ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA DE COMPUTADORES

**Disvise: Arquitectura distribuida para la gestión de notificaciones basadas en el microcontrolador ESP32**

**Disvise: Distributed architecture for notification management based on ESP32 microcontroller**

Realizado por

**David Gómez Delgado**

Tutorizado por

**Alberto Gabriel Saldero Hidalgo**

Departamento

**Lenguaje y Ciencias de la Computación**

UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2017

Fecha defensa: 7 de julio de 2017

Resumen

Realización de un proyecto basado en el microcontrolador ESP32 conectado vía WIFI con peticiones http a una API REST con el objetivo de conseguir un dispositivo portable de avisos para coordinación de un equipo en un espacio controlado, así como poder guardar un registro de todas las peticiones para futuros estudios de coordinación.

La memoria contendrá un diseño en profundidad con diferentes explicaciones para todo el código del proyecto, con diagramas explicativos tanto de la estructura del código como la interacción entre los distintos componentes del sistema distribuido y fragmentos de código importante, así como los requisitos funcionales y no funcionales.

Al final del proyecto se encuentra un manual de instalación donde se recalcarán los parámetros del código que deben ser modificados así como toda la tecnología necesaria, junto con un manual de usuario explicativo de cómo funciona Disvise.

\*Completar para 150\*

**Palabras clave:**

ESP32, microcontrolador, API, base de datos, programa cargable.

Abstract

Development of a project based on the ESP32 microcontroller connected via Wi-Fi through HTTP requests to a API REST, with the aim of creating a portable notification device for coordinating a team within a controlled environment, as well as storing a log of all requests for future coordination analysis.

The report will include detailed explanations of all the project’s code, accompanied by explanatory diagrams illustrating both the structure of the code and the interaction between the various components of the distributed system, along with relevant code snippets.

At the end of the report, there is an installation manual highlighting the code parameters that require modification, as well as the complete list of necessary technologies. Additionally, a user manual is provided, explaining how the Disvise system operates.

**Keywords:**

ESP32, microcontroler, API, data base, flashable program.

Índice

Resumen 1

Abstract 1

Índice 1

Introducción 1

1.1 Motivación 1

1.2 Objetivos y tecnologías a usar 1

1.3 Estructura de la memoria 3

Referencias 9

Manual de Instalación 11

Requerimientos: 11

1

Introducción

1.1 Motivación

Muchas empresas y organizaciones están organizadas en distintos puestos dependientes entre ellos para poder cumplir su función. Cada puesto, conformado por distintas personas, debe coordinarse en determinados momentos, lo que en ocasiones de cierta urgencia genera momentos de caos debido a los retrasos debidos a tener que hacer el aviso en persona o por un teléfono móvil. Algunas empresas emplean redes sociales para coordinar a los trabajadores. Esta solución, que está lejos de ser la adecuada, es inviable en la mayoría de los casos, pues en muchas organizaciones se tiene prohibido el uso de móviles en los

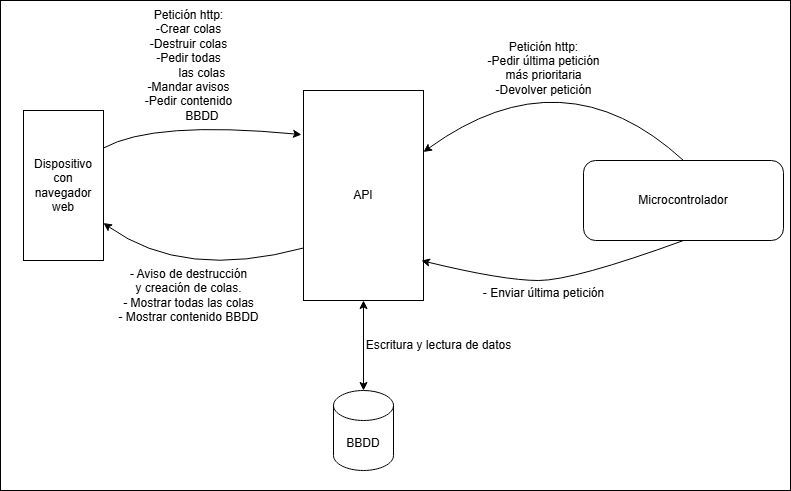
puestos de trabajo.

En puestos como cajero y reponedor de supermercados los avisos suelen ser realizados a través de un micrófono, con una calidad de audio pésima. A este problema se le suman los casos en el que la persona avisada está ocupado con otra labor y no puede avisar de vuelta para que avisen a otra persona. Otro problema generado es el tiempo perdido de tener que parar la labor del puesto de trabajo para coger el micrófono, dar el aviso y continuar trabajando.

En este trabajo pretendemos dar solución a este problema desarrollando un sistema de avisos basado en dispositivos desarrollados ad hoc. Estos dispositivos, basados en el microcontrolador ESP32, estarán conectados a la red de la organización de forma inalámbrica y notificarán a los trabajadores los avisos de forma individualizada. Los trabajadores podrán utilizar también el dispositivo para responder al aviso. Todos los avisos serán gestionados desde una aplicación que implementará un sistema de colas que tendrá en cuenta en todo momento la situación de los trabajadores en la organización.

