{
   Serial.begin(115200);
   SPI.begin(); // Init SPI bus
   rfid.PCD_Init(); // Init MFRC522
   Serial.println();
   Serial.print(F("Reader :"));
   rfid.PCD_DumpVersionToSerial();
   for (byte i = 0; i < 6; i++) {
     key.keyByte[i] = 0xFF;
   }

void loop() {
     if ( ! rfid.PICC_IsNewCardPresent()){return;}
     if ( ! rfid.PICC_ReadCardSerial()){return;}

     Serial.print(F("PICC type: "));
     MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);
     Serial.println(rfid.PICC_GetTypeName(piccType));
     if (piccType != MFRC522::PICC_TYPE_MIFARE_MINI &&  piccType !=
     MFRC522::PICC_TYPE_MIFARE_1K && piccType != MFRC522::PICC_TYPE_MIFARE_4K)
     {
       Serial.println("Su Tarjeta no es del tipo MIFARE Classic.");
       return;
     }
       for (byte i = 0; i < 4; i++) {nuidPICC[i] = rfid.uid.uidByte[i];}

       DatoHex = printHex(rfid.uid.uidByte, rfid.uid.size);
       Serial.print("Codigo Tarjeta: "); Serial.println(DatoHex);
          if(UserReg_1 == DatoHex){
              Serial.println("USUARIO 1 - PUEDE INGRESAR");
              digitalWrite(relayPin1, LOW);
              delay(2000);
              digitalWrite(relayPin1, HIGH);
          }
         else if(UserReg_2 == DatoHex){
             Serial.println("USUARIO 2 - PUEDE INGRESAR");
             digitalWrite(relayPin2, LOW);
             delay(2000);
             digitalWrite(relayPin2, HIGH);
         }
       else

    {
     Serial.println("NO ESTA REGISTRADO - PROHIBIDO EL INGRESO");
     digitalWrite(relayPin1, LOW);
     digitalWrite(relayPin2, LOW);
    }
    Serial.println();
```

Ilustración 53. Configuración completa del modo RFID

MODO BLE

En este último apartado se describe la programación del código utilizado para el correcto funcionamiento del módulo ESP32 a partir de la tecnología inalámbrica BLE.

Al comienzo del código de programación, en el Arduino IDE, hay que añadir las librerías necesarias para poder implementar la tecnología BLE en el proyecto:

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
```

Ilustración 54. Librerías modo BLE

Tras realizar el paso anterior, se instancian una serie de variables que son necesarias para llevar a cabo la comunicación BLE. Por un lado, se crea la instancia de las características y el servidor BLE que se van a utilizar en el proyecto con el objetivo de transferir e intercambiar datos. Por otro lado, se definen las variables de control con el fin de controlar cuando se lleva a cabo la conexión y desconexión con el ESP32:

```
BLEServer* pServer = NULL;
BLECharacteristic* pCharacteristic1 = NULL;
BLECharacteristic* pCharacteristic2 = NULL;

// Variables de control utilizadas
bool deviceConnected = false;
bool oldDeviceConnected = false;
```

Ilustración 55. Definición de variables BLE

Después, se definen los servicios y características necesarios para poder arrancar un servidor BLE, ya que se requiere la creación de al menos un servicio y una característica. En este caso se han instanciado dos servicios y dos características para el monitoreo individual de cada taquilla. Cada servicio y característica lleva asociado un identificador único denominado UUID. Este id se genera a través de la página web UUID Generator:

```
//Rele 1 APERTURA
#define SERVICE_UUID_1 "0df4012c-78f2-4ce3-b479-2230c9973a5b"
#define CHARACTERISTIC_UUID_1 "43eefbf8-c5a2-4ba3-b5d0-b162ca4e938d"

//RELE 2 APUERTURA
#define SERVICE_UUID_2 "d575da54-2004-4ce3-bc92-30d2fdc56b83"
#define CHARACTERISTIC_UUID_2 "27162251-6614-4847-8392-263f311d6551"
```

Ilustración 56. Creación de servicios y características

Seguidamente, se configuran cada uno de los pines empleados por el módulo relé:

```
#define RELE1 12
#define RELE2 13
```

Ilustración 57. Pines de cada relé

Cada característica definida necesita un "callback" asociado. Éstos se encargan de notificar cuando se ha escrito en su característica correspondiente. En este proyecto, se han empleado tres, dos correspondientes a cada una de las taquillas y uno que se encarga de controlar el momento en el que se conecta o se desconecta cada dispositivo móvil.

```
class MyCallbacks1: public BLECharacteristicCallbacks {

  void onWrite(BLECharacteristic *pCharacteristic1) {

    std::string value1 = pCharacteristic1->getValue();

    if (value1.length() > 0) {
        Serial.print("Valor asignado a la Taquilla 1: ");
        Serial.println(value1[0]);
        if (value1[0] == 48) {
          digitalWrite(RELE1, HIGH);
          delay(2000);
          digitalWrite(RELE1, LOW);
          Serial.print("Taquilla 1 Abierta con el valor: ");
          Serial.println(value1[0]);
        }
        else {
          Serial.println("Prueba con otro valor para abrir la Taquilla 1");
        }
class MyCallbacks2: public BLECharacteristicCallbacks {

  void onWrite(BLECharacteristic *pCharacteristic1) {
```

Ilustración 58. Creación del primer callback

```
    if (value2.length() > 0) {
        Serial.print("Valor asignado a la Taquilla 2: ");
        Serial.println(value2[0]);
        if (value2[0] == 49) {
          digitalWrite(RELE2, HIGH);
          delay(2000);
          digitalWrite(RELE2, LOW);
          Serial.print("Taquilla 2 Abierta con el valor: ");
          Serial.println(value2[0]);
        }
        else {
          Serial.println("Prueba con otro valor para abrir la Taquilla 2");

        }
      }
```

Ilustración 59. Creación del segundo callback

Cada uno de los “callbacks” llamados “MyCallbacks” contiene una función encargada de escribir los datos enviados en su correspondiente taquilla:

```
void onWrite(BLECharacteristic *pCharacteristic1)
void onWrite(BLECharacteristic *pCharacteristic2)
```

Ilustración 60. Función de escritura de los datos

En cambio, el “callback*”* *ll*amado “MyServerCallbacks*”* contiene las funciones necesarias para notificar la conexión y desconexión de los dispositivos al ESP32. Estas funciones se llaman “onConnect()” y “onDisconnect()”.

```
class MyServerCallbacks: public BLEServerCallbacks {

    // Funcion invocada cuando se conecte un cliente
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
      Serial.println("Dispositivo conectado al ESP - 32 mediante BLE");
    };

    // Funcion invocada cuando se desconecte un cliente
    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
      Serial.println("Dispositivo desconectado del ESP - 32");
      pServer->startAdvertising(); // Reiniciamos el 'advertising'
      Serial.println("\nEsperando dispositivos para conectarse al ESP - 32 ...");
    }
};
```

Ilustración 61. Funciones de conexión y desconexión del ESP32

Una vez definidos, se llevan a cabo dos configuraciones que aparecen en cada uno de los códigos de programación mencionados anteriormente.

En primer lugar, se encuentra la función “void *setup ()”*. En ella se llevan a cabo una serie de configuraciones necesarias en el entorno de ejecución:

- Se configura la comunicación serial y la velocidad de transmisión, que en este caso es la misma que la utilizada en la tecnología wifi. Además, se definen los pines de cada uno de los relés como pines de salida:

```
Serial.begin(115200);
pinMode(RELE1, OUTPUT);
pinMode(RELE2, OUTPUT);
```

Ilustración 62. Pines de salida de los relés

- Seguidamente, se define tanto el nombre del dispositivo que se empleará para la conexión BLE, como el servidor BLE, encargado de la devolución del ‘’callback’’ una vez se establezca la conexión con el dispositivo móvil.

```
BLEDevice::init("UMH ESP32 JASS");
BLEServer *pServer = BLEDevice::createServer();
pServer->setCallbacks(new MyServerCallbacks());
```

Ilustración 63. Definición de la red de acceso del modo BLE.

- Después, se establecen cada uno de los servicios asignándole su correspondiente identificador UUID, definido anteriormente.

```
// 3.1. Definimos un SERVICIO con el UUID definido anteriormente
BLEService *pService1 = pServer->createService(SERVICE_UUID_1);

// 3.2. Definimos un SERVICIO 2 con el UUID definido anteriormer
BLEService *pService2 = pServer->createService(SERVICE_UUID_2);
```

Ilustración 64. Configuración de servicios con su UUID

- Una vez realizado el paso anterior, se asignan sus correspondientes características y propiedades específicas, en concreto, de lectura y escritura.

```
pCharacteristic1 = pService1->createCharacteristic(
                    CHARACTERISTIC_UUID_1,
                    BLECharacteristic::PROPERTY_READ |
                    BLECharacteristic::PROPERTY_WRITE);

pCharacteristic2 = pService2->createCharacteristic(
                    CHARACTERISTIC_UUID_2,
                    BLECharacteristic::PROPERTY_READ |
                    BLECharacteristic::PROPERTY_WRITE);
```

Ilustración 65. Configuración de las propiedades de cada servicio

- A continuación, se inicia cada uno de los servicios con el fin de atender las solicitudes entrantes y realizar el envío continuo de paquetes publicitarios (advertising).

```
pService1->start();
pService2->start();

BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
pAdvertising->addServiceUUID(SERVICE_UUID_1);
pAdvertising->addServiceUUID(SERVICE_UUID_2);
```

Ilustración 66. Inicio de los servicios

- Tras lanzar los paquetes, se recogen las respuestas para ver que clientes han mostrado interés en ellos y se inicializan de forma física.

```
pAdvertising->setScanResponse(false);
pAdvertising->setMinPreferred(0x0);

 Iniciamos físicamente el 'advertising'
BLEDevice::startAdvertising();
Serial.println("\nEsperando dispositivos para conectarse al ESP - 32 ...");
```

Ilustración 67. Recogida de respuestas advertising

- Por último, se configuran cada uno de los "callbacks*"* asignados a las características definidas al principio del código:

```
pCharacteristic1->setCallbacks(new MyCallbacks1());
pCharacteristic2->setCallbacks(new MyCallbacks2());
```

Ilustración 68. Configuración de los callbacks de las características

En segundo lugar, se encuentra la función "loop()". En esta tecnología tampoco se lleva a cabo ninguna configuración:

```
void loop() {
  delay(2000);
}
```

