Grado en Ingeniería Informática

Trabajo Final de Grado

Desarrollo de una aplicación móvil para localización en interiores

Autor:
Adrián PUERTAS CABEDO

Supervisor:
Sergio AGUADO GONZÁLEZ
Tutor académico:
Ismael SANZ BLASCO

Fecha de lectura: 09 de Septiembre de 2016
Curso académico 2015/2016

Resumen

Este documento describe el proceso de desarrollo de un sistema informático para la localización de dispositivos móviles, y por consiguiente sus usuarios, en entornos interiores utilizando el mapa de señales Wi-Fi presente. El sistema se presenta como una alternativa a los actuales sistemas de geolocalización basados en constelaciones de satélites como el GPS debido a la drástica pérdida de la señal que tienen en el interior de edificios.

El sistema está compuesto por el *frontend*: una aplicación móvil para dispositivos Android (principalmente *smartwatches* con Android Wear) que serán el objetivo a localizar y por el *backoffice*: una aplicación web que permita administrar el sistema. Además se precisará del *backend*: una aplicación en el servidor que dé servicio al *frontend*.

El desarrollo de este proyecto se ha llevado a cabo durante la estancia en prácticas realizada en la empresa Soluciones Cuatroochenta y como trabajo final para el grado en Ingeniería Informática de la Universitat Jaume I.

Palabras clave

Android Wear, MQTT, Elasticsearch, Aprendizaje automático.

Keywords

Android Wear, MQTT, Elasticsearch, Machine Learning.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Índice del capítulo

Este capítulo introductorio tiene como finalidad explicar el contexto donde nace la idea y la motivación que ha impulsado a llevar a cabo el proyecto, así como definir los objetivos que se esperan del mismo.

1.1. Contexto y motivación del proyecto

En esta sección se presenta Soluciones Cuatroochenta, la empresa donde se va a llevar a cabo la estancia en prácticas, y se justifica la necesidad y utilidad del proyecto.

1.1.1. La empresa

Soluciones Cuatroochenta es una empresa especializada en el desarrollo integral de aplicaciones para smartphones y tablets, así como programación avanzada a medida para mejorar procesos de trabajo [53].

La empresa se dedica principalmente a desarrollar aplicaciones móviles y aplicaciones web, pero dedica parte de sus recursos a experimentar en otros sectores como los videojuegos o la realidad virtual. Es una empresa en constante evolución, por lo que siempre investiga nuevas tecnologías que puedan aplicarse a futuros productos.

El proyecto que se ha propuesto se va a desarrollar en el ámbito de investigación y desarrollo, con la finalidad de evaluar la viabilidad de un producto que pueda satisfacer las necesidades de clientes potenciales.

1.1.2. El proyecto

Los sistemas de localización en entornos exteriores basados en constelaciones de satélites (*GPS*, *GLONASS*, el futuro *GALILEO*) han demostrado ser una solución robusta que está siendo utilizada en multitud de aplicaciones (localización y guiado de flotas de camiones, operaciones de salvamento, etcétera). Sin embargo, esta solución es impracticable en el interior de edificios, debido fundamentalmente a la drástica pérdida de la señal en ambientes interiores.

Por ello, han aparecido distintas soluciones, de entre las que cabe destacar la localización en interiores basada en huellas de la señal Wi-Fi. Este tipo de soluciones aprovecha el mapa de señales Wi-Fi presentes en el interior de los edificios, y que son generadas por los puntos de acceso Wi-Fi, para localizar a un usuario dentro de un edificio.

Gran parte de los sistemas actuales se basan en el trabajo de Paramvir Bahl [52], que utiliza un procedimiento en dos pasos: en el primer paso se crea el mapa de señales Wi-Fi o radio-mapa, en el segundo paso se localiza al usuario a partir de la señal recibida en su dispositivo móvil, generalmente un teléfono móvil, y del mapa creado en el primer paso.

Un sistema de localización en interiores puede tener muchas aplicaciones. Por ejemplo, la localización de los miembros de un equipo de trabajo desplegados en el interior de un edificio, el estudio de patrones de conducta de clientes o pacientes, etcétera.

1.2. Objetivos del proyecto

El principal objetivo de este proyecto es desarrollar una aplicación móvil que permita la localización de los usuarios en entornos interiores. La aplicación móvil será para *smartwatches* (relojes inteligentes) usando la plataforma *Android Wear*, aunque se deja abierta a una posible ampliación para soportar *smartphones* Android.

Como objetivo secundario cabe destacar la necesidad de desarrollar una aplicación web para la gestión del sistema por parte de los administradores.

Por tanto, podemos desglosar los objetivos generales en:

- Desarrollar una aplicación móvil para *smartwatches* (*Android Wear*) que permita:
  - Construir un mapa de señales Wi-Fi del escenario.
  - Tomar y clasificar muestras de las señales Wi-Fi.
  - Recolectar muestras de forma automática para el seguimiento del dispositivo.

- Desarrollar una aplicación web que permita:
  - Gestionar los dispositivos.
  - Gestionar los usuarios del sistema.

Al tratarse de un proyecto desarrollado durante la estancia en prácticas, existen una serie de objetivos personales:

- Aprender nuevas tecnologías.
- Trabajar en un entorno profesional.
- Aplicar una metodología de trabajo.

Capítulo 2

Descripción del proyecto

Índice del capítulo

El propósito de este capítulo es, dada la idea de producto que abarca el proyecto, formalizar la descripción del mismo. Para ello, se va a realizar una descripción general que explica en qué consiste y en qué partes se divide. Seguidamente, se va a detallar la funcionalidad del sistema a grandes rasgos y se va a concretar el alcance que va a tener y las posibles restricciones.

2.1. Descripción general

El departamento de investigación y desarrollo de Soluciones Cuatroochenta ha decidido desarrollar internamente un sistema informático que permita la localización en escenarios interiores de dispositivos móviles, y por consiguiente sus usuarios, mediante el uso de señales Wi-Fi. La finalidad del proyecto es investigar las tecnologías necesarias para llevar a cabo el desarrollo y finalmente evaluar la efectividad del mismo.

Podemos dividir el nuevo sistema informático en tres componentes:

- Una **aplicación móvil**, que hará las funciones de *frontend* y que usarán los usuarios. Con ella se creará un mapa de las señales Wi-Fi con las localizaciones de interés para el usuario y se tomarán muestras para que luego puedan ser procesadas y analizadas.
- Una **aplicación servidor**, que hará las funciones de *backend* y que será usada como puente entre la aplicación móvil y la base de datos. También será la encargada del procesamiento de las muestras tomadas, tanto para la creación del mapa inicial como para la clasificación de muestras posteriores.

- Una **aplicación web**, que hará las funciones de *backoffice* y que usarán los administradores tanto para gestionar los usuarios como los dispositivos del sistema.

2.2. Funcionalidad

La funcionalidad del sistema, a grandes rasgos, se puede resumir para cada una de las partes como se hace a continuación.

La aplicación móvil debe permitir:

- Registrar el dispositivo en el sistema[1].
- Obtener la configuración específica para el usuario vinculado desde el servidor.
- Construir un mapa de las señales Wi-Fi con las localizaciones de interés.
- Tomar, clasificar y validar muestras de forma manual.
- Tomar y clasificar muestras de forma periódica y automática.

La aplicación web debe permitir:

- Gestionar los usuarios del sistema.
- Gestionar los dispositivos del sistema.
- Vincular un usuario a un dispositivo.
- Desvincular un usuario de un dispositivo.

La aplicación servidor tiene que:

- Mediar entre la aplicación móvil y la base de datos para:
  - Registrar el dispositivo en el sistema. [1]
  - Servir los datos del usuario vinculado al dispositivo.
- Procesar las muestras tomadas por la aplicación móvil para:
  - Crear el mapa de señales Wi-Fi.
  - Clasificar las muestras tomadas con el mapa creado.

[1] Funcionalidad introducida tras el cambio explicado en el apartado 7.4.2.3

2.3. Alcance

El alcance de este proyecto **incluye** el desarrollo integral de la aplicación móvil así como de la aplicación web y la aplicación servidor. Sin embargo, **no incluye** facilitar la lectura de los resultados obtenidos llevando a cabo algún tipo de integración con software específico de análisis de terceros, quedando así los datos expuestos en la base de datos para ser interpretados como mejor le convenga al cliente.

Podemos detallar el alcance desde los siguientes puntos de vista:

Organizativo: el sistema será utilizado por los usuarios, gestionado por los administradores y mantenido por personal técnico.

Funcional: el sistema ha de cubrir la funcionalidad necesaria para gestionar los usuarios y dispositivos, y llevar un registro de las localizaciones donde han estado.

Informático: el sistema es nuevo y no sustituye ningún sistema actual. Los resultados se podrán extraer de la base de datos para ser utilizados por software de análisis de terceros.

2.4. Restricciones

La principal restricción que existe es la **limitación temporal**, dado que se dispone de un plazo fijo de 300 horas para llevar a cabo el proyecto. Además, al tratarse de jornadas de trabajo de 8 horas, el lapso de tiempo a largo plazo se reduce a aproximadamente 8 semanas, por lo que cualquier obstáculo a medio camino supone un gran retraso.

Por otro lado, otra restricción es la **cantidad de personal** dedicado al desarrollo del proyecto, en este caso solamente una persona. Esto influye directamente en la velocidad a la que puede avanzar el desarrollo.

Por último, cabe destacar que también existe una restricción **económica**, ya que el proyecto se realiza durante la estancia en prácticas y no se dispone de ningún presupuesto.

Capítulo 3

Tecnologías y herramientas

Índice del capítulo

En este capítulo se resumen las tecnologías y herramientas utilizadas durante el desarrollo del proyecto.

3.1. Fase de investigación

Al inicio del proyecto se ha dedicado un tiempo a investigar posibles tecnologías y herramientas a utilizar durante el desarrollo del mismo.

Los requisitos iniciales que la empresa ha decidido son:

- Usar una base de datos *NoSQL*, concretamente *Elasticsearch*, debido a que permite almacenar datos con un esquema dinámico y a las capacidades de búsqueda que puede aportar al trabajar con un gran volumen de datos.
- Usar el protocolo *MQTT* (MQ Telemetry Transport) para la comunicación entre los dispositivos móviles y el *backend*, debido a que es un protocolo más ligero que *HTTP* (HyperText Transfer Protocol).

Otro motivo de los requisitos anteriores es promover el uso y conocimiento de nuevas tecnologías.

A partir de aquí, se ha tenido libertad para elegir el resto de tecnologías y herramientas.

3.2. Tecnologías y herramientas utilizadas

En este apartado se van a resumir todas las tecnologías y herramientas utilizadas. Se explicará brevemente que es cada una y para que se ha utilizado en este proyecto.

3.2.1. Lenguajes

3.2.1.1. Java 8

Java [20] es un lenguaje de programación de propósito general. Es un lenguaje con un paradigma orientado a objetos y concurrente. Es uno de los lenguajes más populares y destaca por ser un lenguaje multiplataforma que puede ser ejecutado en cualquier sistema que disponga de una JVM (Java Virtual Machine). Java 8 es la versión actual y ha introducido importantes mejoras como: expresiones *lambda*, referencias a métodos, métodos por defecto para interfaces, programación funcional con las colecciones *stream* con soporte de procesado paralelo, etc.

En este proyecto se ha utilizado Java como lenguaje de programación en todas las aplicaciones para así mantener un lenguaje común. En concreto se ha utilizado la versión Java 8 para poder aprovechar sus nuevas funcionalidades.

3.2.1.2. HTML 5

HTML5 [17], la quinta versión de HTML (HyperText Markup Language), es el lenguaje estándar para la maquetación de páginas web. Estructura el contenido mediante etiquetas semánticas de forma jerárquica, cada etiqueta es un elemento que puede tener diversos atributos así como un contenido. La versión 5 de HTML introduce nuevas etiquetas, algunas para aumentar la semántica (nav, header, footer, section, article, aside, etc) y otras para aumentar la funcionalidad (canvas, video, audio, etc), además introduce nuevas APIs JavaScript para dotar a las aplicaciones web en navegadores de funcionalidades como Drag&Drop, 2D y 3D, geolocalización, almacenamiento local, sockets, workers, etc.

En este proyecto se ha utilizado HTML5 para estructura el contenido de las vistas de la aplicación *backoffice*.

3.2.1.3. CSS 3

CSS (Cascading Style Sheets) [7] es un lenguaje utilizado para definir los estilos visuales a la hora de presentar un documento HTML. Separar el contenido de la presentación permite disponer de presentaciones diferentes para un mismo contenido, por ejemplo para diferenciar entre un vista de escritorio o una vista móvil. También ayuda a la hora de tratar la accesibilidad para diferentes tipos de personas o dispositivos. CSS 3, es la versión actual e introduce novedades como las media queries, nuevos selectores, soporte oficial para más estilos, animaciones, etc.

En este proyecto se ha utilizado CSS para definir los estilos visuales de las vistas de la aplicación *backoffice*.

3.2.1.4. JavaScript

JavaScript [21] es un lenguaje de programación interpretado de alto nivel, dinámico, no tipado y multiparadigma que implementa la especificación ECMAScript. Aunque actualmente se ha popularizado su uso en el lado del servidor con implementaciones como Node.js, siempre ha sido utilizado junto con HTML y CSS para dotar a las aplicaciones web de nueva funcionalidad en el lado del cliente (el navegador).

En este proyecto se ha utilizado JavaScript para la funcionalidad en el lado de cliente de la aplicación *backoffice*.

3.2.1.5. SQL

SQL [38] (Structured Query Language) es un lenguaje declarativo diseñado especialmente para lidiar con bases de datos relacionales. Permite realizar diversas operaciones como definir el esquema de la base de datos y como la consulta, inserción, modificación o eliminación de datos. Su uso más extendido es para realizar la consulta de datos aplicando filtros, reglas de unión, agrupación, ordenamiento, etc.

En este proyecto se ha utilizado SQL para trabajar con la base de datos SQLite embebida en la aplicación *frontend*.

3.2.2. Frameworks y librerías

3.2.2.1. Android SDK

Android SDK [1] es el *Software Development Kit* que incluye las librerías y herramientas para el desarrollo de aplicaciones Android. El desarrollo de aplicaciones Android se realiza en Java pero se precisa de las librerías incluidas en el SDK para poder construir aplicaciones Android y poder acceder a la funcionalidad del sistema operativo y al hardware del dispositivo. El SDK también incluye herramientas para compilar y depurar aplicaciones, gestionar los dispositivos mediante *shell*, emular dispositivos, etc.

En este proyecto se ha utilizado Android SDK para el desarrollo de la aplicación *frontend* debido a que es una aplicación para dispositivos Android.

3.2.2.2. Android Wear

Android Wear [61] es una extensión de la plataforma Android para una nueva generación de dispositivos móviles *wearables* como por ejemplo relojes. Extiende la plataforma base con nuevas funcionalidades especiales para este tipo de dispositivos y para la comunicación entre el *smartphone* y el *wearable*. También especifica un nuevo diseño de interfaz y da soporte a funciones de voz para mayor accesibilidad.

En este proyecto se ha utilizado Android Wear para el desarrollo de la aplicación *frontend* debido a que es una aplicación enfocada principalmente para su uso en *smartwatches* Android.

3.2.2.3. RxJava y RxAndroid

RxJava [73] es la implementación Java de las extensiones reactivas [33] (ReactiveX): una librería de funciones asíncronas y basadas en eventos que permite utilizar un paradigma de programación reactiva. Se basa en el uso de *observables*, los cuales pueden ser encadenadas para formar secuencias y que permiten el procesado de flujos de datos y/o eventos. También añade operadores para componer y transformar las secuencias al estilo de la programación funcional. Destacar que RxJava se encarga de los detalles de bajo nivel para la concurrencia usando hilos y para la sincronización de los mismos.

RxAndroid [72] es una extensión de RxJava que añade los componentes necesarios para el uso de RxJava en aplicaciones Android. Concretamente tiene un manejo especial de los hilos debido a que el hilo principal en Android es especial y se encarga de el renderizado de la interfaz gráfica.

En este proyecto se ha utilizado RxJava y su extensión RxAndroid en la aplicación *frontend* dada la naturaleza asíncrona de las aplicaciones Android y con el fin de evitar el anidamiento desmesurado de *callbacks*.

3.2.2.4. Retrolambda

Retrolambda [66] es un *backport* de Java 8 que permite utilizar expresiones lambda y referencias a métodos en Java 7, 6 o 5. También ofrece soporte limitado para métodos por defecto en interfaces y métodos estáticos en interfaces.

En este proyecto se ha utilizado Retrolambda en la aplicación *frontend* debido a que actualmente Android no tiene soporte oficial para Java 8.

3.2.2.5. Jackson

Jackson [58] es una librería Java que permite el *marshalling* y *unmarshalling* [25] de objetos Java en formato JSON, BSON, CSV, XML, YAML, entre otros.

En este proyecto se ha utilizado Jackson en las aplicaciones *frontend* y *backend* para poder enviar los objetos en formato JSON a través del broker de mensajería.

3.2.2.6. Spring Framework

Spring Framework [69] es uno de los frameworks más populares en el desarrollo Java. Su núcleo es un contenedor IoC (Inversión de control) que nos facilita la inyección de dependencias. Otros módulos que incluye permiten: la programación orientada a aspectos, la autenticación y autorización, el trabajo con bases de datos, el desarrollo de aplicaciones web MVC y de servicios web RESTful, soporte para el testing y mucho más.

En este proyecto se ha utilizado Spring Framework como framework base en las aplicaciones *backend* y *backoffice*, que luego ha sido extendido de diferente manera en cada una.

3.2.2.7. Spring Boot

Spring Boot [67] pretende facilitar el uso de Spring Framework y todo su ecosistema de extensiones, al mismo tiempo que permite producir aplicaciones autónomas. Utiliza la convención sobre la configuración por lo que elimina la necesidad de la extensa configuración XML. Dispone de meta-paquetes de dependencias para facilitar la configuración Maven. Si es necesario incluye un servidor (Tomcat, Jetty u otros) a la hora de empaquetar la aplicación por lo que es totalmente autónoma y no se necesita ningún despliegue especial.

En este proyecto se ha utilizado Spring Boot con la intención de mejorar Spring Framework en las aplicaciones *backend* y *backoffice*.

3.2.2.8. Spring Security

Spring Security [71] es el framework de Spring que provee tanto autenticación como autorización. Ofrece muchos tipos de autenticación: autenticación en memoria, autenticación mediante JDBC o LDAP, etc. También permite ser extendido con un proveedor de autenticación a medida. Tiene detalles como formulario web, autenticación anónima, opción de recordar, roles, etc. También aporta protección contra ataques de sesión, robo de clicks, CSRF (Cross site request forgery), etc.

En este proyecto se ha utilizado Spring Security para proteger el acceso a la aplicación *backoffice*, que solo debe ser accedida por los administradores del sistema.

3.2.2.9. Spring Data

Spring Data [68] es el framework de Spring que facilita el trabajo con bases de datos tanto relacionales como no relacionales. Realmente se compone de muchos sub-proyectos cada uno enfocado a un tipo de bases de datos concreto. Es comúnmente utilizado para trabajar con bases de datos SQL a través del driver JDBC siguiendo la especificación JPA, aunque puede trabajar con bases de datos NoSQL. Se integra con frameworks ORM (Object-relational mapping) como Hibernate. Ofrece clases repositorio que abstraen el funcionamiento a bajo nivel y soporte para transacciones, etc.

En este proyecto se ha utilizado Spring Data para trabajar con la base de datos elasticsearch en las aplicaciones *backend* y *backoffice*.

3.2.2.10. Spring Integration

Spring Integration [70] es el framework de Spring que da soporte a los EIP [11] (Enterprise Integration Patterns). Permite la mensajería entre aplicaciones y soporta la integración con sistemas externos a través de adaptadores. Proporciona todo lo necesario para la implementación de los EIP: enrutadores, canales, transformadores, adaptadores, filtros, activadores, etc.

En este proyecto se ha utilizado Spring Integration para la integración del broker MQTT con la aplicación *backend*, facilitando la recepción y envío de mensajes.

3.2.2.11. Thymeleaf

Thymeleaf [42] es un motor de plantillas XML/XHTML/HTML5 para aplicaciones Java, no necesariamente web. Es una de las muchas alternativas al tradicional JSP [22] (JavaServer Pages) e implementa el concepto de *Natural Templates*, es decir, plantillas que pueden abrirse directamente en un navegador y ser renderizadas correctamente. Dispone de soporte para la internacionalización y de módulos denominados dialectos que amplían el conjunto de etiquetas y atributos disponibles, entre ellos, para la integración con Spring Framework.

En este proyecto se ha utilizado Thymeleaf para el renderizado de plantillas HTML5 en la aplicación *backoffice*, debido a que es el estándar utilizado en Spring Framework.

3.2.2.12. jQuery

jQuery [23] es una librería JavaScript que facilita la manipulación y navegación del DOM HTML, la gestión de eventos, la animación de los elementos, las peticiones AJAX, etc. También hace estandariza todas estas funciones entre distintos navegadores.

En este proyecto se ha utilizado jQuery en la aplicación *frontend* para el uso en el lado del cliente.

3.2.2.13. Bootstrap

Bootstrap [5] es un framework HTML, CSS y JavaScript para el desarrollo de aplicaciones web *responsive*. Permite la rápida y fácil construcción de interfaces web de calidad que se adaptan a cualquier tipo de pantalla y ofrece diversos componentes visuales listos para usar.

En este proyecto se ha utilizado Bootstrap en la aplicación *frontend* para el diseño de la interfaz web.

3.2.3. Herramientas de construcción

3.2.3.1. Maven

Maven [48] es una herramienta para gestionar las dependencias y el ciclo de vida de construcción (compilación, ejecución de tests, empaquetado, etc.) del software. Maven se utiliza principalmente para proyectos Java aunque puede ser utilizado para otros lenguajes. Utiliza XML para la configuración, pero pone la convención sobre la configuración.

En este proyecto se ha utilizado Maven para gestionar las dependencias y automatizar la construcción de las aplicaciones *backend* y *backoffice*, puesto que ambos usan Spring y es el estándar.