Con un sistema embebido con el microcontrolador ESP32 podemos obtener mejores tiempos de reacción para acudir a los avisos, teniendo en cuenta también, como aspecto añadido, conseguimos mejoras en la protección de datos del trabajador y un ambiente más adecuado para personas con autismo e hipersensibilidad sensorial evitando el ruido alto que pueden llegar a generar los altavoces.

Otra ventaja añadida del uso del microcontrolador ESP32, es que es económico y se le pueden añadir fácilmente funciones de vibración, pantalla y más extensiones para poder personalizar a gusto del presupuesto disponible y funcionalidad requerida.



**Figura 1.1** Diagrama de la idea de flujo del dispositivo.

1.2 Objetivos y tecnologías a usar

Para obtener el Sistema Distribuido necesitaremos diseñar varios elementos como lo son la API REST, encargada de recibir peticiones URL desde un dispositivo móvil o sobremesa como un ordenador o un microcontrolador, así como enviar confirmación de haber realizado los procesos correctamente al dispositivo origen.

Necesitaremos gestionar las peticiones en colas con RabbitMQ ya que aseguran la entrega de los mensajes y proporciona prioridad entre mensajes, dos aspectos muy importantes para nuestro sistema, tal y como menciona Manuel León.  
Como objetivo principal, queremos conectar el microcontrolador ESP32 a la API REST como dispositivo principal que consumirá y mostrará los avisos almacenados en RabbitMQ, dado que su bajo consumo energético y conexión WI-FI sumado a su bajo costo, según el estudio "Design and Implementation of ESP32-Based IoT Devices", son características suficientes para elegir el uso de este dispositivo.  
La creación de una base de datos ayudará a tener una gestión de todas las peticiones realizadas, eligiendo como base de datos SQLite ya al ser una base de datos relacional podemos tener claves únicas según necesidades futuras.

Como objetivo final, querremos una interfaz web con HTML, CSS y JavaScript, ya que como dice César Pablo Córcoles Briongos, son actualmente lenguajes muy potentes y que evolucionan constantemente.

Para lograr los objetivos además usaremos tecnologías como drawio.com para la realización de los diagramas a lo largo de la memoria, plataformio para la realización del código en el microcontrolador sobre Visual Studio Code y finalmente Maven sobre Intellij Idea Community para poder lanzar la API REST.

1.3 Metodología

Usaremos la metodología Srcum, este se basa en la gestión de proyectos de manera eficiente a través de un trabajo ágil. Según el artículo de Clara Gil "Scrum aumenta la productividad, la calidad del producto y la satisfacción del cliente. Estudios demuestran un aumento de hasta el 66 % en productividad y una reducción del 30 % en el tiempo de desarrollo." Esto junto con el flujo de trabajo en el que se realizan ciclos iterativos para desarrollo incremental y la adaptación constante a las necesidades que van surgiendo, es una metodología muy fiable para implementar.

Lo primero a realizar será la planificación del producto, en nuestro caso el programa instalable en microcontroladores para la gestión de avisos en un sistema distribuido.

Dividiremos el trabajo en 10 ciclos los cuales seguirán una estructura similar, primero planificación del ciclo, luego su ejecución y una vez realizado se revisará.   
Los 10 ciclos son:

1. Creación de una API REST encargada de la generación de notificaciones a partir de una URL.

2. Creación de la aplicación consumidora de notificaciones.

3. Implementación de las funciones a realizar en el dispositivo ESP32 con el uso de temporizadores y botones.

4. Implementación de la apliación consumidora en el dispositivo ESP32.

5. Enlazar los avisos consumidos con el dispositivo ESP32 con las funciones implementadas.

6. Implementar generación de peticiones desde el dispositivo hacia la API.

7. Implementación de una base de datos que recoja todas las notificaciones generadas.

8. Unión de la base de datos con la API REST.

9. Creación de una interfaz gráfcia para realizar las peticiones.

10. Refactorización del proyecto.

Finalmente para cumplir los 5 puntos principales que define Clara Gil se incluye un ciclo de retrospectectiva, en el cual lo enfocaremos a las futuras mejoras que podría tener el proyecto.

1.4 Estructura de la memoria

La memoria estará divida en 5 puntos principales:

Primero la fase de análisis y diseño, encontrando diagramas sobre la estructura del proyecto y requisitos funcionales y no funcionales, destacando que se dividirá en dos subproyectos a analizar y diseñar, la API y el código del microcontrolador.

Seguidamente la implementación del código, se mostrarán las partes del código más relevante para el funcionamiento de la API y el programa cargable en el microcontrolador.

Una conclusión donde se hará una revisión total de objetivos cumplidos y una visión futura del proyecto.

Y finalmente dos manuales, uno de instalación con la explicación de qué parámetros hay que editar y otro de usuario con una explicación de cómo funciona el sistema distribuido.

2

Fase de análisis y diseño

2.1 API REST

2.1.1 Requisitos funcionales

RF1: Encolar y desencolar peticiones en colas.