Ilustración 69. Función loop

A continuación, se muestra el código completo empleado en la tecnología BLE:

```
#define RELE1 12
#define RELE2 13

class MyCallbacks1: public BLECharacteristicCallbacks {

  void onWrite(BLECharacteristic *pCharacteristic1) {

    std::string value1 = pCharacteristic1->getValue();

    if (value1.length() > 0) {
        Serial.print("Valor asignado a la Taquilla 1: ");
        Serial.println(value1[0]);
        if (value1[0] == 48) {
          digitalWrite(RELE1, HIGH);
          delay(2000);
          digitalWrite(RELE1, LOW);
          Serial.print("Taquilla 1 Abierta con el valor: ");
          Serial.println(value1[0]);
        }
        else {
          Serial.println("Prueba con otro valor para abrir la Taquilla 1");
        }
      }
  }
};

class MyCallbacks2: public BLECharacteristicCallbacks {

  void onWrite(BLECharacteristic *pCharacteristic2) {

    std::string value2 = pCharacteristic2->getValue();
```



```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
//#include <BLE2902.h>

BLEServer* pServer = NULL;
BLECharacteristic* pCharacteristic1 = NULL;
BLECharacteristic* pCharacteristic2 = NULL;

bool deviceConnected = false;
bool oldDeviceConnected = false;

#define SERVICE_UUID_1 "0df4012c-78f2-4ce3-b479-2230c9973a5b"
#define CHARACTERISTIC_UUID_1 "43eefbf8-c5a2-4ba3-b5d0-b162ca4e938d"

#define SERVICE_UUID_2 "d575da54-2004-4ce3-bc92-30d2fdc56b83"
#define CHARACTERISTIC_UUID_2 "27162251-6614-4847-8392-263f311d6551"
```

```
    if (value2.length() > 0) {
        Serial.print("Valor asignado a la Taquilla 2: ");
        Serial.println(value2[0]);
        if (value2[0] == 49) {
          digitalWrite(RELE2, HIGH);
          delay(2000);
          digitalWrite(RELE2, LOW);
          Serial.print("Taquilla 2 Abierta con el valor: ");
          Serial.println(value2[0]);
        }
        else {
          Serial.println("Prueba con otro valor para abrir la Taquilla 2");

        }
      }
  }
};

class MyServerCallbacks: public BLEServerCallbacks {

    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
      Serial.println("Dispositivo conectado al ESP - 32 mediante BLE");
    };

    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
      Serial.println("Dispositivo desconectado del ESP - 32");


      pServer->startAdvertising(); // Reiniciamos el 'advertising'
      Serial.println("\nEsperando dispositivos para conectarse al ESP - 32 ...");
    }
};

void setup() {

  Serial.begin(115200);
  pinMode(RELE1, OUTPUT);
  pinMode(RELE2, OUTPUT);

  BLEDevice::init("UMH ESP32 JASS");
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());

  BLEService *pService1 = pServer->createService(SERVICE_UUID_1);

  BLEService *pService2 = pServer->createService(SERVICE_UUID_2);

      pCharacteristic1 = pService1->createCharacteristic(
                         CHARACTERISTIC_UUID_1,
                         BLECharacteristic::PROPERTY_READ |
                         BLECharacteristic::PROPERTY_WRITE);

      pCharacteristic2 = pService2->createCharacteristic(
                         CHARACTERISTIC_UUID_2,
                         BLECharacteristic::PROPERTY_READ |
                         BLECharacteristic::PROPERTY_WRITE);


pCharacteristic1->setValue("Hello World");
pCharacteristic2->setValue("Hello World");
```

```
pService1->start();
pService2->start();

      BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
      pAdvertising->addServiceUUID(SERVICE_UUID_1);
      pAdvertising->addServiceUUID(SERVICE_UUID_2);

      pAdvertising->setScanResponse(false);
      pAdvertising->setMinPreferred(0x0);

      BLEDevice::startAdvertising();
      Serial.println("\nEsperando dispositivos para conectarse al ESP - 32 ...");

      pCharacteristic1->setCallbacks(new MyCallbacks1());
      pCharacteristic2->setCallbacks(new MyCallbacks2());
}


void loop() {
  delay(2000);
}
```

Ilustración 70. Código completo tecnología BLE

7.2. Implementación de la aplicación Android

En este apartado se describe el funcionamiento de la aplicación Android desarrollada en el programa Android Studio. Esta aplicación se emplea para la monitorización y control de dos taquillas inteligentes a través de tres tecnologías empleadas en la actualidad: WiFi, BLE y RFID.

7.2.1. Desarrollo Android basado en *fragments*

Activity y fragments

Una *activity* es una clase que representa una pantalla o una interfaz de usuario con la que el usuario puede interactuar. Cada pantalla o ventana en una aplicación Android es una instancia de la clase *Activity* y cada una de ellas puede tener uno o varios *fragments.* Además, su creación y gestión se lleva a cabo a través de ellas.

Por otro lado, los *fragments* son componentes de la interfaz de usuario que representan una parte de la interfaz de una *Activity*. Su finalidad es dividir la interfaz de usuario en partes más pequeñas y modulares, permitir la reutilización del código en diferentes partes de la aplicación y adaptar la interfaz de usuario a los diferentes tamaños de pantalla y orientaciones además de crear interfaces de usuario complejas e interactivas.

Android incorpora los *fragments* en Android 3.0 debido a la gran variedad de tamaños de pantalla que estaban surgiendo en el mercado y a la capacidad de orientación de la interfaz. Estas características permiten a las aplicaciones Android admitir diseños más dinámicos y flexibles en pantallas de mayor tamaño, como las *tablets*. Los *fragments* admiten esos diseños sin la necesidad de realizar cambios complejos en la jerarquía de vistas. Separar el diseño de una actividad en *fragments*, permite modificar el diseño de la actividad durante el tiempo de ejecución y mantener esos cambios en una pila de actividades controlada por la actividad.

En la imagen que se encuentra a continuación se muestra el ejemplo de cómo se pueden combinar dos *fragments* en una *activity* para un diseño de *tablet* y como sería la representación por separado para un diseño de teléfono:

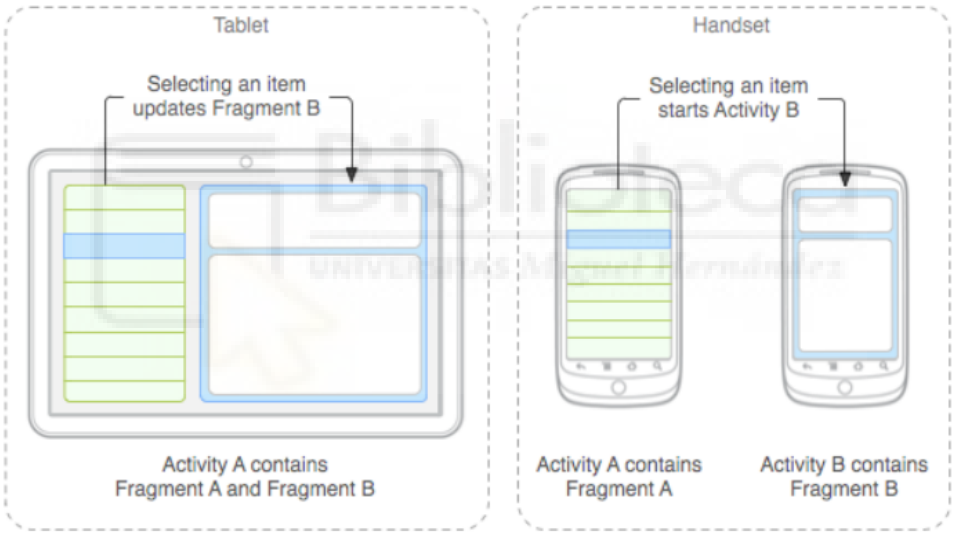

Ilustración 71. Representación gráfica de fragments en tablets y en teléfonos móviles

Ciclo de vida de un fragment

Un *fragment* siempre debe estar alojado en una *activity*, por lo que su ciclo de vida se ve afectado directamente al de la *activity* donde se ha definido:

- Al pausar una *activity* también se pausan todos los *fragments* que se encuentren alojados dentro de esta.

- Si una *activity* se elimina, todos los *fragments* que se encuentren alojados también se eliminan.

- Si la *activity* está en ejecución, es posible manipular cada *fragment*, de manera independiente, incluso eliminarlo.

Al hablar del ciclo de vida de un fragment, hace referencia a las etapas por las que atraviesa desde su creación hasta su destrucción. Al igual que una activity, un fragment puede existir en tres estados distintos:

1. Creado: Es el primer estado al que entra un fragment. En este estado, el fragment ha sido instanciado, pero aún no ha sido mostrado al usuario. Aquí se lleva a cabo la inicialización de variables y se prepara el fragment para su visualización.

2. Ejecución: En esta etapa el fragment es visible y el usuario puede interactuar con él. Aquí se ejecutan operaciones principales del fragment y se controlan eventos e interacciones con el usuario.

3. Detenido: El *fragment* ya no se encuentra visible debido a que la *activity* que lo aloja se ha detenido o la *activity* ya no lo contiene y se ha añadido a una pila de actividades. Un fragmento detenido aún está activo, pero ya no es visible para el usuario y se cerrará si finaliza la *activity*. Aquí se liberan recursos y se realizan tareas de limpieza.

En la mayoría de las aplicaciones se deben implementar al menos estos tres métodos en el ciclo de vida de cada fragment: “onCreate()”, “onCreateView()” y “onPause()”.

A continuación, se muestra el ciclo de vida de la actividad en el ciclo de vida del fragment.

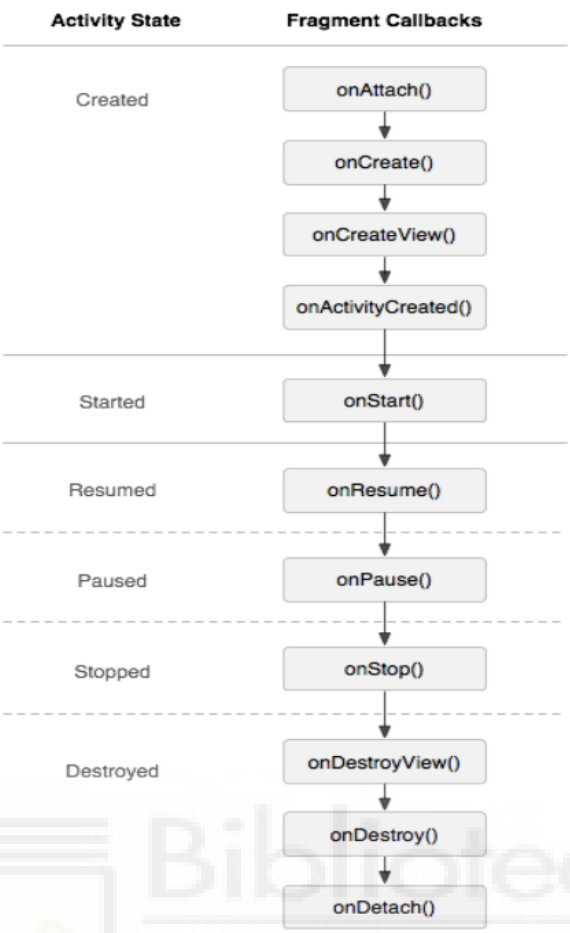

Ilustración 72. Efecto del ciclo de vida de la actividad en el fragment

En la siguiente tabla se encuentran las distintas funciones de la clase fragment:

OnAttach()	Se utiliza cuando el fragment se ha embebido a la actividad anfitriona.
onCreate()	Este método se emplea cuando el *fragment* se está creando. En él, se inician los componentes que se quieren guardar si el *fragment* se encuentra pausado o detenido.
onCreateView()	Es invocado en el momento en el que el *fragment* es dibujado por primera vez en la interfaz del usuario. En este método se crea la jerarquía de vistas que el fragment muestra en la pantalla.
onActivityCreated()	Se llama cuando termina el método onCreate() de la actividad anfitriona.

onStart()	Se nombra cuando el *fragment* se encuentra visible ante el usuario.
onResume()	Se ejecuta cuando el *fragment* se encuentra activo e interactuando con el usuario.
onPause()	Se invoca cuando se detecta que el usuario orienta el foco por fuera del *fragment*.
onStop()	Este método se utiliza cuando el *fragment* ya no es visible, debido a que la actividad anfitriona se ha detenido o se están gestionando una operación de *fragments* dentro de la actividad.
onDestroyView()	Se emplea cuando se elimina la jerarquía de vistas que se encuentra asociada con el *fragment*.
onDetach()	Se utiliza cuando el *fragment* ya no se encuentra asociado a la actividad.

