Grado en Ingeniería Informática

Trabajo de Final de Grado

Geolocalización en interiores utilizando Bluetooth Low Energy

Autor:
Guillermo Alaejos López

Supervisor:
Miguel Salas Alegre
Tutor académico:
Pablo Boronat Pérez

Fecha de lectura: 14 de julio de 2020
Curso académico 2019/2020

Resumen

En esta memoria se explica el desarrollo de mapas de calor de recintos cerrados a través de la tecnología Bluetooth Low Energy y usando un dispositivo ESP32. Un dispositivo ESP32 es un *System on Chip* (SoC), es decir, que cuenta con todos sus módulos en un mismo chip. Está diseñado por la empresa china Espressif, y tiene como predecesor el ESP8266, diseñado también por la misma compañía.

El ESP32 detecta continuamente las balizas a su alcance y realiza una petición HTTP cada vez que detecta una baliza válida, es decir, una baliza de la empresa. El servidor que recoge la petición guarda en la base de datos los datos recibidos para su uso posterior.

Un mapa de calor es una representación gráfica de la cantidad de personas que se encuentran en una zona, poniendo colores más cálidos a mayor cantidad de personas y colores más fríos a menor número de personas. Para este módulo se ha creado su almacenamiento en el servidor, junto a su petición y respuesta HTTP.

Por último, respecto a la ocupación por zonas, el servidor recibe una petición con un listado de zonas y responde con el número de personas que se encuentran en esas zonas actualmente.

El alcance de este proyecto no incluye la visualización gráfica del mapa de calor ni de la ocupación por zonas.

Palabras clave

Bluetooth Low Energy, BLE, ESP32, Arduino, Kotlin, geolocalización

Keywords

Bluetooth Low Energy, BLE, ESP32, Arduino, Kotlin, geolocation

Índice general

1. Introducción **7**

1.1. Contexto y motivación del proyecto 7

1.1.1. La empresa 7

1.1.2. El proyecto 7

1.2. Objetivos del proyecto 9

1.3. Descripción del proyecto 10

2. Planificación del proyecto **13**

2.1. Metodología 13

2.2. Planificación 13

2.2.1. Metodología Ágil 13

2.3. Estimación de recursos 15

2.4. Costes del proyecto 15

2.4.1. Modelo de estimación COCOMO 15

2.5. Seguimiento del proyecto 17

2.6. Gestión de riesgos 18

2.6.1. Identificación de riesgos 18

2.6.2. Análisis de riesgos 18

3. Análisis y diseño del sistema **21**

3.1. Análisis del sistema 21

3.1.1. Diagrama de casos de uso 21

3.1.2. CU01: Detectar dispositivo 22

3.1.3. CU02: Enviar datos al servidor 23

3.1.4. CU03: Ver mapa de calor 24

3.1.5. CU04: Ver ocupación de las zonas 25

3.2. Diseño de la arquitectura del sistema 26

3.2.1. Bluetooth Low Energy 26

3.2.2. Recepción de datos 27

3.2.3. Mapas de calor 27

3.2.4. Ocupación por zonas 27

3.2.5. Base de Datos 27

4. Implementación y pruebas **29**

4.1. Detalles de implementación 29

4.1.1. Bluetooth Low Energy 29

4.1.2. Recepción de datos 32

4.1.3. Mapas de calor 32

4.1.4. Ocupación por zonas 33

4.1.5. Base de datos 33

4.2. Verificación y validación 33

5. Tecnologías y Herramientas **35**

5.1. Herramientas de gestión 35

5.1.1. Teamwork 35

5.2. Herramientas de comunicación 35
5.2.1. Teamwork Chat 36
5.2.2. Google Meet 36
5.2.3. Anydesk 36
5.3. Herramientas de documentación 36
5.3.1. Swagger 36
5.4. Control de versiones 37
5.4.1. GitLab 37
5.5. Lenguajes de programación 37
5.5.1. C++ 37
5.5.2. Kotlin 37
5.6. Base de datos 37
5.6.1. MySQL 37
5.7. IDEs 38
5.7.1. Arduino IDE 38
5.7.2. IntellJ 38
5.8. Frameworks y librerías 38
5.8.1. ESP32 Snippets 38
5.8.2. JOOQ 38
5.9. Pruebas 39
5.9.1. Spek 39
5.9.2. Beeceptor 39
5.9.3. Gitlab CI 39
5.9.4. XML 39

6. Conclusiones y posibles mejoras **41**

6.1. Conclusiones 41

6.2. Posibles mejoras 41

Bibliografía **43**

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

En este TFG se va a estudiar y analizar el desarrollo de la tecnología Bluetooth Low Energy aplicada al mundo empresarial, en concreto, al desarrollo de una aplicación de geolocalización en interiores.

1.1.1. La empresa

Easygoband es una empresa fundada en 2017, con un equipo de ocho personas dedicadas a la creación de *software* para la gestión de eventos, hoteles y parques de atracciones. Sus dos productos principales son los siguientes:

- GoFun: Pulsera inteligente para la gestión de eventos (como festivales de música) que permite el pago de productos usando tecnología *contactless*.
- GoGuest: Producto para facilitar de gestión de hoteles y parques de atracciones.

1.1.2. El proyecto

Desde Easygoband se desea implementar un sistema de geolocalización en interiores. Actualmente, el sistema de geolocalización más utilizado es el GPS, con una gran precisión para la localización en exteriores, pero de peor calidad para interiores debido a las posibles interferencias y obstáculos que se puede encontrar. Para solucionar eso, se propone utilizar Bluetooth Low Energy a través de un dispositivo Arduino. El dispositivo elegido es un ESP32 [33], que proporciona servicio de Bluetooth y Wi-Fi. Mientras que el primero se usa para la localización, el segundo es necesario para el envío de los datos a un servidor para que estos puedan ser almacenados y tratados. En la figura 1.1 se puede ver un dispositivo ESP32.

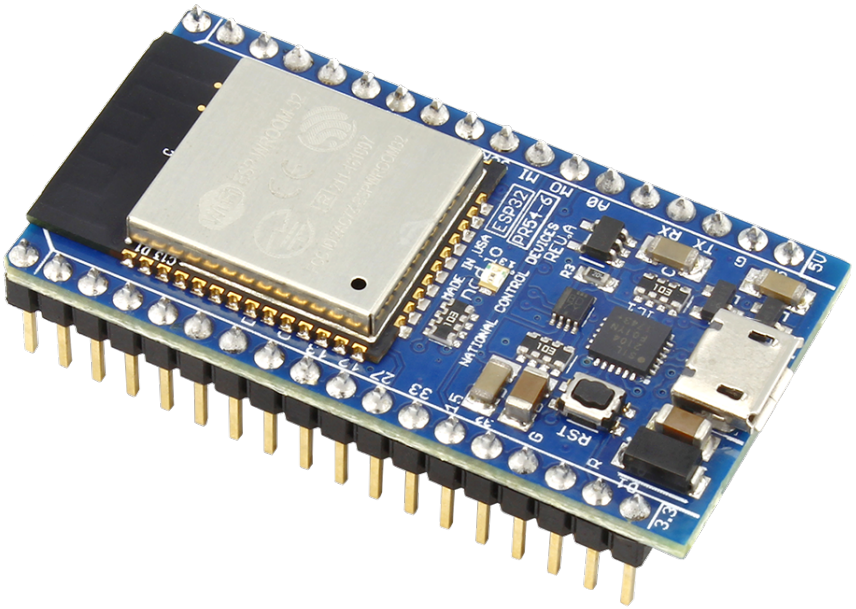

Figura 1.1: ESP32

En la figura 1.2 se puede ver una baliza emisora. Estas balizas emiten cada pocos segundos un mensaje con el que anuncian su presencia a los dispositivos receptores.

Figura 1.2: Baliza emisora

En la tabla 1.1 se puede ver una comparativa entre Bluetooth y Bluetooth Low Energy. Como se puede ver, Bluetooth Low Energy consume muy poca energía sacrificando el ancho de banda en el envío de datos. Sin embargo, para este proyecto, donde los datos a enviar son mínimos, reducir el consumo de energía resulta una ventaja muy importante.