RF2: Gestionar peticiones entre distintos microcontroladores.

RF3: Registrar todas las peticiones en una base de datos.

RF4: Obtener registro de todas las peticiones realizadas.

RF5: Crear y destruir colas de peticiones.

RF6: Encolar según prioridad y en formato FIFO.

2.1.2 Requisitos no funcionales

RNF1: Sistema de seguridad de recepción de peticiones por un entero.

RNF2: Base de datos en MySQLite.

RNF3: Serializar mensajes con JSON.

RNF4: Gestión de colas usando RabbitMQ.

RNF5: Spring Boot como framework base.

2.1.3 Actores del Sistema

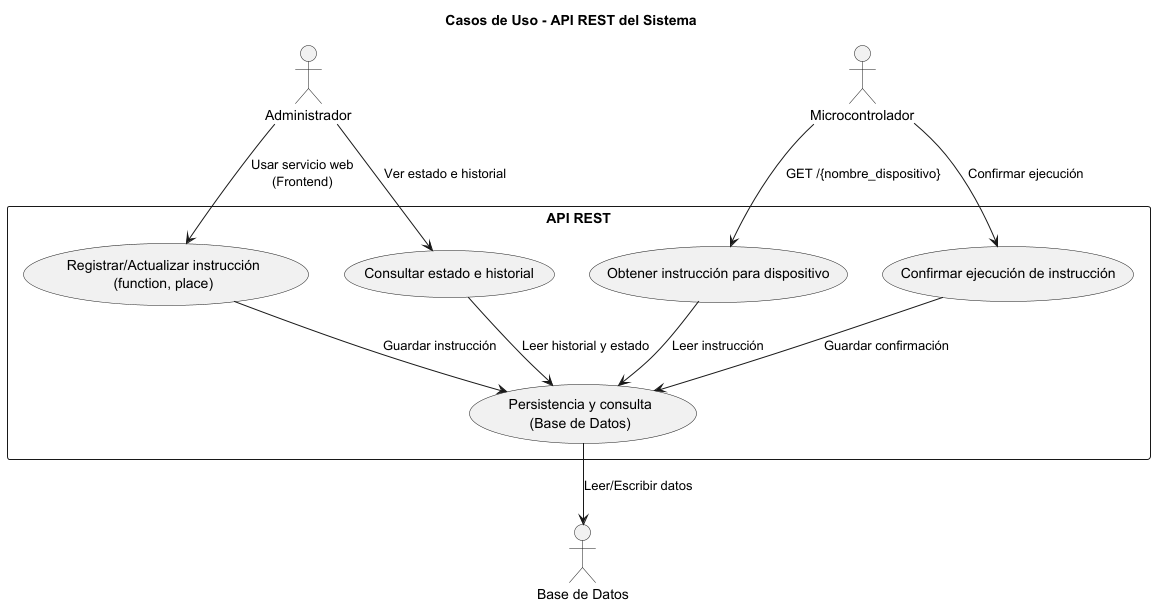
Usuario administrador: encargado de crear y destruir colas, además mandará las peticiones para que sea futuramente encoladas. Finalmente podrá pedir un listado de todas las peticiones almacenadas en al base de datos.

Microcontroladores: consumidores de peticiones así como podrán enviar el mismo mensaje consumido de vuelta al siguiente microcontrolador.

Base de datos SQLite: almacenará cada petición realizada tanto por el administrador como por el dispositivo.