Tabla 8. Funciones de la clase fragment

Creación de un fragment

Para crear un *fragment* es necesario crear una subclase de fragment. La clase *fragment*, al igual que la clase *activity*, contiene funciones de tipo *callback* para cuando una función es pasada como argumento a otra función. Para poder utilizar los fragments en la aplicación, hay que añadirlos a una activity ya existente o crear una nueva. Algunas de las funciones vienen definidas en la tabla 8.

7.3. Estructura de la aplicación

La estructura que se ha utilizado en el desarrollo de la aplicación Android sigue el patrón Modelo Vista VistaModelo (MVVM). En esta configuración se pueden diferenciar tres partes:

- Modelo: Permite el encapsulado de los datos de la aplicación y la lógica de negocios, permitiendo máxima integridad y consistencia de los datos. Dependiendo de los requisitos de la aplicación, ésta puede contener una o más clases modelo.

- Vista: Es la encargada de definir la interfaz de usuario de las aplicaciones. Representa la estructura, diseño, estilos de la interfaz de usuario y comportamientos visuales.

- VistaModelo: Proporciona la encapsulación de la lógica de presentación y los datos de la vista. También se encarga de gestionar las interacciones entre las vistas y los eventos de notificación a través de la aplicación de propiedades y los comandos en el ViewModel. Esta clase no contiene ninguna interfaz de usuario.

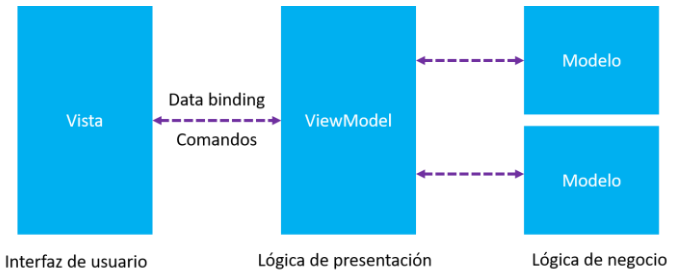

Ilustración 73. Esquema general de la arquitectura MVVM

VENTAJAS	DESVENTAJAS
- Lógica desacoplada de la interfaz de usuario - Fácil mantenimiento - Permite realizar pruebas unitarias para el modelo y vista-modelo, sin hacer referencia a la vista - Reutilización de componentes - Simplificación de los mantenimientos	- Adaptación a una estructura predefinida, aumentando la complejidad del sistema - Curva de aprendizaje superior a los otros modelos - Creación y mantenimiento de un mayor número de ficheros

Tabla 9. Ventajas y desventajas del uso de MVVM

7.3.1. Gestión de permisos

Los permisos que se implementan en las aplicaciones se utilizan para mantener segura la privacidad del usuario ya que proporcionan seguridad tanto en datos como acciones que se encuentran restringidas. Android clasifica los permisos en diferentes tipos:

- **Permisos en el momento de instalación:** Otorgan a la app acceso limitado a datos restringidos o permite realizar acciones que no afectan al sistema. Dentro de esta categoría se encuentran dos subtipos: permisos normales y permisos de firma.

- **Permisos de tiempo de ejecución o permisos peligrosos:** permiten a la aplicación acceso adicional a datos restringidos o permite realizar acciones que afectan significativamente al sistema.

- **Permisos especiales:** corresponden a operaciones particulares de la aplicación.

Permisos utilizados en la aplicación

Durante el desarrollo de la aplicación se han empleado una gran cantidad de permisos, los cuales están directamente relacionados con las tecnologías implementadas en el proyecto.

Por un lado, al utilizar la tecnología BLE, se han necesitado permisos relacionados con el bluetooth y la ubicación. Asimismo, para acceder a la red se han empleado permisos específicos asociados a la tecnología Wifi.

Por otro lado, se han requerido permisos que corresponden con el almacenamiento de la base de datos.

La mayoría de estos permisos tienen que permitirse por el usuario durante la ejecución de la aplicación. Por lo tanto, se solicitan en el momento que se lanza la aplicación, a menos que el usuario ya los haya concedido previamente.

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Ilustración 74. Permisos utilizados en la aplicación

A continuación, se muestra un fragmento del código utilizado donde aparecen implementados una serie de permisos que son necesarios para el monitoreo de la aplicación móvil:

```
private boolean comprobarRequisitosPrevios(){
    if(ActivityCompat.checkSelfPermission(requireActivity(), Manifest.permission.BLUETOOTH) != PackageManager.
            PERMISSION_GRANTED){
        Toast.makeText(getActivity(), "Error. Permiso Bluetooth no declarado", Toast.LENGTH_SHORT).show();
        return false;
    }

    if(ActivityCompat.checkSelfPermission(requireActivity(), Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.
            PERMISSION_GRANTED){
        Toast.makeText(getActivity(), "Error. Permiso de localización no declarado.", Toast.LENGTH_SHORT).show();
        return false;
    }

    if(ActivityCompat.checkSelfPermission(requireActivity(), Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.
            PERMISSION_GRANTED){
        Toast.makeText(getActivity(), "Error. Permiso de ubicación no declarado", Toast.LENGTH_SHORT).show();
        return false;
    }
```

Ilustración 75. Solicitud de permisos bluetooth, localización y ubicación

Tras acceder a las vistas de la aplicación, si los permisos previos no están concedidos, aparecen los siguientes mensajes:

- Si faltan permisos relacionados con el bluetooth y la ubicación, se despliega un mensaje indicando que es necesario solicitar dichos permisos.

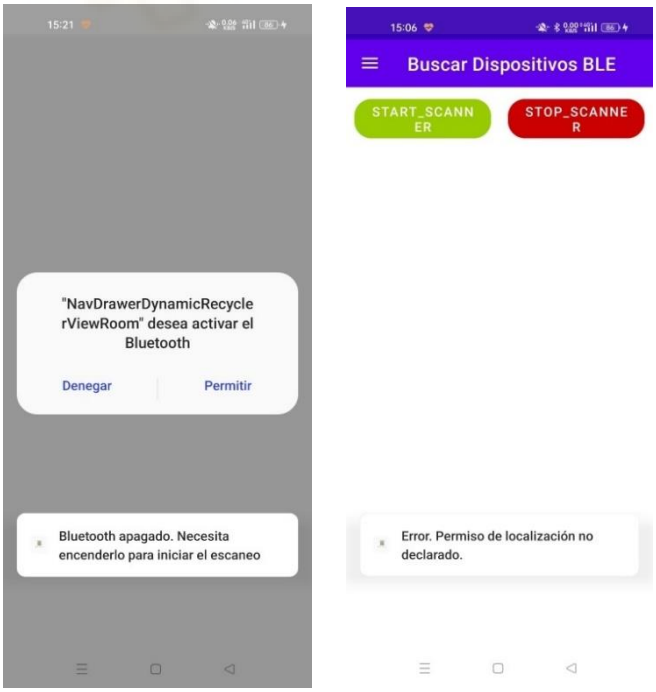

Ilustración 76. Permiso ubicación y bluetooth

7.3.2. *Fragments* de la aplicación

Durante el desarrollo de la aplicación Android, se han empleado una gran cantidad de *fragments*, cada uno de ellos desempeñando una función diferente. A continuación, se muestra un estudio de los diferentes fragments empleados en el desarrollo de la aplicación, destacando sus funciones y sus características.

Welcome fragment

Este fragment es uno de los más sencillos que se han implementado en la aplicación en comparación con otros empleados. Su objetivo principal es introducir y dar una vista general de la aplicación. Es por ello, que se muestra al inicio una vez ejecutada la aplicación.

```
public static WellcomeFragment newInstance(String param1, String param2) {
    WellcomeFragment fragment = new WellcomeFragment();
    Bundle args = new Bundle();
    args.putString(ARG_PARAM1, param1);
    args.putString(ARG_PARAM2, param2);
    fragment.setArguments(args);
    return fragment;
}

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Sin Factory, cogiendo la ViewModelFactory del objeto Application
    theAppFactory = ViewModelProvider.AndroidViewModelFactory.getInstance(getActivity().getApplication());
    if (getArguments() != null) {
        mParam1 = getArguments().getString(ARG_PARAM1);
        mParam2 = getArguments().getString(ARG_PARAM2);
    }
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    // Inflate the layout for this fragment
    return inflater.inflate(R.layout.fragment_wellcome, container,  attachToRoot: false);
}
```

Ilustración 77. Welcome Fragment

Cada *fragment* tiene asociado su *layout* correspondiente. Este está formado por 3 *textview*, dos *imageview* y una animación de una puerta. A continuación, se muestran algunos fragmentos que lo forman:

```
<pl.droidsonroids.gif.GifImageView
    android:id="@+id/gifImageView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/puerta"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.498"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.794" />
```

```
<TextView
    android:id="@+id/textView20"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="TAQUILLA"
    android:textColor="@color/marron_clarito"
    android:textSize="30sp"
    android:textStyle="bold"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.21" />
```

Ilustración 78. Animación y TextView1

```
<ImageView
    android:id="@+id/imageView5"
    android:layout_width="65dp"
    android:layout_height="72dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.216"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.289"
    app:srcCompat="@drawable/ic_ble_item" />
```

```
<ImageView
    android:id="@+id/imageView7"
    android:layout_width="71dp"
    android:layout_height="82dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.752"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.284"
    app:srcCompat="@drawable/icowifi" />
```

Ilustración 79. ImageViews

Buscar dispositivos BLE fragment

El siguiente fragment tiene como función principal gestionar la conectividad BLE, centrándose en el escaneo y búsqueda de dispositivos que utilicen esta tecnología. Para poder establecer la conexión, primero hay que obtener el adaptador del dispositivo y solicitar un objeto de tipo scanner.

```
bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
scanner = bluetoothAdapter.getBluetoothLeScanner();
private BluetoothLeScanner scanner;
private BluetoothAdapter bluetoothAdapter;
```

Ilustración 80. Fragment buscar dispositivos BLE

A continuación, se muestran los filtros utilizados para llevar a cabo la búsqueda de elementos y determinar cuáles de ellos deben añadirse al *Recyclerview* definido en el fragment.

```
scanSettings = new ScanSettings.Builder()
        .setScanMode(ScanSettings.SCAN_MODE_LOW_POWER)
        .setCallbackType(ScanSettings.CALLBACK_TYPE_ALL_MATCHES)
        .setMatchMode(ScanSettings.MATCH_MODE_AGGRESSIVE)
        .setNumOfMatches(ScanSettings.MATCH_NUM_ONE_ADVERTISEMENT)
        .setReportDelay(0L)
        .build();
```

Ilustración 81. Filtros de búsqueda

Una vez se presiona el botón *Start Scanner*, se inicia el proceso de escaneo. Dependiendo de la situación en la que se encuentre el dispositivo, el escaneo se realizará de manera inmediata o no.

En primer lugar, si el dispositivo no se encuentra conectado a una red, el escaneo se lleva a cabo a través de la función “startScanner()”.

En segundo lugar, si el dispositivo está enlazado a una red, se mostrará un mensaje de advertencia a través de un *AlertDialog*.