Característica	Bluetooth	Bluetooth Low Energy
Ancho de banda	50 Mbps	200 Kbps
Consumo de energía	100 %	1 %-5 %
Duración de la batería	N/A	1-5 años

Tabla 1.1: Diferencias entre Bluetooth y Bluetooth Low Energy

Una vez implementada la geolocalización, se propone el desarrollo de mapas de calor para, en un futuro, visualizar gráficamente las zonas más transitadas de los hoteles y parques de atracciones en determinados momentos. Un mapa de calor es un gráfico que representa mediante colores el número de personas que hay en cada zona, dando colores más intensos a valores más alto, y colores más fríos a valores menores.

Por último, se tiene que poder realizar una consulta que permita conocer el número de personas que se encuentran actualmente en unas determinadas zonas.

1.2. Objetivos del proyecto

El proyecto desarrollado tiene como su principal objetivo la creación de un sistema que permita guardar la última zona visitada por un cliente. Cada cliente llevará una baliza emisora Bluetooth Low Energy, y según se vaya desplazando, los dispositivos ESP32 irán detectando su localización y enviándola a un servidor.

Una vez el servidor haya recibido la información, la guardará en una base de datos para su uso posterior, por ejemplo, conocer el número de personas que se encuentran actualmente en una zona determinada o la creación de mapas de calor.

Por último, los responsables de los hoteles y parques de atracciones podrán ver cuántos usuarios hay actualmente en cada zona. El alcance de esta última parte solo incluye la parte de *backend*, la parte de *frontend* queda excluida.

El objetivo final es que este producto sea usado por los hoteles y parques de atracciones para gestionar la afluencia de personas en las distintas zonas. Además, con este sistema se pretende facilitar la gestión a los hoteles y parques de atracciones y ofrecer una mayor información de sus clientes que puedan utilizar para obtener más rentabilidad económica a su negocio. Las balizas emisoras que tendrán que llevar los clientes, en algunos casos ya lo llevan para facilitar pagos, por lo que para el cliente no implica ningún cambio, mientras que para los hoteles y parques será necesario una instalación de los dispositivos ESP32; sin embargo, una vez instalados, su coste de mantenimiento es prácticamente nulo.

1.3. Descripción del proyecto

Para este proyecto se han desarrollado cuatro módulos. El primero de ellos es la geolocalización a través de Bluetooth Low Energy con el ESP32, programado en C++ en el IDE Arduino.

Los otros tres módulos restantes son más similares entre sí, puesto que todos se han programado en *Kotlin*, utilizando y ampliando la API de la empresa.

- **Recepción de mensajes enviados por el ESP32**: se tiene que tratar la petición HTTP recibida por el servidor y guardar los datos enviados en la base de datos.
- **Creación de mapas de calor**: ampliación de la base de datos para almacenar los mapas de calor. Tratamiento de la petición HTTP recibida por el servidor y su correspondiente respuesta.
- **Visualización de la ocupación por zonas**: tratamiento de la petición HTTP recibida por el servidor, cálculo de la ocupación que hay en cada zona solicitada y su respuesta.

Bluetooth Low Energy es una tecnología de comunicación inalámbrica de corto alcance desarrollada y comercializada por Bluetooth SIG [31]. A diferencia del Bluetooth tradicional, Bluetooth Low Energy se caracteriza tener un consumo de energía muy bajo manteniendo un radio de acción similar.

La programación del ESP32 ha partido desde cero, pero se ha basado en un ejemplo para la detección de dispositivos que venía en la librería utilizada.

Por otra parte, la programación del resto de los módulos se ha realizado siguiendo el esquema habitual dentro de Easygoband. En primer lugar, se ha empezando documentando las rutas que luego se crearán. De esta manera se establece desde un primer momento qué se quiere hacer y cómo, facilitando así su paso a código. Para la documentación se ha utilizado *Swagger* [25], una herramienta diseñada para facilitar y unificar la documentación de aplicaciones.

Una vez realizada y validada la documentación, se empieza el desarrollo. Todo el desarrollo se ha realizado en *Kotlin* [16], que es un lenguaje de programación creado en 2012 por la empresa *JetBrains* [14].

En cuanto a la base de datos, se ha utilizado y ampliado la base de datos de la empresa, de tipo *MySQL*, que, al seguir el estándar SQL, ha hecho que la adaptación haya sido muy rápida. No obstante, para la realización de las consultas no se ha trabajado con *MySQL* [19] directamente, sino que se ha utilizado *JOOQ* [15] (*Java Object Oriented Querying*), un *framework* diseñado para trabajar consultas SQL siguiendo la sintaxis de Java.

Generic Access Profile

Dentro de Bluetooth Low Energy existen dos roles definidos según el *Generic Access Profile* [30].

El primero de ellos hace referencia a los periféricos, dispositivos emisores cuya función es el envío periódico de mensajes.

El segundo rol es del de los dispositivos centrales, cuya función principal es detectar los mensajes enviados por los periféricos y contestar si procede.

Un periférico solo puede estar conectado a un dispositivo central a la vez, por tanto, cuando se conecta deja de enviar mensajes para notificar su presencia, y la comunicación entre el periférico y el dispositivo central se establece a través de los servicios y características GATT (***G****eneric* ***ATT****ribute Profile*, de las que se hablará a continuación.

Generic Attribute Protocol

La comunicación entre un periférico y un dispositivo central se realiza a través de los servicios y características de los perfiles definidos en el *Generic Attribute Protocol*. La comunicación se realiza siguiendo el paradigma cliente-servidor, donde el periférico actúa como servidor y envía los datos que el cliente, en este caso el dispositivo central, necesita. La lista de perfiles definidos en el GATT se puede encontrar en la especificación proporcionada por Bluetooth [32], donde se puede ver que destacan perfiles orientados al ámbito sanitario, junto con los de geolocalización.

Para la detección de dispositivos BLE se ha utilizado la librería creada por Neil Kolban[35] para trabajar con Bluetooth Low Energy con el ESP32. Esta librería proporciona soporte para las principales funciones de Bluetooth Low Energy, como pueden ser la detección y envío de mensajes, además de proporcionar soporte para los distintos protocolos de comunicación que existen, como *Eddystone* o *iBeacon*. Para la realización de las prácticas se ha trabajado con este último protocolo.

Los mensajes enviados por un periférico *iBeacon* siguen siempre el mismo formato, como se puede ver en la tabla 1.2.

Campo	**Tamaño**	**Descripción**
UUID	*16 bytes*	Acrónimo de ***U****niversally* ***U****nique* ***ID****entifier*, es el identificador del periférico que envía el mensaje. Suele ser el mismo para aquellos dispositivos que comparten una misma finalidad. Siguen el formato "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx", combinando letras y números.
Major	*2 bytes*	Usado para dividir en zonas más pequeñas las zonas que tienen un mismo UUID.
Minor	*2 bytes*	Usado para identificar cada dispositivo dentro de una misma zona con un mismo *major*.

Tabla 1.2: Formato de los mensajes del protocolo *iBeacon*

Capítulo 2

Planificación del proyecto

2.1. Metodología

A continuación se va a detallar la metodología seguida para el desarrollo de este proyecto. Dentro de Easygoband se sigue una metodología ágil basada en *Scrum* [23]. A diferencia de esta, que propone reuniones diarias de unos 15 minutos de duración, en Easygoband se hacen dos reuniones semanales de unos 20 ó 25 minutos donde todo el equipo de desarrollo expone su avance en esos días y comenta qué tiene planteado hacer hasta la siguiente reunión.

Debido al estado de alarma que se anunció el 13 de marzo provocado por la crisis del COVID-19, se estableció un teletrabajo para asegurar la salud de los trabajadores. Con el fin de asegurar que ninguna persona se quede rezagada en su trabajo y no disminuir la comunicación entre el equipo, se decidió cambiar el sistema de reuniones de uno bisemanal de 25-30 minutos de duración a uno diario de unos 15 minutos de duración. Estas reuniones *online* se realizaron a través de *Google Meet*.

2.2. Planificación

2.2.1. Metodología Ágil

Como se ha comentado en el apartado anterior, la metodología elegida para el desarrollo del proyecto es una metodología ágil basada en *Scrum*.