3.2.3.2. Gradle

Gradle [14] es una herramienta para gestionar las dependencias y el ciclo de vida de construcción del software, como Maven. La diferencia es que Gradle utiliza un DSL (domain-specific language) en vez de XML para la configuración. El DSL de Gradle es un lenguaje de programación basado en Groovy, por lo que aporta mayor flexibilidad y reduce la verbosidad si se compara con XML. Gradle se utiliza principalmente en Java, Groovy y Scala, pero puede ser utilizado para otros lenguajes.

En este proyecto se ha utilizado Gradle para gestionar las dependencias y automatizar la construcción de la aplicación *frontend* ya que es el estándar para proyectos Android.

3.2.4. Bases de datos

3.2.4.1. SQLite

SQLite [39] es un SGBD (sistema gestor de bases de datos) relacional contenido en una pequeña librería software. A diferencia de otros SGBD, SQLite no tiene una arquitectura cliente-servidor sino que es embedido en la aplicación final. Está disponible en muchos lenguajes de programación y es muy popular para su uso como almacenamiento local en navegadores web y dispositivos móviles debido a su pequeño tamaño.

En este proyecto se ha utilizado SQLite para la persistencia de datos de forma local en la aplicación *frontend* cuando no se dispone de conexión con la aplicación *backend*.

3.2.4.2. ORMLite

ORMLite [31] es una librería Java para el mapeado objeto-relacional que facilita la persistencia y consulta de objetos Java en una base de datos relacional. Tiene soporte para bases de datos SQLite pero también para MySQL, Postgres, H2, entre otras. Permite definir el esquema de la base de datos directamente con anotaciones sobre las clases Java y crea automáticamente las tablas. Abstrae el acceso a los datos con objetos DAO (Data Access Object) y permite la construcción de consultas SQL.

En este proyecto se ha utilizado ORMLite para el trabajo con la base de datos SQLite utilizada en la aplicación *frontend*.

3.2.4.3. Elasticsearch

Elasticsearch [10] es un motor de búsqueda distribuido, escalable y de alta disponibilidad basado en Apache Lucene [2] (un motor de búsqueda full-text). Aunque está enfocado como motor de búsqueda, también es una base de datos NoSQL orientada a documentos (JSON), lo que permite no seguir un esquema de datos fijo y relacionado. Dispone de una driver nativo para Java y de una intefaz RESTful que permite utilizarlo desde cualquier aplicación. Elasticsearch tiene varios complementos o plugins para su monitorización, análisis de los datos, etc.

En este proyecto se ha utilizado Elasticsearch como base de datos principal ya que la empresa así lo ha requerido debido a sus capacidades y con un afán de probar esta tecnología. Elasticsearch se ha utilizado desde las aplicaciones *backend* y *backoffice*.

3.2.5. Testing

3.2.5.1. JUnit

JUnit [24] es un framework que facilita la escritura de tests unitarios y de integración y la ejecución automática de los mismos. Incluye un conjunto de funciones para verificar el resultado del código a testear y permite parametrizar tests con un conjunto de diferentes valores, entre otras funcionalidades.

En este proyecto se ha utilizado JUnit para la implementación de diversos tests unitarios y de integración en las aplicaciones, en especial en el *backend*.

3.2.5.2. Mockito

Mockito [26] es un framework para construir *mocks* que nos va a permitir aislar el código a testear de sus dependencias reales, pudiendo manipular el resultado y comportamiento de esas dependencias. En conjunto con JUnit se utiliza para la implementación de tests unitarios libres de dependencias.

En este proyecto se ha utilizado Mockito para la implementación de diversos test unitarios en los que había que asilar el código de las dependencias reales.

3.2.6. Machine Learning

3.2.6.1. Weka

Weka [65] es una colección de algoritmos de aprendizaje automático o *Machine Learning* [3], por ejemplo: redes neuronales, redes bayesianas, árboles de decisión, etc. Los algoritmos pueden ser aplicados directamente sobre un conjunto de datos o llamados desde código Java. Weka permite la creación y configuración de los algoritmos, su entrenamiento y su uso. Dispone de una interfaz gráfica para el uso de los algoritmos y la visualización de los resultados.

En este proyecto se ha utilizado Weka para crear los clasificadores de muestras Wi-Fi tomadas en la aplicación *backend*.

3.2.7. MQTT

MQTT (MQ Telemetry Transport) es un protocolo ligero de mensajería que sigue el patrón Publicador-Subscriptor sobre TCP/IP. Es habitual su uso en conexiones móviles donde el ancho de banda es limitado.

Se requiere de un broker que es el responsable de distribuir los mensajes a los clientes suscritos al *topic* del mensaje. Los clientes se conectan al broker y publican mensajes o se suscriben a los *topics* de interés para recibir mensajes.

3.2.7.1. Apache Apollo

Apache Apollo [47] es un *broker* de mensajería construido desde las bases del original Apache ActiveMQ [46], no tan completo pero más rápido y ligero. Es un broker multi-protocolo y soporta: MQTT, AMQP, STOMP y OpenWire. También ofrece soporte para SSL y su comunicación puede ser a través de TCP o WebSockets. Disponse de una API RESTful para su gestión y de un panel de administración web.

En este proyecto se ha utilizado Apache Apollo como broker de mensajería debido a su soporte para el protocolo MQTT, ya que la empresa lo había requerido dado su ligereza frente al protocolo HTTP. Apache Apollo se ha utilizado desde las aplicaciones *frontend* y *backend* para la comunicación entre si.

3.2.7.2. Eclipse Paho

Eclipse Paho [57] es un cliente de mensajería para el protocolo MQTT. Ofrece clientes para C/C++, Java, Android, JavaScript, Python, C# .Net y Go. Permite conectarse a un broker MQTT para publicar mensajes en diferentes *temas* así como suscribirse a temas para la recepción de los mensajes publicados.

En este proyecto se ha utilizado Eclipse Paho en la aplicación *frontend* para la comunicación con el *backend* a través del broker de mensajería. La aplicación *backend* también utiliza Eclipse Paho pero no directamente sino a través de Spring Integration.

3.2.8. Control de versiones

3.2.8.1. Git y GitHub

Git [12] es un sistema de control de versiones distribuido. Permite almacenar el código de nuestra aplicación en un repositorio y mantener un control de los cambios efectuados en el mismo. Algunas características de Git son las ramas que permiten trabajar en paralelo, repositorio local pudiendo efectuar cambios que luego se fusionarán en el repositorio remoto, etc. Git permite muchos tipos de *workflows* [49], de los que cabe destacar GitFlow [63].

GitHub [13] es un servicio que ofrece el alojamiento de repositorios Git públicos de forma gratuita y repositorios privados para cuentas de estudiante o de pago. GitHub es la mayor forja de repositorios de software libre y uno de los principales servicios de alojamiento de repositorios Git, seguido de BitBucket de Atlassian.

En este proyecto se ha utilizado Git como sistema de control de versiones para el código fuente de las aplicaciones *backend*, *frontend* y *backoffice*, así como de los scripts para el despliegue con Docker. Se ha seguido el *workflow* GitFlow a la hora de trabajar con Git y se han alojado los repositorios en GitHub.

3.2.8.2. SourceTree

SourceTree [51] es una herramienta gráfica para trabajar con repositorios Git y Mercurial. Permite visualizar y navegar el historial de commits y ramas así como realizar todo tipo de acciones como commit, push, pull, branch, merge, stash, etc. Un añadido muy interesante es la integración del workflow GitFlow ya que facilita mucho su aplicación, aunque su uso es opcional. También permite gestionar las cuentas en servicios remotos y tener fácil acceso a todos los repositorios alojados.

En este proyecto se ha utilizado SourceTree como cliente gráfico para trabajar con los repositorios Git.

3.2.9. Despliegue

3.2.9.1. Docker

Docker [9] es una herramienta que permite empaquetar una aplicación junto con todas sus dependencias en un contenedor que puede ser desplegado de forma trivial en cualquier máquina que disponga del motor Docker. Los contenedores ofrecen un entorno aislado para la aplicación por lo que evita conflictos con otro software y proporciona seguridad, además son más ligeros que una máquina virtual ya que comparten el kernel del sistema operativo anfitrión lo que los hace más rápidos y consumen menos memoria RAM. Dispone de un extenso repositorio central de imágenes base que pueden ser utilizadas y extendidas. Las imágenes son instantáneas del sistema de archivos y se construyen por capas por lo que permite compartir capas intermedias y ahorrar espacio. Docker automatiza el despliegue de aplicaciones por lo que reduce el tiempo invertido y elimina los errores humanos y las inconsistencias del entorno.

En este proyecto se ha utilizado Docker para realizar el despliegue de las aplicacioens *backend* y *backoffice*, así como del *broker* de mensajería y de la base de datos.

3.2.10. Entornos de desarrollo

3.2.10.1. IntelliJ IDEA

IntelliJ IDEA [62] es un IDE (Integrated Development Environment o Entorno de Desarrollo Integrado) para el desarrollo de aplicaciones Java con infinidad de características y funcionalidades, extensiones, integraciones con otros lenguajes y frameworks, herramientas, etc. Con todo esto, facilita el desarrollo de aplicaciones y con ello aumenta la producitividad.

En este proyecto se ha utilizado IntelliJ IDEA como IDE para las aplicaciones *backend* y *backoffice*.

3.2.10.2. Android Studio

Android Studio [60] es el IDE oficial para Android. Está basado en IntelliJ IDEA pero añade integración con Android SDK y todo su ecosistema de herramientas. También dispone de un editor de layouts para las vistas de las aplicaciones, entre otras cosas.

En este proyecto se ha utilizado Android Studio como IDE para la aplicación *frontend*.

3.2.10.3. SublimeText

SublimeText [40] es un sofisticado editor de texto enfocado al desarrollo de código. Es altamente configurable y tiene infinidad de extensiones y soporte para casi cualquier lenguaje. No ofrece todas las características de un IDE pero a cambio es extremadamente liviano y rápido.

En este proyecto se ha utilizado SublimeText como editor de texto de uso general y para el desarrollo de los scripts Docker.

3.2.11. Gestión de proyectos

3.2.11.1. Jira

Jira [50] es una herramienta para la gestión y planificación de proyectos software. Dispone de multitud de extensiones y es altamente configurable. Incluye herramientas para la aplicación de metodologías ágiles como Scrum y Kaban.

En este proyecto se ha utilizado Jira para gestionar y planificar el propio proyecto, es decir, registrar las historias de usuario y tareas y supervisar su progreso.

3.2.12. Diseño

3.2.12.1. MagicDraw UML

MagicDraw UML [64] es una herramienta CASE [15] (Computer Aided Software Engineering o Ingeniería de Software Asistida por Computadora) que nos permite realizar el modelado mediante UML [43] (Unified Modeling Language o Lenguaje Unificado de Modelado) de nuestro sistema en las fases de análisis y diseño.

En este proyecto se ha utilizado MagiDraw UML para la realización de los diagramas UML necesarios, por ejemplo, diagramas de casos de uso y diagramas de clases.

3.2.12.2. Moqups

Moqups [75] es una aplicación web que nos permite crear *wireframes* o *mockups*, es decir, prototipos visuales de la interfaz de usuario de nuestra aplicación.

En este proyecto se ha utilizado Moqups para el diseño de los prototipos de las vistas o interfaces de usuario de las apliacciones *backoffice* y *frontend*.

Capítulo 4

Planificación del proyecto

Índice del capítulo

En este capítulo se explica la metodología a seguir durante la etapa de desarrollo técnico del proyecto y se realiza una primera planificación global del mismo ajustada al tiempo disponible. Por último, se realiza una estimación del coste del proyecto, se explica el seguimiento que se ha hecho y se exponen las desviaciones que han surgido durante el desarrollo.

4.1. Metodología

Para llevar a cabo el proceso de desarrollo de software es conveniente seguir una metodología que defina el modelo del proceso, y que nos proporcione técnicas y herramientas para dar soporte a las actividades que hay que llevar a cabo.

Existen dos grandes grupos de metodologías: clásicas y ágiles. En este proyecto se ha optado por seguir una metodología ágil, concretamente Scrum, aunque no de forma estricta. En este caso se ha realizado primero análisis y diseño, apoyándonos en ciertos aspectos de las metodologías tradicionales como los diagramas UML. Posteriormente se ha seguido la filosofía Scrum haciendo una implementación iterativa con *Sprints*.

Primero se va a comparar brevemente las metodologías tradicionales y ágiles, luego se va a explicar más en detalle el proceso Scrum y algunas técnicas a utilizar como las historias de usuario.

4.1.1. Metodologías tradicionales vs. ágiles

Las **metodologías tradicionales** imponen distintas fases que se deben llevar a cabo durante el proceso de desarrollo de software. Típicamente estas fases son:

- **Definición de requisitos:** definir qué se espera del sistema.
- **Análisis:** especificar qué hará el sistema.
- **Diseño:** especificar cómo lo hará.
- **Implementación:** construir el sistema.
- **Pruebas:** verificar y validar el sistema.
- **Despliegue:** llevar el sistema a un entorno de producción.

Aunque cada fase tiene su razón de ser, es común encontrar la imposición de ejecutarlas de forma secuencial. Esto lleva a elaborar excesiva documentación en las fases iniciales con la intención de definir al máximo como llevar a cabo las fases posteriores, lo que resulta en un producto cerrado a futuros cambios y un consumo considerable de tiempo. Además, no se entrega el producto hasta el final, por lo que no se puede recibir ningún tipo de retroalimentación que ayude a reconducir el proyecto si fuese necesario.

Por otro lado, las **metodologías ágiles** proponen dar más valor al producto que a la documentación y admiten cambios que nos permitan reconducir el proyecto. Por tanto, se llevan a cabo iteraciones, donde se pretende conseguir un producto funcional que irá creciendo en cada iteración.

Esto no elimina las fases propuestas por las metodologías tradicionales, sino que las reorganiza. Inicialmente se llevará a cabo una definición de requisitos, pero esta no será demasiado detallada y podrá estar sujeta a cambios futuros. Posteriormente, en cada iteración se deberá llevar a cabo una parte de los requisitos, realizando el correspondiente análisis y diseño si fuese necesario, así como la implementación y pruebas, terminando en un producto funcional. También recomiendan realizar la fase de implementación y pruebas de forma conjunta para evitar problemas al final y asegurar la calidad del software. Tras cada iteración se revisa el producto y si este se da por terminado se puede realizar el despliegue.

4.1.2. Scrum

Scrum es un modelo de referencia que define un conjunto de prácticas y roles, y que puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto [36].

Los roles principales en Scrum son el **Scrum Master**, que procura facilitar la aplicación de Scrum, el **Product Owner**, que representa a los **stakeholders** (los interesados externos o internos), y el **Team**, que ejecuta el desarrollo y demás elementos relacionados.

Durante cada sprint, un periodo entre una y cuatro semanas (la magnitud es definida por el equipo y debe ser lo más corta posible), el equipo crea un incremento de software potencialmente entregable (utilizable).

Inicialmente se debe crear el **Product Backlog**, que es un conjunto de requisitos de alto nivel (historias de usuario) priorizados que definen el trabajo a realizar (PBI, Product Backlog Item).

Una vez creado el Product Backlog se puede empezar el **Scrum Sprint Cycle** que se muestra en la figura 4.1.

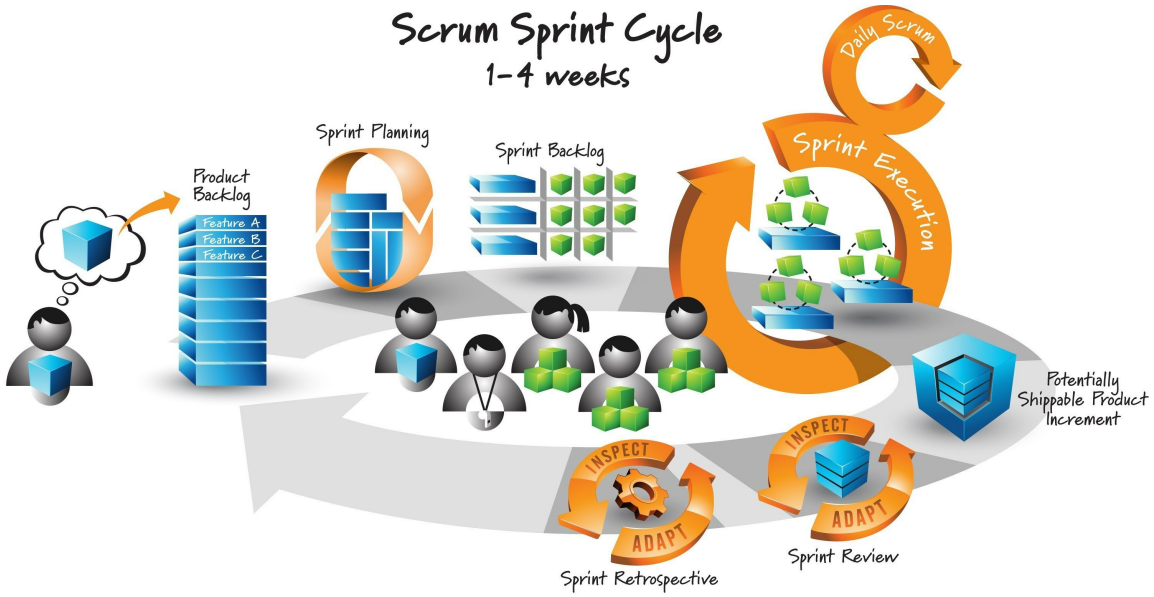


Figura 4.1: Ciclo típico de un sprint en Scrum. Fuente [74]

Los elementos del Product Backlog que forman parte del sprint se determinan durante la reunión de **Sprint Planning**. Durante esta reunión, el Product Owner identifica los elementos del Product Backlog que quiere ver completados y los hace del conocimiento del equipo. Entonces, el equipo conversa con el Product Owner buscando claridad y magnitud adecuadas para luego determinar la cantidad de ese trabajo que puede comprometerse a completar durante el siguiente sprint.

Durante el sprint, nadie puede cambiar el **Sprint Backlog**, lo que significa que los requisitos están congelados durante el sprint.

Una vez se tiene claro que trabajo se va a realizar durante el sprint y en que tareas se descompone, se puede empezar con la ejecución del sprint, donde se realiza la implementación del software. Idealmente, se sigue la práctica del TDD (Test-Driven Development) [41] realizando de forma conjunta implementación y pruebas. Si fuera necesario, se realizan análisis y diseño del software a implementar.

Cada día se hace la reunión **Daily Scrum** donde los miembros del equipo pone al día al resto de la situación actual del trabajo.

Una vez llegado al final del sprint, se debe realizar la reunión **Sprint Review**, donde se valida si el producto terminado cumple las expectativas. Es entonces cuando se decide si el producto está terminado o no, en cuyo caso seguiría una nueva iteración.

Por último, es recomendable realizar una reunión **Sprint Retrospective**, en la cual todos los miembros del equipo dejan sus impresiones sobre el sprint recién superado. El propósito de la retrospectiva es realizar una mejora continua del proceso.

4.1.3. Historias de usuario

Una historia de usuario es una representación de un requisito, escrito en una o dos frases, utilizando el lenguaje común del usuario [16]:

```
Como [rol], quiero [acción] para así [motivo].
```

Las historias de usuario son utilizadas en las metodologías de desarrollo ágiles para la especificación de requisitos, ya que son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Además, las historias de usuario permiten responder rápidamente a los requisitos cambiantes.

Las historias de usuario deben ser [19]:

- **Independientes** unas de otras: De ser necesario, combinar las historias dependientes o buscar otra forma de dividir las historias de manera que resulten independientes.
- **Negociables:** La historia en sí misma no es lo suficientemente explícita como para considerarse un contrato, la discusión con los usuarios debe permitir esclarecer su alcance y éste debe dejarse explícito bajo la forma de pruebas de validación.
- **Valoradas** por los clientes o usuarios: Los intereses de los clientes y de los usuarios no siempre coinciden, pero en todo caso, cada historia debe ser importante para alguno de ellos más que para el desarrollador.
- **Estimables:** Un resultado de la discusión de una historia de usuario es la estimación del tiempo que tomará completarla. Esto permite estimar el tiempo total del proyecto.
- **Pequeñas:** Las historias muy largas son difíciles de estimar e imponen restricciones sobre la planificación de un desarrollo iterativo.
- **Verificables:** Las historias de usuario cubren requerimientos funcionales, por lo que generalmente son verificables. La verificación debe automatizarse, de manera que pueda ser verificada en cada entrega del proyecto.

En el proyecto se van a hacer uso de las historias de usuario como plantilla para documentar los requisitos pedidos por el usuario. Las historias de usuario pueden también incluir una lista de posibles escenarios que luego pueden ser convertidos en pruebas de aceptación. Los escenarios se especifican en el siguiente formato:

```
Dado cierto contexto inicial
Cuando sucede algo
Entonces este es el resultado
```

4.2. Planificación

La primera fase del proyecto consiste en desarrollar una propuesta técnica. Durante la primera fase se ha dedicado un tiempo a investigar y probar posibles tecnologías y herramientas para decidir cómo se llevará a cabo el proyecto.