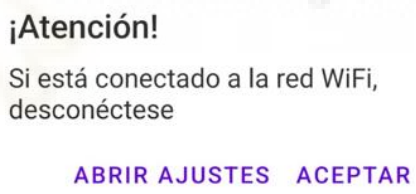

Ilustración X. AlertDialog interferencias red wifi

El objetivo de esta advertencia es prevenir posibles interferencias entre las tecnologías wifi y bluetooth. Una vez desconectado de la red, el escaneo se lleva a cabo a través de la función mencionada en el caso anterior.

```
AlertDialog.Builder builder = new AlertDialog.Builder(getContext());
AlertDialog alert = builder.setTitle("¡Atención!").setMessage("Si está conectado a la red WiFi, desconéctese")
        .setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
            }
        })
        .setNegativeButton("Abrir ajustes", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                startActivity(new Intent(Settings.ACTION_BLUETOOTH_SETTINGS));
            }
```

Ilustración 82. Proceso de escaneo BLE mediante AlertDialog

El método “iniciarScanner()” se encarga de verificar si el objeto “scanner” ha sido creado para iniciar el proceso de búsqueda. En caso de que el objeto no haya sido creado, este método se encarga de crearlo con el objetivo de iniciar la búsqueda.

Cada dispositivo que cumpla con los filtros establecidos se añadirá al *Recyclerview* que se encuentra asociado al *fragment*, indicando su nombre y su dirección MAC. Los *recyclerView* son los encargados de almacenar y mostrar vistas relacionadas con los datos de la aplicación, permitiéndonos crear listas con diferentes formatos según nuestras necesidades. Cada elemento individual de la lista está asociado a un objeto de las vistas.

La conexión con el dispositivo se lleva a cabo una vez se hayan añadido e impreso todos los dispositivos por pantalla.

```
private void conectarDispositivo(int position){
    BluetoothDevice device = bluetoothAdapter.
            getRemoteDevice(viewModel.getDevices().getValue().get(position).getDireccionMac());
    connectedDevice = device;
    @SuppressLint("MissingPermission") BluetoothGatt gatt = device.connectGatt(getContext(),
            autoConnect: false, bluetoothGattCallback, TRANSPORT_LE);
}
```

Ilustración 83. Conexión BLE

Cada dispositivo del *recyclerView* contiene métodos relacionados con la conexión, con el objetivo de obtener los servicios de cada uno de ellos. Una vez establecida la conexión con un dispositivo, se puede navegar a la ventana correspondiente para llevar a cabo la monitorización mediante la tecnología BLE, siempre que se haya aceptado el *alertDialog* que se muestra a continuación:

Ilustración 84. AlertDialog al realizar la conexión BLE

Por un lado, se muestra la parte del código empleada para obtener los servicios de los dispositivos:

```
@Override
public void run() {
    deviceConnectViewModel = new ViewModelProvider(requireActivity()).
            get(ControlTaquillaBleViewModel.class);
    deviceConnectViewModel.setBluetoothGatt(gatt);
    ArrayList<GattServiceModel> gattServiceModels = new ArrayList<>();

    for (int i = 2; i < services.size(); i++){
        gattServiceModels.add(new GattServiceModel(services.get(i)));
    }

    deviceConnectViewModel.setServices(gattServiceModels);
```

Ilustración 85. Obtención de los servicios de los dispositivos

Por otro lado, se muestra el fragmento de código que se utiliza para llevar a cabo el desplazamiento de vista tras la conexión:

```
ArrayList<GattServiceModel> gattServiceModels = new ArrayList<>();

for (int i = 2; i < services.size(); i++){
    gattServiceModels.add(new GattServiceModel(services.get(i)));
}
deviceConnectViewModel.setServices(gattServiceModels);
viewModel.connectDevice(connectedDevice.getAddress());
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
builder.setTitle("Conexión BLE realizada");
builder.setMessage("¿Desea ir a la pantalla 'Dispositivo conectado'?");
builder.setIcon(getContext().getDrawable(R.drawable.escaneo_ble));
builder.setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // User clicked OK button
        Navigation.findNavController(getView()).
                navigate(R.id.ControlTaquillaModoBle_dest, args: null, new NavOptions.Builder().
                        setPopUpTo(Navigation.findNavController(getView()).getGraph().
                                getStartDestinationId(), inclusive: true).build());
        //Navigation.findNavController(getView()).navigate(R.id.device_connect_dest);
    }
});
```

Ilustración 86. Desplazamiento de fragment tras la conexión

Control taquilla BLE fragment

Este *fragment* es el encargado de realizar el intercambio de datos entre un dispositivo móvil y el módulo ESP32, a través de la tecnología BLE. Esta monitorización se lleva a cabo a través de dos estados.

- Cuando el switch se encuentra en el estado “ischecked()”, se instancia la función “WriteCharacteristic()”. Esta función es la encargada de escribir en el módulo ESP32 que taquilla se ha de abrir, dependiendo del valor que se escriba en el módulo, se abrirá una u otra.
- Una vez que el switch vuelve a su estado inicial, se instancia la función “ReadCharacteristic()”. Esta función es la responsable de obtener las características que han sido leídas y añadirlas a la base de datos de la aplicación.

```
@Override
public Controlbles onClickItem(int position, boolean isChecked) {
    if(isChecked){
        writeCharacteristic(viewModel.getServices().get(position).getService().getCharacteristics().get(0));
    }
    else
        readCharacteristic(viewModel.getServices().get(position).getService().getCharacteristics().get(0));
    viewModel.changeActived(position, isChecked);

    return null;
}
```

Ilustración 87. Características de lectura y escritura BLE

En la función “WriteCharacteristic()” se configuran los servicios y las características asociadas a cada una de las taquillas. Además, se asigna a cada característica un valor específico que es escrito en el ESP32 una vez se active el estado “ischecked”. Este valor, que puede ser 0 o 1, depende de la taquilla.

```
@SuppressLint("MissingPermission")
public void writeCharacteristic(BluetoothGattCharacteristic bluetoothGattCharacteristic){

    viewModel.getBluetoothGatt().writeCharacteristic(bluetoothGattCharacteristic);
    BluetoothGattService mService1 = viewModel.
            getBluetoothGatt().getService(UUID.fromString("0df4012c-78f2-4ce3-b479-2230c9973a5b"));
    BluetoothGattCharacteristic characteristic1 = mService1.
            getCharacteristic(UUID.fromString("43eefbf8-c5a2-4ba3-b5d0-b162ca4e938d"));

    BluetoothGattService mService2 = viewModel.getBluetoothGatt().
            getService(UUID.fromString("d575da54-2004-4ce3-bc92-30d2fdc56b83"));
    BluetoothGattCharacteristic characteristic2 = mService2.
            getCharacteristic(UUID.fromString("27162251-6614-4847-8392-263f311d6551"));

    viewModel.getBluetoothGatt().writeCharacteristic(characteristic1);
    byte[] value1 = "0".getBytes(StandardCharsets.UTF_8);
    characteristic1.setValue(value1);

    viewModel.getBluetoothGatt().writeCharacteristic(characteristic2);
    byte[] value2 = "1".getBytes(StandardCharsets.UTF_8);
    String string2 = new String(value2, StandardCharsets.UTF_8);
    System.out.println(string2);
    characteristic2.setValue(value2);
}
```

Ilustración 88. Función WriteCharacteristic

El método “ReadCharacteristic()” se encarga de obtener las características leídas de nuestro objeto BluetoothGatt y posteriormente añadirlas a la base de datos para llevar un registro de la apertura de las taquillas.

```
@SuppressLint("MissingPermission")
public void readCharacteristic(BluetoothGattCharacteristic bluetoothGattCharacteristic){

    viewModel.getBluetoothGatt().readCharacteristic(bluetoothGattCharacteristic);

}
```

Ilustración 89. Función ReadCharacteristic

Registro apertura taquillas fragment

El objetivo de este fragment es registrar el momento en el que se lleva a cabo el monitoreo de las taquillas. Para ello, obtiene una lista de las características leídas de la base de datos, las cuales han sido previamente añadidas a través de la función “ReadCharacteristic()”.
En el caso de que se quiera eliminar los registros, dentro de este fragment se ha configurado un botón que permite eliminar todos los datos almacenados en la base de datos.

```
viewModel.getListaLecturas().observe(requireActivity(), list -> {
    taquillaReadModelList.clear();
    taquillaReadModelList.addAll(list);
    adapter.notifyDataSetChanged();
    });

//binding = FragmentSensorReadDataBaseBinding.inflate(inflater, container, false);

binding.btnhistorials.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
                AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
                builder.setTitle("¿Desea eliminar la base de datos?")
                        .setMessage("¡ATENCIÓN!, la base de datos se eliminará")
                        .setIcon(R.drawable.ic_warning)
                        .setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int id) {
                                viewModel.deleteTable();
                                Toast.makeText(getActivity(), text: "Base de datos eliminada",
                                        Toast.LENGTH_SHORT).show();
                            }
                        })
                        .setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int id) {
                                // User cancelled the dialog
                            }
                        }).create().show();
    }
});
```

Ilustración 90. Registro monitoreo de las taquillas

Buscar conexiones Wifi fragment

El siguiente fragment es el encargado de realizar la búsqueda de las redes Wifi que se encuentran disponibles. Para ello, se ha configurado un botón que se encarga de iniciar el proceso de búsqueda. Sin embargo, en caso de que el dispositivo tenga conectado el bluetooth, se activará un *alertDialog* con el fin de prevenir posibles conflictos con la conexión:

```
public class BuscarConexionesWifi extends Fragment {

    private FragmentBuscarConexionesWifiBinding binding;

    public BuscarConexionesWifi() {
        // Required empty public constructor
    }

    @Override
    public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        binding = FragmentBuscarConexionesWifiBinding.inflate(inflater,container, attachToParent: false);

        binding.btnAbrirAjustesWifi.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                startActivity(new Intent(Settings.ACTION_WIFI_SETTINGS));

            }
        });

        return binding.getRoot();
    }
}
```

Ilustración 91. Conectividad Wifi

Servidor web fragment

El objetivo principal de este fragment es la monitorización de las taquillas a través del servidor web. Para ello, se ha configurado un botón que cuando se presiona, permita redirigirnos a la dirección URL que se ha definido en su interior.

En caso de que el dispositivo móvil esté previamente conectado a la red del ESP32 definida, aparece un servidor Web encargado de la monitorización de las taquillas. Sin embargo, si la conexión no se ha llevado a cabo, al pulsar el botón, se redirige a la dirección IP establecida, pero sin cargar ni mostrar el servidor web.

```
binding.btnServidorWeb.setOnClickListener(new View.OnClickListener() {
    @SuppressLint("MissingPermission")
    @Override
    public void onClick(View view) {

        BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

        if(bluetoothAdapter != null && bluetoothAdapter.isEnabled()){

            AlertDialog.Builder builder = new AlertDialog.Builder(getContext());
            AlertDialog alert = builder.setTitle("¡Atención!").
                    setMessage("Si está conectado el adaptador Bluetooth del dispositiv...")
                    .setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                        }
                    })
                    .setNegativeButton("Abrir ajustes", new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                            startActivity(new Intent(Settings.ACTION_BLUETOOTH_SETTINGS));
                        }
                    }).create();
            alert.show();

        }else{
            Uri uri = Uri.parse(url);
            Intent intent = new Intent(Intent.ACTION_VIEW, uri);
            startActivity(intent);
        }
    }
});
```

Ilustración 92. Conectividad Servidor Web

Imágenes taquilla fragment

La funcionalidad de este fragment es proporcionar una vista general del proyecto a través de la aplicación Android. Para ello, se muestran varias imágenes de las taquillas en diferentes perspectivas.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    // Inflate the layout for this fragment
    return inflater.inflate(R.layout.fragment_imagenes_taquilla,
            container,  attachToRoot: false);
}
```

Ilustración 93. Imágenes taquilla

Debido a la simplicidad del código, se incluyen también algunos fragmentos del layout correspondiente:

```
<ImageView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:adjustViewBounds="true"
    android:src="@drawable/fototaquilla_1" />
```

```
<ImageView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:adjustViewBounds="true"
    android:src="@drawable/fototaquilla2" />
```

Ilustración 94. Fragmentos del layout Imágenes Taquilla

Ubicación taquilla fragment

El último fragment empleado se encarga de mostrar la ubicación de cada taquilla a través de la aplicación de mapas que se encuentra en el dispositivo. Esto se consigue a través de dos variables definidas llamadas “latitude” y “longitude”, las cuales se configuran con las coordenadas exactas de cada taquilla.