En las tablas 2.1 y 2.2 se pueden ver la pila del producto y las historias de usuario, respectivamente. Hay más tareas en la pila del producto que historias de usuario puesto que hay varias tareas que solo se han realizado a nivel de servidor, sin que se sepa cómo va a interactuar un agente externo. Cada punto de historia representa 10 horas, y hay un total de 30 puntos de historia, es decir, 300 horas en total estimadas para el desarrollo del proyecto.

Pila del producto

Código	Nombre	Estimación	Prioridad	Sprint
1	Detectar dispositivo	1	Alta	1
2	Filtrar dispositivo	1	Alta	1
3	Enviar datos al servidor	2	Alta	1
4	Recepción de los datos en el servidor	7	Alta	2
5	Creación de mapas de calor	13	Media	3
6	Creación de un listado con la ocupación por zonas	6	Media	3

Tabla 2.1: Pila del producto

Para el primer *sprint* está planificado hacer todas las tareas relacionadas con el ESP32, y luego centrarse en el resto de la funcionalidad a desarrollar.

Historias de usuario

Pila del producto	
ID	**Historia de usuario**
1	Como ESP32 Quiero poder detectar dispositivos BLE Estimación: 1
2	Como ESP32 Quiero poder filtrar los dispositivos detectados Estimación: 1
3	Como ESP32 Quiero poder enviar los datos al servidor Para que sean guardados Estimación: 2

Tabla 2.2: Historias de usuario

2.3. Estimación de recursos

A continuación se muestran los recursos necesarios para el desarrollo de este proyecto, estando divididos en dos categorías según sean de *hardware* o *software*.

Tipo recurso	Descripción	Precio
Hardware	Ordenador portátil para la realización de las prácticas	500 €
	Segunda pantalla para facilitar el desarrollo del proyecto	80 €
	ESP32	10 €
	Balizas emisoras con las que realizar las pruebas	20 €
Software	IDE Arduino	Gratuito
	IntellJ	Gratuito
	MySQL	Gratuito

Tabla 2.3: Recursos utilizados durante el proyecto

2.4. Costes del proyecto

El objetivo de este apartado es calcular el coste asociado al proyecto realizado como si este fuese desarrollado por un trabajador profesional contratado por la empresa.

2.4.1. Modelo de estimación COCOMO

El modelo de estimación COCOMO [18] se utiliza para estimar el coste de un proyecto en función de fórmulas matemáticas basadas en las líneas de código que este tiene. En función del número de líneas se establece el tipo de proyecto que se va a desarrollar. Si el proyecto se estima menor a 50 000 líneas de código se considera orgánico, si tiene entre 50 000 y 300 000 se considera semi-acoplado, y si tiene más de 300 000 se considera embebido.

Módulo	Lenguajes de programación	Líneas de código	Líneas de código (Tests)
Geolocalización	C++	150	0
Recepción de datos	Kotlin/SQL/XML	140	130
Mapas de calor	Kotlin/SQL/XML	620	260
Ocupación por zonas	Kotlin/SQL/XML	115	100

Tabla 2.4: Desglose de las líneas de código del proyecto

Se puede observar que el desarrollo de tests es una parte importante del desarrollo dentro de Easygoband, pudiendo llegando a ocupar en número de líneas la misma cantidad el código a ejecutar. Sin embargo, esto no ha sido así para el módulo de geolocalización, puesto que al estar programado en C++ y para simplificar el desarrollo, se decidió no hacer ningún test y, en

su lugar, asegurar el funcionamiento haciendo pruebas con *mocks* que simulen la recepción de peticiones HTTP. De esta manera, se comprueba tanto el envío de mensajes como la detección y filtración de dispositivos detectados.

En total se han escrito 1515 líneas de código, como son menores a 50 000, el tipo de proyecto desarrollado es orgánico, por lo que se toman los coeficientes necesarios para COCOMO en función a eso. La tabla 2.6 muestra los coeficientes de cada parámetro en función del tipo de proyecto.

Tipo de proyecto	**a**	**b**	**c**	**d**
Orgánico	2,4	1,05	2,5	0,38
Semiacoplado	3	1,12	2,5	0,35
Empotrado	3,6	1,20	2,5	0,32

Tabla 2.5: Coeficientes de COCOMO en función del tipo de proyecto

Una vez hecho esto, ya es posible calcular los costes del proyecto siguiendo una serie de fórmulas predeterminadas.

Esfuerzo persona/mes (E):

$$E = a \times KLDC^b = 2,4 \times 1,515^{1,05} = 3,71$$

Tiempo estimado (meses):

$$D = c \times E^d = 2,5 \times 3,71^{0,38} = 4,11 meses$$

Como se puede ver, el tiempo estimado se aproxima casi con total exactitud al tiempo real invertido, que ha sido casi 3 meses. En la planificación inicial, la fecha de fin era tres semanas más tarde, pero este tiempo se ha acortado debido a cambios en el horario. Estos cambios han sido provocados por la cuarentena establecida por el COVID-19, que me ha permitido hacer más horas semanales (de 25,5 horas a 36 horas) y que ha hecho que el proyecto se termine antes.

Una vez calculados estos datos se puede deducir el número de personas necesarias para completar este proyecto, siendo este de casi 1 persona.

Por tanto, los costes finales del proyecto se pueden ver en la tabla 2.6.

	Costes
Programador	1200 €/ mes * 4 meses = 4800 €
Gastos de contratación	4800 €* 25 % = 1200 €
Costes indirectos	6000 €* 20 % = 1200 €
Recursos	610 €
Total	7810 €

Tabla 2.6: Costes finales del proyecto

2.5. Seguimiento del proyecto

En la tabla 2.7 se puede ver un resumen del proyecto, donde se puede ver el desarrollo del proyecto y una breve explicación sobre qué se ha realizado en cada *sprint* y la cantidad de horas invertidas en cada uno.

Sprint	**Fecha de inicio**	**Fecha de fin**	**Tareas realizadas**	**Horas invertidas**
Sprint 0	10/02/2020	17/02/2020	Investigación sobre el funcionamiento de Bluetooth Low Energy	25,5
Sprint 1	17/02/2020	08/03/2020	Detección de dispositivos BLE por parte del ESP32 y envío de mensajes con los datos a través de Wi-Fi	55,5
Sprint 2	09/03/2020	05/04/2020	Ampliación de la API para facilitar la recepción de los mensajes enviados por el ESP32. Resolución de errores en la detección de dispositivos	115,5
Sprint 3	06/04/2020	30/04/2020	Creación de mapas de calor en función de los datos obtenidos por BLE y creación de un listado de la ocupación de las zonas en ese momento	103,5

Tabla 2.7: Desarrollo del proyecto

En la tabla 2.8 se puede ver una comparativa entre el tiempo estimado y el tiempo real invertido para cada tarea del proyecto.

Tarea	**Estimación (horas)**	**Tiempo invertido (horas)**	**Variación**
Bluetooth Low Energy	40	100	+60
Recepción de mensajes	70	95	+25
Mapas de calor	130	75	-55
Ocupación por zonas	60	30	-30

Tabla 2.8: Comparativa entre el tiempo estimado y el tiempo invertido

Como se puede observar, hay grandes variaciones entre la estimación y el tiempo invertido. En el primer *sprint*, por una parte, el desarrollo de la programación del ESP32 empezó una semana más tarde, tiempo que se invirtió en entender el funcionamiento de Bluetooth Low Energy. Además, aprender esa nueva tecnología, no solo a nivel teórico sino a nivel práctico, llevó más tiempo de lo estimado. A esto se sumó tener que programar tanto en un lenguaje de programación nuevo como en un entorno de desarrollo nuevo. Asimismo, cada ejecución del programa para comprobar su funcionamiento llevaba varios minutos de compilación, puesto que usar librerías de Bluetooth o Wi-Fi incrementa este tiempo.

En el segundo *sprint* tuve que aprender otro lenguaje de programación nuevo, junto a entender la API de la empresa y saber cómo realizar un caso de uso desde cero. En este caso, si bien el tiempo invertido fue mayor al estimado, gracias a la ayuda recibida por parte de varios miembros del equipo de desarrollo de la empresa, se pudo avanzar más rápidamente el desarrollo, y permitió que en el último *sprint* se pudiera recuperar el tiempo invertido anteriormente.