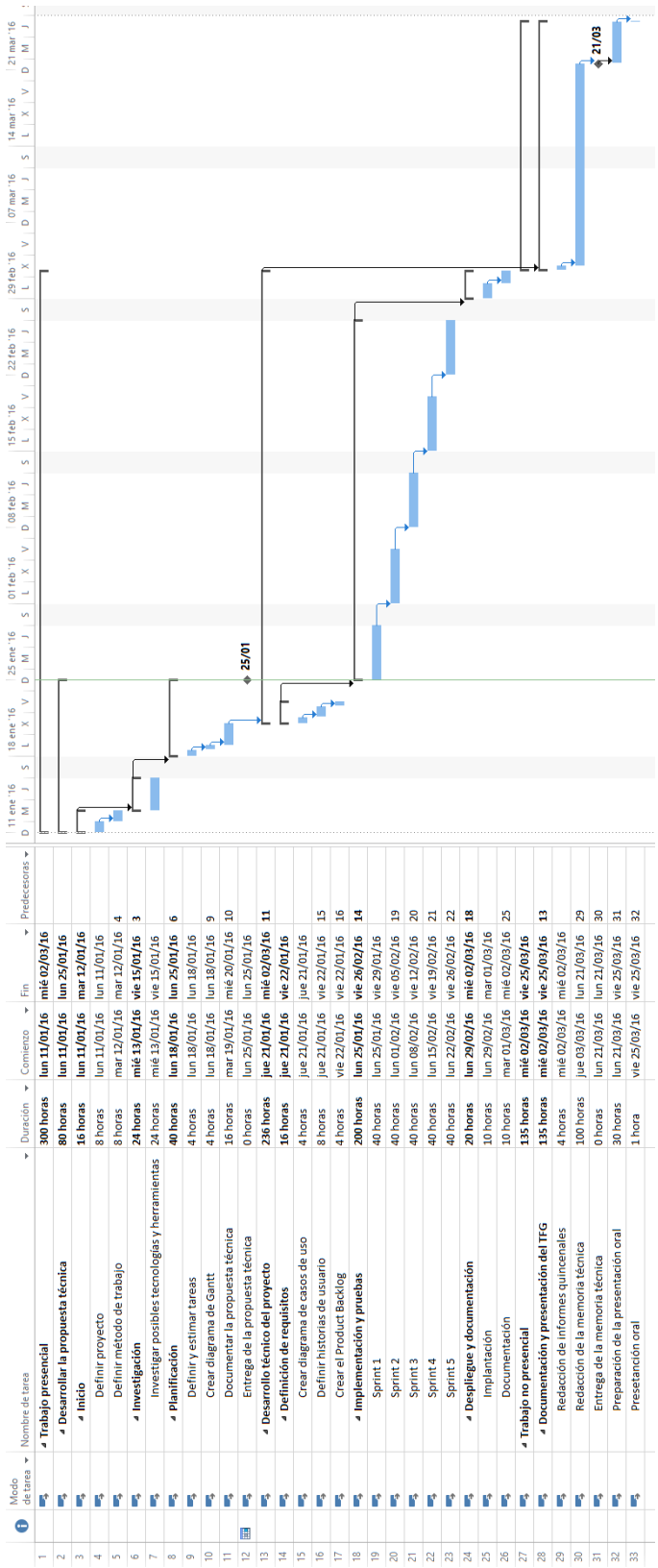
La segunda fase consiste en el desarrollo técnico del sistema, es decir todo el ciclo de vida que la metodología implica.

Primero se realiza una etapa de definición de requisitos donde se crea un diagrama de casos de uso que ayude a tener una visión rápida del sistema. Tras ello, se definen las historias de usuario que finalmente serán estimadas y priorizadas formando así el Product Backlog.

El siguiente paso es la implementación y pruebas del sistema. Como ya se ha explicado, se va a seguir la metodología Scrum, y dado el tiempo disponible, se ha acordado dividir el tiempo en 5 sprints. Cada sprint implica una reunión inicial (Sprint Planning, 4h) donde se elige en que historias de usuario trabajar y estas se descomponen en tareas. Si es necesario, se realizará tanto análisis como diseño de la parte del software que se vaya a implementar. La mayor parte de cada sprint se dedica a implementación y pruebas, de forma conjunta. Cada día hay una reunión (Daily Scrum, 15m) para asegurar que todo marcha bien y al final del sprint hay una reunión (Sprint Review, 4h) donde se valida el producto terminado y opcionalmente se hace una reunión (Scrum Retrospective, 2h) donde se debate sobre la metodología, el equipo y cómo mejorar.

Por último, queda media semana para realizar la implantación y posible documentación, o dado el caso, usarla como tiempo extra para terminar el proyecto.

A continuación se muestra un diagrama de Gantt [8] con la planificación propuesta en la figura 4.2.



	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		**Trabajo presencial**	**300 horas**	**lun 11/01/16**	**mié 02/03/16**	
2		**Desarrollar la propuesta técnica**	**80 horas**	**lun 11/01/16**	**lun 25/01/16**	
3		**Inicio**	**16 horas**	**lun 11/01/16**	**mar 12/01/16**	
4		Definir proyecto	8 horas	lun 11/01/16	lun 11/01/16	
5		Definir método de trabajo	8 horas	mar 12/01/16	mar 12/01/16	4
6		**Investigación**	**24 horas**	**mié 13/01/16**	**vie 15/01/16**	**3**
7		Investigar posibles tecnologías y herramientas	24 horas	mié 13/01/16	vie 15/01/16	
8		**Planificación**	**40 horas**	**lun 18/01/16**	**lun 25/01/16**	**6**
9		Definir y estimar tareas	4 horas	lun 18/01/16	lun 18/01/16	
10		Crear diagrama de Gantt	4 horas	lun 18/01/16	lun 18/01/16	9
11		Documentar la propuesta técnica	16 horas	mar 19/01/16	mié 20/01/16	10
12		Entrega de la propuesta técnica	0 horas	lun 25/01/16	lun 25/01/16	
13		**Desarrollo técnico del proyecto**	**236 horas**	**jue 21/01/16**	**mié 02/03/16**	**11**
14		**Definición de requisitos**	**16 horas**	**jue 21/01/16**	**vie 22/01/16**	
15		Crear diagrama de casos de uso	4 horas	jue 21/01/16	jue 21/01/16	
16		Definir historias de usuario	8 horas	jue 21/01/16	vie 22/01/16	15
17		Crear el Product Backlog	4 horas	vie 22/01/16	vie 22/01/16	16
18		**Implementación y pruebas**	**200 horas**	**lun 25/01/16**	**vie 26/02/16**	**14**
19		Sprint 1	40 horas	lun 25/01/16	vie 29/01/16	
20		Sprint 2	40 horas	lun 01/02/16	vie 05/02/16	19
21		Sprint 3	40 horas	lun 08/02/16	vie 12/02/16	20
22		Sprint 4	40 horas	lun 15/02/16	vie 19/02/16	21
23		Sprint 5	40 horas	lun 22/02/16	vie 26/02/16	22
24		**Despliegue y documentación**	**20 horas**	**lun 29/02/16**	**mié 02/03/16**	**18**
25		Implantación	10 horas	lun 29/02/16	mar 01/03/16	
26		Documentación	10 horas	mar 01/03/16	mié 02/03/16	25
27		**Trabajo no presencial**	**135 horas**	**mié 02/03/16**	**vie 25/03/16**	
28		**Documentación y presentación del TFG**	**135 horas**	**mié 02/03/16**	**vie 25/03/16**	**13**
29		Redacción de informes quincenales	4 horas	mié 02/03/16	mié 02/03/16	
30		Redacción de la memoria técnica	100 horas	jue 03/03/16	lun 21/03/16	29
31		Entrega de la memoria técnica	0 horas	lun 21/03/16	lun 21/03/16	30
32		Preparación de la presentación oral	30 horas	lun 21/03/16	vie 25/03/16	31
33		Presetanción oral	1 hora	vie 25/03/16	vie 25/03/16	32

Figura 4.2: Diagrama de Gantt con la planificación propuesta.

4.3. Estimación de recursos y costes del proyecto

En Soluciones Cuatroochenta, se estima el precio por hora de un desarrollador en aproximadamente 30 €, incluyendo gastos directos e indirectos. Aunque suele presupuestarse por bastante más al cliente final para poder hacer frente a posibles complicaciones, cambios, retrasos, etc.

En este proyecto solo va a participar un desarrollador y la duración del proyecto se estima en un cuantía fija de 300 horas. Por tanto, el coste económico del desarrollo resulta en:

$$\frac{30€}{\text{hora}} * 300\text{horas} = 9000€$$

4.4. Seguimiento del proyecto

Como ya se ha explicado, en Scrum o en una metodología ágil en general, se lleva un seguimiento exhaustivo del proyecto debido a su naturaleza cíclica.

En cada iteración se realizan reuniones para: planificar el proyecto (Sprint Planning), ponerse al día (Daily Scrum), revisar la evolución del producto (Sprint Review) e intentar mejorar el proceso de desarrollo (Sprint Retrospective).

4.5. Desviaciones en la planificación

En todo proyecto pueden surgir contratiempos que requieran desviarse de la planificación inicial. Estos pueden traer consigo un impacto en mayor o menor medida a la productividad debido a los cambios que deban hacerse. Los cambios pueden venir de distintas fuentes: el propio cliente, alteraciones en el presupuesto, leyes o entidades externas, etc.

Por suerte, en un desarrollo ágil es posible hacer frente a estos cambios en cualquier etapa del desarrollo sin que ello suponga un gran problema.

En este proyecto se ha sufrido un cambio en una etapa avanzada del proyecto y por ello se ven a lo largo del documento referencias a este cambio que se explica en detalle en la sección 7.4.2.3. El problema surge de una limitación en la API de Android 6.0 que elimina la posibilidad de obtener la dirección MAC de los dispositivos, que hasta entonces se había utilizado para identificar a los dispositivos ante el servidor. Esto ha implicado realizar cambios de código y diseño, además de suponer un tiempo de trabajo extra.

Capítulo 5

Definición de requisitos

Índice del capítulo

En este capítulo se documenta la definición de requisitos del proyecto. Antes de especificar los requisitos es conveniente realizar un diagrama de casos de uso que muestre, *grosso modo*, los usos que se le van a dar al sistema. Al seguir una metodología ágil, los requisitos se especifican en forma de historias de usuario. Por último, se creará el Product Backlog.

5.1. Diagramas de casos de uso

En este apartado se muestran los diagramas de casos de uso elaborados para cada componente del sistema y se comentan brevemente los casos de uso que lo requieran.

Como se ha explicado en el capítulo 2 (Descripción del proyecto) el sistema se compone de:

- Una aplicación web (*backoffice*) para administrar el sistema.
- Una aplicación móvil (*frontend*) para *smartwatches Android Wear*.
- Una aplicación servidor (*backend*).

5.1.1. Diagrama de casos de uso del backoffice

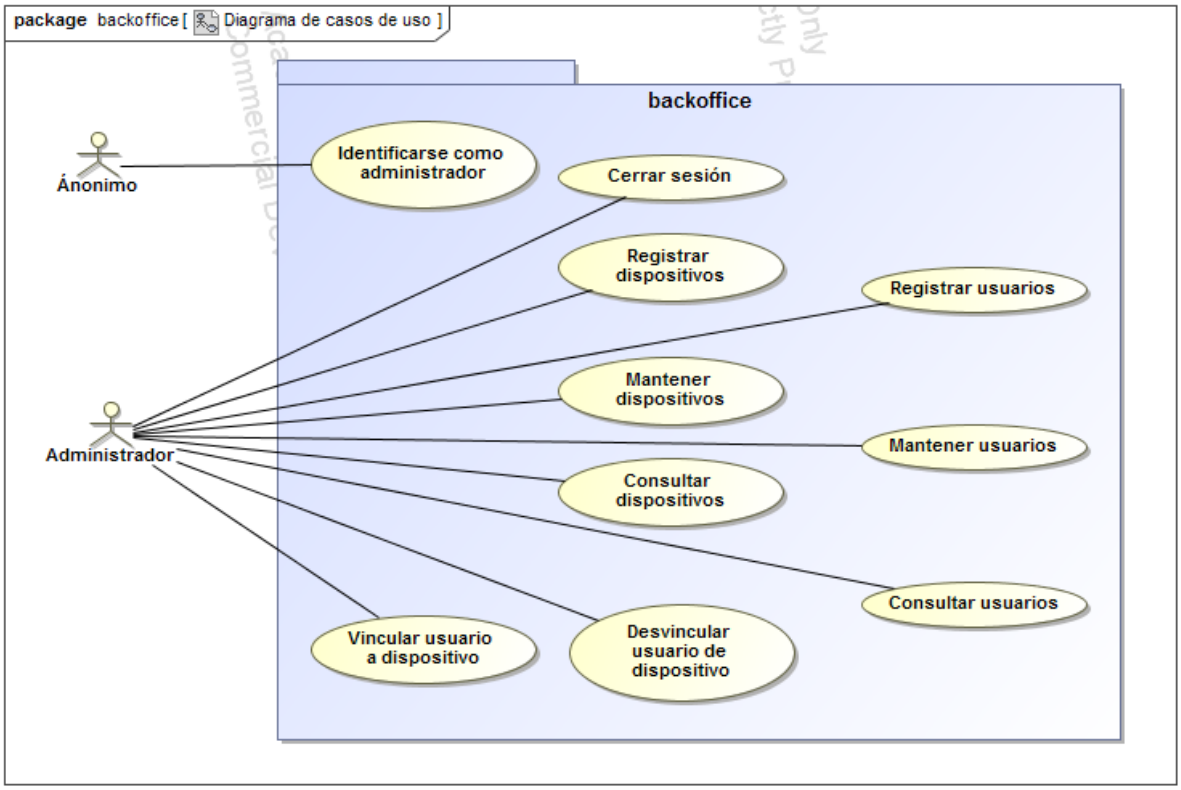


Figura 5.1: Diagrama de casos de uso del backoffice.

Como muestra el diagrama de casos de uso de la figura 5.1, en el backoffice el usuario inicialmente va a tener que iniciar sesión con las credenciales apropiadas para identificarse como administrador. Una vez ha sido identificado, en cualquier momento tiene la opción de cerrar la sesión.

El administrador puede registrar dispositivos y usuarios nuevos o editar parcialmente los ya existentes, pero no va a poder eliminarlos, esto se ha decidido así por mantener siempre la relación con las muestras tomadas. Los dispositivos y usuarios pueden ser consultados tanto como un listado como individualmente en detalle.

Por último, el administrador puede vincular y desvincular un usuario con un dispositivo, esto es: que cuando un usuario está vinculado a un dispositivo, las muestras tomadas con ese dispositivo se asignarán a ese usuario.

5.1.2. Diagrama de casos de uso del frontend

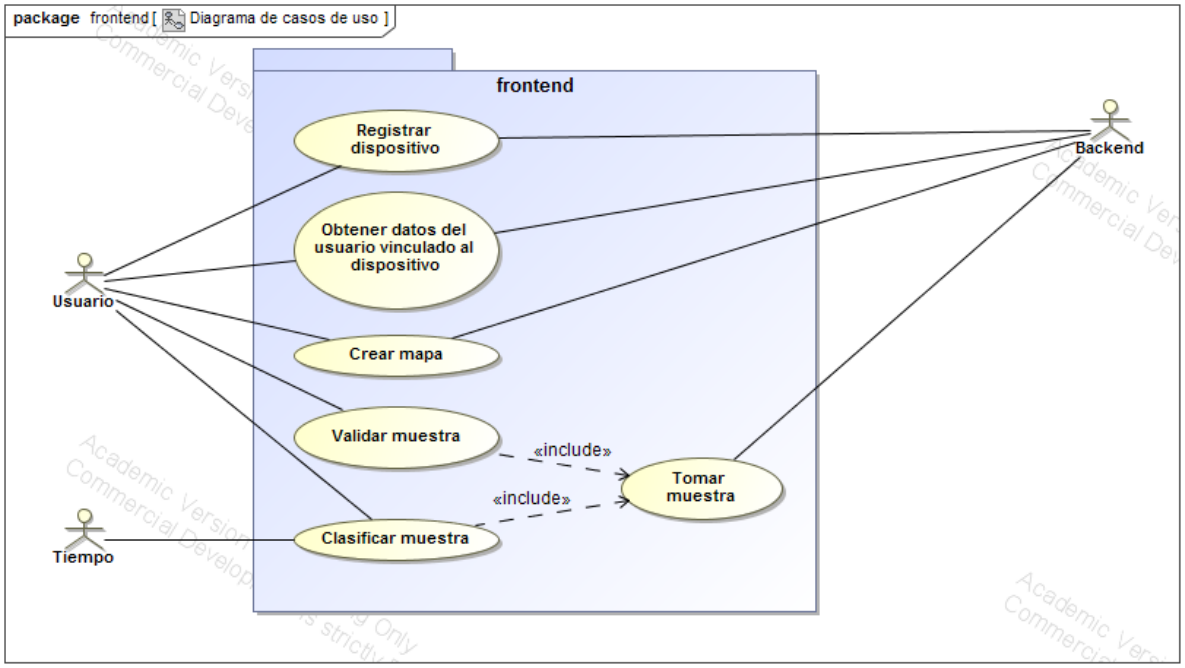


Figura 5.2: Diagrama de casos de uso del frontend.

Como muestra el diagrama de casos de uso de la figura 5.2, en el frontend el usuario debe registrar el dispositivo ante el servidor la primera vez que se ejecuta la aplicación. Una vez registrado, cuando el dispositivo no tenga un usuario vinculado podrá preguntar al servidor por el usuario que tiene vinculado (si lo tiene).

Una vez el dispositivo conoce el usuario vinculado, si ese usuario todavía no tiene el mapa creado, se guiará al usuario por un proceso de configuración que pedirá ir tomando muestras en los lugares de interés para luego poder crear el mapa inicial.

Finalmente, cuando todo este listo, el usuario podrá tomar manualmente clasificar una muestra apoyandose en el backend o validar una muestra (indicando previamente el lugar donde está) para retroalimentar el mapa en el backend. Ambos casos deben tomar una muestra de las señales Wi-Fi y está funcionalidad podrá ser reutilizada.

El tiempo es un actor pues disparará cada cierto tiempo un evento para tomar una muestra y mandarla al servidor para su procesado.

5.1.3. Diagrama de casos de uso del backend

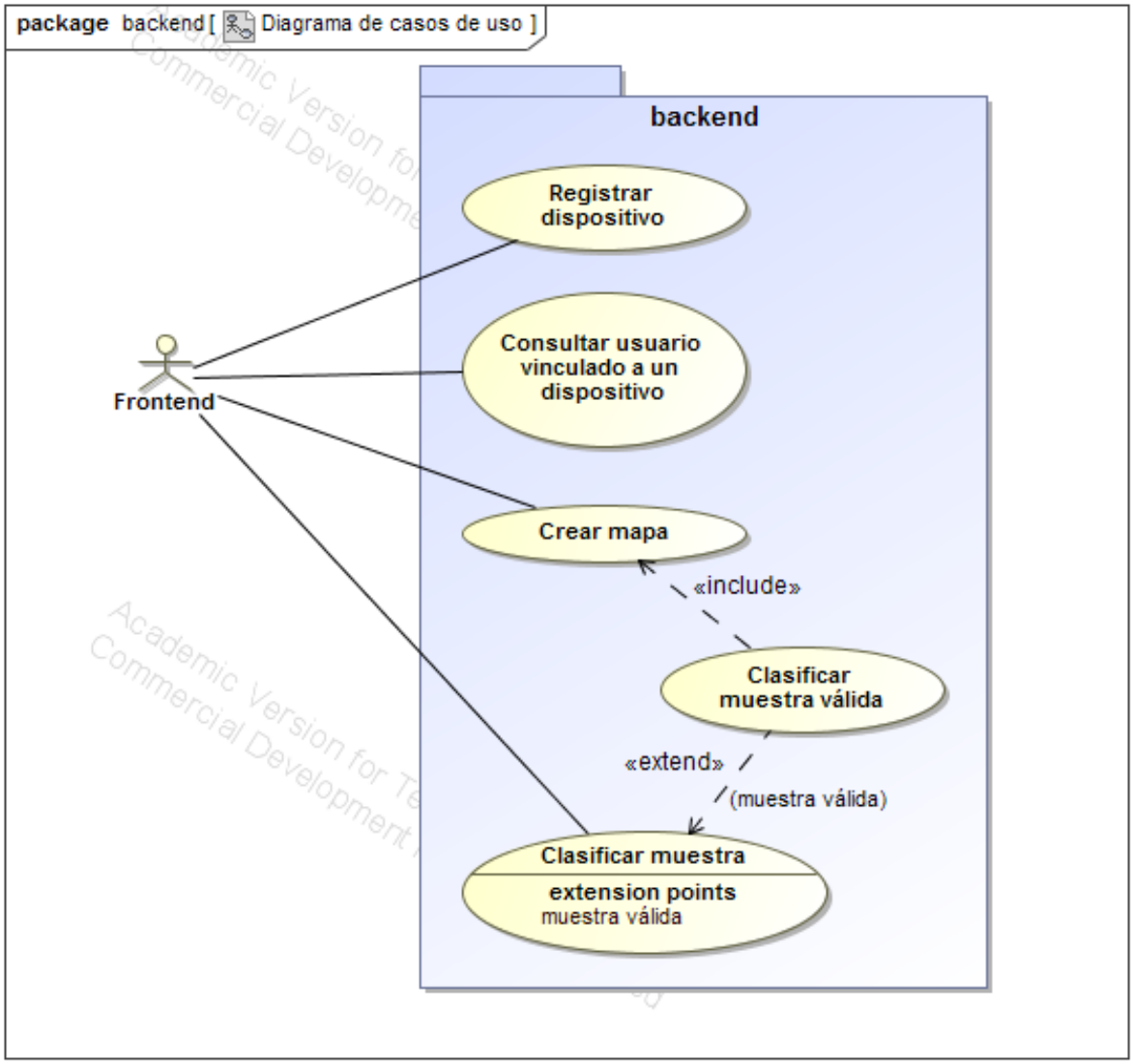


Figura 5.3: Diagrama de casos de uso del backend.

Como muestra el diagrama de casos de uso de la figura 5.3, en el backend es el frontend quien hace las peticiones: registrar un dispositivo, consultar el usuario que tiene vinculado el dispositivo, crear el mapa para un usuario o clasificar una muestra.

En cuanto a clasificar muestras, las clasificará usando el mapa del usuario, guardará el resultado en la base de datos y se lo devolverá también al frontend. Sin embargo, cuando se recibe una muestra previamente validada desde el frontend, está simplemente se utiliza para mejorar el mapa, que se regenera utilizando las muestras validas que ya tuviera y la nueva.

5.2. Historias de usuario

En este apartado se especifican las historias de usuario acompañadas de sus posibles escenarios. También se señala a que componentes afecta cada historia de usuario.

5.2.1. HU01 – Login backoffice

Como administrador quiero poder autenticarme en el backoffice para poder administrar el sistema.