```
public void mapa1(){
    double latitude =38.345751;
    double longitude = -0.508598;

    String label = "Restaurante Las Arenas";

    String uriBegin = "geo:" + latitude + "," + longitude;
    String query = latitude + "," + longitude + "(" + label + ")";
    String encodedQuery = Uri.encode(query);
    String uriString = uriBegin + "?q=" + encodedQuery + "&z=16";
    Uri uri = Uri.parse(uriString);
    Intent mapIntent = new Intent(android.content.Intent.ACTION_VIEW, uri);
    startActivity(mapIntent);
}
```

Ilustración 95. Fragmento de código Ubicación Taquilla

7.3.3. Diseño de la base de datos

Con el fin de almacenar los valores recibidos de cada taquilla, se ha implementado una base de datos local a través del programa SQLite. Esta base de datos se organiza y estructura en diferentes clases:

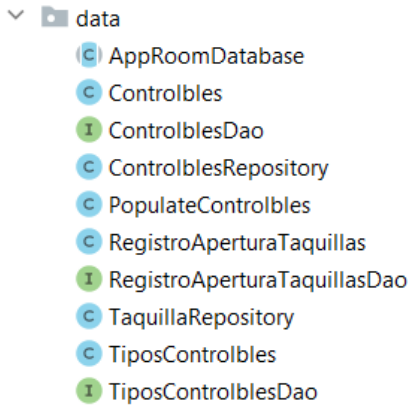

Ilustración 96. Distribución base de datos

La clase *AppRoomDatabase* es la más importante de las que se encuentran relacionadas con la base de datos. Esta clase desempeña dos funciones principales:

- Por un lado, se encarga de declarar cada una de las tablas y entidades que se van a utilizar en la base de datos, en ella se encuentran: *RegistroAperturaTaquillas, Controlbles y TiposControlbles*.
- Por otro lado, si la base de datos no se ha creado aún, esta clase se encarga de generarla según la configuración definida en @database.

```
@Database(entities = {RegistroAperturaTaquillas.class, TiposControlbles.class, Controlbles.class},
        version = 1, exportSchema = false)
@TypeConverters({DateConverter.class})
abstract public class AppRoomDatabase extends RoomDatabase {
    public abstract TiposControlblesDao tiposControlblesDao();
    public abstract ControlblesDao controlblesDao();
    public abstract RegistroAperturaTaquillasDao registroAperturaTaquillasDao();


    private static volatile AppRoomDatabase INSTANCE;
    private static final int NUMBER_OF_THREADS = 1;
    public static final ExecutorService databaseWriteExecutor =
            Executors.newFixedThreadPool(NUMBER_OF_THREADS);

    public static AppRoomDatabase getDatabase(final Context context) {
        if (INSTANCE == null) {
            synchronized (AppRoomDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                            AppRoomDatabase.class,  name: "controltaquilla")
                            .addCallback(sRoomDatabaseCallback)
                            .build();
                }
            }
        }
        return INSTANCE;
    }
```

Ilustración 97. Clase AppRoomDatabase

Dentro de la estructura de la base de datos también se encuentran cada una de las interfaces que van asociadas a las tablas y entidades que se han mencionado previamente.

La primera interfaz y la más importante es *RegistroAperturaTaquillasDao*. En ella, se codifica cada uno de los métodos que se utilizan en la base de datos, es decir, permite gestionar los registros a través de operaciones de lectura, escritura, eliminación y actualización. A través de ella, se registra el momento exacto en el que se lleva a cabo el monitoreo de las taquillas. Cada una de las funciones que se utilizan se encuentran definidas en la clase *TaquillaRepository*.

```
@Dao
public interface RegistroAperturaTaquillasDao {

    @Query("SELECT * FROM RegistroAperturaTaquillas ORDER BY dateTime DESC")
    List<RegistroAperturaTaquillas> getAll();


    @Insert(onConflict = OnConflictStrategy.IGNORE)
    void insertRead(RegistroAperturaTaquillas taquillaReadModel);

    @RawQuery
    List<RegistroAperturaTaquillas> getAllFiltered(SupportSQLiteQuery query);

    @Query("DELETE FROM RegistroAperturaTaquillas")
    void deleteTable();

    @Query("DELETE FROM RegistroAperturaTaquillas where id = 'dummy'")
    void dummyDelete();

    @Query("SELECT * FROM RegistroAperturaTaquillas ORDER BY id ASC")
    List<RegistroAperturaTaquillas> getPedidos();
}
```

Ilustración 98. Interfaz RegistroAperturaTaquillasDao

A continuación, se muestra una imagen del registro que se lleva a cabo a través de la interfaz mencionada previamente:

	id	uuid	taquilla	dateTime
1	26	27162251-6614-4847-8392-263	Abrir Taquilla 2	1686506088253
2	27	43eefbf8-c5a2-4ba3-b5d0-b16	Abrir Taquilla 1	1686506088658

Ilustración 99. Registo en base de datos

Dado que actualmente solo se está registrando el monitoreo de las taquillas a través de la tecnología BLE. Las siguientes interfaces llamadas *ControlblesDao* y

TiposControlblesDao se han implementado en el proyecto con la finalidad de incluir en un futuro, también las tecnologías Wifi y RFID, para facilitar la distinción de los diferentes tipos de tecnologías que se emplean en el proyecto. Las funciones empleadas se encuentran definidas en la clase *CotrolblesRepository*. Además, se ha definido una clase denominada *PopulateControlbles* con el objetivo de asociar los tipos de tecnologías empleados a las diferentes taquillas que se encuentran disponibles.

```
@Dao
public interface TiposControlblesDao {
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    void insert(TiposControlbles tiposControlbles);

    @Query("SELECT id FROM tiposcontrolbles WHERE tipo = :strTipoControlble")
    LiveData<List<Integer>> getTipoId(String strTipoControlble);

    @Query("SELECT id FROM tiposControlbles")
    LiveData<List<Integer>> getIdsTiposControlbles();

    @Query("SELECT * FROM tiposControlbles")
    LiveData<List<TiposControlbles>> getTiposControlbles();

    @Query("SELECT * FROM tiposControlbles WHERE tipo = :strTipoControlble")
    List<TiposControlbles> getTipoControlbles(String strTipoControlble);
}
```

Ilustración 100. Interfaz TiposControlblesDao

A continuación, se adjunta una imagen del diseño de la tabla de *TiposControlbles* donde se pueden apreciar los tres tipos de tecnologías mencionadas previamente:

id	tipo
1	Modo Bluetooth
2	Modo Wifi
3	Modo RFID

Ilustración 101. Tipos de tecnología en base de datos

La forma en que se ejecuta el código en la base de datos depende de la complejidad de los métodos utilizados. A continuación, se explican algunas anotaciones utilizadas en las operaciones de la base de datos:

@RawQuery: permite realizar consultas en tiempo de ejecución. Normalmente se emplea cuando se necesita ejecutar consultas complejas que no se pueden definir de antemano.

@Insert: permite insertar nuevos datos en la tabla de la base de datos.
‘SELECT’: permite obtener registros de la base de datos y ordenarlos según se requiera.
‘DELETE’: permite eliminar todos los registros de la tabla de la base de datos.

7.3.4. Otras funcionalidades

Además de las características principales de la aplicación mencionadas anteriormente, es importante destacar otros aspectos relacionados con su desarrollo. A continuación, se muestran otras funcionalidades empleadas en el desarrollo de la aplicación:

Fichero uuid_data.json

Cada uno de los servicios y características implementados en el proyecto tienen una identificación única llamada UUID. Esta UUID es uno de los valores que el controlador se encarga de leer, pero el formato que utilizan las UUID no es compatible con él. Para solucionar este problema, se ha diseñado una especie de traductor a partir de un archivo. json que se encarga de convertir cada UUID de los servicios y características a un formato compatible para el controlador.

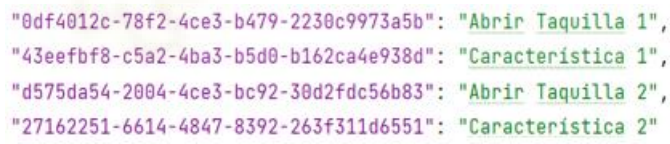

Ilustración 102. Uuid de los servicios y características BLE

Este archivo, a su vez se encuentra asociado a la clase UuisListSingleton. Esta clase se encarga de acceder al archivo mencionado y extraer la información contenida en él. Además, actúa como intermediario entre el archivo y otras partes del programa que requieren utilizar la información contenida:

```
public class UuidListSingleton {
    private static UuidListSingleton uuidList;
    private Map<String, String> uuidMap = new HashMap<>();

    public UuidListSingleton(Context context) {
        InputStream inputStream = context.getResources().openRawResource(R.raw.uuid_data);
        BufferedReader buffer = new BufferedReader(new InputStreamReader(inputStream));

        String cadena = "";
        String json = "";

        while(cadena != null){
            try {
                json += cadena;
                cadena = buffer.readLine();
            } catch (IOException e) {
                cadena = null;
            }
        }
        uuidMap = new Gson().fromJson(json,uuidMap.getClass());
    }

    public Map<String, String> getUuidMap() { return uuidMap; }
public static UuidListSingleton getUuidListSingleton(Context context){
    if(uuidList == null){
        uuidList = new UuidListSingleton(context);
    }
    return uuidList;
}
```

Ilustración 103. Clase UuidListSingleton

Esta clase se emplea en el método "onBindViewHolder()" de la clase *ServiciosTaquillasAdapter*. Su función es añadir los servicios definidos en el RecyclerView de la vista *ControlTaquillaBle*.

```
public void onBindViewHolder(@NonNull ServiciosTaquillasAdapter.ViewHolder holder, int position) {
    GattServiceModel service = gattServiceModels.get(position);
    holder.tvTaquilla.setText(UuidListSingleton.getUuidListSingleton(context).getUuidMap().
            get(service.getService().getUuid().toString()));

    if(service.isActivado()){
        holder.swServicios.setChecked(true);
    }
}
```

Ilustración 104. Función que añade los servicios definidos en el RecyclerView

DateConverter

A la hora de introducir la fecha de apertura de las taquillas se utilizan datos de tipo 'date'. Estos datos son demasiado complejos para que la base de datos pueda almacenarlos directamente, ya que solo admite objetos básicos como datos de tipo long.

La finalidad de esta clase es convertir estos datos complejos de tipo “date” a objetos sencillos de tipo “long”, de modo que puedan ser almacenados en la base de datos.