2.6. Gestión de riesgos

En esta sección se habla sobre los posibles riesgos que se puedan dar durante el desarrollo de las prácticas.

2.6.1. Identificación de riesgos

En la tabla 2.9 se pueden ver los riesgos que se han identificado para este proyecto, junto al impacto y la probabilidad de que ocurran.

ID	Descripción	Impacto	Probabilidad
R01	Tareas poco claras o mal definidas	Medio/Alto	Bajo
R02	Falta de ayuda en la empresa	Medio	Muy bajo
R03	Falta de tiempo	Medio/Alto	Medio

Tabla 2.9: Posibles riesgos de este proyecto

2.6.2. Análisis de riesgos

R01: Tareas poco claras o mal definidas

Al inicio de cada *sprint* se tienen que tener claras las tareas a realizar. En caso de que no lo estén, será necesario destinar tiempo a reunirse con el supervisor para dejarlas claras.

R02: Falta de ayuda en la empresa

En caso de que el supervisor no esté disponible para resolver dudas, es posible que el proyecto se quede bloqueado. Para evitar esto, se creó un canal de comunicación entre varios miembros del equipo de desarrollo para agilizar las posibles dudas que fueran surgiendo.

R03: Falta de tiempo

Es posible que la estimación de tiempo para las tareas sea inferior al real, lo que implique que no dé tiempo a acabar todas las tareas. Este riesgo se dio durante los primeros módulos, pero pudo corregirse a tiempo gracias a la ayuda recibida del equipo de trabajo.

Capítulo 3

Análisis y diseño del sistema

3.1. Análisis del sistema

En este apartado se habla acerca del análisis realizado sobre este proyecto.

3.1.1. Diagrama de casos de uso

En la figura 3.1 se puede ver el diagrama de casos de uso de este proyecto, junto a los actores implicados. Cada caso de uso representa cada módulo desarrollado.

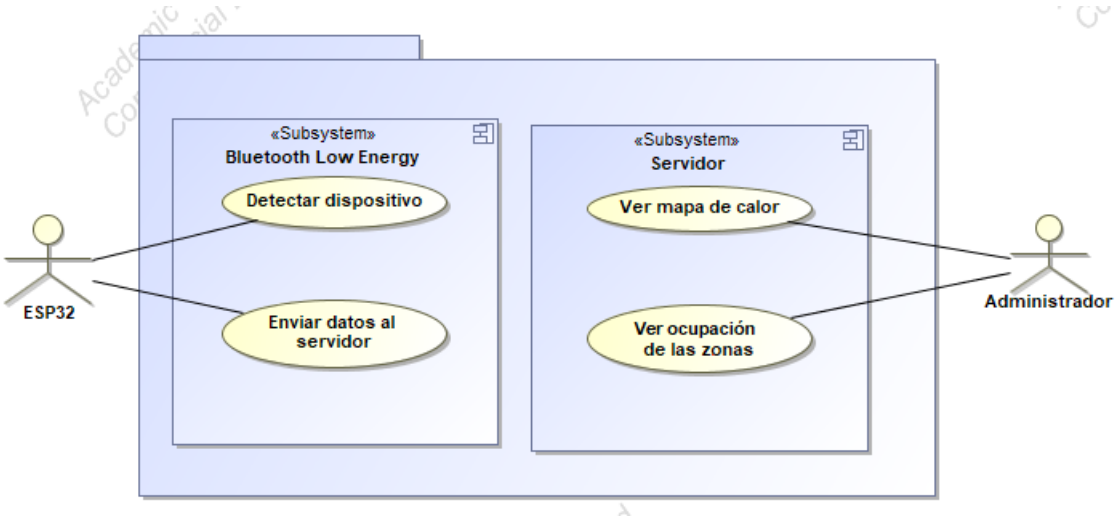

Figura 3.1: Diagrama de casos de uso del proyecto

3.1.2. CU01: Detectar dispositivo

Especificación del caso de uso	
Identificador	CU01
Nombre	Detectar dispositivo
Versión	1.0
Autor	Guillermo Alaejos López
Descripción	El dispositivo ESP32 ha de escanear balizas emisoras llevadas por los huéspedes para determinar su posición.
Alcance	El alcance de este caso de uso es la detección de una baliza emisora que lleva un cliente
Actor principal	ESP32
Actores secundarios	Huéspedes
Relaciones	Enviar datos al servidor
Precondición	El huésped lleva una baliza emisora
Trigger	El huésped pasa cerca de un dispositivo ESP32
Secuencia normal	Acción
1	El huésped se acerca a la zona donde está situado el ESP32
2	El ESP32 detecta la baliza emisora del huésped
Frecuencia esperada	Una vez cada 5 segundos

3.1.3. CU02: Enviar datos al servidor

Especificación del caso de uso	
Identificador	CU02
Nombre	Enviar datos al servidor
Versión	1.0
Autor	Guillermo Alaejos López
Descripción	El dispositivo ESP32 tiene que mandar los datos de la baliza detectada al servidor
Alcance	El alcance de este caso de uso llega desde la detección de las balizas hasta el envío de los datos al servidor
Actor principal	ESP32
Actores secundarios	Huéspedes
Relaciones	Detectar dispositivo
Precondición	El huésped lleva una baliza emisora
Trigger	El huésped ha sido detectado por el ESP32
Secuencia normal	Acción
1	El ESP32 recoge los datos del huésped que acaba de detectar
2	El ESP32 envía vía Wi-Fi al servidor los datos del huésped
Frecuencia esperada	Una vez cada 5 segundos

3.1.4. CU03: Ver mapa de calor

Especificación del caso de uso	
Identificador	CU03
Nombre	Ver mapa de calor
Versión	1.0
Autor	Guillermo Alaejos López
Descripción	Creación en el backend de mapas de calor sobre zonas
Alcance	El alcance de este caso de uso solo incluye la parte de backend (petición y respuesta del servidor), no la creación ni visualización de la imagen del mapa de calor
Actor principal	Administrador
Secuencia normal	Acción
1	El servidor recoge la petición hecha por el administrador
2	El servidor responde
Frecuencia esperada	Una vez cada 5 segundos

3.1.5. CU04: Ver ocupación de las zonas

Especificación del caso de uso	
Identificador	CU04
Nombre	Ver ocupación de las zonas
Versión	1.0
Autor	Guillermo Alaejos López
Descripción	El administrador ve un listado de la ocupación actual de las zonas que haya seleccionado.
Alcance	El alcance de este caso de uso incluye la petición y respuesta del servidor, no la creación y visualización del listado con las zonas
Actor principal	Administrador
Secuencia normal	Acción
1	El servidor recoge la petición hecha por el administrador
2	El servidor calcula cuántas personas hay en cada zona
3	El servidor responde con los datos calculados sobre las zonas
Frecuencia esperada	Una vez cada varios días

3.2. Diseño de la arquitectura del sistema

3.2.1. Bluetooth Low Energy

En este apartado se explica la relación entre los dispositivos emisores o balizas, y los dispositivos receptores o centrales.

Cada persona lleva encima una baliza emisora, que se encuentra emitiendo permanentemente. Estas balizas emisoras emiten cada pocos segundos un mensaje de unos pocos *bytes*. Este mensaje es utilizado para ser detectado por las balizas receptoras. Los dispositivos receptores se encuentran situados en las paredes o techos de las salas. Cuando una persona se acerca a un dispositivo receptor, este detecta la baliza emisora de la persona y envía sus datos al servidor.

Desde el punto de vista del receptor, hay cuatro fases:

Fase 1: escaneo

El ESP32 se encuentra escaneando permanentemente a la espera de detectar algún dispositivo cercano a él.

Fase 2: detección

El ESP32 ha detectado un dispositivo Bluetooth Low Energy cercano.

Fase 3: filtrado

Es posible que se haya detectado un dispositivo Bluetooth Low Energy que no sea una baliza emisora usada para este propósito, por lo que es necesario filtrar los dispositivos detectados. Primero se comprueba que el protocolo al que pertenece el dispositivo detectado sea *iBeacon*, y una vez asegurado eso, se comprueba si su UUID se encuentra en una lista de UUIDs que pertenecen a la empresa. En caso de que ambas condiciones se cumplan, el protocolo y el UUID, se puede asegurar que el dispositivo detectado es de la empresa.