Componentes: Backoffice

Escenarios:

- Dado que no estoy autenticado y me encuentro en el formulario de autenticación, cuando introduzco el usuario y la contraseña correctos y procedo con la autenticación, entonces accedo al backoffice autenticado como administrador.
- Dado que no estoy autenticado y me encuentro en el formulario de autenticación, cuando introduzco un usuario y/o una contraseña incorrectos y procedo con la autenticación, entonces se me informa de que las credenciales no son válidas.

5.2.2. HU02 – Logout backoffice

Como administrador quiero poder cerrar sesión en el backoffice para terminar de administrar el sistema y evitar accesos indebidos.

Componentes: Backoffice

Escenarios:

- Dado que estoy autenticado en el backoffice, cuando pulso sobre el botón de cerrar sesión del menú, entonces la sesión es destruida y se pierde la autenticación.

5.2.3. HU03 – Registrar dispositivo

Como administrador quiero poder registrar un dispositivo para que pueda hacer uso del sistema.

Componentes: Backoffice

Escenarios:

- Dado que estoy autenticado y me encuentro en el formulario de registrar dispositivo y se ha generado un identificador único por defecto para el nuevo dispositivo, cuando mantengo el identificador generado o introduzco uno manualmente que no estuviera ya registrado e introduzco una dirección MAC válida y un nombre, entonces se registra un nuevo dispositivo en el sistema con los datos proporcionados.
- Dado que estoy autenticado y me encuentro en el formulario de registrar dispositivo y se ha generado un identificador único por defecto para el nuevo dispositivo, cuando cambio el identificador generado por uno que ya está registrado e introduzco una dirección MAC válida y un nombre, entonces se muestra un error advirtiendo de que el identificador ya está en uso.
- Dado que estoy autenticado y me encuentro en el formulario de registrar dispositivo y se ha generado un identificador único por defecto para el nuevo dispositivo, cuando cambio el identificador generado por uno vacío e introduzco una dirección MAC válida y un nombre, entonces se muestra un error advirtiendo de que el identificador no puede estar vacío.
- Dado que estoy autenticado y me encuentro en el formulario de registrar dispositivo y se ha generado un identificador único por defecto para el nuevo dispositivo, cuando mantengo el identificador generado o introduzco uno manualmente que no estuviera ya registrado e introduzco una dirección MAC inválida y un nombre, entonces se muestra un error advirtiendo de que la dirección MAC es inválida.
- Dado que estoy autenticado y me encuentro en el formulario de registrar dispositivo y se ha generado un identificador único por defecto para el nuevo dispositivo, cuando mantengo el identificador generado o introduzco uno manualmente que no estuviera ya registrado, no introduzco una dirección MAC peros si un nombre, entonces se muestra un error advirtiendo de que la dirección MAC no puede estar vacía.
- Dado que estoy autenticado y me encuentro en el formulario de registrar dispositivo y se ha generado un identificador único por defecto para el nuevo dispositivo, cuando mantengo el identificador generado o introduzco uno manualmente que no estuviera ya registrado e introduzco una dirección MAC válida pero no un nombre, entonces se muestra un error advirtiendo de que el nombre no puede estar vacío.

5.2.4. HU04 – Listar dispositivos

Como administrador quiero poder consultar una lista de dispositivos registrados en el sistema para así poder acceder a las opciones para gestionarlos.

Componentes: Backoffice

Escenarios:

- Dado que estoy autenticado y existen dispositivos dados de alta en el sistema, cuando pulso sobre la sección Dispositivos, entonces se muestra un listado de los dispositivos registrados con la información más relevante y las opciones de consultar o editar.
- Dado que estoy autenticado y no existen dispositivos dados de alta en el sistema, cuando pulso sobre la sección Dispositivos, entonces se muestra un listado vacío indicando que no existen dispositivos registrados.
- Dado que estoy autenticado y me encuentro en el listado de dispositivos, cuando selecciono una cantidad diferente de dispositivos a mostrar en el desplegable correspondiente, entonces se muestra un listado con la cantidad de dispositivos seleccionada.
- Dado que estoy autenticado y me encuentro en el listado de dispositivos, cuando introduzco un texto en el campo de filtrado y pulso sobre el botón de filtrar, entonces se muestra un listado de los dispositivos registrados cuyo id o nombre coincida total o parcialmente con el texto introducido.

5.2.5. HU05 – Consultar dispositivo

Como administrador quiero poder consultar un dispositivo para poder ver en detalle sus datos.

Componentes: Backoffice

Escenarios:

- Dado que estoy autenticado y me encuentro en el listado de dispositivos, cuando pulso sobre el botón de consultar un dispositivo, entonces se muestra un ficha con los detalles del dispositivo y opciones para gestionarlo.

5.2.6. HU06 – Editar dispositivo

Como administrador quiero poder editar un dispositivo para poder asignarle un nombre que me ayude a identificarlo.

Componentes: Backoffice

Escenarios:

- Dado que estoy autenticado y me encuentro en el formulario de editar un dispositivo, cuando introduzco un nombre y pulso sobre el botón de guardar, entonces se modifica el nombre del dispositivo y se vuelve a la ficha del mismo.
- Dado que estoy autenticado y me encuentro en el formulario de editar un dispositivo, cuando introduzco un nombre vacío y pulso sobre el botón de guardar, entonces se muestra un error diciendo que el nombre no puede estar vacío.
- Dado que estoy autenticado y me encuentro en el formulario de editar un dispositivo, cuando pulso sobre el botón de cancelar, entonces se vuelve a la ficha del dispositivo sin modificarlo.

5.2.7. HU07 – Registrar usuario

Como administrador quiero poder registrar un usuario para que pueda hacer uso del sistema.

Componentes: Backoffice

Escenarios:

- Dado que estoy autenticado y me encuentro en el formulario de registrar usuario y se ha generado un identificador único por defecto para el nuevo usuario, cuando mantengo el identificador generado o introduzco uno manualmente que no estuviera ya registrado e introduzco un nombre y 2 o más lugares, entonces se registra un nuevo usuario en el sistema con los datos proporcionados.
- Dado que estoy autenticado y me encuentro en el formulario de registrar usuario y se ha generado un identificador único por defecto para el nuevo usuario, cuando cambio el identificador generado por uno que ya está registrado e introduzco un nombre y 2 o más lugares, entonces se muestra un error advirtiendo de que el identificador ya está en uso.
- Dado que estoy autenticado y me encuentro en el formulario de registrar usuario y se ha generado un identificador único por defecto para el nuevo usuario, cuando cambio el identificador generado por uno vacío e introduzco un nombre y 2 o más lugares, entonces se muestra un error advirtiendo de que el identificador no puede estar vacío.
- Dado que estoy autenticado y me encuentro en el formulario de registrar usuario y se ha generado un identificador único por defecto para el nuevo usuario, cuando cambio el identificador generado por uno vacío e introduzco un nombre vacío y 2 o más lugares, entonces se muestra un error advirtiendo de que el nombre no puede estar vacío.
- Dado que estoy autenticado y me encuentro en el formulario de registrar usuario y se ha generado un identificador único por defecto para el nuevo usuario, cuando mantengo el identificador generado o introduzco uno manualmente que no estuviera ya registrado e introduzco un nombre y menos de 2 lugares, entonces se muestra un error advirtiendo de que se requieren al menos 2 lugares.

5.2.8. HU08 – Listar usuarios

Como administrador quiero poder consultar una lista de usuarios registrados en el sistema para así poder acceder a las opciones para gestionarlos.

Componentes: Backoffice

Escenarios:

- Dado que estoy autenticado y existen usuarios dados de alta en el sistema, cuando pulso sobre la sección Usuarios, entonces se muestra un listado de los usuarios registrados con la información más relevante y las opciones de consultar o editar.
- Dado que estoy autenticado y no existen usuarios dados de alta en el sistema, cuando pulso sobre la sección Usuarios, entonces se muestra un listado vacío indicando que no existen usuarios registrados.
- Dado que estoy autenticado y me encuentro en el listado de usuarios, cuando selecciono una cantidad diferente de usuarios a mostrar en el desplegable correspondiente, entonces se muestra un listado con la cantidad de usuarios seleccionada.
- Dado que estoy autenticado y me encuentro en el listado de usuarios, cuando introduzco un texto en el campo de filtrado y pulso sobre el botón de filtrar, entonces se muestra un listado de los usuarios registrados cuyo id o nombre coincida total o parcialmente con el texto introducido.

5.2.9. HU09 – Consultar usuario

Como administrador quiero poder consultar un usuario para poder ver en detalle sus datos.

Componentes: Backoffice

Escenarios:

- Dado que estoy autenticado y me encuentro en el listado de usuarios, cuando pulso sobre el botón de consultar un usuario, entonces se muestra un ficha con los detalles del usuario y opciones para gestionarlo.

5.2.10. HU10 – Editar usuario

Como administrador quiero poder editar un usuario para poder asignarle un nombre que me ayude a identificarlo.

Componentes: Backoffice

Escenarios:

- Dado que estoy autenticado y me encuentro en el formulario de editar un usuario, cuando introduzco un nombre y pulso sobre el botón de guardar, entonces se modifica el nombre del usuario y se vuelve a la ficha del mismo.
- Dado que estoy autenticado y me encuentro en el formulario de editar un usuario, cuando introduzco un nombre vacío y pulso sobre el botón de guardar, entonces se muestra un error diciendo que el nombre no puede estar vacío.
- Dado que estoy autenticado y me encuentro en el formulario de editar un usuario, cuando pulso sobre el botón de cancelar, entonces se vuelve a la ficha del usuario sin modificarlo.

5.2.11. HU11 – Vincular usuario a un dispositivo

Como administrador quiero poder vincular un usuario a un dispositivo para que pueda desde el dispositivo se pueda obtener la información del usuario actualmente vinculado.

Componentes: Backoffice

Escenarios:

- Dado que estoy autenticado y desde la ficha de un dispositivo que no tiene ningún usuario vinculado he pulsado el botón de vincular usuario y se me ha mostrado la lista de usuarios, cuando pulso sobre el botón de vincular de uno de los usuarios, entonces ese usuario queda vinculado al dispositivo y se vuelve a la ficha del dispositivo.
- Dado que estoy autenticado y desde la ficha de un dispositivo que no tiene ningún usuario vinculado he pulsado el botón de vincular usuario y se me ha mostrado la lista de usuarios, cuando vuelvo atrás o navego a otra página, entonces ningún usuario es vinculado al dispositivo.

5.2.12. HU12 – Desvincular usuario de un dispositivo

Como administrador quiero poder desvincular un usuario de un dispositivo para así posteriormente poder vincular ese usuario a otro dispositivo o otro usuario a ese dispositivo.

Componentes: Backoffice

Escenarios:

- Dado que estoy autenticado y me encuentro en la ficha de un dispositivo que tiene un usuario vinculado, cuando pulso sobre el botón de desvincular usuario, entonces el usuario es desvinculado del dispositivo y se actualiza la ficha del dispositivo.

5.2.13. HU13 – Obtener datos del usuario vinculado desde el dispositivo

Como usuario necesito poder obtener los datos del usuario que tenga vinculado el dispositivo para así poder hacer uso del sistema.

Componentes: Frontend, Backend

Escenarios:

- Dado que al abrir la aplicación móvil no existen datos de usuario y se muestra un diálogo de confirmación explicando que es necesario vincular un usuario, teniendo en cuenta que no es posible establecer una conexión con el backend, cuando se pulsa sobre el botón de confirmar y se procede a obtener los datos del usuario vinculado, entonces se muestra un error advirtiendo de que no es posible conectar con el servidor.
- Dado que al abrir la aplicación móvil no existen datos de usuario y se muestra un diálogo de confirmación explicando que es necesario vincular un usuario, teniendo en cuenta que ese dispositivo no ha sido registrado en el sistema, cuando se pulsa sobre el botón de confirmar y se procede a obtener los datos del usuario vinculado, entonces se muestra un error advirtiendo de que el dispositivo no ha sido registrado.
- Dado que al abrir la aplicación móvil no existen datos de usuario y se muestra un diálogo de confirmación explicando que es necesario vincular un usuario, teniendo en cuenta que ese dispositivo ha sido registrado en el sistema pero no tiene vinculado ningún usuario, cuando se pulsa sobre el botón de confirmar y se procede a obtener los datos del usuario vinculado, entonces se muestra un error advirtiendo de que el dispositivo no tiene ningún usuario vinculado.
- Dado que al abrir la aplicación móvil no existen datos de usuario y se muestra un diálogo de confirmación explicando que es necesario vincular un usuario, teniendo en cuenta que ese dispositivo ha sido registrado en el sistema y tiene vinculado un usuario, cuando se pulsa sobre el botón de confirmar y se procede a obtener los datos del usuario vinculado, entonces se obtienen y guardan los datos del usuario en el dispositivo y se anvaza a la siguiente pantalla.

5.2.14. HU14 – Crear mapa de señales Wi-Fi

Como usuario necesito poder crear un mapa de señales Wi-Fi con la aplicación móvil para así poder ser localizado posteriormente.

Componentes: Frontend, Backend

Escenarios:

- Dado que el dispositivo tiene un usuario vinculado y la aplicación ya está sincronizada pero el usuario no tiene todavía creado un mapa Wi-Fi, cuando pulso sobre crear mapa y sigo las instrucciones para tomar muestras en cada uno de los puntos de interés y al terminar se establece la conexión con el backend, entonces los datos recopilados son enviados al backend para generar el mapa Wi-Fi y se vuelve a la pantalla principal.

- Dado que el dispositivo tiene un usuario vinculado y la aplicación ya está sincronizada pero el usuario no tiene todavía creado un mapa Wi-Fi, cuando pulso sobre crear mapa y sigo las instrucciones para tomar muestras en cada uno de los puntos de interés pero antes de hacer todos cancelo el proceso, entonces vuelve a la pantalla principal.

- Dado que el dispositivo tiene un usuario vinculado y la aplicación ya está sincronizada pero el usuario no tiene todavía creado un mapa Wi-Fi, cuando pulso sobre crear mapa y sigo las instrucciones para tomar muestras en cada uno de los puntos de interés y al terminar no se puede establecer la conexión con el backend, entonces se muestra un aviso pero no se vuelve a la pantalla principal.

5.2.15. HU15 – Clasificar muestra

Como usuario quiero poder tomar una muestra y que está sea clasificada para así poder comprobar la efectividad del sistema.

Componentes: Frontend, Backend

Escenarios:

- Dado que estoy en la pantalla principal, cuando pulso en tomar muestra, entonces se toma una muestra y se establece la comunicación con el backend para enviar la muestra para su clasificación y registro.

- Dado que estoy en la pantalla principal, cuando pulso en tomar muestra, entonces se toma una muestra y si no se puede establecer la comunicación con el backend la muestra es almacenada localmente para su posterior envío cuando se disponga de conexión.

5.2.16. HU16 – Validar muestra

Como usuario quiero poder tomar una muestra válida para así poder mejorar el clasificador.

Componentes: Frontend, Backend

Escenarios:

- Dado que estoy en la pantalla principal, cuando pulso en tomar muestra y elijo el lugar donde estoy, entonces se toma una muestra y se establece la comunicación con el backend para enviar la muestra para su registro y se vuelve a generar el mapa Wi-Fi usando todas las muestra válidas de ese usuario.
- Dado que estoy en la pantalla principal, cuando pulso en tomar muestra y elijo el lugar donde estoy, entonces se toma una muestra y si no se puede establecer la comunicación con el backend la muestra es almacenada localmente para su posterior envío cuando se disponga de conexión.

5.2.17. HU17 – Clasificar muestras automáticamente

Como usuario necesito que la aplicación móvil tome muestras periódicamente y las envíe de forma automática para su clasificación para así poder realizar un seguimiento del dispositivo.

Componentes: Frontend, Backend

Escenarios:

- Dado que el dispositivo está registrado y tiene un usuario vinculado, cuando termino de crear el mapa Wi-Fi y a partir de entonces cada vez que arranque el dispositivo, entonces se registra una alarma que disparará de cada cierto tiempo y en segundo plano el proceso de tomar y clasificar una muestra.

5.2.18. HU18 – Tomar muestra de señales Wi-Fi

Como desarrollador necesito tomar una muestra de las señales Wi-Fi para así poder utilizarla en otros procesos como la creación del mapa o la clasificación de muestras.

Componentes: Frontend

Escenarios:

- Dado que el sensor Wi-Fi está encendido, cuando se pide realizar un escaneo de los puntos de acceso del entorno, entonces se obtienen los resultados y se guardan los datos de interés de cada uno.

- Dado que el sensor Wi-Fi está apagado, cuando se pide realizar un escaneo de los puntos de acceso del entorno, entonces se activa el sensor Wi-Fi y una vez está listo se obtienen los resultados y se guardan los datos de interés de cada uno.

5.2.19. HU19 – Registrar instalación en dispositivo

Como usuario necesito registrar el dispositivo donde ha sido instalada la aplicación para que así pueda formar parte del sistema.

Componentes: Frontend, Backend

Escenarios:

- Dado que se acaba de instalar la aplicación en el dispositivo móvil, cuando pulso sobre el botón para registrar el dispositivo y se establece la comunicación con el backend, entonces el dispositivo queda registrado.
- Dado que se acaba de instalar la aplicación en el dispositivo móvil, cuando pulso sobre el botón para registrar el dispositivo y no se consigue establecer la comunicación con el backend, entonces el dispositivo queda no puede registrarse y se muestra una aviso.

5.3. Product Backlog

Tras analizar las historias de usuario y realizar una estimación apropiada de las mismas en puntos de historia (PH), se ha creado el Product Backlog o pila de producto que se muestra en la tabla 5.1. Están ordenadas de más a menos prioritarias.

Historia de usuario	**Estimación**
H18 - Tomar muestra de señales Wi-Fi	8 PH
H14 - Crear mapa de señales Wi-Fi	16 PH
H15 - Clasificar muestra	8 PH
H13 - Obtener datos del usuario vinculado desde el dispositivo	5 PH
H17 - Clasificar muestras automáticamente	12 H
H01 - Login backoffice	3 PH
H02 - Logout backoffice	1 PH
H03 - Registrar dispositivo	4 PH
H07 - Registrar usuario	6 PH
H04 - Listar dispositivos	9 PH
H08 - Listar usuarios	9 PH
H11 - Vincular usuario a un dispositivo	3 PH
H12 - Desvincular usuario de un dispositivo	1 PH
H05 - Consultar dispositivo	3 PH
H09 - Consultar usuario	3 PH
H06 - Editar dispositivo	2 PH
H10 - Editar usuario	2 PH
H16 - Validar muestra	6 PH
H19 - Registrar instalación en dispositivo	4 PH
Total	**105 PH**

Tabla 5.1: Product Backlog o pila de producto. Ordenada por prioridad.

En total suman 105 PH, que divido entre 5 sprints, sale a una media de 21 PH/sprint.

Capítulo 6

Análisis y diseño del sistema

Índice del capítulo

En este capítulo se recoge el análisis y el diseño del sistema que se ha ido realizando a lo largo de la etapa de desarrollo del proyecto. Al ser un proyecto ágil no se ha hecho todo el análisis y el diseño al inicio sino que se ha ido haciendo bajo demanda.

6.1. Análisis del sistema

El análisis de un sistema informático consiste en especificar qué es lo que debe hacer basándonos en los requisitos previamente definidos. Típicamente esto implica el modelado del sistema a nivel lógico: diagramas de clases, diagramas de actividad, etc.

6.1.1. Diagrama de clases

El diagrama de clases que se muestra en la figura 6.1, obtenido a partir de un análisis de los requisitos, resume qué entidades tenemos en nuestro modelo de datos y qué datos relevantes contiene cada una. El diagrama se ha simplificado al máximo, pues solo se usa para tener una visión global del modelo de datos compartido por todos los componentes del sistema, así que se han omitido pequeños detalles diferentes en cada componente y también clases que interactúan con nuestro modelo pero que forman parte de paquetes de terceros.

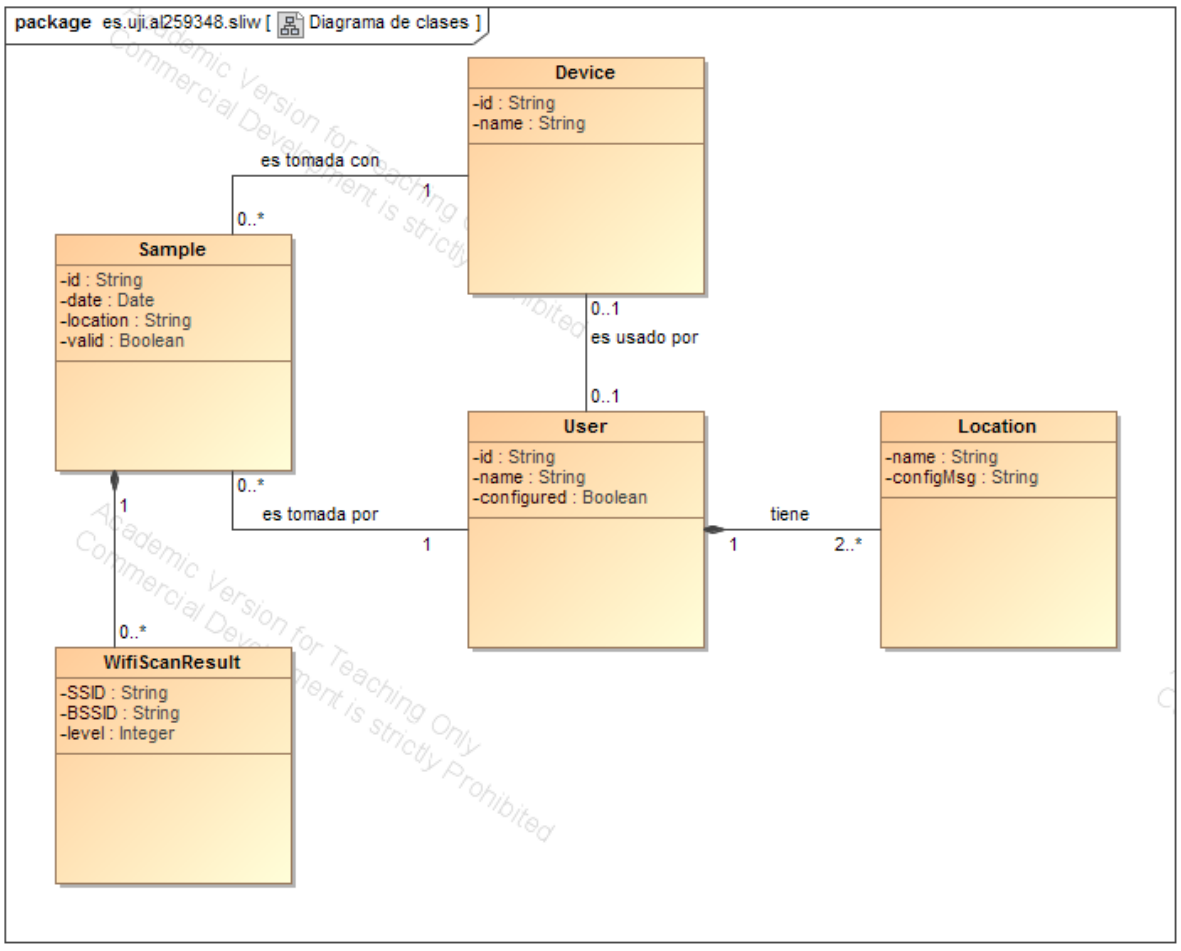


Figura 6.1: Diagrama de clases.