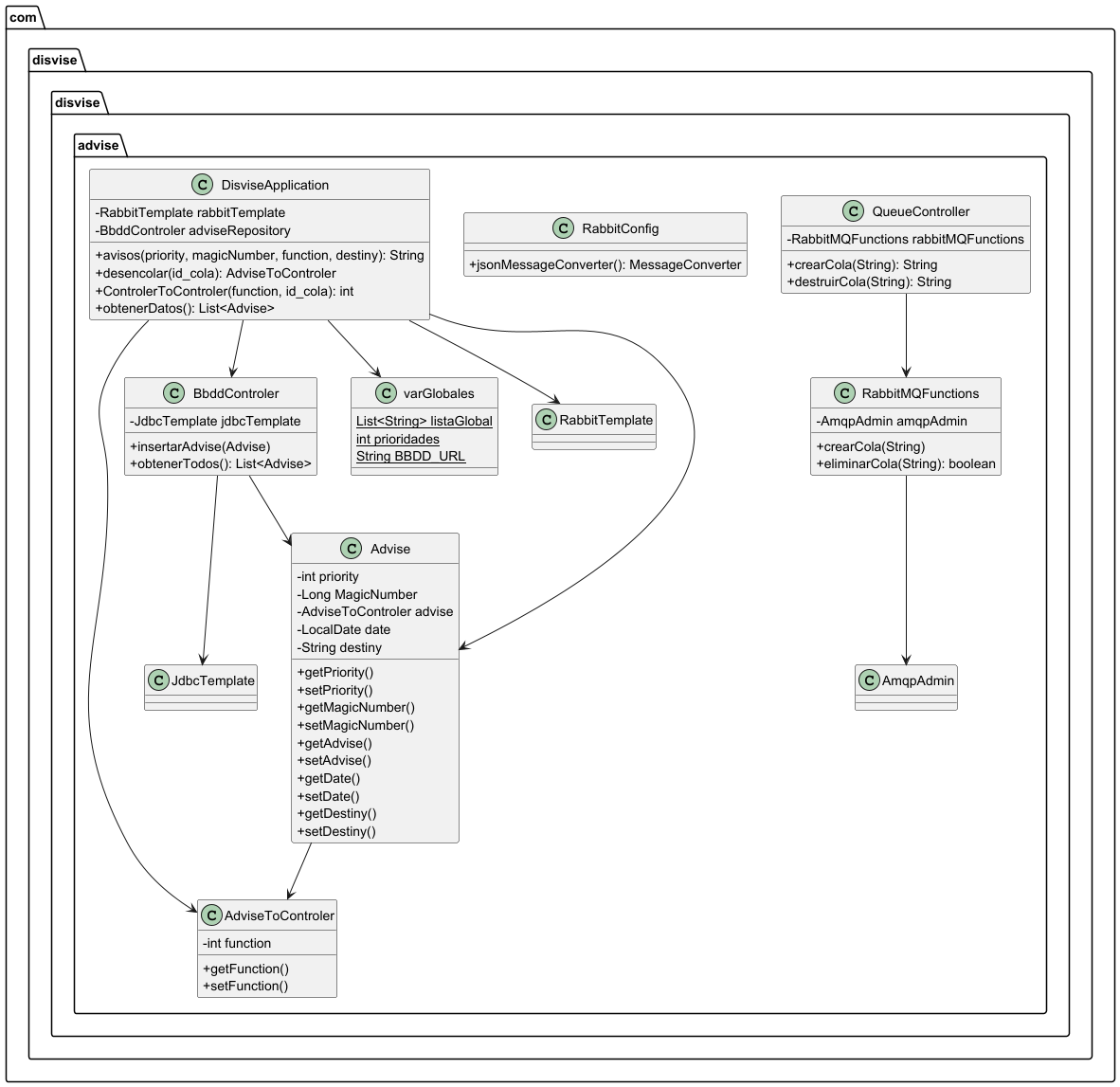
RabbitMQ: Aplicación encargada de gestionar todas las peticiones con prioridad y formato FIFO.

2.1.4 Diagrama de Casos de Uso



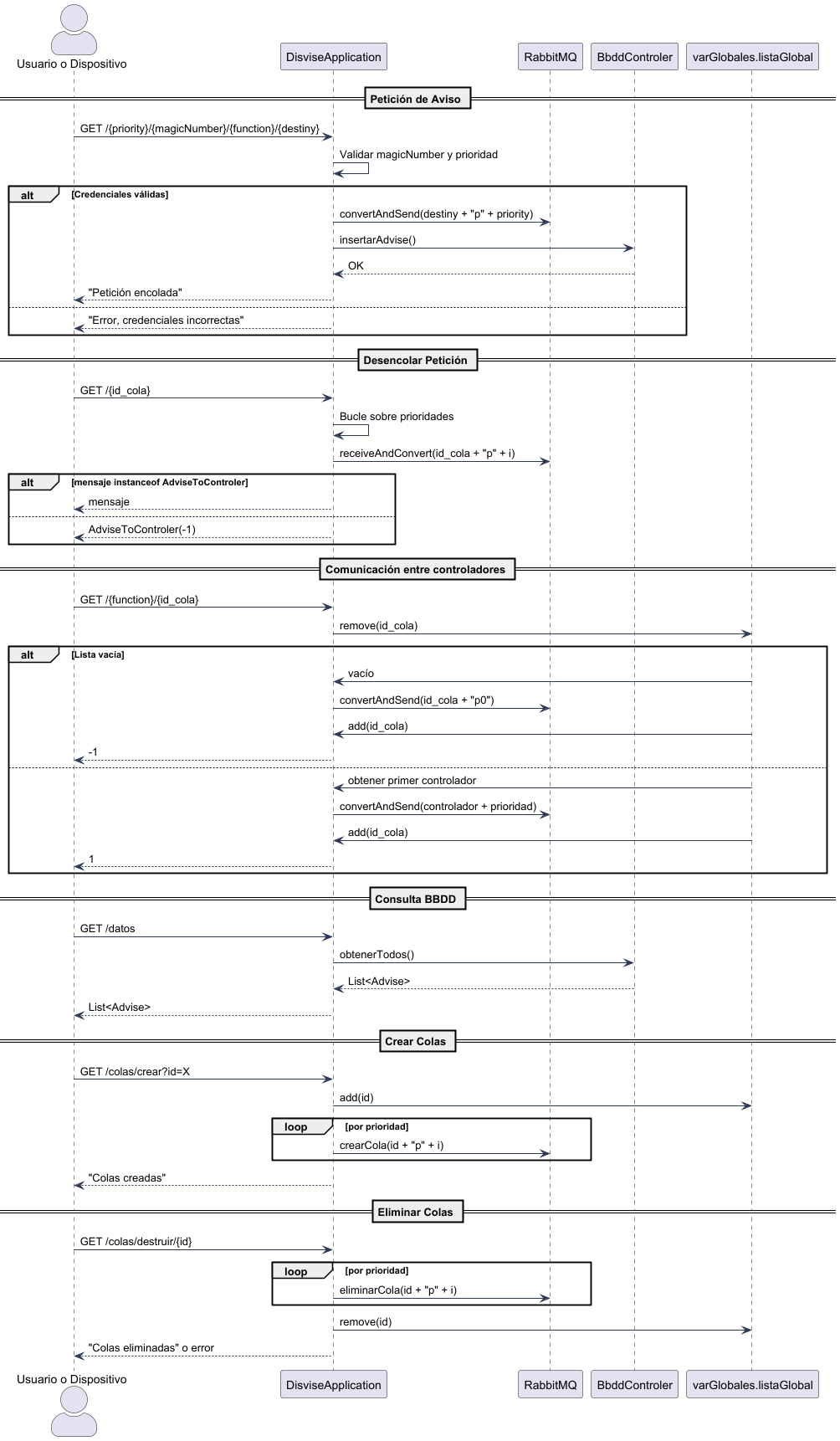
**Figura 2.1** Diagrama de casos de uso API