Fase 4: envío de datos

Una vez confirmado que el dispositivo detectado es de la empresa, se recuperan y guardan el *major* y el *minor* y se envía un mensaje al servidor vía Wi-Fi a través de una petición HTTP PUT para guardar los datos en el servidor.

3.2.2. Recepción de datos

Este es el primer módulo de este proyecto realizado en *Kotlin* a través de la API de Easygoband. Para la realización de este módulo se ha empezado validando la petición recibida en el servidor, es decir, comprobando que el usuario recibido como parámetro existe en la base de datos. En caso contrario, se devuelve un código de error.

Una vez validada la petición, se realiza una operación llamada *Upsert* [6], lo que supone añadir la localización a la persona si no tenía ninguna, o actualizándosela si ya tenía una previamente.

3.2.3. Mapas de calor

Este módulo ha sido el más extenso de los que se han tenido que desarrollar. La creación de los mapas de calor ha implicado el desarrollo de una nueva entidad dentro del programa, que ha implicado un mayor trabajo. No ha sido así con los otros dos módulos, donde se ha ampliado funcionalidad sobre entidades ya existentes.

Se ha dado soporte a las operaciones de lectura, listado, y a un *Upsert*, que implica creación si no existe, y actualización si existe.

Cada mapa de calor contiene un id, un nombre y un listado de zonas destacadas dentro del mapa. Cada una de estas zonas contiene unas coordenadas para su localización y un valor que representa la forma en la que se representará esa zona concreta en el mapa, como un punto o un cuadrado.

3.2.4. Ocupación por zonas

La ocupación por zonas ha sido el último módulo desarrollado. Para este módulo, el servidor recibe una petición con un listado de zonas como parámetros. Una vez validadas todas las zonas, el servidor calcula la ocupación de cada una teniendo en cuenta la última localización de cada persona. Por último, una vez calculada la ocupación de cada zona, el servidor genera una respuesta con los datos solicitados.

3.2.5. Base de Datos

En este apartado se explica el diseño de la base de datos realizado, junto a un diagrama en la figura 3.2 que permita su visualización. Por simplicidad, en el diagrama solo se ha incluido aquellas tablas que se han tenido que realizar, junto a una tabla ya existente que permite comunicar las tablas creadas.

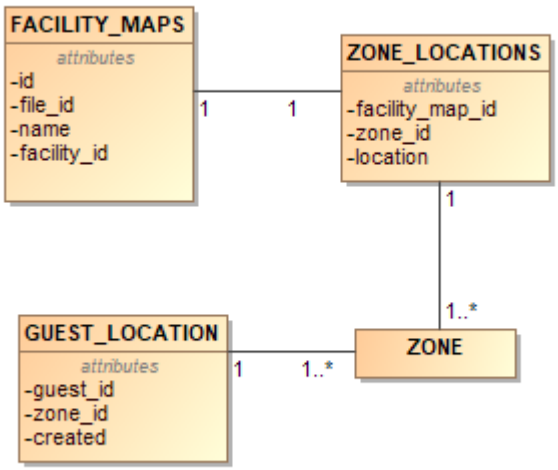

Figura 3.2: Diseño conceptual de la base de datos

En la tabla *GUEST_LOCATION* solo se guarda la última localización del huésped, no un histórico de sus movimientos. En la tabla *ZONE* no se muestra ningún atributo puesto que no se ha modificado esa tabla ni hay ningún atributo relevante para el trabajo realizado, solo se incluye para ver la relación entre la tabla *GUEST_LOCATION* con el resto del sistema.

Capítulo 4

Implementación y pruebas

4.1. Detalles de implementación

4.1.1. Bluetooth Low Energy

Para la realización de este módulo se ha utilizado un dispositivo ESP32 programado desde cero utilizando el IDE Arduino. Para facilitar la programación se ha utilizado una librería creada por Neil Kolban que facilita trabajar con Bluetooth Low Energy para el ESP32 y proporciona soporte para *iBeacon*, el protocolo elegido para trabajar [35].

En un principio, la librería utilizada estaba pensada para que el ESP32 actuara como periférico y enviara mensajes, no para recibirlos. Afortunadamente, a partir de un *issue* [36] de GitHub, el desarrollador de la librería hizo los cambios necesarios para que pudiera funcionar también como escáner.

Para la realización de este módulo, la funcionalidad se ha realizado en tres partes: escaneo y detección, filtrado y envío de datos.

Escaneo y detección

La función de este submódulo es escanear y detectar todos los dispositivos BLE al alcance del ESP32, independientemente de si los dispositivos emisores son aquellos que nos interesa detectar o son otros dispositivos. La base del código para este submódulo se ha realizado a partir de un ejemplo proveniente de la propia librería, que ya realiza esta función.

A este ejemplo se le han probado y cambiado diferentes parámetros de configuración para entender en qué cambia el comportamiento del ESP32 y, finalmente, elegir la más adecuada. Entre los parámetros de configuración se encuentran el tiempo entre escaneos o si se quiere detectar dispositivos repetidos durante un mismo escaneo.

Al final se ha decidido optar por un tiempo de 5 segundos entre escaneos y que no se detecten dispositivos repetidos. Permitir la detección de dispositivos repetidos implicaba que algún dispositivo se detectara hasta cinco o incluso más veces en un mismo escaneo, lo que aumentaría innecesariamente el número de peticiones al servidor.

Filtrado

Una vez detectado un dispositivo Bluetooth Low Energy, es necesario comprobar si es de las balizas emisoras de los clientes. Esta comprobación se realiza comparando el UUID del dispositivo detectado pertenece a una lista ya conocida de UUIDs válidos, es decir, de la empresa.

Para acceder al UUID de un dispositivo detectado se llama al método *getUUID()*, proporcionado por la clase *BLEAdvertisedDevice*, que es a la que pertenece un dispositivo al ser detectado; sin embargo, debido a que *iBeacon* utiliza una estructura de mensajes diferente, usar este método provoca un error puesto que no da un UUID válido.

Por tanto, para solucionar este problema, es necesario convertir el dispositivo detectado de la clase *BLEAdvertisedDevice* a la clase *BLEBeacon*. Esta clase proporciona funcionalidad específica para trabajar solo con dispositivos *iBeacon*; no obstante, no existe un método específico para convertirlo a esa clase, debido a que, originalmente, la librería estaba pensada para que el ESP32 actuara como baliza emisora y no como dispositivo receptor. Tras estar buscando una solución, en un *issue* de GitHub una persona comentaba este mismo problema y el propio Neil Kolban modificó la librería para ofrecer esa funcionalidad [36].

Una vez convertido el dispositivo detectado a la clase *BLEBeacon* ya es posible acceder correctamente a su UUID a través del método *getProximityUUID()*. Una vez hecho esto, el UUID obtenido se compara con una lista de UUIDs para comprobar que el dispositivo detectado pertenece a la empresa.

Envío de mensajes

Una vez que se sabe que el dispositivo detectado pertenece a la empresa, se accede al *major* y *minor* del dispositivo y se envían a través de Wi-Fi al servidor con una petición PUT de HTTP. En concreto, los datos que se envían son el *major* y *minor*, la hora, el UUID del ESP32 que lo ha detectado y la zona.

En la figura 4.1 se puede ver de manera gráfica este módulo, donde cada ESP32 detecta por Bluetooth Low Energy varios dispositivos y envía sus datos por Wi-Fi al servidor.