```java
public class DateConverter {
    private static DateFormat dfFechaHora = new SimpleDateFormat( pattern: "HH:mm:ss / dd-MM-YY");
    private static DateFormat dfHora = new SimpleDateFormat( pattern: "HH:mm:ss");
    private static DateFormat dfFecha = new SimpleDateFormat( pattern: "dd/MM/yy");
    @TypeConverter
    public static Date fromTimestamp(Long value) { return value == null ? null : new Date(value); }
    @TypeConverter
    public static Long dateToTimestamp(Date date) { return date == null ? null : date.getTime(); }

    public static String getFechaHoraFromDate(Date value) { return dfFechaHora.format(value); }

    public static String getHoraFromDate(Date value) { return dfHora.format(value); }

    public static String getFecha(Date value){ return dfFecha.format(value); }

    public static Date getDateFromString(String value){
        try {
            return dfFecha.parse(value);
        } catch (ParseException e) {
            return null;
        }
    }
}
```

Ilustración 105. Clase DateConverter

8. RESULTADOS

En este apartado se expondrá, por un lado, los resultados de la interconexión de cada uno de los componentes hardware empleados en el proyecto y, por otro lado, el resultado final de la aplicación Android, donde se mostrará tanto la interfaz como las diferentes funcionalidades que tiene cada una de las pantallas.

8.1. Interconexión Componentes Hardware

El propósito del proyecto es la monitorización de diferentes taquillas a través de tres tecnologías de comunicación como son el Wifi, el RFID y el BLE. Para poder llevarlo a cabo, se han empleado los siguientes componentes:

- Pestillos eléctricos solenoide
- Módulo ESP32
- Módulo Relé 2 canales
- Lector y tarjetas RFID
- Cableado interconexión
- Fuente de alimentación

A continuación, se muestran diferentes perspectivas del sistema completo montado:

ALZADO

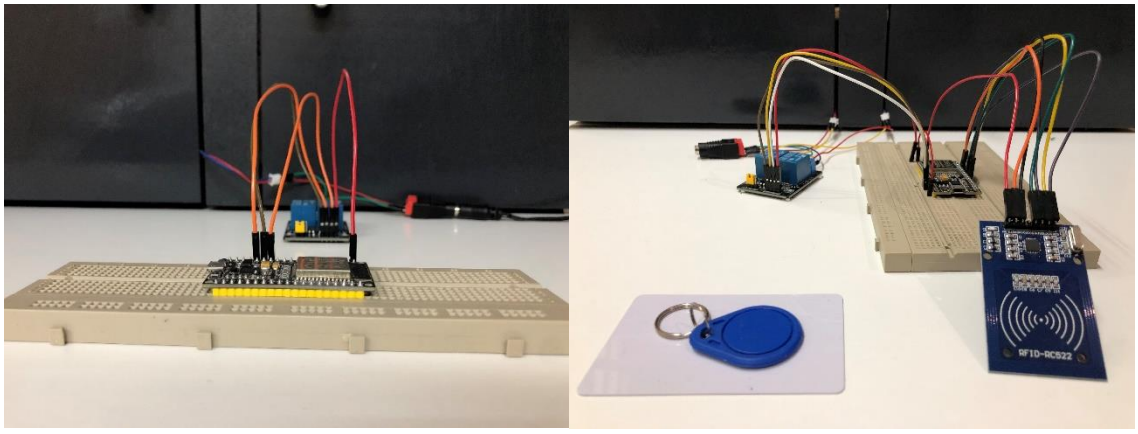

Ilustración 106. Alzado conexión componentes hardware

PERFIL

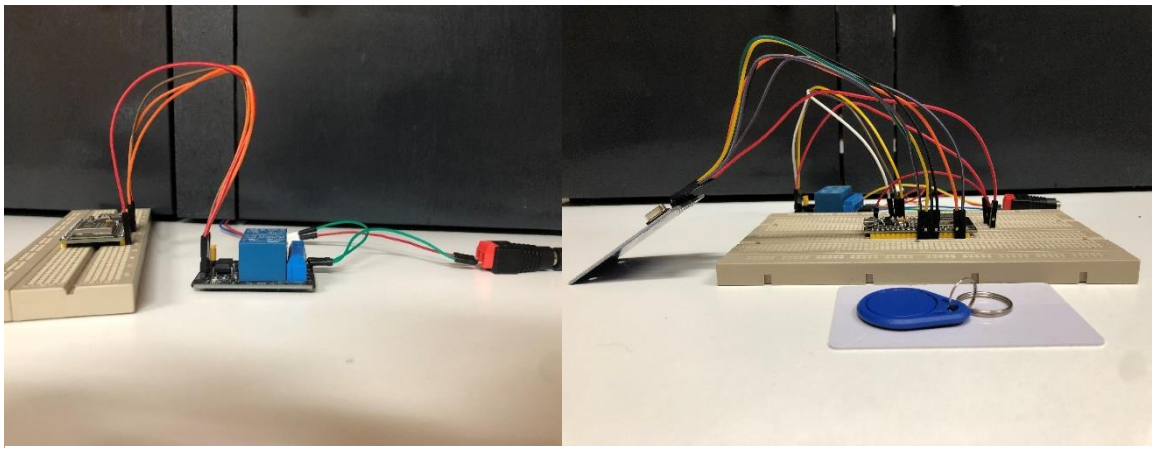

Ilustración 107. Perfil Conexión componentes Hardware

PLANTA

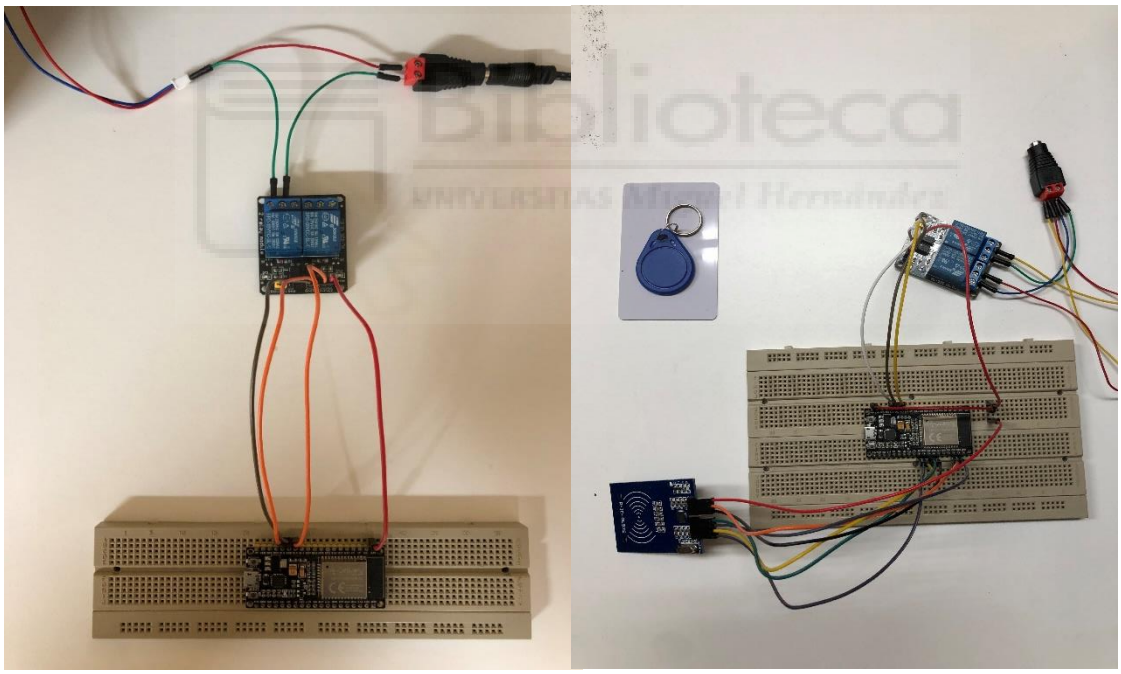

Ilustración 108. Planta conexión componentes hardware

8.2. Vistas aplicación Android

En este apartado se expondrá algunas de las pantallas de la aplicación Android para mostrar el resultado de éstas.

INICIO

Al iniciar la aplicación, nos dirige a la pantalla de bienvenida, en esta vista se muestra el nombre de la aplicación y un gif que simula la apertura y cierre de una puerta. También se observa en la parte superior izquierda un icono de desplegable para poder navegar por las diferentes pantallas de la aplicación. En esta vista, se solicitará al usuario el permiso de la ubicación, ya que es imprescindible para la monitorización de la taquilla a través de la tecnología BLE.

Ilustración 109.
Pantalla de bienvenida
de Aplicación Android

NAVEGAR POR LA APLICACIÓN

En el icono del desplegable mencionado previamente, permite navegar rápidamente por todas las vistas que componen la aplicación. Este menú se encuentra segmentado en 5 partes, donde cada una de éstas contienen funciones bien definidas.

1. Botón de retorno a la vista de inicio
2. Conectividad BLE
   2.1. Búsqueda de dispositivos BLE
   2.2. Monitorización de ambas taquillas
   2.3. Registro de apertura de cada una de las taquillas monitorizadas
3. Conectividad Wifi

3.1. Búsqueda de redes Wifi

3.2. Monitorización de las taquillas a través de un servidor web

4. Galería de imágenes de las taquillas
5. Localización de cada una de las taquillas

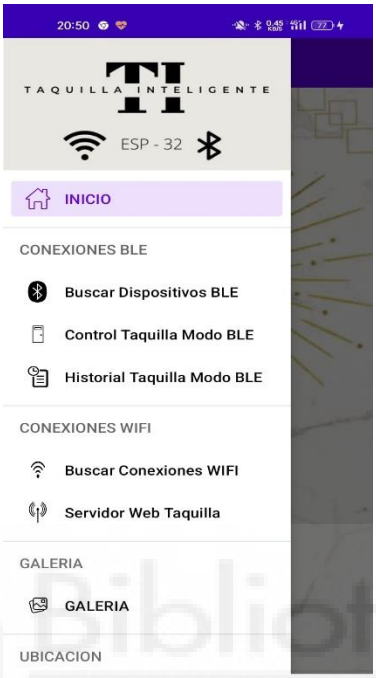

Ilustración 110. Navegación App Android

BUSCAR DISPOSITIVOS BLE

Al seleccionar el botón de buscar dispositivos BLE nos aparece una vista con dos opciones a elegir. El botón de color verde se encarga de iniciar la búsqueda de los dispositivos BLE que coincidan con las características definidas y el segundo botón se utiliza para parar la ejecución una vez nos aparece el dispositivo al que se quiere vincular.

En el momento que se realiza la búsqueda, aparecerán varias alertas, dos avisarán de que es necesario tener tanto el bluetooth como la ubicación activada y la otra recordará de que el Wifi debe estar desactivado. Una vez solucionado las diferentes alertas, los resultados se mostrarán en forma de listado, donde cada uno de los dispositivos tiene asociado un botón para realizar el emparejamiento.

Al realizar el emparejamiento, la vista muestra la última alerta. Ésta permite la opción de desplazarse directamente a la pantalla de dispositivo conectado.

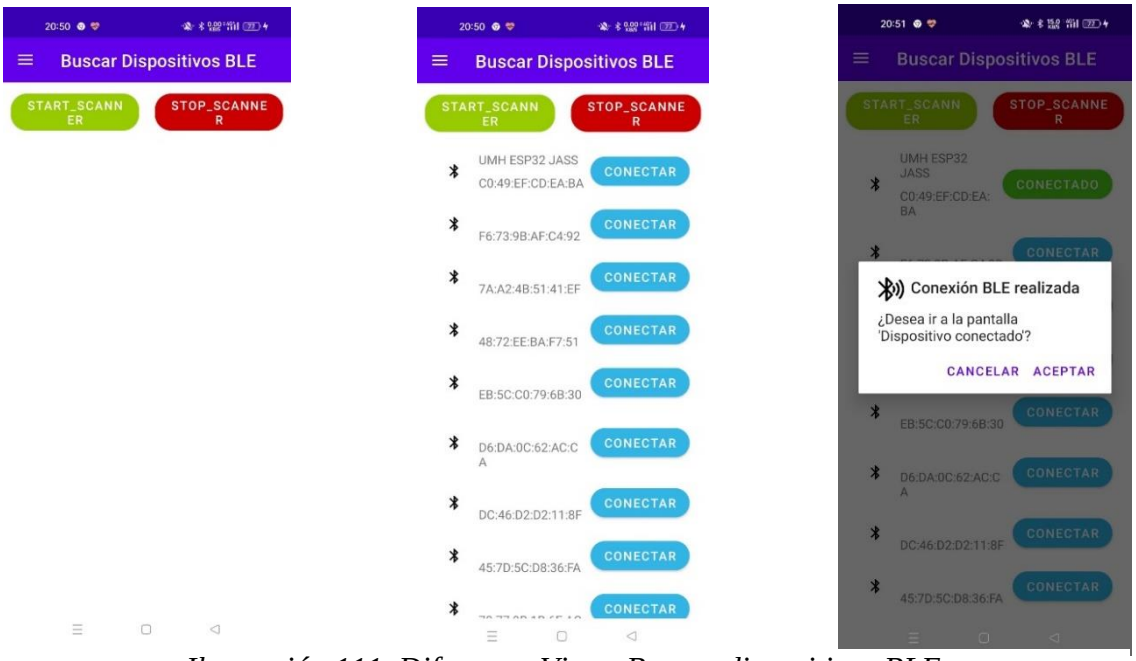

Ilustración 111. Diferentes Vistas Buscar dispositivos BLE

DISPOSITIVO CONECTADO

Una vez se ha realizado el emparejamiento de forma correcta y nos encontramos en la vista de dispositivo conectado, aparecerán cada uno de los servicios y características definidos cuyo objetivo es el monitoreo de ambas taquillas. Cada uno de estos servicios tiene asociado un deslizador el cual se encargan de la apertura de su correspondiente taquilla.

Ilustración 112. Vista de dispositivo conectado BLE

HISTORIAL

En el momento que se ha realizado la monitorización de las taquillas, se realiza un registro en una base de datos local. Éste se podrá visualizar en la pantalla de historial, donde se mostrará un listado en el que se define en columnas tanto la taquilla como la hora y el día de la monitorización. También, en el caso de querer realizar un borrado total de la base de datos local, en la parte inferior se encuentra un botón encargado de ello, donde nos avisará si estamos seguros de realizar dicha acción.

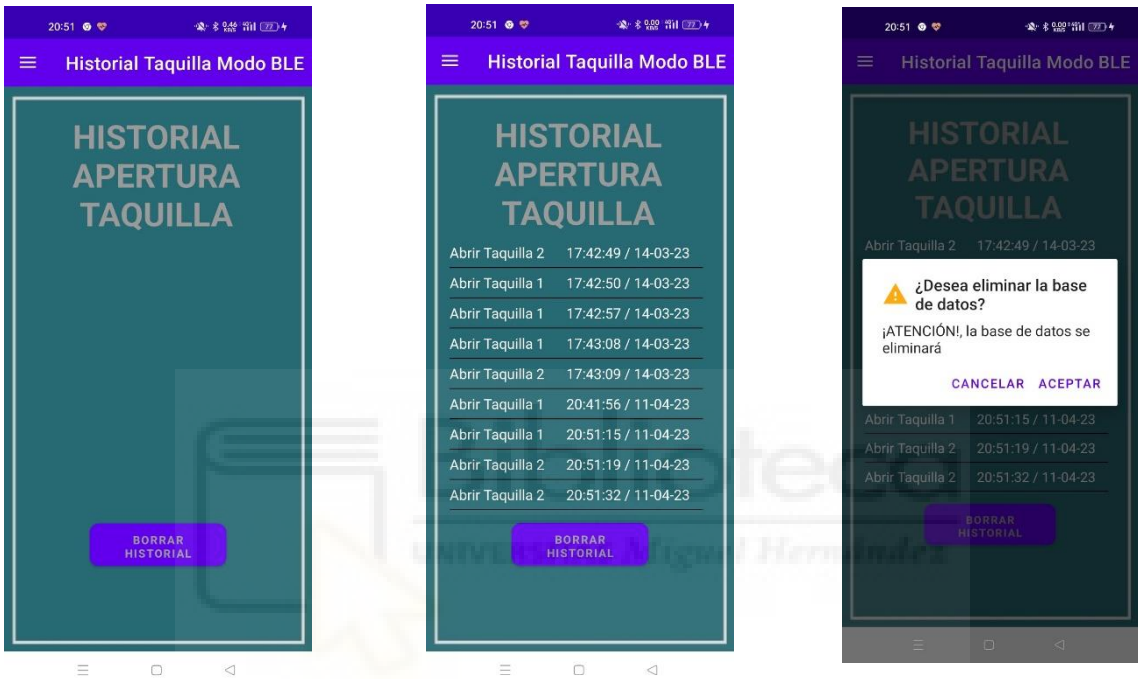

Ilustración 113. Diferentes vistas del historial

BUSCAR CONEXIONES WIFI

Cuando se quiere realizar la monitorización a través de la tecnología Wifi es necesario desplazarnos hacia la vista de buscar conexiones wifi, en ella se muestran, por un lado, el nombre de la red y la contraseña para realizar la conectividad y, por otro lado, en la parte inferior, el botón que nos envía al listado de las diferentes redes a enlazar. En el caso, de tener habilitado el bluetooth nos saltará una alerta para desactivarlo, y en el caso, de no tener habilitado el wifi, nos aparecerá un aviso de que es necesario habilitarlo.