2.1.5 Diagrama de Clases



**Figura 2.2** Diagrama de clases API

2.1.4 Diagrama de secuencia

** Figura 2.3** Diagrama de secuencia API

2.1.5 Base de datos

La base de datos contendrá un registro de cada aviso realizado, por tanto tendrá que almacenar cada dato de la estructura. Al usar una base de datos relacional necesitaremos el uso de una clave única, sin embargo, al no necesitar realmente de clave única lo dejaremos como el número incremental como el número de elemento de entrada.  
Siguiendo finalmente la estructura:

CREATE TABLE advise (

id INTEGER PRIMARY KEY AUTOINCREMENT,

priority INTEGER NOT NULL,

magic\_number INTEGER,

advise STRING NOT NULL,

date TEXT,

destiny TEXT

);

**2.2 Microcontrolador ESP32**

Tenemos un microcontrolador ESP-32 implementado en el dispositivo Lilygo T-Display S3, el cual \*Buscar alguna fuente de información que hable de las ventajas del dispositivo\*.

2.2.1 Requisitos funcionales

RF1: Conectar a una red WIFI.

RF2: Consultar a una API REST datos en JSON.

RF3: Mostrar información en la pantalla del dispositivo según la función recibida.

RF4: Uno de los botones debe pedir la siguiente petición.

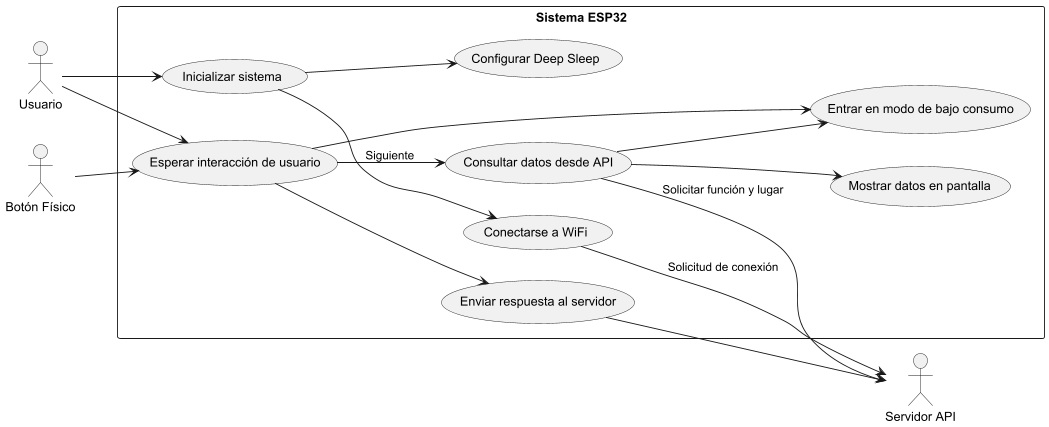
RF5: Uno de los botones debe poner el dispositivo en modo suspensión.

2.2.2 Requisitos no funcionales

RNF1: El dispositivo entrará en un modo de suspensión profunda para ahorrar batería.