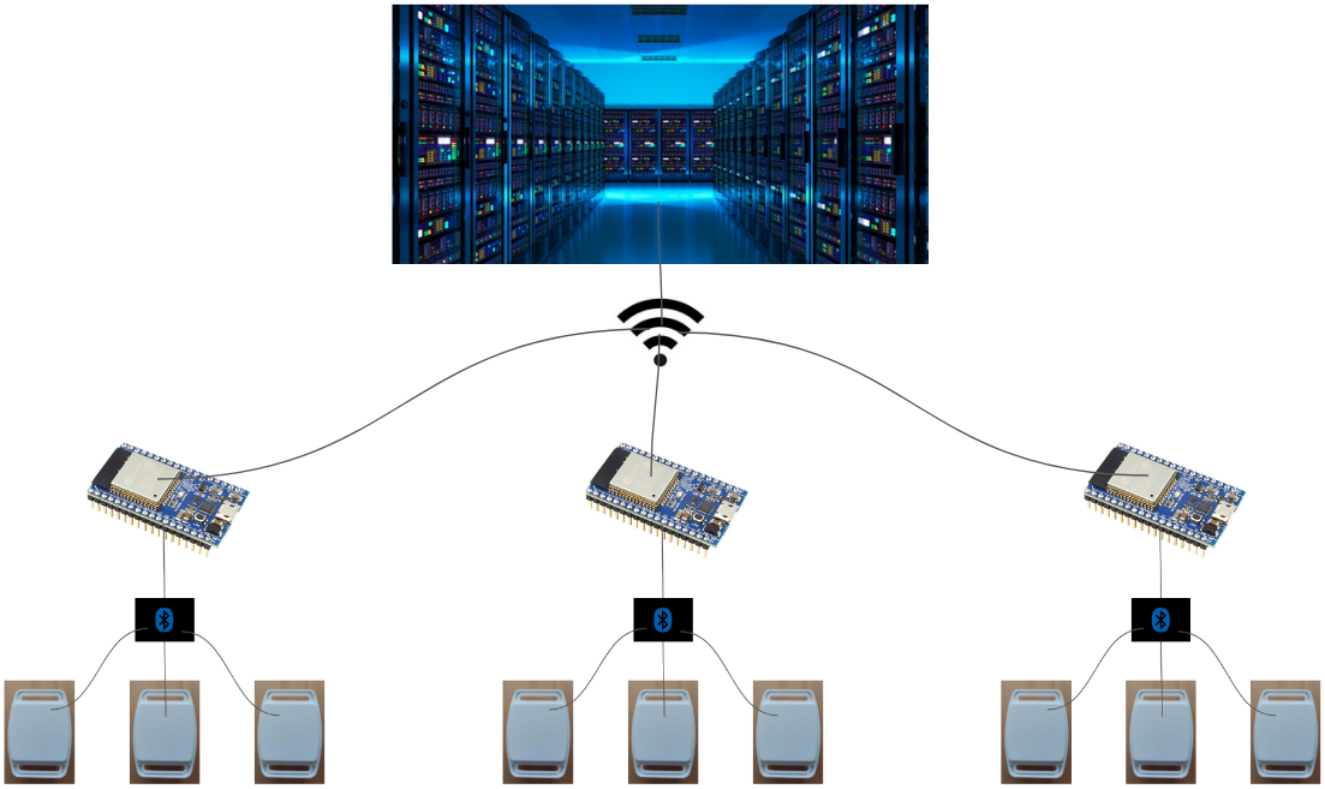

Figura 4.1: Esquema del módulo de geolocalización

Se ha intentado que el envío se realizara a través de HTTPS y no HTTP, pero por problemas desconocidos con el certificado de seguridad, se ha tenido que descartar.

Para terminar con este módulo, durante su desarrollo se han producido algunos errores conocidos como *Guru Meditation Error* [34]. Estos errores han sido producidos por accesos a la hora de almacenar datos en sectores de memoria prohibidos y lectura de datos con punteros con valor NULL en lugar de una dirección de memoria válida. La salida que produce este error es ininteligible, como se puede ver en la figura 4.2.

```
Guru Meditation Error: Core  1 panic'ed (Cache disabled but cached memory region accessed)
Register dump:
PC      : 0x400d8f4c  PS      : 0x00060034  A0      : 0x400829fc  A1      : 0x3ffc0bf0
A2      : 0x00000003  A3      : 0x3ffc4dac  A4      : 0x00000008  A5      : 0x400d8f4c
A6      : 0x00000003  A7      : 0x00060623  A8      : 0x8008120c  A9      : 0x00000001
A10     : 0x00060623  A11     : 0x3ffd7d20  A12     : 0x80085e90  A13     : 0x3ffd6a90
A14     : 0x00000003  A15     : 0x00060a23  SAR     : 0x00000016  EXCCAUSE: 0x00000007
EXCVADDR: 0x00000000  LBEG    : 0x4000c2e0  LEND    : 0x4000c2f6  LCOUNT  : 0x00000000

Backtrace: 0x400d8f4c:0x3ffc0bf0 0x400829f9:0x3ffc0c10 0x4000bfed:0x00000000

Rebooting...
```

Figura 4.2: Salida con error *Guru Meditation Error*

Afortunadamente, he podido encontrar un *plugin* que transforma el mensaje lanzado por la excepción a otro con una sintaxis similar a la de una excepción de Java [28]. El último error que he tenido ha sido la escasez de memoria en el dispositivo ESP32. Esto ha sido debido a que el programa usa librerías Wi-Fi y Bluetooth y el ejecutable una vez compilado supera 1 MB, el tamaño establecido en la partición por defecto [21]. Es posible crear una partición personalizada que cumpla con los requisitos necesarios [5], pero Arduino IDE provee varias configuraciones ya hechas. Al final se decidió optar por una de estas configuraciones de Arduino IDE que admite una mayor memoria, con lo que se solucionó ese problema.

4.1.2. Recepción de datos

Por cada mensaje recibido en el servidor, este comprueba si la persona asociada existe y si el dispositivo ya está registrado en la base de datos. Si la petición no pasa esas dos comprobaciones, se lanza un mensaje de error indicando todo lo que ha fallado. En caso de que sí pase, se realiza una operación *Upsert* [6], es decir, si la persona no tiene ninguna localización asociada se realiza un *Insert*, pero si ya tiene un valor en la base de datos, se realiza un *Update*.

4.1.3. Mapas de calor

El desarrollo de este módulo ha supuesto la creación de una nueva entidad en el programa. La creación de una nueva entidad implica un mayor trabajo, puesto que hay que dar soporte a operaciones básicas. Estas operaciones son similares a CRUD (*Create, Read, Update, Delete*) [20], pero con unas pequeñas variaciones. Al final son tres operaciones en lugar de cuatro: *Create* y *Update* se combinan en un *Upsert*, mientras que *Delete* se cambia por un *List*, y *Read* se queda igual.

Debido a la arquitectura elegida por Easygoband, la realización del *Upsert* es la única que conlleva escribir código, mientras que para las otras dos, es suficiente con crear y extender una clase, y es la clase padre la que tiene el código que se ejecutará.

4.1.4. Ocupación por zonas

Para este módulo, el servidor recibe una petición con un listado de zonas de las que obtener su ocupación actual como parámetro. Una vez recibido el listado, se comprueba que todas las zonas son válidas, en caso contrario se lanza un código de error. Si la petición pasa esa comprobación, el servidor realiza una consulta a la base de datos obteniendo la cantidad de huéspedes que tienen como última localización cada zona del listado recibido como parámetro.

Tras la realización de la consulta y la obtención de los datos, el servidor devuelve cada zona con su respectiva ocupación.

4.1.5. Base de datos

La base de datos utilizada en Easygoband es *MySQL*, que, como es una base de datos relacional y sigue el estándar SQL, la adaptación ha sido muy rápida. El desarrollo de la base de datos se ha ido realizando de manera fragmentada, realizando en cada módulo su parte correspondiente. El diagrama de clases se puede ver en la figura 3.2 en la página 28.

4.2. Verificación y validación

Para la realización de las pruebas, se ha seguido la metodología TDD (*Test Driven Development*) [29], o desarrollo dirigido por pruebas. Esta metodología consiste en implementar primero las pruebas antes que el código. Así, una vez desarrollado el código, este se puede probar al momento y comprobar su correcto funcionamiento. Para la implementación de las pruebas se ha utilizado el *framework* Spek [24]. Este *framework* está diseñado para trabajar sobre *Kotlin*.

Para comprobar el correcto funcionamiento de las peticiones HTTP realizadas por el ESP32 para el envío de datos se ha utilizado un *mock* que simule la recepción de la peticiones. El *mock* utilizado ha sido *Beeceptor* [3], que no solo simula la recepción de peticiones, sino que además simula la existencia de rutas y subrutas que reciben la petición, sin necesidad de crearlas, lo que facilita la comprobación del correcto funcionamiento.