6.2. Diseño de la arquitectura del sistema

En este apartado se describe la arquitectura del software, es decir, la estructura, funcionamiento e interacción entre las partes del software [4].

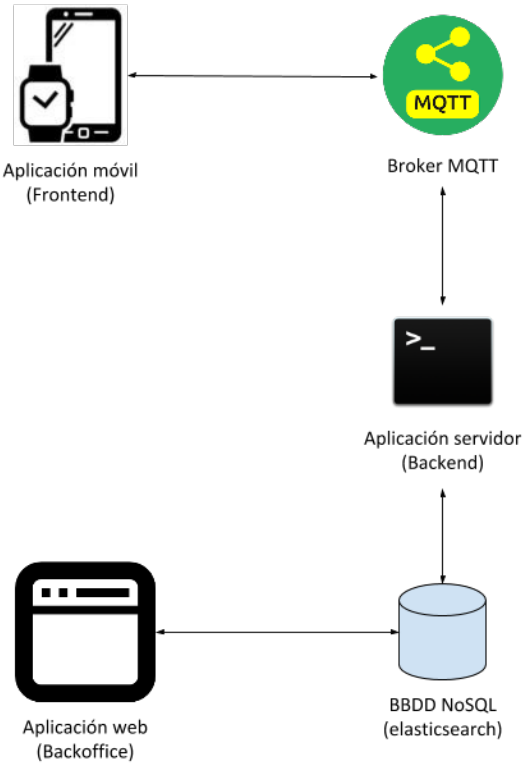


Figura 6.2: Arquitectura del sistema.

Como muestra el diagrama de la figura 6.2, la aplicación móvil o *frontend* y la aplicación servidor o *backend* se comunican a través del broker de mensajería MQTT y tanto esta última como la aplicación web o *backoffice* leen y escriben en la base de datos.

6.2.1. Arquitectura cliente-servidor

La arquitectura cliente-servidor, que se muestra en el diagrama de la figura 6.3, divide un sistema en dos partes:

- El servidor, que provee los recursos y servicios.
- El cliente, que demanda esos recursos y servicios.

El cliente siempre es quien realiza la petición solicitando al servidor el acceso a un recurso o servicio y espera a que el servidor devuelva una respuesta.

El servidor está a la espera de peticiones por parte de los clientes, cuando recibe una petición la procesa y devuelve la respuesta apropiada.

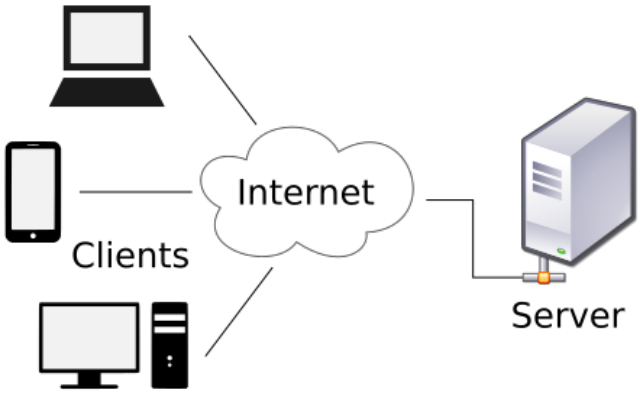


Figura 6.3: Arquitectura cliente-servidor.

Por lo general suele haber más clientes que servidores debido a que un servidor tiene la capacidad de atender a varios clientes de forma simultánea, pero si hay un gran número de clientes se deberían introducir más servidores (lo que se conoce como escalado horizontal) y balancear la carga entre ellos para que el sistema pueda seguir funcionando adecuadamente.

En este proyecto se ha empleado una arquitectura cliente-servidor en la que la aplicación móvil o *frontend* se comunica con la aplicación servidor o *backend* bien sea para solicitar o para mandar datos. El servidor será el encargado de procesar los datos e interactuar con la base de datos.

6.2.2. Arquitectura MVC

La arquitectura MVC (Modelo-Vista-Controlador) [28], que se muestra en el diagrama de la figura 6.4, se basa en la separación de conceptos y propone separar la aplicación en 3 componentes:

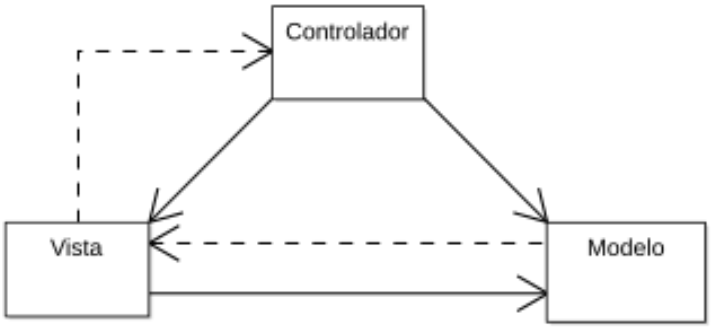


Figura 6.4: Arquitectura MVC.

- Modelo, la representación de los datos con los que el sistema trabaja. Es actualizado por el controlador y notifica a la vista cuando ha cambiado.
- Vista, la interfaz de usuario donde se presentan los datos del modelo y con la que el usuario interactúa. Consulta el modelo de datos y dispara eventos en el controlador.
- Controlador, implementa la lógica de negocio. Responde a los eventos generados por la vista y actualiza el modelo y la vista si fuese necesario.

La arquitectura MVC aporta un desacoplamiento que permite sustituir cualquiera de las partes por otra implementación siempre que esta implemente la misma interfaz.

Existen muchas variantes tales como MVVM [27], MVP [29], entre otros, que se diferencian en como los 3 componentes interactúan entre si.

En este proyecto se ha seguido una arquitectura MVC en todas las aplicaciones.

6.3. Diseño de la interfaz de usuario

En este apartado se muestran los prototipos elaborados tanto de la aplicación web como de la aplicación móvil. Para cada prototipo se muestra también una captura de pantalla del resultado real.

6.3.1. Prototipos backoffice

A continuación se muestran los prototipos de las pantallas de la aplicación web desde la figura 6.5 a la figura 6.15.

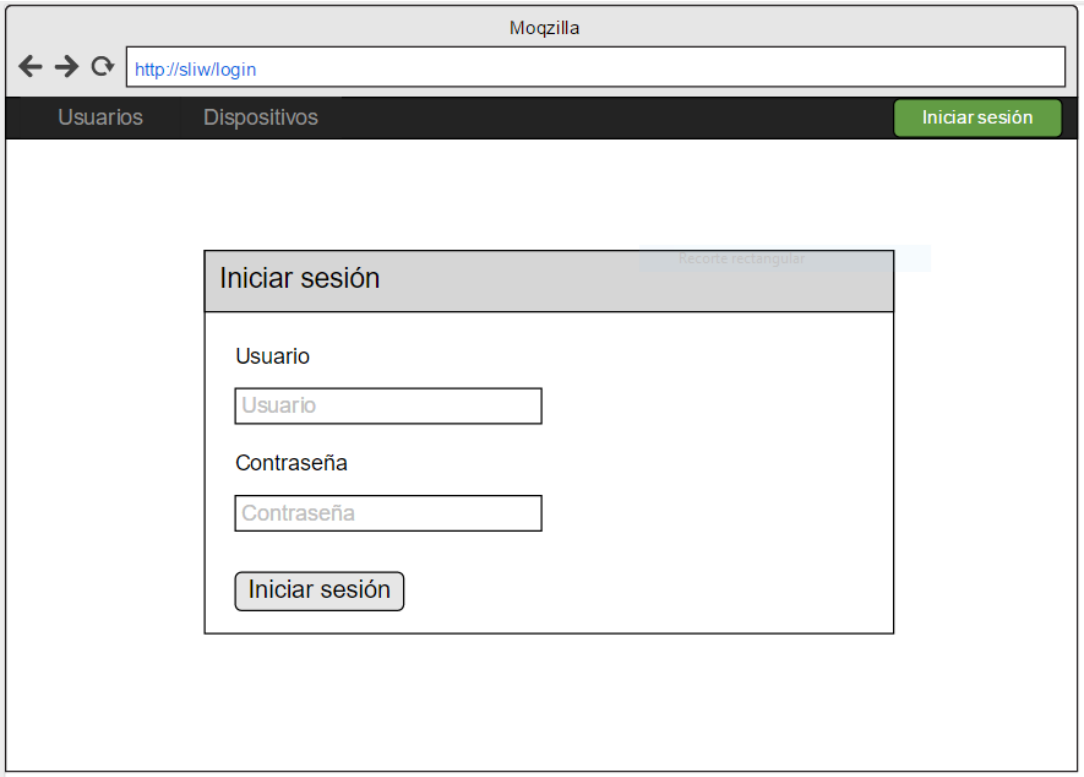


Figura 6.5: Mockup del formulario de iniciar sesión.

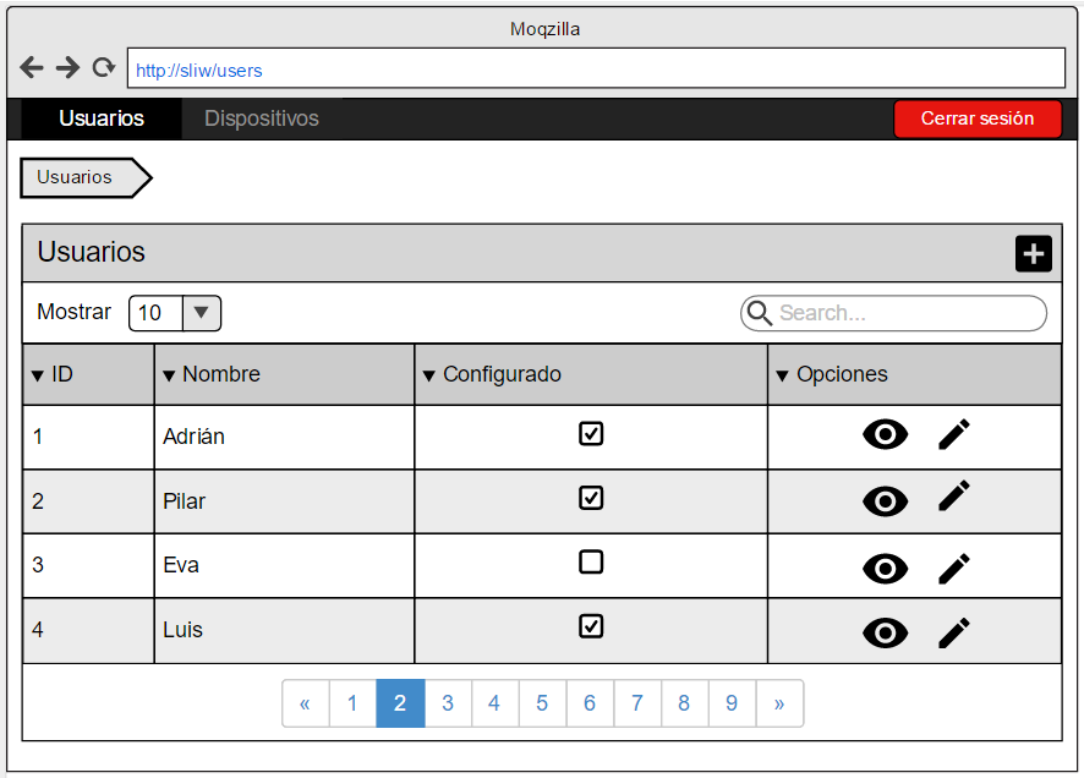


Figura 6.6: Mockup del listado de usuarios.

Figura 6.7: Mockup del formulario de crear usuario.

Figura 6.8: Mockup del formulario de editar usuario.

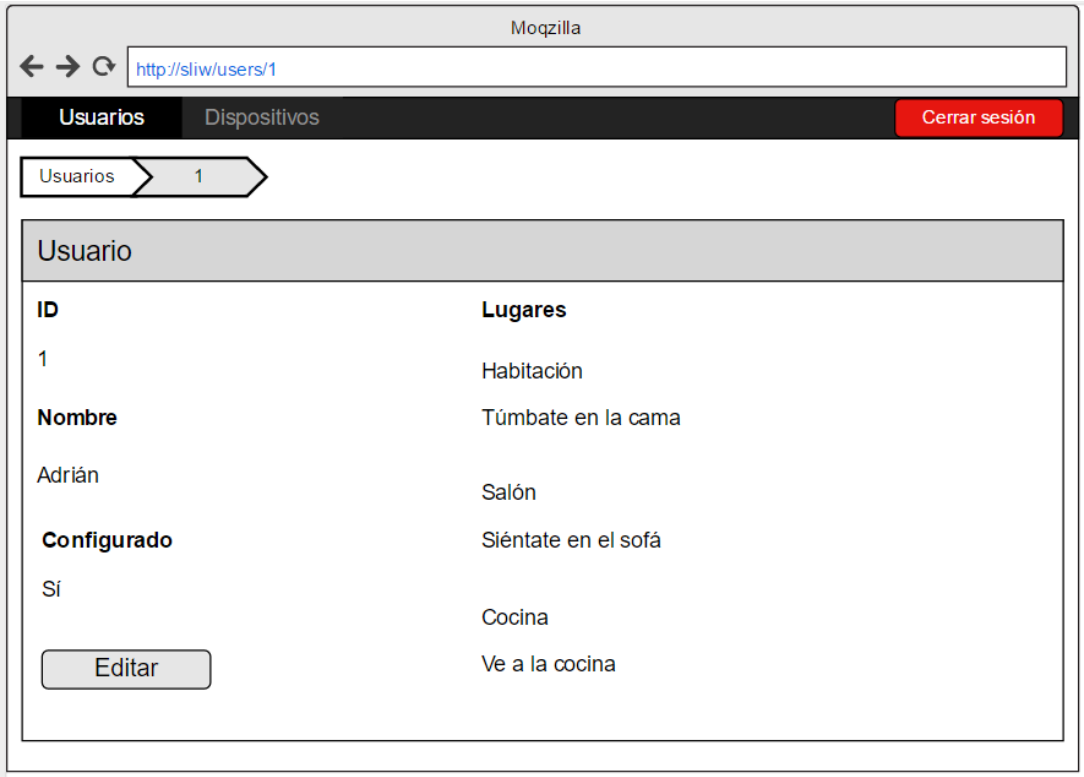


Figura 6.9: Mockup de la ficha de usuario.

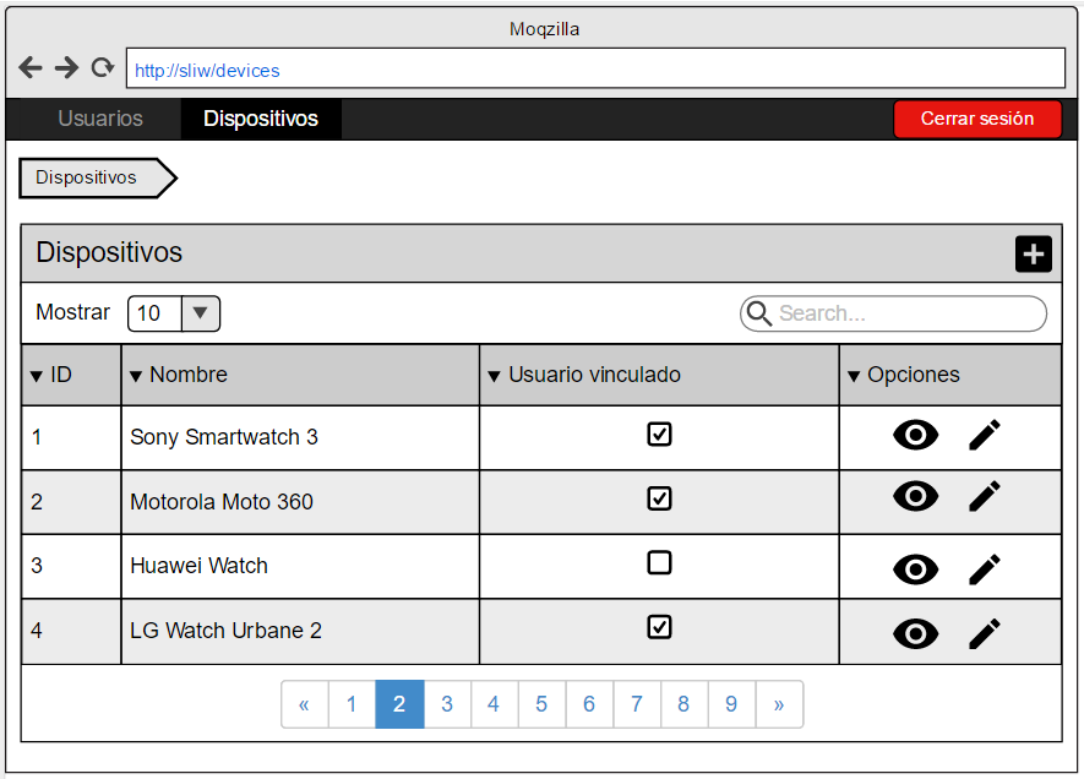


Figura 6.10: Mockup del listado de dispositivos.

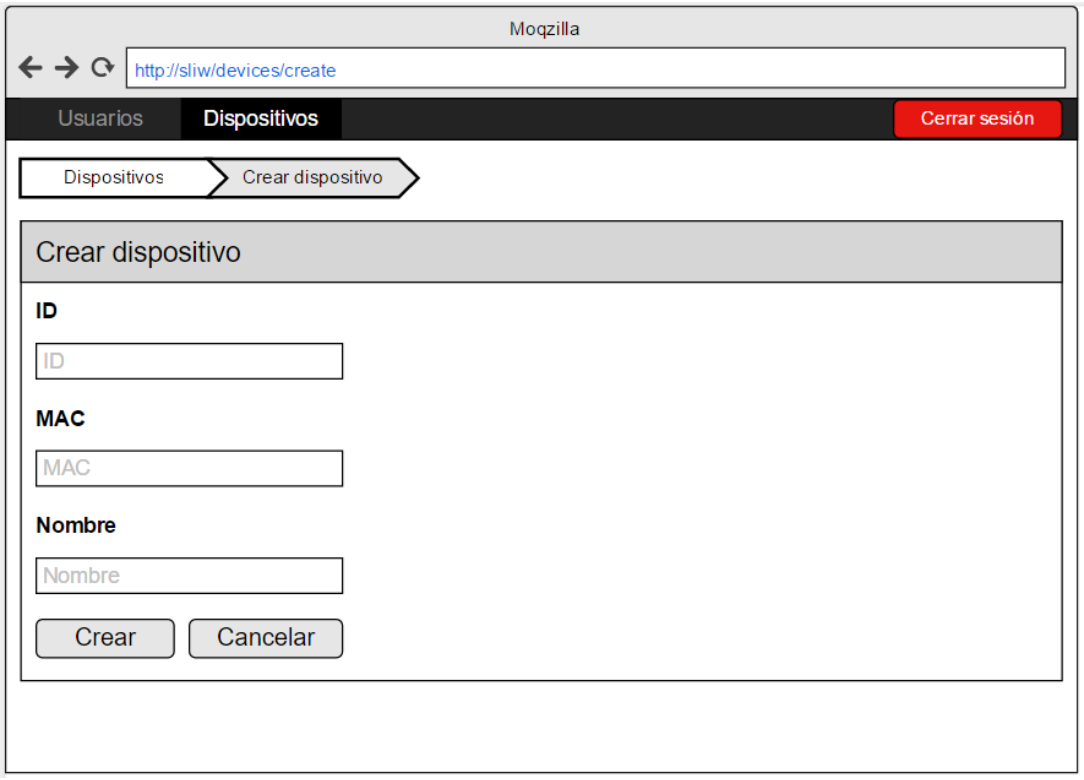


Figura 6.11: Mockup del formulario de crear dispositivo.

Figura 6.12: Mockup del formulario de editar dispositivo.