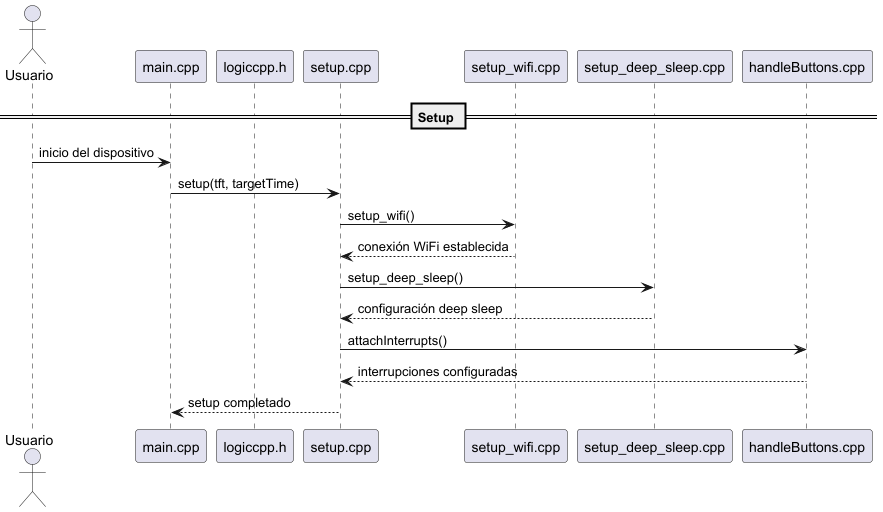
RNF2: El dispositivo esperará un tiempo prudencial a que se pulse algún botón.

RNF3: Tolerancia a errores WIFI o HTTP.

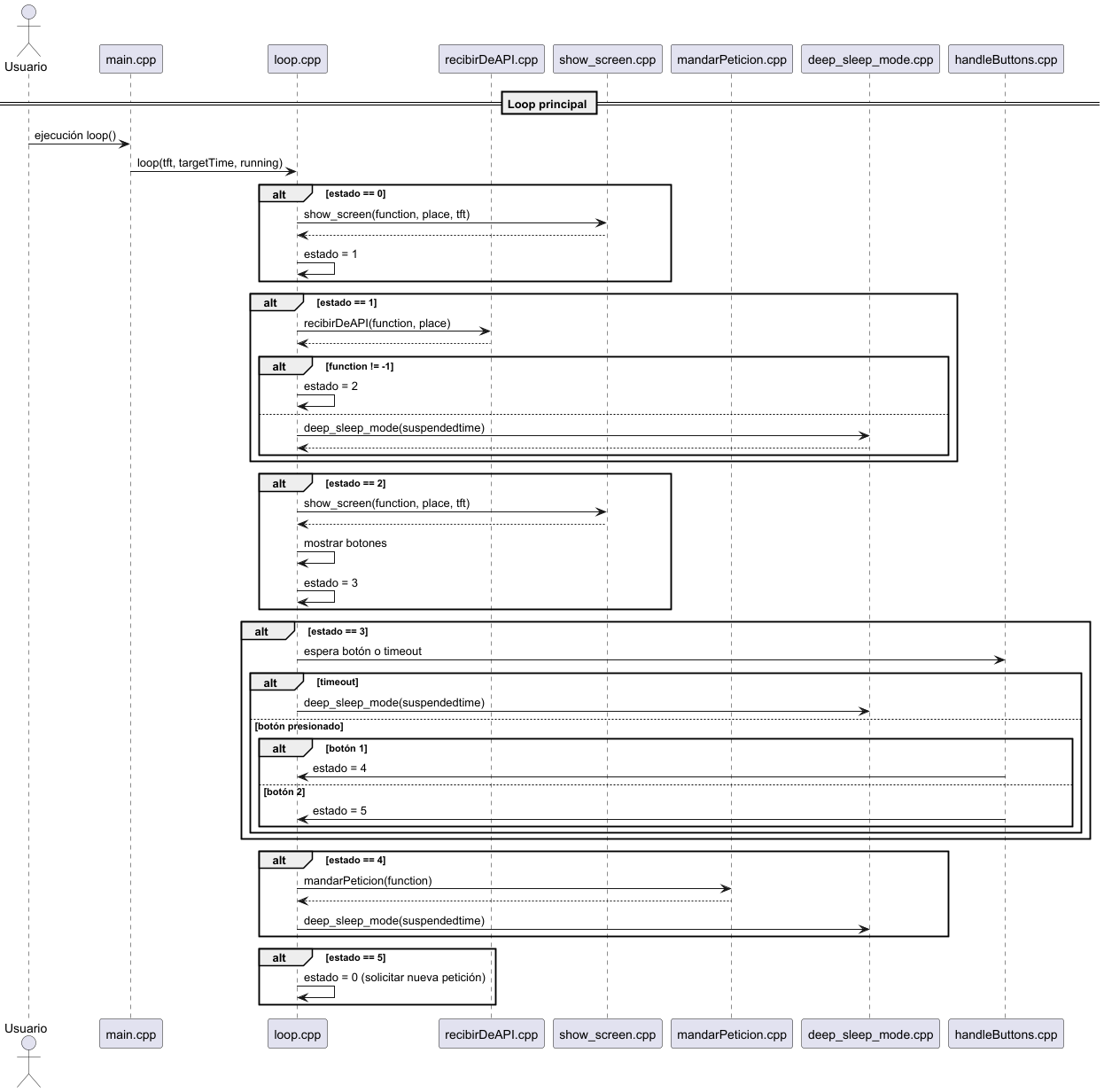
2.2.3 Diagrama de casos de uso

**Figura 2.4** Diagrama de casos de uso microcontrolador

2.2.4 Diagrama de secuencia

Diagrama de secuencia del inicializador del microcontolador:  


**Figura 2.5** Diagrama de secuencia inicialización microcontrolador

Diagrama de secuencia para el bucle principal:  


**Figura 2.6** Diagrama de secuencia bucle principal microcontrolador

**2.2.4 Diagrama de estados**  
Diagrama

El contenido generado por IA puede ser incorrecto.