Además, se han realizado tests de repositorio para comprobar el correcto funcionamiento de los accesos a la base de datos. Para realizar estos tests eliminando la dependencia externa de la base de datos, se han realizado a través de ficheros XML con unos datos iniciales y unos datos esperados tras la ejecución de la consulta. Solo si los datos esperados coinciden con los iniciales más los de la ejecución se considera superado el test.

Para terminar, al hacer un *commit* y subir el código implementado a GitLab [8], este ejecuta unos tests de CI (Continuous Integration) [4] [9] que facilitan los procesos de integración continua. A través de un fichero *yaml* [13] se configuran las tareas a realizar, tests solamente en este caso.

Una vez que el código ha superado estos tests, recibe una revisión manual por parte del desa-

rrollador jefe para comprobar determinados aspectos que no se pueden comprobar automáticamente, desde algo tan simple como el nombre de una variable hasta buscar una forma más eficiente de realizar una consulta SQL, por ejemplo. En la figura 4.3 se puede ver un esquema que representa el proceso de comprobaciones una vez se realiza un *commit*.

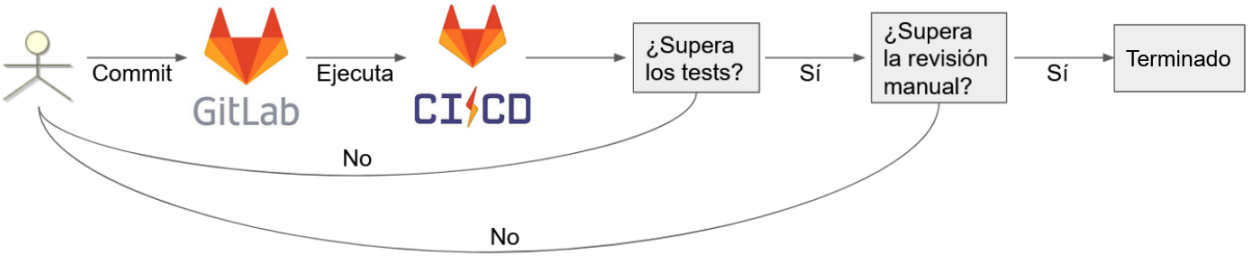

Figura 4.3: Esquema de pruebas al realizar un commit

Capítulo 5

Tecnologías y Herramientas

En este capítulo se presentan las tecnologías y herramientas utilizadas a lo largo del desarrollo del proyecto. Casi todas estas tecnologías han tenido que ser aprendidas por primera vez para este proyecto.

5.1. Herramientas de gestión

5.1.1. Teamwork

Teamwork [26] es un programa de gestión de proyectos independiente de la metodología, pues da soporte tanto a una metodología ágil como a una más tradicional. Permite, además, tener un tablero *Kanban* donde poder ver las tareas a realizar, en qué estado de desarrollo se encuentran y quién es el encargado.

Este programa cuenta con varios tipos de licencias, desde la más básica y gratuita, hasta otras con una mayor funcionalidad y con un coste de unos 10 ó 15 euros por usuario al año.

Esta herramienta es utilizada por más de 20 000 compañías entre las que se incluyen Paypal, Spotify, Honda, HP o Disney, lo que demuestra el éxito y el buen funcionamiento de esta herramienta [7].

5.2. Herramientas de comunicación

La comunicación con el tutor y el resto del equipo de desarrollo ha sido algo vital durante este proyecto debido al confinamiento establecido por el COVID-19.

5.2.1. Teamwork Chat

Además de ser una herramienta de gestión, *Teamwork* proporciona una serie de chats que permiten al equipo comunicarse entre ellos. Haciendo una analogía con *WhatsApp*, *Teamwork* proporciona tanto chats individuales como grupales, además de poder crearse tantos como sean necesarios [27].

Para solucionar dudas sobre la API, se decidió crear un grupo entre los alumnos en prácticas y varias personas encargadas de su mantenimiento con el fin de agilizar las respuestas y minimizar el tiempo de espera.

5.2.2. Google Meet

Al seguir en Easygoband una metodología basada en *Scrum* es necesario la realización de reuniones de seguimiento. Durante el primer mes estas reuniones se realizaban en la propia empresa de manera presencial, pero tras el confinamiento, estas pasaron a ser *online*, realizándose a través de esta herramienta proporcionada por *Google* [10].

5.2.3. Anydesk

Anydesk [1] es un programa de escritorio remoto que permite a un usuario tomar el control del dispositivo de otro usuario. Se ha utilizado entre el supervisor de la empresa y el alumno para solucionar fácilmente dudas y revisar el trabajo realizado.

Esta herramienta ofrece varias licencias diferentes según las necesidades, gratuita con funcionalidad limitada o de pago con una mayor funcionalidad.

5.3. Herramientas de documentación

5.3.1. Swagger

Swagger [25] facilita y homogeneiza la documentación de rutas de una aplicación, permitiendo no solo ver de manera gráfica qué tipo de petición recibe cada ruta, sino que también se pueden detallar los parámetros que se tienen que recibir, qué tipo y formato tienen y si son opcionales u obligatorios.

Antes de empezar a desarrollar una nueva característica es necesario documentar primero la ruta que se quiere crear, qué parámetros y qué petición HTTP recibe.

Al igual que *Anydesk* o *Teamwork*, *Swagger* ofrece distintas licencias según la funcionalidad que se desee contratar, teniendo una licencia gratuita básica y varias de pago con mayor o menor funcionalidad en función del precio.

5.4. Control de versiones

5.4.1. GitLab

GitLab [8] es un servicio web de control de versiones *open source* basado en *Git*. Además de proporcionar un repositorio, también ofrece soporte para una *wiki* o documentación y control de errores.

GitLab ofrece funcionalidad básica gratuita y planes mensuales o anuales con características extra, cuyos precios van desde los 5 euros hasta los 100 euros en función del plan que de desee contratar.

5.5. Lenguajes de programación

5.5.1. C++

C++ es un lenguaje de programación creado en 1983 con la intención de ser una extensión de C e incluyendo la funcionalidad de las clases y objetos.

Se ha utilizado para la programación del ESP32 junto a la librería de Neil Kolban [35].

5.5.2. Kotlin

Kotlin [16] es un lenguaje de programación creado en 2012 por la empresa *JetBrains*. Esta empresa también ha desarrollado *IDEs* como *IntellJ* o *Pycharm* [14]. Este lenguaje adquirió una mayor popularidad cuando *Google* decidió proclamarlo como lenguaje de programación oficial de *Android*, junto a *Java*.

5.6. Base de datos

5.6.1. MySQL

MySQL [19] es un sistema gestor de base de datos relacional, multiplataforma y *open-source* desarrollado por Oracle. Su lanzamiento fue en 1995 y continúa en activo, siendo su última versión en abril de 2020.

5.7. IDEs

5.7.1. Arduino IDE

El IDE Arduino [2] es una herramienta de desarrollo multiplataforma y *open-source* que facilita el desarrollo de microprocesadores.

5.7.2. IntellJ

Al igual que *Kotlin*, *JetBrains* también ha desarrollado *IDEs* como *IntellJ* [11] o *Pycharm*. Debido a que tanto *IntellJ* como *Kotlin* han sido desarrollados por la misma empresa, este *IDE* proporciona soporte completo para el lenguaje. Además, existe un *plugin*, creado también por *JetBrains*, para aprender a programar en este lenguaje dentro del propio *IDE* [22].

5.8. Frameworks y librerías

5.8.1. ESP32 Snippets

ESP32 Snippets [35] es una librería creada por Neil Kolban. Esta librería aporta funcionalidad a un ESP32 para trabajar fácilmente con Bluetooth Low Energy, independientemente del protocolo con el que se quiera trabajar. Proporciona soporte tanto para *Eddystone* como para *iBeacon*, pero para este proyecto solo se ha utilizado la parte de *iBeacon*. Aunque actualmente Neil Kolban no se encargua de su mantenimiento, cuenta con una comunidad bastante activa dispuesta a resolver las dudas expuestas por otros usuarios a través de *issues* de GitHub.

5.8.2. JOOQ

JOOQ [15] es un *framework* de generación de código para sentencias SQL. Este *framework* funciona a través de clases y constantes autogeneradas en función de la estructura de la base de datos. Esto implica un menor número de errores al no depender de poner el nombre exacto de la tabla y el atributo sobre el que se realiza la consulta. Sin embargo, este *framework* tiene sus limitaciones, puesto que su uso se limita a las consultas y no permite la creación de nuevas tablas o modificación de las existentes.