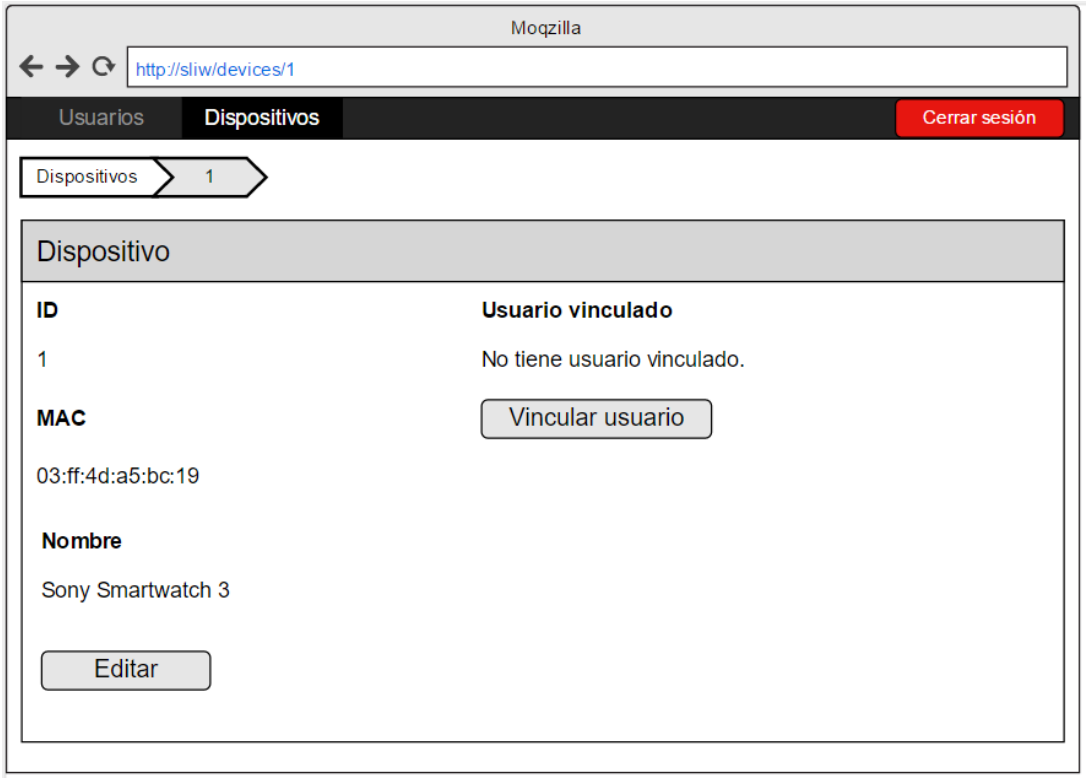


Figura 6.13: Mockup de la ficha de dispositivo sin usuario vinculado.

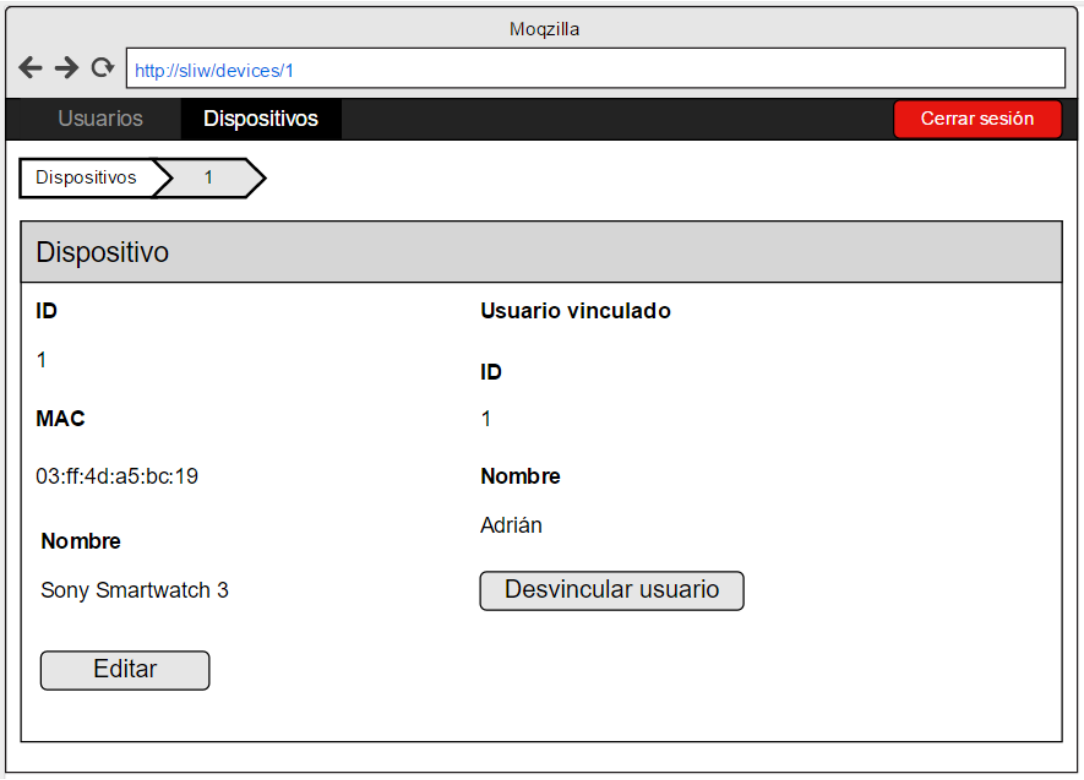


Figura 6.14: Mockup de la ficha de dispositivo con usuario vinculado.

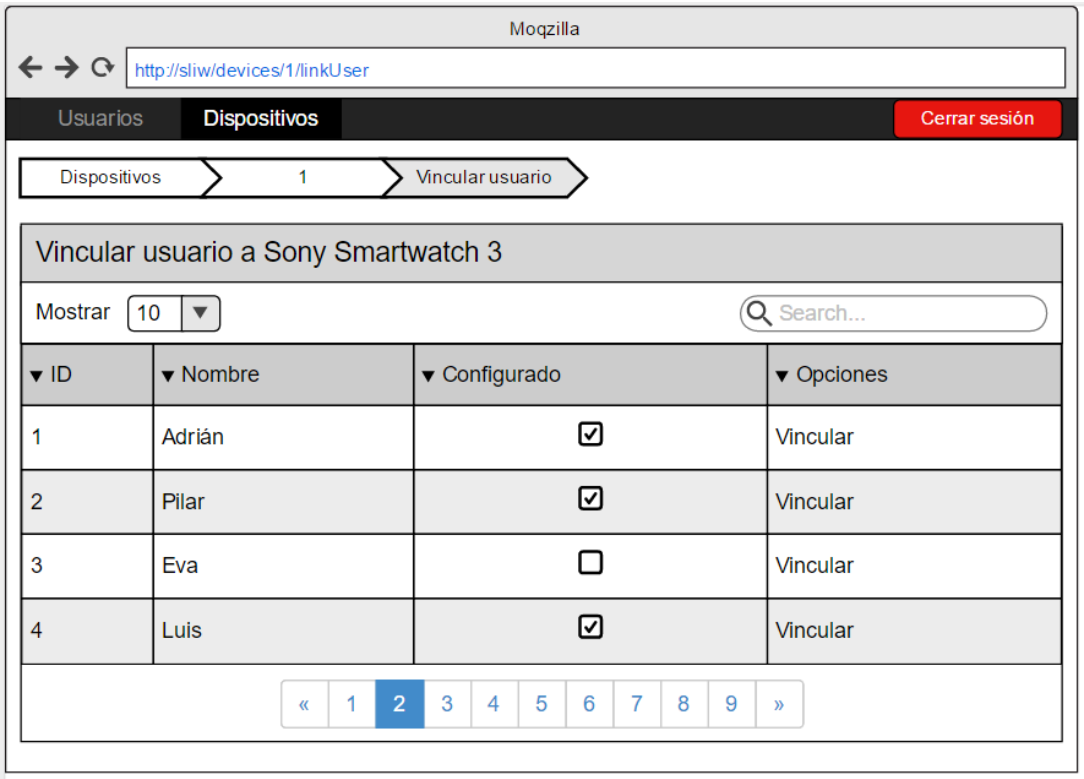


Figura 6.15: Mockup del listado de usuarios para vincular al dispositivo.

6.3.2. Prototipos frontend

A continuación se muestran los prototipos de algunas pantallas de la aplicación para Android Wear desde la figura 6.16 a la figura 6.24. No están todas, debido a que muchas reutilizan la misma plantilla pero cambiando los textos, por ejemplo: pantalla de confirmación, pantalla de espera, pantalla de progreso, etc.

Figura 6.16: Mockup de la pantalla de confirmación para registrar el dispositivo.

Figura 6.17: Mockup de la pantalla de espera de registrar el dispositivo.

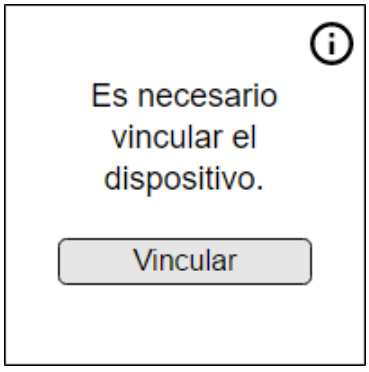


Figura 6.18: Mockup de la pantalla de confirmación para vincular el dispositivo.

Figura 6.19: Mockup de la pantalla de espera de vincular el dispositivo.

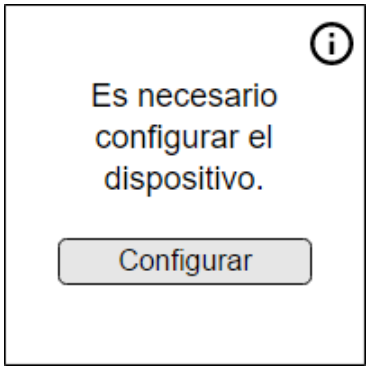


Figura 6.20: Mockup de la pantalla de confirmación para configurar el dispositivo.

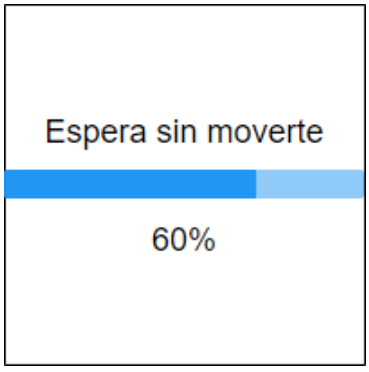


Figura 6.21: Mockup de la pantalla de progreso al tomar muestras.

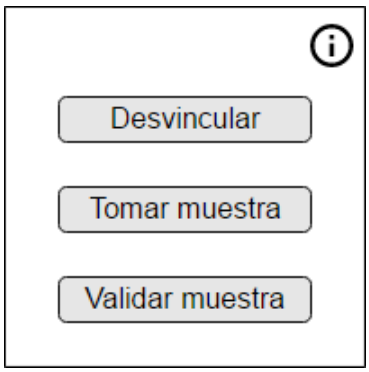


Figura 6.22: Mockup de la pantalla principal.

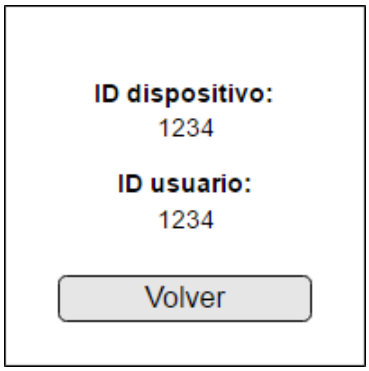


Figura 6.23: Mockup de la pantalla de información.

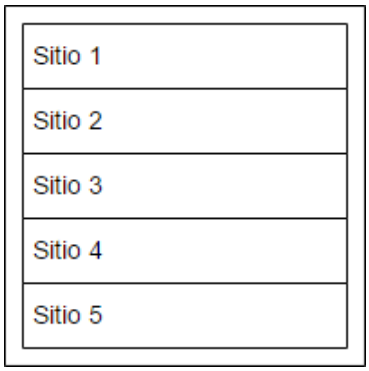


Figura 6.24: Mockup de la pantalla de validación de muestra.

6.3.3. Árbol de pantallas del backoffice

A continuación se muestra el árbol de pantallas de la aplicación web en la figura 6.25.

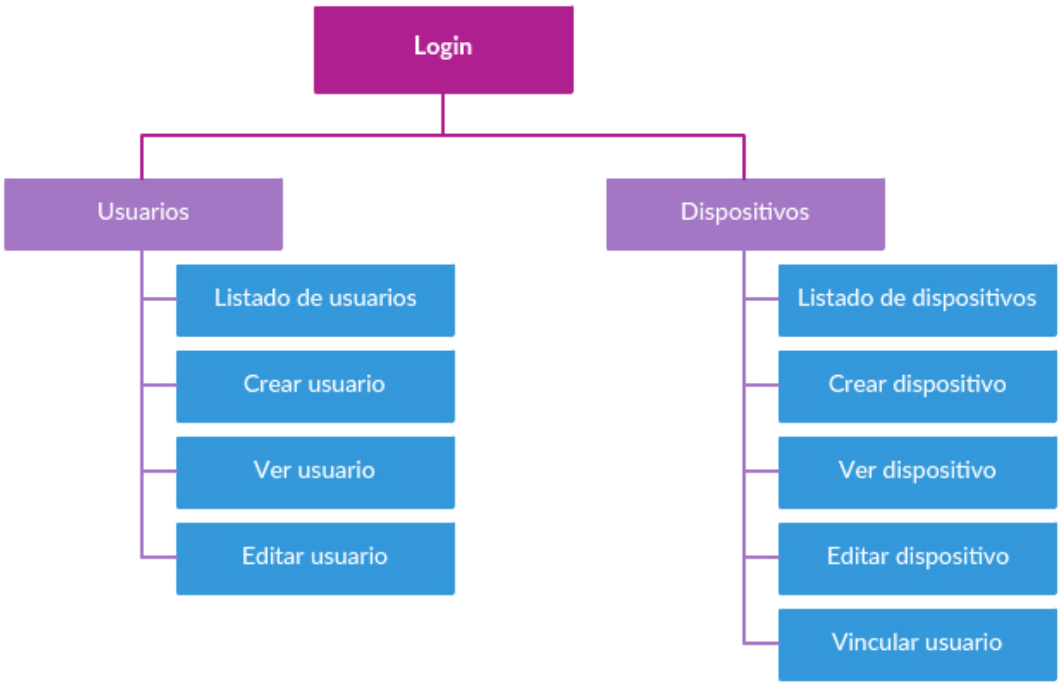


Figura 6.25: Árbol de pantallas del backoffice.

6.3.4. Árbol de pantallas del frontend

A continuación se muestra el árbol de pantallas de la aplicación móvil en la figura 6.26. Debido a la que la navegación entre pantallas en el frontend es más estricta se incluye también el sentido de la misma.

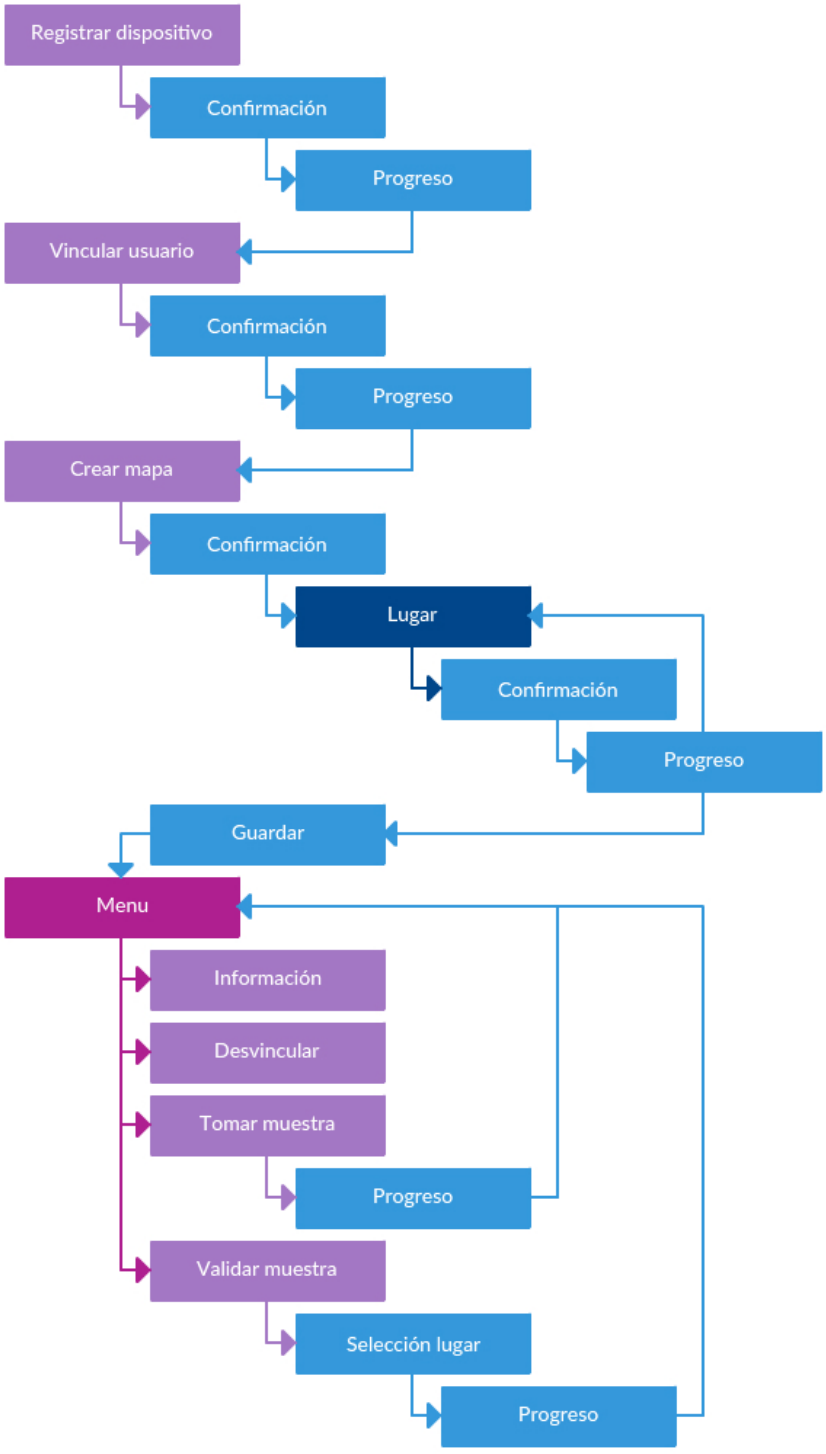


Figura 6.26: Árbol de pantallas y de navegación del frontend.

Capítulo 7

Implementación y pruebas

Índice del capítulo

En este capítulo se va a explicar como se han llevado a cabo los sprints, tanto la planificación como la ejecución, donde se darán algunos detalles de implementación. Por último se explicarán que tipos de pruebas se han realizado.

7.1. Sprint 1

7.1.1. Planificación del sprint

Para poder escoger que items del Product Backlog se van a realizar durante el sprint, hay que atender tanto a la prioridad como a la magnitud (Puntos de Historia) que nos indica aproximadamenet la cantidad de tiempo necesaria para completar una historia.

Como se ha explicado en 5.3, el Product Backlog completo suma 105 PH y sale una media de 21 PH por cada sprint. Vamos a tratar de cubrir el máximo en cada sprint para ganar tiempo en sprints posteriores, pero sin exceder demasiado la media para evitar un sprint incompleto.

En el primer sprint se ha revisado el Product Backlog y se han escogido los siguientes items:

- H18 - Tomar muestra de señales Wi-Fi (8 PH)
- H14 - Crear mapa de señales Wi-Fi (16 PH)

Todos los items escogidos hacen un total de 24 PH.

7.1.2. Ejecución del sprint

7.1.2.1. Estructura del código del backend y el frontend

Se ha seguido un organización del código similar en todas las aplicaciones. Se han agrupado las clases en paquetes según su papel.

Algunos paquetes como por ejemplo el que contiene los controladores o el modelo están presentes en todas las aplicaciones, sin embargo existen variaciones. Por ejemplo, la aplicación *backend* no dispone de interfaz gráfica por lo que no tiene el paquete *view* y la aplicación *frontend* dispone de paquetes para componentes específicos de la plataforma Android.

La aplicación *frontend* ha sido además separada en varios módulos: *core* o *núcleo* que contiene toda la lógica de negocio común, *wear* que contiene todas las vistas de la aplicación para *smartwatches*, y *mobile* que, aunque queda fuera del alcance del proyecto, podrá en un futuro realizarse una aplicación para *smartphones* tan solo implementado la vistas y aprovechando toda la lógica de negocio ya implementada.

Todas las aplicaciones tienen los dos entornos por convención de Maven, que es la herramienta utilizada para gestionar el código: *main* que concentra el código real y los recursos estáticos o de configuración para este entorno y *test* donde se encuentran los tests unitarios y de integración, así como recursos específicos para este entorno si fuese necesario.

Los paquetes del módulo *core* de la aplicación Android son los siguientes:

- core/android, contiene las clases con la implementación de componentes específicos de Android tales como *BroadcastReceivers* y *Services*.
- core/controller, contiene la interfaz y la implementación de los controladores, es decir aquí es donde se decide como actuar ante los eventos que se generan.
- core/model, contiene las clases que representan el modelo de datos, son simples clases contenedoras de datos aunque algunas contiene anotaciones para su uso adecuado con Jackson y ORMLite.
- core/repositories, contiene la interfaz de los repositorios de datos y una implementación para SQLite.
- core/services, contiene la interfaz e implementación de la lógica de negocio, cada clase tiene todas las funciones relacionadas con una responsabilidad.
- core/view, contiene las interfaces de las vistas necesarias para la comunicación desde el *core* al módulo de vista correspondiente.
- res/values, contiene archivos XML con valores que serán cargados en el código: mensajes, valores de configuración, etc.

Los paquetes del módulo *wear* de la aplicación Android son los siguientes:

- wear/activities, contiene la implementación de las interfaces de las vistas en forma de *Activities*.
- wear/fragments, contiene la implementación de los *Fragments* que son usados en las *Activities*.
- wear/receivers, contiene *BroadcastReceivers* que están a la espera de terminados eventos.

- res/layout, contiene archivos XML que definen el diseño de las *Activities* y los *Fragments*.
- res/values, contiene archivos XML con valores que serán cargados en el código: mensajes, valores de configuración, etc.