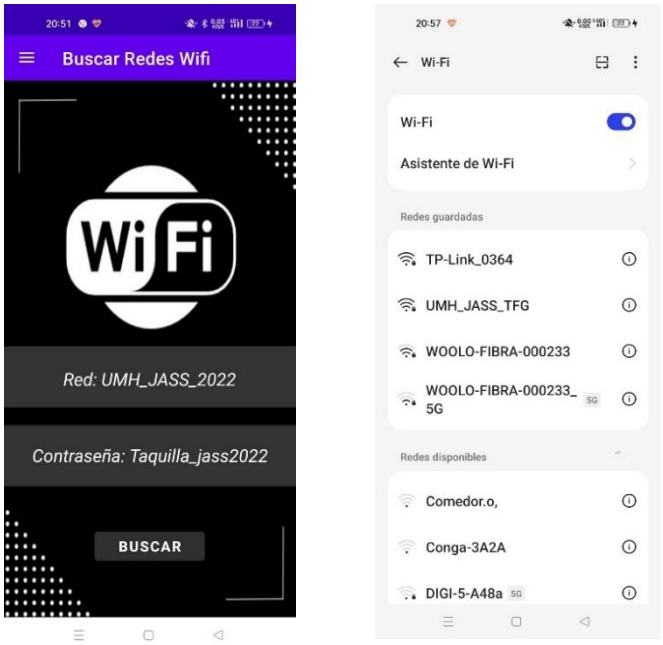

Ilustración 114. Diferentes vistas buscar conexiones wifi

SERVIDOR WEB

Una vez efectuada la conectividad con la red wifi correspondiente se podrá realizar el monitoreo de la taquilla desplazándose a la siguiente vista denominada *Servidor web*. En ella, se muestra en la parte central de la vista la dirección IP que enviará al usuario en el caso de estar conectados y, en la parte inferior, el botón que será el encargado de desplazarnos a dicha dirección IP para el monitoreo ambas taquillas. En ésta se encuentran varios deslizadores encargados de llevar a cabo el control. En el caso de no estar conectados a la red, nos enviará a la misma dirección IP, pero no se mostrará la página web definida.

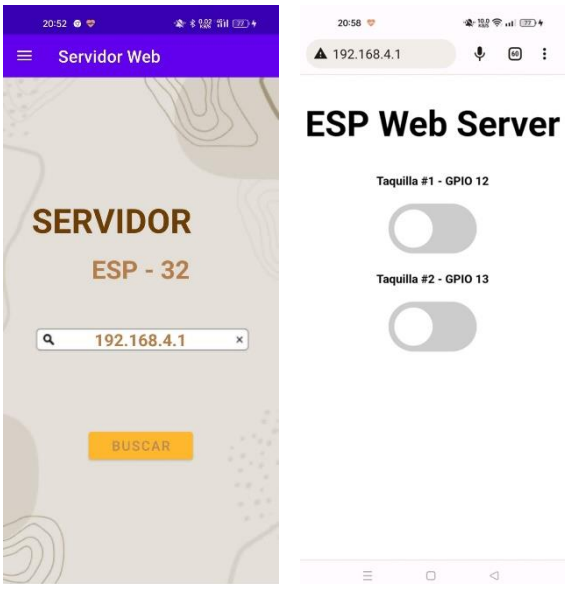

Ilustración 115. Vista de la página Web

GALERIA

En esta vista se muestran varias imágenes en diferentes perspectivas del diseño de las taquillas, tanto abiertas como cerradas. Para poder observar todas las imágenes de forma correcta, la pantalla se puede deslizar hacia arriba y abajo.

Ilustración 116. Galería de imágenes de la taquilla

UBICACIÓN

Por último, se encuentra la vista de la localización. En ella se muestra en la parte superior de la vista un contacto de soporte, éste se utiliza en el caso de tener alguna incidencia con las taquillas. A continuación, se indica cada una de las direcciones de las taquillas, esta opción puede visualizarse en forma de texto o con un botón que permite abrir directamente la ubicación de las taquillas a través de la aplicación de *Google Maps*.

Ilustración 117. Vista de la ubicación de las taquillas

9. PRESUPUESTO Y PLANIFICACIÓN

En este anexo se detallan la planificación y el presupuesto de la realización de este proyecto.

PRESUPUESTO

En esta tabla se muestra el presupuesto necesario para llevar a cabo el desarrollo del proyecto, teniendo en cuenta los costes de los componentes hardware:

MATERIALES	**PRECIO**
ESP-WROOM-32	11,49€
Módulo Relé 2 canales	8,49€
Protoboard x2	1,78€
Pestillo eléctrico x2	25,02€
Fuente de alimentación 12V	6,49€
Conector Jack Hembra	3,99€
Cable de Alimentación microUSB	3,65€
Lector y tarjeta RFID	6,49€
Cables de interconexión	0,79€
Maqueta taquilla madera	15€
TOTAL	**83,19€**

Tabla 10. Costes del proyecto

Los costes del proyecto podrían reducirse considerablemente de cara a futuros proyectos similares si la mayoría de los diferentes componentes se adquieren a proveedores como Aliexpress o similares.

El equipamiento y los softwares empleados en este proyecto no han requerido ningún coste económico, debido a que, por un lado, se ha optado por utilizar herramientas de código abierto y software libre, y, por otro lado, se ha recurrido a dispositivos electrónicos de uso común, como ordenadores portátiles y dispositivos móviles.

PLANIFICACIÓN

En las siguientes tablas se recogen tanto el número de días (jornadas de 4 horas) como las dependencias requeridas para la realización del proyecto:

Realización prototipo taquilla

	DESCRIPCIÓN	**DEPENDENCIA**	**DURACIÓN**
1	Investigación de los componentes hardware		7 días
2	Investigación del software		30 días
3	Construcción (cableado y componentes Hardware)	1	15 días
4	Diseño del algoritmo (Arduino IDE)	2	35 días
5	Codificación y comprobación	3,4	14 días

Tabla 11. Realización prototipo taquilla

Desarrollo Aplicación Android

	DESCRIPCIÓN	**DEPENDENCIA**	DURACIÓN
1	Diseño de la estructura de la App		
2	Desarrollo de la aplicación a través de Android Studio	1	50 días
3	Comprobación del funcionamiento	1,2	15 días

Tabla 12. Desarrollo aplicación Android

Integración y ejecución del proyecto

	DESCRIPCIÓN	**DEPENDENCIA**	**DURACIÓN**
1	Integración de la App móvil y el prototipo de la taquilla	Tabla 1 y 2	28 días
2	Comprobación del correcto funcionamiento y entrega del proyecto	Tabla 1 y 2	35 días

Tabla 13. Integración y ejecución del proyecto

10. CONCLUSIÓN Y LÍNEAS FUTURAS

En este apartado se recogen las conclusiones del proyecto, así como posibles desarrollos futuros para la aplicación realizada.

CONCLUSIÓN

En este proyecto se ha llevado a cabo tanto el desarrollo de la aplicación Android como el funcionamiento de los diferentes elementos empleados en el proyecto. La conclusión a la que se ha llegado es que se ha visto cumplido el principal objetivo del proyecto, que era la monitorización de las taquillas a través de las tres tecnologías utilizadas mediante una aplicación móvil Android.

La aplicación desarrollada, a pesar de ser completamente funcional, puede utilizarse como apoyo para futuras implementaciones que se puedan realizar sobre ella.

En lo que respecta a la perspectiva personal, existen varias conclusiones que se pueden destacar. En primer lugar, se ha tomado conciencia de la complejidad y esfuerzo que supone desarrollar una aplicación Android. Además, se ha logrado aumentar los conocimientos relacionados con el desarrollo de aplicaciones móviles Android a través del lenguaje Java. Asimismo, se ha brindado la ocasión de someter a prueba una diversidad de aptitudes y saberes adquiridos a lo largo del programa académico del estudiante.

En definitiva, este proyecto ha supuesto un hito importante en mi trayectoria personal y profesional, y estoy convencido de que los conocimientos adquiridos y la experiencia obtenida me serán de gran ayuda en futuros proyectos y desafíos en este ámbito.

LÍNEAS FUTURAS

A pesar de haberse cumplido con la totalidad de los objetivos y requerimientos establecidos en la memoria del proyecto, se presenta a continuación una lista de mejoras que han sido ideadas durante el desarrollo del proyecto, con el propósito de ser consideradas para futuros desarrollos:

- Incluir en la base de datos de la aplicación el registro de la apertura de las taquillas a través de las tecnologías Wifi y RFID, con el fin de poder visualizar en la vista del historial, la monitorización de las taquillas a través de las tres tecnologías empleadas en el proyecto.