JOOQ cuenta con una serie de licencias en función de las necesidades, gratuitas, de pago mensual o anual o de un único pago. La licencia gratuita, al ser una licencia de Apache, se puede utilizar para el desarrollo de *software* comercial [17].

5.9. Pruebas

5.9.1. Spek

Spek [24] es un *framework* orientado a la creación y ejecución de pruebas a través de funciones *lambda*.

Dentro de *Spek* se pueden utilizar dos tipos de estilo:

- Gherkin: con escenarios válidos e inválidos.
- Especificación: tiene una sintaxis basada en *Jasmine*, un *framework* de pruebas orientado a *JavaScript*.

5.9.2. Beeceptor

Beeceptor [3] es un *mock* de recepción de peticiones HTTP *online*. De esta manera se puede ver cuál es el código HTTP recibido como respuesta y comprobar que los datos enviados son correctos.

En este proyecto se ha utilizado para comprobar el correcto envío de los datos desde el ESP32 hasta el servidor.

5.9.3. Gitlab CI

Además de las funcionalidades que se han mencionado anteriormente de *GitLab*, *GitLab CI* [4] proporciona una serie de *tests* automatizados al realizar un *commit* a través de *pipelines*. Estos *tests* proporcionan una capa más de seguridad y de protección ante errores en el código.

5.9.4. XML

XML [12] se ha utilizado como herramienta de pruebas para comprobar los accesos a la base de datos. Para eliminar esa dependencia externa, el código de las pruebas relativas a la base de datos requiere de unos ficheros XML que simulen esa dependencia. Se trata de una serie de ficheros en el que uno de ellos muestra el estado inicial de los datos y otro, estado final. Si el fichero resultante tras la modificación al estado inicial es igual al estado final, se considera el *test* superado.

Capítulo 6

Conclusiones y posibles mejoras

6.1. Conclusiones

Como conclusión, decir que he aprendido numerosas técnicas y herramientas que sin duda serán muy útiles y utilizaré en el futuro. Además, he aprendido también sobre geolocalización, pero utilizando una tecnología no tan conocida. Bluetooth Low Energy tiene mucho potencial debido a su bajo coste, pero, a su vez, es una tecnología limitada puesto que no da una ubicación exacta. Por ejemplo, sirve para decir que una persona ha entrado en una sala pero no para decir en qué punto exacto de ella.

Por otra parte, he aprendido a valorar la importancia de la realización de tests, y la confianza que da tenerlos para realizar un programa correcto.

Además, he visto la importancia de mantener un código limpio, siguiendo los principios SOLID, y cómo esto, junto a un amplio conjunto de tests, hacen que trabajar con un programa sea muy fácil.

Por último, a nivel profesional, he conocido cómo es trabajar en una empresa, el ambiente que tiene y las exigencias que se tienen que cumplir, y cómo es trabajar y centrarte en un único proyecto.

6.2. Posibles mejoras

Este proyecto cuenta con varias posibles mejoras que se le pueden aplicar, las más inmediatas son la creación del fichero con la imagen del mapa de calor y el listado con la ocupación de las zonas. Además, se podría modificar ligeramente el servidor y la base de datos para que almacene un histórico de las localizaciones de los huéspedes y, una vez hecho esto, aplicar una inteligencia artificial con los datos obtenidos para establecer patrones de movimiento y orientar a los hoteles y parques de atracciones con posibles cambios y mejoras.

Bibliografía

[1] Anydesk. https://anydesk.com/es. [Consulta: 26 de junio de 2020].

[2] Arduino ide. https://www.arduino.cc/en/Main/Software. [Consulta: 30 de junio de 2020].

[3] Beeceptor. https://beeceptor.com/. [Consulta: 26 de junio de 2020].

[4] Conceptos básicos de gitlab ci. https://elfreneticoinformatico.com/conceptos-basicos-de-gitlab-ci/. [Consulta: 26 de junio de 2020].

[5] Creación de particiones en el esp32. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/partition-tables.html. [Consulta: 26 de junio de 2020].

[6] Definición de *Upsert*. https://www.definitions.net/definition/upsert. [Consulta: 26 de junio de 2020].

[7] Empresas que utilizan teamwork. https://www.teamwork.com/customer-stories/#page1. [Consulta: 26 de junio de 2020].

[8] Gitlab. https://about.gitlab.com/. [Consulta: 26 de junio de 2020].

[9] Gitlab ci. https://about.gitlab.com/webcast/automate-security-ci/. [Consulta: 26 de junio de 2020].

[10] Google meet. https://apps.google.com/meet/how-it-works/. [Consulta: 26 de junio de 2020].

[11] Intellj. https://www.jetbrains.com/idea/. [Consulta: 30 de junio de 2020].

[12] Introducción a xml. https://www.mundolinux.info/que-es-xml.htm. [Consulta: 30 de junio de 2020].

[13] Introducción a yaml. https://www.tutorialspoint.com/yaml/index.htm. [Consulta: 26 de junio de 2020].

[14] Jetbrains. https://www.jetbrains.com/. [Consulta: 26 de junio de 2020].

[15] Jooq. https://www.jooq.org/. [Consulta: 30 de junio de 2020].

[16] Kotlin. https://kotlinlang.org/. [Consulta: 26 de junio de 2020].

[17] Licencias de jooq. https://www.jooq.org/legal/licensing. [Consulta: 30 de junio de 2020].

[18] Modelo de estimación cocomo. https://www.eoi.es/blogs/cesaraparicio/2012/05/06/el-modelo-cocomo-para-estimar-costes-en-un-proyecto-de-software/. [Consulta: 26 de junio de 2020].

[19] Mysql. https://www.mysql.com/. [Consulta: 26 de junio de 2020].

[20] Operaciones *CRUD*. https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/crud-las-principales-operaciones-de-bases-de-datos/. [Consulta: 26 de junio de 2020].

[21] Particiones del esp32. https://esp32.com/viewtopic.php?t=5549. [Consulta: 26 de junio de 2020].

[22] Plugin para aprender a programar en kotlin. https://www.jetbrains.com/edu-products/download/#section=idea-Kotlin. [Consulta: 30 de junio de 2020].

[23] Scrum. https://proyectosagiles.org/que-es-scrum/. [Consulta: 26 de junio de 2020].

[24] Spek. https://www.spekframework.org/specification/. [Consulta: 26 de junio de 2020].

[25] Swagger. https://www.chakray.com/es/swagger-y-swagger-ui-por-que-es-imprescindible-para- [Consulta: 26 de junio de 2020].

[26] Teamwork. https://www.teamwork.com/es/. [Consulta: 26 de junio de 2020].

[27] Teamwork chat. https://www.teamwork.com/es/chat/. [Consulta: 26 de junio de 2020].

[28] *Plugin* utilizado para decodificar el mensaje con las excepciones. https://github.com/me-no-dev/EspExceptionDecoder. [Consulta: 26 de junio de 2020].

[29] *Test Driven Development*. https://www.guru99.com/test-driven-development.html. [Consulta: 26 de junio de 2020].

[30] Adafruit. Gap. https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap. [Consulta: 26 de junio de 2020].

[31] Bluetooth. Bluetooth low energy. https://web.archive.org/web/20170310111443/https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/low-energy. [Consulta: 26 de junio de 2020].

[32] Bluetooth. Gatt. https://www.bluetooth.com/specifications/gatt/. [Consulta: 26 de junio de 2020].

[33] Espressif. Esp32. https://www.espressif.com/en/products/socs/esp32/overview. [Consulta: 26 de junio de 2020].

[34] Espressif. Guru meditation errors. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/fatal-errors.html#loadprohibited-storeprohibited. [Consulta: 26 de junio de 2020].

[35] Neil Kolban. Librería utilizada para el desarrollo con el esp32. https://github.com/nkolban/esp32-snippets/tree/master/cpp_utils. [Consulta: 26 de junio de 2020].

[36] Neil Kolban. *Issue* de github para convertir el dispositivo detectado a la clase blebeacon. https://github.com/nkolban/esp32-snippets/issues/271#issuecomment-355439602. [Consulta: 26 de junio de 2020].