Los paquetes de la aplicación *Backend* son los siguientes:

- config, contiene clases de configuración de Spring Boot.
- exceptions, contiene clases que definen nuevas excepciones.
- ml, contiene clases de utilidad para trabajar con Weka (Machine Learning).
- model, contiene las clases que representan el modelo de datos, son simples clases contenedoras de datos aunque algunas contiene anotaciones para mapeado con Elasticsearch.
- repositories, contiene las interfaces de los repositorios de Elasticsearch, Spring Data se encarga de generar la implementación automáticamente.
- services, contiene la interfaz e implementación de la lógica de negocio, cada clase tiene todas las funciones relacionadas con una responsabilidad.
- resources, contiene archivos estáticos o de configuración.
- test.it, contiene test de integración.
- test.ut, contiene test unitarios.

7.1.2.2. Acceso al chip Wi-Fi

Para poder tomar muestras hemos de utilizar la API de Android para acceder y controlar el chip Wi-Fi del dispositivo. Deberemos declarar en el AndroidManifest (el fichero donde se especifican los permisos, pantallas, servicios, etc. que utiliza la aplicación) diversos permisos para que nuestra aplicación pueda hacer uso de estas funciones.

Lo primero que hay que hacer es comprobar si el Wi-Fi está activado, en caso contrario hay que solicitar su activación. Utilizando la clase WifiManager de la API de Android podemos consultar el estado del chip de forma sencilla y solicitar su activación si estuviera apagado, pero esta operación es asíncrona por lo que necesitamos implementar un BroadcastReceiver que permanezca atento al cambio de estado del chip Wi-Fi y que en caso de éxito llamará a la función que continua con el proceso. Además, para evitar solicitar varias veces la activación en caso de una segunda petición mientras todavía se está activando se ha utilizado una cola donde se suscribe cada llamada, tan solo el primero activará el chip Wi-Fi pero todos serán avisados cuando el chip este listo para su uso.

Una vez el chip Wi-Fi está activado se puede solicitar que realice un barrido con el fin de obtener todas los puntos de acceso detectados y con que intensidad, que es la información que nos interesa conocer. Del mismo modo que antes, la operación es asíncrona por lo que habrá que usar un BroadcastReceiver donde se recibirá el resultado y este será devuelto allá donde se requirió inicialmente.

7.1.2.3. Programación reactiva

Como ya hemos visto, en Android muchas tareas se deben realizar de forma asíncrona. Lo mismo sucede cuando se realizan comunicaciones a través de la red.

No debemos bloquear el hilo principal donde se ejecuta la interfaz de usuario, o la aplicación se bloqueará y el sistema la cerrará. Por tanto, lo adecuado es que al solicitar una operación asíncrona se especifique una función encargada de atender la respuesta cuando esta esté lista. Mientras tanto, la aplicación debe poder seguir funcionando y es conveniente informar al usuario de que la tarea se está ejecutando.

La forma más sencilla y clásica para indicar la función receptora de la respuesta es mediante *callbacks*, generalmente implementados en Java por clases anónimas que cumplen con la interfaz esperada. Esto puede resultar en un código bastante engorroso, por lo que con Java 8 o Retrolambda en Android podemos usar funciones lambda y referencias a métodos, lo que consigue un código más limpio.

Pero esto no es suficiente cuando necesitamos encadenar diversas acciones asíncronas, ya que surgen problemas como el anidamiento excesivo de callbacks y la incorrecta gestión y propagación de la respuesta o los errores. Por ello, se ha utilizado la programación reactiva gracias a la librería RxJava y RxAndroid, que nos permiten componer funciones asíncronas y construir programas que reaccionan a eventos, además de ocuparse de la gestión de los hilos.

La programación reactiva se basa en el patrón Observador, entre otros y en la programación funcional. Básicamente al llamar a un función que será asíncrona creamos y devolvemos un objeto Observable, al cual nos suscribiremos pudiendo combinar varios observables con diferentes operadores y configurar los hilos que se usarán, etc. Cuando se realiza una suscripción a un objeto observable se ejecuta el código y este llamará a los métodos del suscriptor correspondientes cuando tenga éxito o falle y cuando termine.

A lo largo de este proyecto se ha utilizado la programación reactiva en todas las tareas asíncronas y en las comunicaciones a través de la red.

7.1.2.4. Protocolo MQTT

Para la comunicación entre el *frontend* y el *backend* se ha utilizado el protocolo MQTT. Este protocolo utiliza un *broker* que hace de intermediario donde se conectan los clientes para publicar o recibir mensajes. Básicamente el protocolo utiliza *topics* que podemos ver como canales en los que se pueden publicar mensajes que recibirán aquellos que esten conectados y suscritos a él.

Hay un par de detalles interesantes:

- Los *mensajes retenidos*, son mensajes normales pero se marcan como *retained*. Cada *topic* siempre mantendrá el último mensaje *retained* que haya sido publicado, por lo que un cliente que no estuviera conectado cuando se publicó el mensaje lo recibirá cuando se conecte y suscriba al *topic*. La limitación es que solo se puede retener el último mensaje en cada *topic*.
- Los *wildcards* (patrones textuales) en los nombres de los *topics*. Permite suscribirse a varios temas cuyo nombre coincida con el patrón.

En la aplicación Android se ha utilizado la librería Eclipse Paho que es un cliente MQTT y ofrece las operaciones de conectar, desconectar, suscribirse, desuscribirse y publicar. Las operaciones son asíncronas y se deben especificar callbacks que serán ejecutados cuando las operaciones han sido completadas o ante eventos como la recepción de un mensaje o la perdida de conexión.

Se ha implementado un servicio de mensajería que envuelve esta librería y ofrece los mismos métodos usando observables de programación reactiva. Además, como normalmente se necesita una comunicación del estilo *Request/Response* tipo HTTP, se han implementado métodos que ofrecen está dinámica y encadenan y realizan las operaciones de conexión, publicación del mensaje de petición, suscripción al tema donde recibir la respuesta, recepción del mensaje de respuesta y desconexión.

En la aplicación servidor se ha utilizado Spring Integration, que ya envuelve la librería Eclipse Paho, y mediante componentes como adaptadores, canales y gateways facilita la recepción y publicación de mensajes con el broker.

La aplicación *frontend* siempre será quien inicie la petición que el *backend* procesará y devolverá la respuesta correspondiente. Para ello el *backend* está siempre suscrito a diferentes *topics* donde espera las peticiones. Por su parte, el *frontend* se conectará solo cuando tenga que enviar un mensaje, enviará el mensaje a un determinado *topic* y se suscribirá a otro (único para él) donde esperará la respuesta, por último se dará de baja del canal y se desconectará.

Por ejemplo, un caso es cuando el dispositivo quiere obtener los datos del usuario que tiene vinculado actualmente:

- El *backend* escucha estas peticiones en el *topic* `devices/+/user/request` donde `+` hace de comodín y cada dispositivo usará su identificador.
- El *frontend* se conectará y mandará la petición al *topic* que le corresponde (por ejemplo, `devices/8cdb0af5-3dfd-4215-91bf-b84a9ed84611/user/request`).
- Inmediatamente, se suscribirá al *topic* donde espera recibir la respuesta a su petición (`devices/8cdb0af5-3dfd-4215-91bf-b84a9ed84611/user/response`).
- El *backend* recibirá la petición, obtendrá el usuario del dipositivo en cuestión y se lo devolverá mediante un mensaje al *topic* de respuesta.
- Cuando el *frontend* reciba el mensaje podrá cerrar su conexión con el broker.

7.1.2.5. Aprendizaje automático para la creación del mapa Wi-Fi

Una vez el dispositivo está registrado y ha establecido comunicación con el servidor para obtener los datos del usuario vinculado, se debe crear el mapa de señales Wi-Fi. Para ello, la aplicación móvil guiará al usuario a través del proceso de toma de muestras, que podrá ser cancelado en cualquier momento. Durante este proceso se le pedirá al usuario ir a cada uno de los lugares de interés del escenario y pulsar un botón que tomará un determinado número de muestras (en principio era 10, luego se subió a 50 para mejorar la precisión). Una vez finalizada la toma de las muestras en todas las localizaciones, se enviarán todas a la aplicación *backend* con el fin de que cree el clasificador de muestras. En caso de no haber conexión a internet, se avisará al usuario y se mantendrá el proceso abierto para que pueda reintentarlo o cancelarlo.

En el servidor, al recibir un mensaje de este tipo desde un dispositivo, se guardarán todas las muestras en la base de datos y se procederá a crear el clasificador de muestras. Para ello se utilizarán todas las muestras válidas del usuario que haya en la base de datos.

Para construir el clasificador se ha utilizado la librería Weka, que incluye algoritmos de aprendizaje automático. Los clasificadores que se han utilizado se detallan en la sección 7.2.2.1. En Weka, una muestra se denomina instancia y se compone de atributos, en nuestro caso, uno por cada punto de acceso Wi-Fi del escenario y cuyo valor será el nivel de intensidad de la señal detectado en la muestra. Además, hay un atributo especial que determina la clase, en nuestro caso, uno de los posibles lugares en que puede ser clasificada la muestra.

Una conjunto de muestras se denomina *dataset*, y cuando las muestras son válidas, es decir, se conoce el atributo de clase, se denomina *training set*. Un clasificador se crea con un conjunto de entrenamiento que incluya muestras para todas las clases, cuanto más grande mejor, ya que ajustará los parámetros internos del clasificador según estás muestras.

Destacar que para el procesado de las muestras se ha hecho uso de las ventajas que ofrece Java 8, es decir *stream collections*, funciones *lambda* y referencia a métodos, dando como resultado un código más limpio y fácil de leer. Y para la construcción del *Training Set* se ha creado una clase siguiendo el patrón de diseño *Builder* [6] que permite construir un objeto mediante encadenamientos de métodos (todos devuelven el objeto builder) que van guardando los parámetros especificados y se termina con un método `build` que finalmente construye y devuelve el objeto en cuestión.

Los clasificadores creados se almacenan en la base de datos como texto codificado en base64 para así poder ser recuperados y descodificados de nuevo a objetos Java, evitando así tener que construirlos cada vez que necesitamos utilizarlos.

7.2. Sprint 2

7.2.1. Planificación del sprint

En el segundo sprint se ha revisado el Product Backlog y se han escogido los siguientes items:

- H15 - Clasificar muestra (8 PH)
- H13 - Obtener datos del usuario vinculado desde el dispositivo (5 PH)
- H17 - Clasificar muestras automáticamente (12 PH)

Todos los items escogidos hacen un total de 25 PH.

7.2.2. Ejecución del sprint

7.2.2.1. Clasificación de las muestras Wi-Fi

Una vez creado el clasificador, se puede utilizar para clasificar una muestra o instancia en la que se desconozca el atributo de clase. La muestra debe incluir valores para todos los atributos posibles, pero una lectura de los puntso de acceso Wi-Fi no siempre va a devolver todos los detectados en todas las muestras utilizadas para entrenar el clasificador, por lo que para los atributos cuyo valor se desconoce se le dara un valor de 0.

El clasificador nos devolverá una predicción con un valor para cada clase que indica el grado de similitud, cuanto más alto más probabilidad de acierto. El resultado de clasificar una muestra no deja de ser una predicción y no asegura el acierto, por lo que para conseguir más fiabilidad, se han utilizado 5 clasificadores distintos (número impar para evitar empates) y se da como resultado la clase con más votos. Los clasificadores utilizados han sido:

- *MultilayerPerceptron* es una red neuronal artificial formada por múltiples capas. [35]
- *SMO* es un algoritmo de clasificación mediante máquinas de soporte vectorial. [30]
- *J48*, es un algoritmo de clasificación mediante un árbol de decisión. [44]
- *RandomForest* es una combinación de árboles de decisión. [32]
- *BayesNet* es una red bayesiana. [34]

7.2.2.2. Almacenamiento de datos en Elasticsearch

Elasticsearch es un gestor de bases de datos NoSQL basado en documentos JSON. Se compone de índices, cada uno de los cuales sería análogo a una base de datos SQL. Un índice contiene tipos que representan una clase de documentos, que serían el equivalente a una tabla SQL. Y cada tipo contiene documentos JSON, que serían el equivalente a filas en SQL. La diferencia es que los documentos no siguen un esquema estricto y sus atributos son variables.

Los *mappings* definen cómo un campo de un tipo va a ser procesado y analizado. Por ejemplo, un campo puede ser procesado de la siguiente forma: se convierte a minúsculas, se reemplazan los caracteres especiales por su equivalente ASCII, se crean *tokens* de todas las posibles combinaciones entre un mínimo y un máximo de caracteres. Cuando se realiza una búsqueda *full text* sobre ese campo se comprueba si coincide con algún token. Elasticsearch tiene gran variedad de filtros, *tokenizers* y *analyzers* predefinidos, pero permite crear nuevos con reglas personalizadas que se definien en la configuración del índice.

Todo en Elasticsearch utiliza JSON: las consultas, los resultados, los documentos, la configuración, etc. Dispone de una API REST y también de un driver Java. En este proyecto se ha utilizado Spring Data para trabajar al estilo JPA, mapeando clases Java con la base de datos.

Spring Data ofrece repositorios inteligentes, definiendo métodos en una interfaz cuyo nombre siga ciertas reglas genera de forma transparente la implementación. Aunque se tiene la posibilidad de implementar métodos propios si así se requiere. Por ejemplo, en nuestro caso se ha querido aprovechar la capacidad de Elasticsearch para que los resultados devuelven el resaltado de las coincidencias y se ha creado tanto un método propio para la consulta como un *Result Mapper* que mapea los resultados a objetos de la clase en cuestión.

7.2.2.3. Clasificación automática de las muestras Wi-Fi

Para llevar un registro de la localización del dispositivo, se tomarán muestras de forma automática cada cierto tiempo. Para ello, se utiliza el *AlarmManager* de Android para definir una alarma que disparará un *intent* (descripción abstracta de una operación a realizar que sirve para invocar componentes) cada minuto (no permite lapsos de tiempo menores a un minuto). Un minuto es una frecuencia lo suficientemente alta para el tipo de registro que se quiere llevar y el consumo de batería es aceptable, ya que debe durar al menos una jornada sin necesidad de cargar el dispositivo.

La alarma se define en dos casos: al terminar el proceso de creación del mapa Wi-Fi para el usuario vinculado (la primera vez) y durante el arranque del dispositivo (el resto de veces) ya que las alarmas se borran cuando el dispositivo se reinicia. La alarma también será borrada, si se desvincula al usuario del dispositivo.

Para definir la alarma durante el arranque del dispositivo se ha creado un *Broadcast Receiver* que este pendiente de cualquier *intent* con la acción `android.intent.action.BOOT_COMPLETED` que generará el sistema operativo cuando haya arrancado. Para poder escuchar esta acción es necesario declarar el permiso `android.permission.RECEIVE_BOOT_COMPLETED`. Una vez el

sistema arranca y se llama al código del *Broadcast Receiver* se comprobará que el dispositivo esté registrado, se tenga un usuario vinculado y este haya completado la creación de su mapa Wi-Fi, en caso de cumplir los requisitos se definira la alarma.

Cuando la alarma se dispara la alarma se lanza un *intent* que activará otro *Broadcast Receiver* encargado de tomar una muestra Wi-Fi y enviarla al servidor para su clasificación y almacenamiento en la base de datos.

En caso de no poder establecer conexión con el broker para publicar la muestra, se ha creado un *fallback* que guardará la muestra en un repositorio local. Se trata de una base de datos SQLite en el propio dispositivo y para trabajar con ella se ha utilizado la librería ORMLite.

Por último se ha creado otro *Broadcast Receiver* que estará al tanto de los cambios de conectividad del dispositivo y cuando detecte que se vuelve a tener conexión se lanzará un servicio Android en segundo plano creado para ir publicando las muestras almacenadas en la base de datos local de forma secuencial. Cada muestra publicada con éxito será eliminada de la base de datos local hasta quedar esta vacía o detenerse el proceso en caso de volver a perder la conexión.

7.3. Sprint 3

7.3.1. Planificación del sprint

En el tercer sprint se ha revisado el Product Backlog y se han escogido los siguientes items:

- H01 - Login backoffice (3 PH)
- H02 - Logout backoffice (1 PH)
- H03 - Registrar dispositivo (4 PH)
- H07 - Registrar usuario (6 PH)
- H05 - Consultar dispositivo (3 PH)
- H09 - Consultar usuario (3 PH)
- H06 - Editar dispositivo (2 PH)
- H10 - Editar usuario (2 PH)

Todos los items escogidos hacen un total de 24 PH.

7.3.2. Ejecución del sprint

7.3.2.1. Estructura del código del backoffice

La aplicación *backoffice* sigue una organización similar a la ya explicada para las otras aplicaciones, salvo ciertas diferencias al tratarse de una aplicación web.

Los paquetes de la aplicación *Backoffice* son los siguientes:

- config, contiene clases de configuración de Spring Boot.
- controllers, contiene los controladores de Spring MVC.
- model, contiene las clases que representan el modelo de datos, estás clases utlizan anotaciones para mapear los objetos con la base de datos elasticsearch.
- model/forms, contiene las clases que se usan en los formularios, son clases contenedoras de datos con anotaciones para la validación de los datos.
- model/generators, clases de utilidad para generar datos y entidades aleatorios que han sido usado para hacer pruebas durante el desarrollo.
- model/validation/annotations, contiene anotaciones propias para validaciones personalizadas.

- model/validation/validators, contiene los validadores usados por las anotaciones propias.
- repositories/elasticsearch, contiene las interfaces y clases para los repositorios de elasticsearch, así como clases de utilidad para mapear resultados, consturir queries con resaltado, etc.
- services, contiene la interfaz e implementación de la lógica de negocio, cada clase tiene todas las funciones relacionadas con una responsabilidad.
- resources, contiene ficheros de configuración de la aplicación.
- resources/elasticsearch, contiene ficheros JSON con la configuración para elasticsearch incluyendo análisis personalizado de los datos y mapeado de las clases.
- resources/static, incluye ficheros estáticos usados en la aplicación web: estilos css, fuentes, javascript, etc.
- resources/templates, contiene todas las plantillas Thymeleaf para las vistas de Spring MVC.

7.3.2.2. Seguridad de la apliación web

Debido a que la aplicación web *backoffice* es de uso restringido a los administradores del sistema, se ha protegido el acceso a la misma. Para ello se ha utilizado Spring Security, que permite autorizar el acceso a determinadas rutas y autenticar los usuarios con diferentes métodos: autenticación en memoria, autenticación mediante JDBC o LDAP, etc.

Ciertas rutas no requieren de autorización: todas las de recursos estáticos (css, js e imágenes), la página principal o *home* y la página con el formulario de *login*. El resto de rutas de la aplicación están protegidas y utilizan autenticación en memoria con unas credenciales únicas ya que no se requiere de varios usuarios.

7.3.2.3. Validación de formularios

Para los formularios de creación y edición de usuarios y dispositivos se han creado clases con los campos necesarios para cada formulario. Spring se encarga de mapear el formulario con los objetos de estas clases automáticamente y aplicar la validación correspondiente.

La validación para cada campo se especifica con anotaciones. Algunas anotaciones predefinidas permiten definir que un campo no puede estar vacío, que debe cumplir un patrón (expresión regular), etc. Además, en este proyecto se han creado anotaciones personalizadas que utilizan sus correspondientes validadores para comprobar que el valor introducido en un campo que no admite repetidos este disponible, como el identificador o la dirección MAC, para ello se realizan las correspondientes consultas a la base de datos.

Un ejemplo de validación, para el campo MAC de un dispositivo:

```
@NotBlank(message = "No puede estar vacío.")
@Pattern(regexp = "^([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})\$", message = "No es una direcci
@DeviceMacAvailable
private String mac;
```

7.3.2.4. Capturas de pantalla

A continuación se muestran algunas capturas de pantalla de la aplicación web para las historias de usuario del sprint (Figuras 7.1 a 7.8).

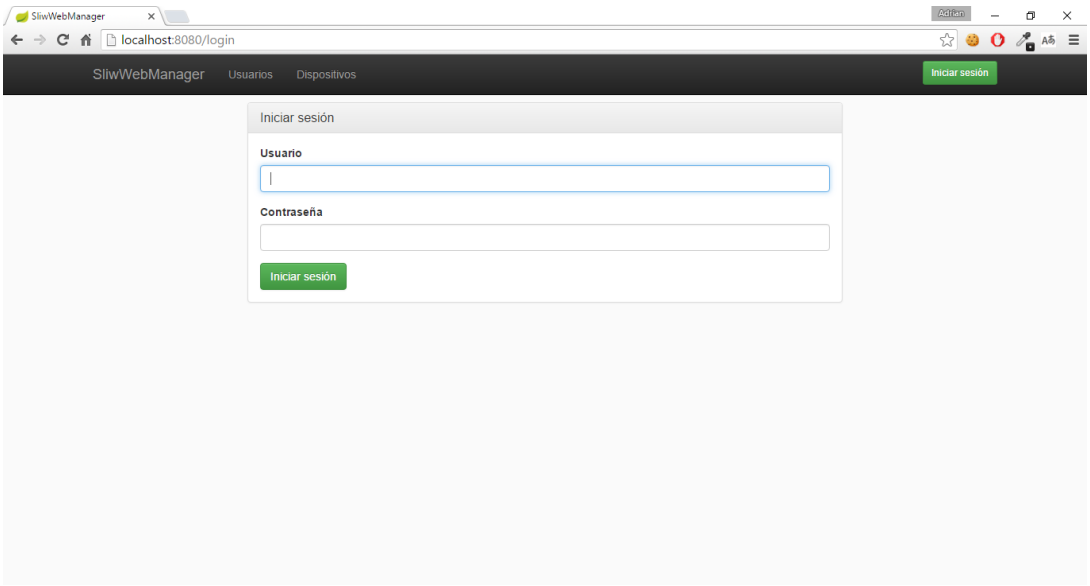


Figura 7.1: Captura del formulario de iniciar sesión.