**Figura 2.7** Diagrama de estados microcontrolador

Estado 0: Mostrando pantalla de carga con mensaje de "Recibiendo petición…"  
Estado 1: Recibiendo de la API la petición y deserialización del mensaje.

Estado 2: Mostrando por pantalla.

Estado 3: Esperando la pulsación de algún botón o que pase el tiempo marcado.

Estado 4: Devolviendo petición a la API.

Estado 5: Marcando el botón 2 para volverse a usar.

3

Implementación

**3.1 API REST**

En la API encontramos cuatro funciones principales que definen el comportamiento de la Figura 2.3, dos de ellas encargadas de darle a RabbitMQ peticiones, una de origen del microcontrolador y otra de una petición http, una tercera función encargada de las consumiciones de peticiones del microcontrolador y finalmente la encargada de devolver como JSON la información de la base de datos.

Texto

El contenido generado por IA puede ser incorrecto.

**Figura 3.1** Función encargada de encolar avisos.

En la figura 3.1 aparecen que se pasan 4 parámetros a la función, la prioridad, un denominado número mágico, el número de la función y el destino. El primer parámetro que tomaremos es el número mágico, un número puesto como medida de seguridad para asegurar que solo se procesarán avisos que conozcan ese valor. Seguidamente se encolará el aviso que, aunque Manuel León mencione en su artículo que es posible que RabbitMQ gestione colas con distintas prioridades, para poder tener mayor control sobre la generación de colas nosotros tendremos nuestras propias colas definidas por un identificador. El aviso encolado será del tipo "AdviseToControler", una clase definida por si en el futuro se requiere pasar más parámetros, en nuestro caso será solo uno, el número de la función. Finalmente se almacenará el aviso con todos los parámetros necesarios en la base de datos.

Texto

El contenido generado por IA puede ser incorrecto.

**Figura 3.2** Función encargada de enviar el mensaje al microcontrolador

En el momento que el microcontrolador hace una llamada a la API, comienza la función mostrada en la Figura 3.2, encargada de recorrer las colas hasta encontrar, si existe el primer aviso más prioritario.

Texto

El contenido generado por IA puede ser incorrecto.

**Figura 3.3** Función encargada de encolar el mensaje del microcontrolador

Tanto en la Figura 2.3 como en la Figura 2.5 se contempla la posibilidad de que el microcontrolador también pueda enviar un aviso a la API para este ser encolado en la cola segunda cola prioritaria si hay más de una cola o en la única cola que exista, dado que consideraremos que una petición reenviada de un microcontrolador debe tener cierta prioridad pero nunca la máxima. La API a su vez guarda un registro de todos los dispositivos activos con sus identificadores, por tanto, si un microcontrolador reenvía una petición, este pasará a ser el último dispositivo que reciba la petición. Se contempla el caso de si solo existe un único dispositivo, en ese caso la petición se reenviará al microcontrolador con la máxima prioridad.  
  
Texto

El contenido generado por IA puede ser incorrecto.

**Figura 3.4** Función encargada de llamar a la lógica que accede a la base de datos

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

**Figura 3.5** Función encargada de devolver una lista con el contenido de la base de datos

Finalmente tenemos en la Figura 3.4 la petición de los datos a la base de datos, el cual hace una llamada a la función mostrada en la Figura 3.5, función que utiliza la librería JdbcTemplate, encargada de manejar toda la lógica relacionada con el acceso a la base de datos.

**3.2 Microcontrolador ESP32**

Nuestro programa cargado en el microcontrolador se centra en un bucle principal que sigue la estructura de la Figura 2.7, dentro del bucle encontramos tres funciones principales, una función encargada de recibir los datos de la API, otra que reenvía los datos a la API y para terminar la función que muestra el contenido por pantalla.

Texto

El contenido generado por IA puede ser incorrecto.

**Figura 3.6** Primera parte del bucle principal del microcontrolador.

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

**Figura 3.7** Segunda parte del bucle principal del microcontrolador.

Texto

El contenido generado por IA puede ser incorrecto.

**Figura 3.8** Tercera parte del bucle principal del microcontrolador.

Texto

El contenido generado por IA puede ser incorrecto.  
**Figura 3.9** Cuarta parte del bucle principal del microcontrolador.

Texto

El contenido generado por IA puede ser incorrecto.

**Figura 3.10** Función encargada de mostrar información por pantalla del dispositivo

Texto

El contenido generado por IA puede ser incorrecto.

**Figura 3.11** Código encargado de deserializar el contenido JSON recibido de la API

Texto

El contenido generado por IA puede ser incorrecto.