- Se propone la migración de la aplicación Android hacia el lenguaje de programación Kotlin, con el fin de estar a la vanguardia con las últimas tendencias en el desarrollo de aplicaciones móviles, debido a su reconocida capacidad para mejorar la eficiencia y legibilidad del código fuente.

- Incorporar en la aplicación móvil un sistema de inicio de sesión que requiera introducir un nombre de usuario y una contraseña para garantizar la privacidad y seguridad de los usuarios de la aplicación.

- Integrar los diferentes componentes del sistema hardware en una misma PCB (Placa de circuito impreso), con el objetivo de proporcionar una mayor integración y compactación de los componentes electrónicos para mejorar la eficiencia y confiabilidad del circuito.

11. BIBLIOGRAFÍA

1. Brea, Victor Manuel. (2018, julio). Internet de las cosas. Horizonte 2050. *Instituto Español de Estudios Estratégicos.* https://www.ieee.es/Galerias/fichero/docs_investig/2018/DIEEEINV17-2018_Internet_de_las_Cosas_Horizonte_2050.pdf

2. Salazar, J, Silvestre, S. Internet de las cosas. *European Virtual Learning Platform for Electrical and Information Engineering.* https://upcommons.upc.edu/bitstream/handle/2117/100921/LM08_R_ES.pdf

3. Fortino, G., Guerrieri, A., Pace, P., Savaglio, C., & Spezzano, G. (2022). IoT Platforms and Security: An Analysis of the Leading Industrial/Commercial Solutions. *Sensors*, *22*. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8952798/pdf/sensors-22-02196.pdf

4. Gokhale, P., Bhat, O., & Bhat, S. (2018). Introduction to IOT. *IARJSET*, *5*(1). https://www.researchgate.net/profile/Omkar-Bhat/publication/330114646_Introduction_to_IOT/links/5c2e31cf299bf12be3ab21eb/Introduction-to-IOT.pdf

5. Vivancos, C., González, R., & Hidalgo, E. (2000). *Diseño y desarrollo de un sistema de precintado remoto sobre FreeRTOS para sistemas IoT.* Carmen Vivancos. https://biblus.us.es/bibing/proyectos/abreproy/93899/fichero/TFG-3899+VIVANCOS+PORRAS%2C+CARMEN.pdf

6. Sanaguado, J., Caina, A., & Delgado, J. (2020). *"DESARROLLO DE UNA APLICACIÓN MÓVIL PARA EL MONITOREO DE PACIENTES CON HIPERTENSIÓN ARTERIAL UTILIZANDO LA TECNOLOGÍA ESP32"*. http://dspace.unach.edu.ec/bitstream/51000/6543/1/DESARROLLO%20DE%20UNA%20APLICACIÓN%20MÓVIL%20PARA%20EL%20MONITOREO.pdf

7. Carmenate, J. G. (2022, 7 marzo). *ESP32 Wifi + Bluetooth en un solo lugar*. Programar fácil con Arduino. https://programarfacil.com/esp8266/esp32/

8. Pravalika, V., & Rajendra, C. (2019). Internet of Things Based Home Monitoring and Device Control Using Esp32. *International Journal of Recent Technology and Engineering*, *1*(14). https://www.researchgate.net/profile/Rajendra-Prasad-Ch/publication/334226986_Internet_of_Things_Based_Home_Monitoring_and_Device_Control_Using_Esp32/links/5d1de7fe458515c11c0ff074/Internet-of-Things-Based-Home-Monitoring-and-Device-Control-Using-Esp32.pdf

9. Bruno, A. (2019). *ESP32 NODE MCU* (1.ª ed.). https://www.microelectronicash.com/downloads/ESP32_MANUAL.pdf

10. Galán, J. (2020). *DISEÑO DE UNA CERRADURA INTELIGENTE E INTEGRACIÓN EN OPENHAB* (1.ª ed.). https://oa.upm.es/66679/8/TFG_JAIME_GALAN_MARTINEZ.pdf

11. Garijo, L. (2016). *DISEÑO Y ESTUDIO DE UN SISTEMA DE COMUNICACIÓN INALÁMBRICO BASADO EN TECNOLOGÍA BLUETOOTH LOW ENERGY CON DESARROLLO DE PROTOCOLO PROPIO DE ENRUTAMIENTO.* (1.ª ed.). https://academica-e.unavarra.es/xmlui/bitstream/handle/2454/21877/TFG%20Protocolo%20de%20Enrutamiento%20-%20Luis%20Garijo.pdf;jsessionid=BD5C3B6390ABC56E7EC4F56691951A15?sequence=1

12. Montoya, S., & Amor, M. (2018). *DESCUBRIENDOS DISPOSITIVOS IoT* (1.ª ed.). https://riuma.uma.es/xmlui/bitstream/handle/10630/17102/StevenmontoyavargasMemoria.pdf?sequence=1&isAllowed=y

13. Valenzuela, J. (1999). *Bluetooth 4.0 Low Energy: Análisis de las prestaciones y aplicaciones para la automoción* (1.ª ed.). https://upcommons.upc.edu/bitstream/handle/2117/82702/memoria.pdf?sequence=1&isAllowed=y

14. Woolley, M. (2020). *Bluetooth Core Specification Version 5.2 Feature Overview* (2.ª ed.). https://www.bluetooth.com/wp-content/uploads/2020/01/Bluetooth_5.2_Feature_Overview.pdf

15. Benito, A., & Villadangos, J. (1999). *Desarrollo de aplicaciones para IoT con módulo ESP32* (1.ª ed.). https://ebuah.uah.es/dspace/bitstream/handle/10017/35420/TFG_Benito_Herranz_2019.pdf?sequence=1&isAllowed=y

16. *Protocolo TCP/IP*. (s. f.). https://www.mheducation.es/bcv/guide/capitulo/8448199766.pdf

17. Santos, S., & Santos, S. (2019). ESP32 Access Point (AP) for Web Server | Random Nerd Tutorials. *Random Nerd Tutorials*. https://randomnerdtutorials.com/esp32-access-point-ap-web-server/

18. Jiménez, M. V. (s. f.). *Vista de Relés electromagnéticos y electrónicos. Parte I: relés y contactores*. Vivat Academia. https://www.vivatacademia.net/index.php/vivat/article/view/373/689

19. Redacción. (2022, 31 marzo). Protoboard ¿Qué es, Cómo funciona y cómo se usa? *DescubreArduino*. https://descubrearduino.com/protoboard/

20. Córcoles, C., Rollo, J., & Lambán, M. (2003). *Análisis de la operatividad y comportamiento de la tecnología RFID UHF en líquidos* (1.ª ed.). https://zaguan.unizar.es/record/96482/files/TAZ-TFG-2020-3431.pdf

21. Espejo, C., & Rodríguez, M. (2018). *Estudio de las aplicaciones de la tecnología RFID y su grado de implantación.* (1.ªed.). https://core.ac.uk/download/pdf/190375379.pdf

22. Aguilar, M., Ormset, A., Ríos, J., Hofboer, R., Zamorano, D., & García, B. (s. f.). *HISTORIA Y TIPOS DE LENGUAJE DE PROGRAMACIÓN*. https://recursos.salonesvirtuales.com/assets/bloques/marina_aguilar_HistoriayTiposdeLenguajesdeProgramacion.pdf

23. Pérez, A., Aldea, M., & González, M. (2020). *Soporte de Aplicaciones de Tiempo Real en Dispositivos Móviles* (1.ª ed.). https://repositorio.unican.es/xmlui/bitstream/handle/10902/21642/Tesis%20APR.pdf?sequence=1&isAllowed=y

24. Martínez, A. (2021). *ESTUDIO COMPARATIVO DE LAS MEJORAS DEL LENGUAJE DE PROGRAMACIÓN KOTLIN Y EL LENGUAJE JAVA EN EL DESARROLLO DE APLICACIONES ANDROID.* (1.ª ed.). http://dspace.utb.edu.ec/bitstream/handle/49000/10535/E-UTB-FAFI-SIST-000244.pdf?sequence=1&isAllowed=y

25. Casado, R. (s. f.). *Introducción a HTML* (1.ª ed.). https://gredos.usal.es/bitstream/handle/10366/139647/BISITE_CasadoVaraR_HTML.pdf?sequence=1&isAllowed=y