**Figura 3.12** Código encargado de mandar función a la API

En el momento en el que el programa esté cargado en el microcontrolador este iniciará la pantalla, el wifi y las variables necesarias. Inmediatamente después de terminar la configuración como vemos en la Figura 3.6 mostrará un mensaje por pantalla de "Recibiendo petición" en un fondo negro, la función encargada de mostrar la información está contenida en la Figura 3.10. Continuando con el bucle del programa pasamos al estado 1, haremos una llamada a la API para recibir un objeto en JSON, este objeto habrá que deserializarlo como se muestra en la Figura 3.11, según el número de la función se almacenará en la variable "place" el nombre del lugar que posteriormente se mostrará en pantalla.

Fijándonos en al Figura 3.7, se puede ver que se encargará de mostrar el contenido correcto en la pantalla, siempre que un dato haya sido leído de la API previamente, volviendo a realizar el mismo código contemplado en la Figura 3.11, añadiendo que se mostrará ahora una indicación en cada botón según la función que desempeñen.

En la Figura 3.8 encontramos el nuevo estado alcanzado, el estado 3, dicho estado consiste en una espera mientras se muestra el contenido por pantalla, esperando a que pase un tiempo acotado o se pulse un botón.

Según el botón pulsado se accederá al estado 4 de la Figura 3.8 o al estado 5 de la Figura 3.9, el estado 4 será el responsable de reenviar la función a la API como en la Figura 3.12 y desconectar el dispositivo más tiempo que el normal y el estado 5 se encargará de volver a empezar el bucle sin necesidad de parar el dispositivo.

4

Conclusión

**4.1 Objetivos cumplidos**

Para conseguir el sistema distribuido necesitábamos tener varios dispositivos conectados entre sí, en nuestro sistema está compuesto por la API, el microcontrolador y el dispositivo con acceso al navegador web.

La API consigue encolar y desencolar peticiones en colas con prioridad usando RabbitMQ, las peticiones llegan correctamente por peticiones http y accede a la base de datos con lecturas y escrituras correctas.

El microcontrolador se conecta a la red WiFi, realiza peticiones http y recibe el objeto serializado en JSON, deserializa el objeto bien para realizar su función mostrando la información por pantalla, ambos botones funcionan y cumplen su función.

La base de datos almacena y devuelve todas las peticiones realizadas de origen distinto al microcontrolador.  
Obtenemos un sistema distribuido aplicable a múltiples ámbitos como supermercados, hospitales, sector hostelería y cualquier ámbito que necesite notificar a una persona rápidamente, pudiendo ser muy determinante en situaciones críticas.

Referencias

León, M., *RabbitMQ, qué es y cuáles son sus funcionalidades, Funcionalidades de RabbitMQ*

Hercog, D., Lerher, T., Truntič, M. y Težak, O., *Desing and Implementation of ESP32-Based IoT Devices*

Documentación SQLite, [*https://sqlite.org/docs.html*](https://sqlite.org/docs.html)

Córcoles Briongos, C., *¿Qué aprender para ser expert@ en desarrollo web?*

Gil, C., *Qué es la metodología Scrum y cómo aplicarla en 5 fases*

Manual JdbcTemplate, [*https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/jdbc/core/JdbcTemplate.html*](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/jdbc/core/JdbcTemplate.html)

Manual platformIO, [*https://docs.platformio.org/en/latest/what-is-platformio.html*](https://docs.platformio.org/en/latest/what-is-platformio.html)

Manual Arduino, [*https://docs.arduino.cc/*](https://docs.arduino.cc/)

Apéndice A

Manual de Instalación

**Requerimientos:**

Se necesitará un servidor que corra la API REST usando maven.

Un servidor dedicado a la base de datos o el mismo servidor de la API.

Un navegador web para realizar peticiones http y ver la interfaz gráfica.

Un microcontrolador con pantalla y dos botones como el Lilygo T-Display S3.

Un compilador de C++ y el programa plataformIO para la compilación del programa cargable en el microcontrolador.

**Constantes y variables a modificar:**

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Figura 6.1 Parámetros configurables para conexión WIFI



Figura 6.2 Parámetro configurable dirección url de la API

En el programa cargable del microcontrolador tendremos que modificar varios parámetros, siendo estos, los parámetros de conexión WiFi, el nombre de la red y la contraseña vistos en la Figura 6.1, la pantalla se tendrá que configurar comentando la línea "#include <User\_Setup.h>" y descomentando la línea "#include <User\_Setups/Setup206\_LilyGo\_T\_Display\_S3.h>" en el fichero "User\_Setup\_Select.h", en caso de ser distinta se deberá tener en cuenta y configurar la pantalla deseada, y configurar el parámetro de la dirección de la API terminado en "/" para asegurar cuando se complete la dirección como podemos ver en la Figura 3.12 que se genera bien la dirección. Esta dirección deberá contener una dirección IP y el puerto si no se cuenta con un servidor DNS.



Figura 6.3 Parámetro configurable dirección url de la base de datos

En la API se deberá modificar el valor que almacena la dirección física de la base de datos iniciando con ".jdbc:sqlite".

Apéndice B

Manual de Usuario

**API REST**