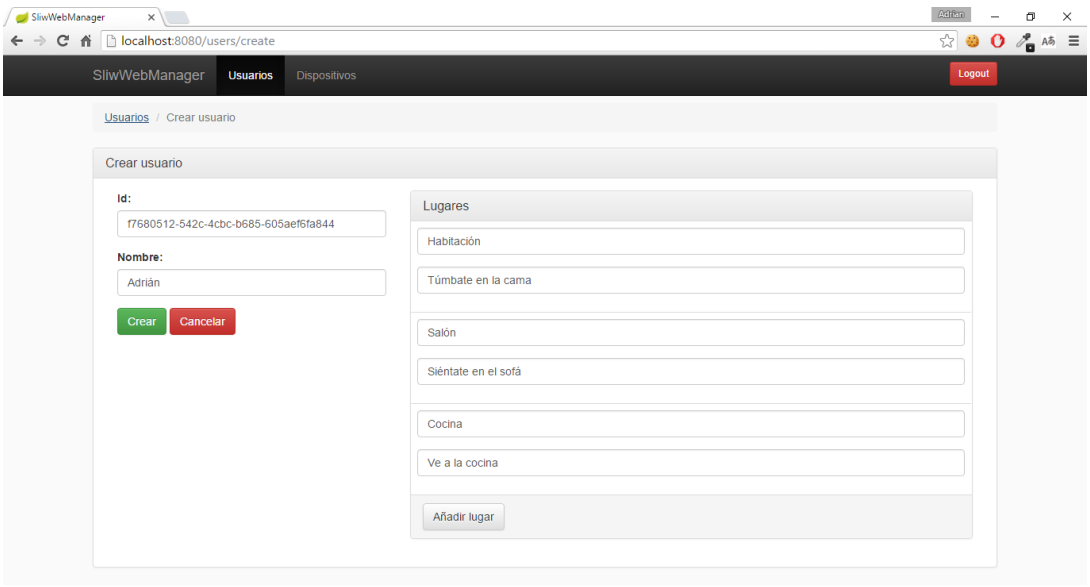


Figura 7.2: Captura del formulario de crear usuario.

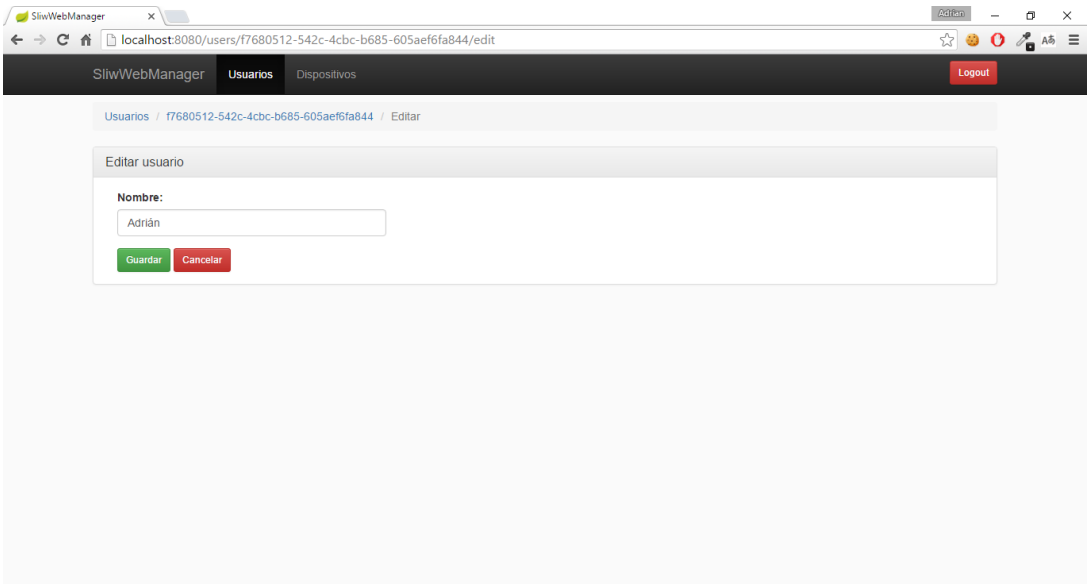


Figura 7.3: Captura del formulario de editar usuario.

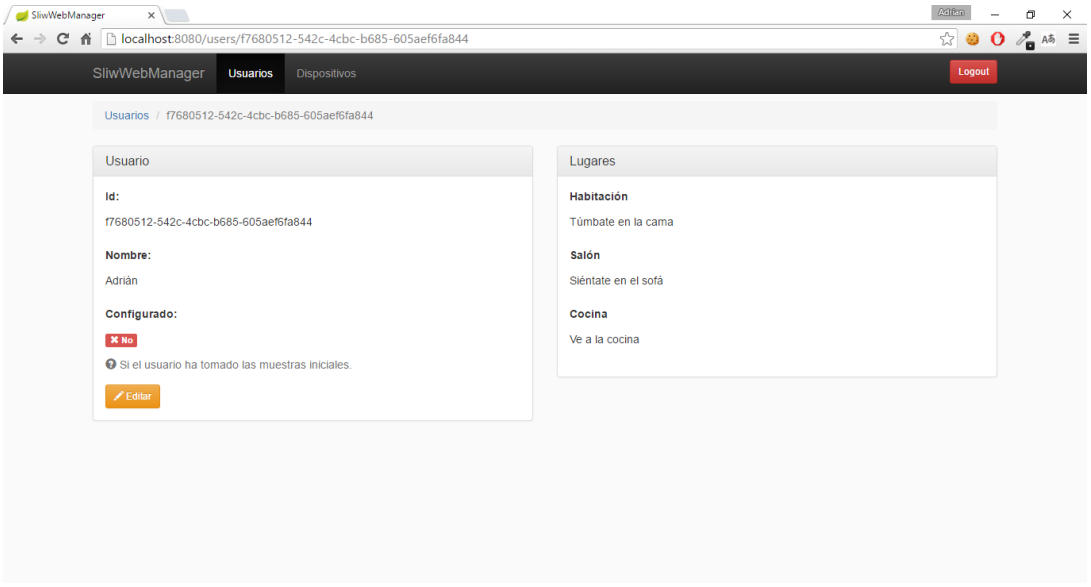


Figura 7.4: Captura de la ficha de usuario.

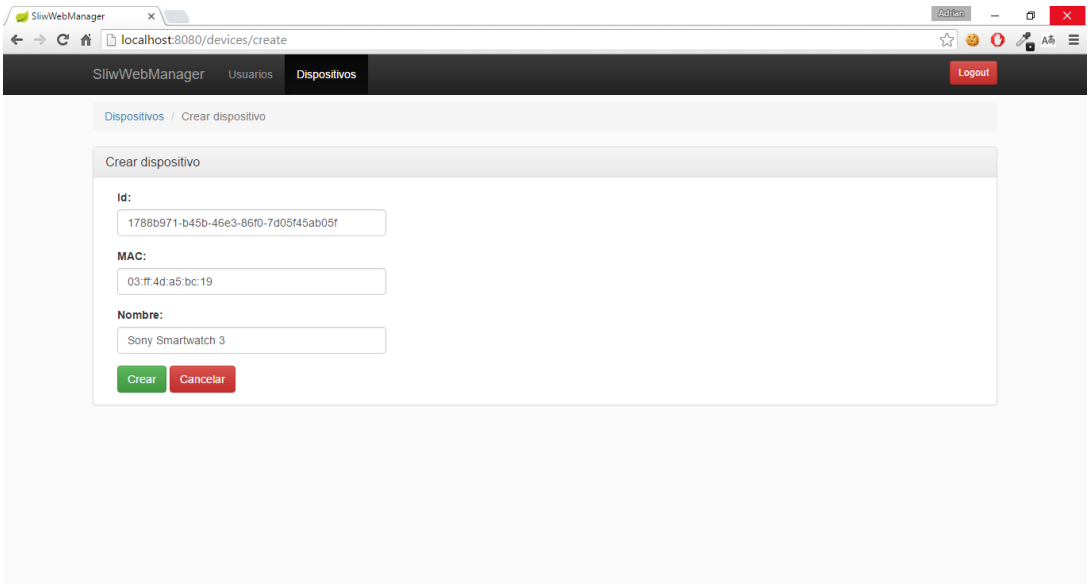


Figura 7.5: Captura del formulario de crear dispositivo.

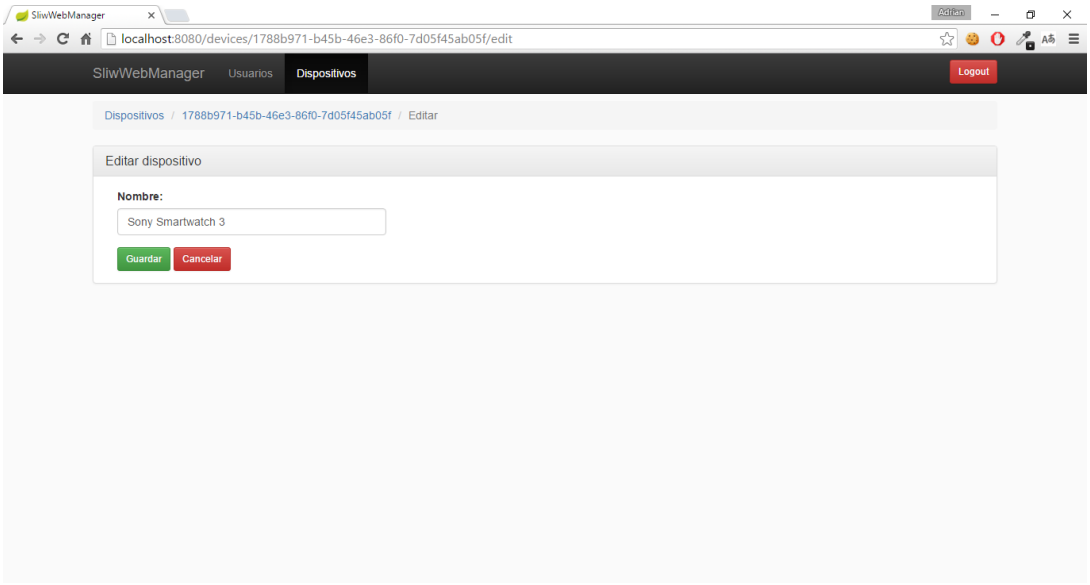


Figura 7.6: Captura del formulario de editar dispositivo.

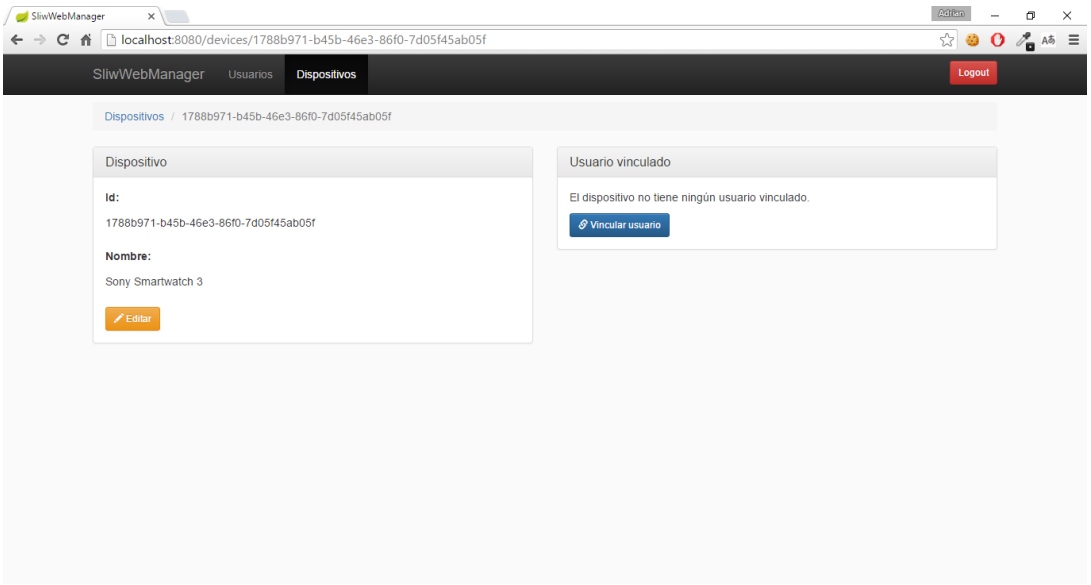


Figura 7.7: Captura de la ficha de dispositivo sin usuario vinculado.

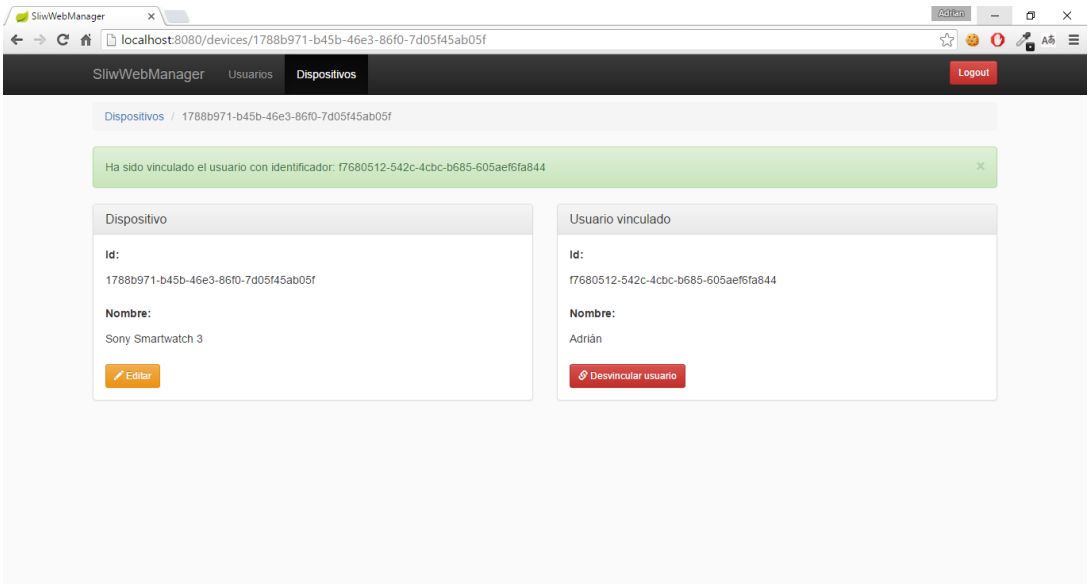


Figura 7.8: Captura de la ficha de dispositivo con usuario vinculado.

7.4. Sprint 4

7.4.1. Planificación del sprint

En el cuarto sprint se ha revisado el Product Backlog y se han escogido los siguientes items:

- H04 - Listar dispositivos (9 PH)
- H08 - Listar usuarios (9 PH)
- H11 - Vincular usuario a un dispositivo (3 PH)
- H12 - Desvincular usuario de un dispositivo (1 PH)

Todos los items escogidos hacen un total de 22 PH.

7.4.2. Ejecución del sprint

7.4.2.1. Listados

Los listados tanto para dispositivos como para usuarios tienen ciertas características comunes:

- Paginación: permite visualizar un determinado número de elementos como máximo por página y navegar entre las diferentes páginas de forma cómoda e intuitiva.
- Tamaño: permite seleccionar el número de elementos máximo a mostrar por página.
- Ordenamiento: permite seleccionar el orden de los elementos, por defecto se ordenan por identificador de forma ascendente.
- Filtrado: permite filtrar los resultados realizando búsquedas *full text* sobre varios campos.
- Resaltado: muestra los resultados coincidentes con el filtro con resaltado en la parte que coincide.

Estos parámetros pueden ser manipulados a través de la interfaz o directamente desde la *query string*.

Para la vinculación de un usuario a un dispositivo se utiliza un listado similar para seleccionar al usuario.

7.4.2.2. Capturas de pantalla

A continuación se muestran algunas capturas de pantalla de la aplicación web para las historias de usuario del sprint (Figuras 7.9 a 7.11).

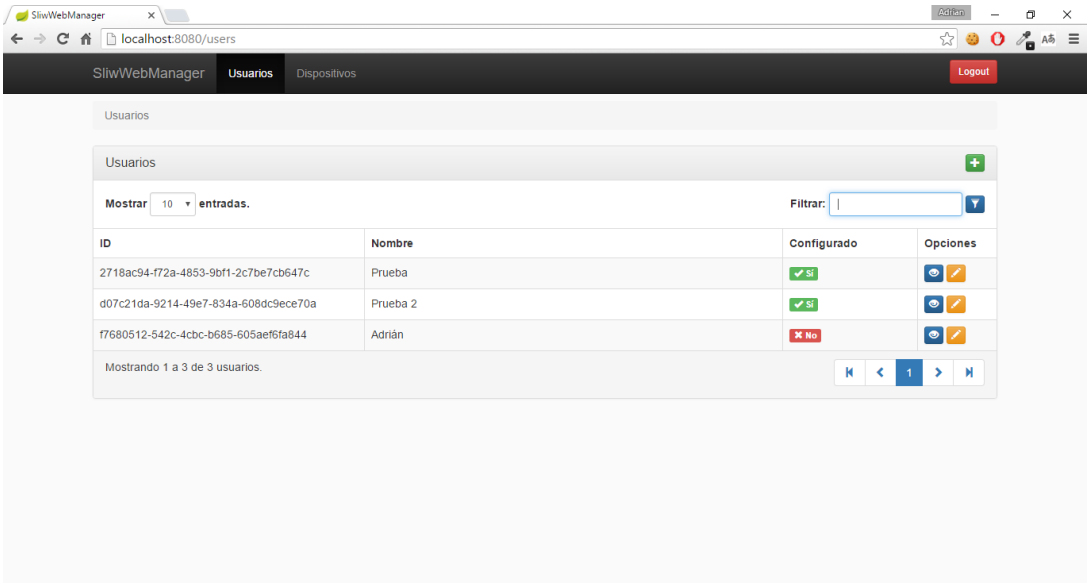


Figura 7.9: Captura del listado de usuarios.

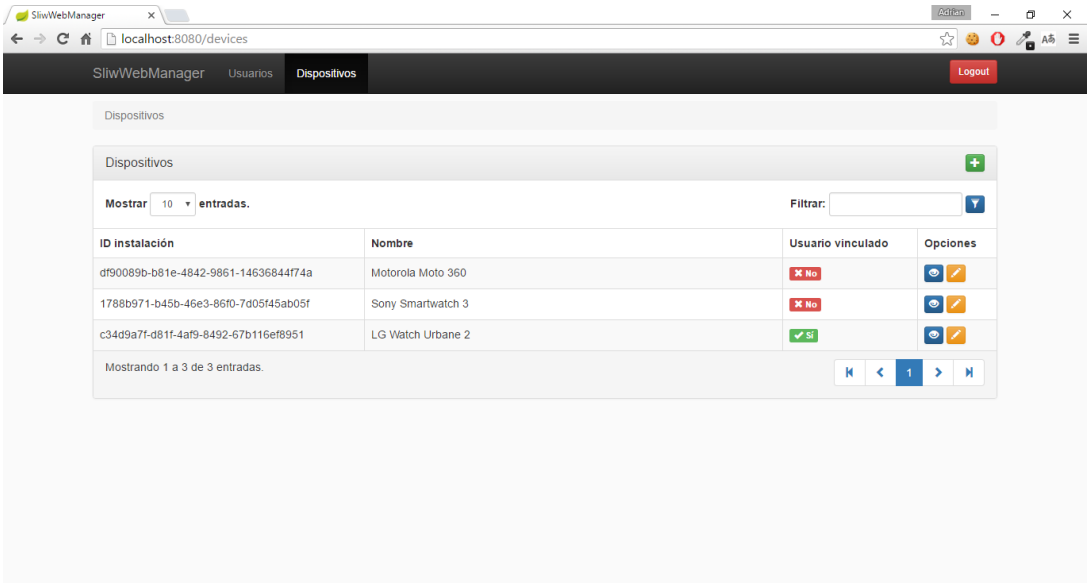


Figura 7.10: Captura del listado de dispositivos.

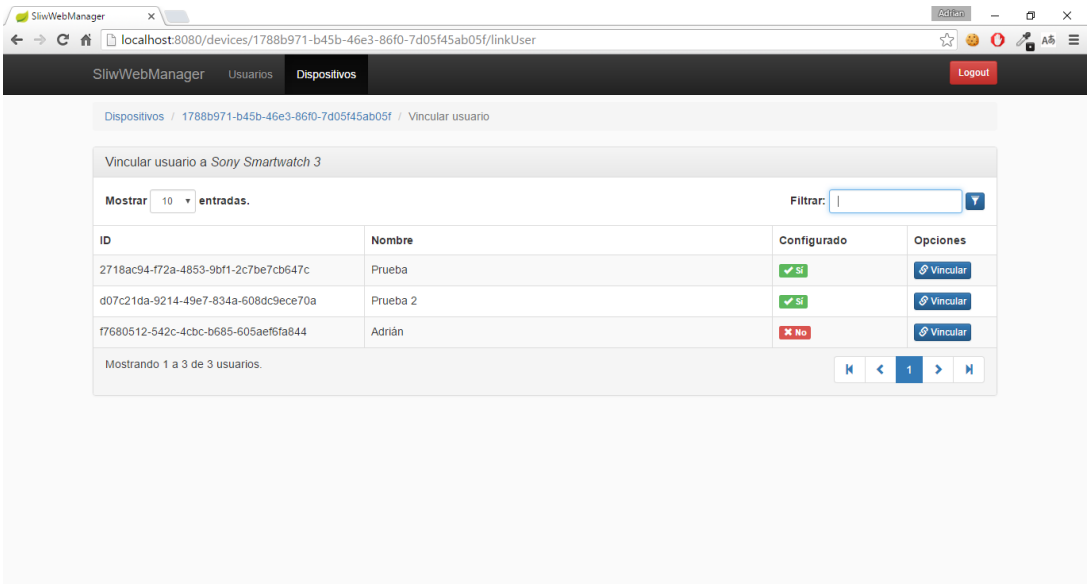


Figura 7.11: Captura del listado de usuarios para vincular al dispositivo.

7.4.2.3. Cambios en la API de Android 6.0

Durante la ejecución del sprint nos hemos encontrado con un cambio en la API de Android 6.0 que nos ha obligado a realizar cambios en el sistema. Para poder comprender el problema, primero es necesario conocer algunos detalles de implementación.

En el sistema un dispositivo es identificado de forma única por un identificador y por la dirección MAC de su chip Wi-Fi. En un principio el administrador registra un dispositivo en el sistema a través del backoffice, donde puede dejar el identificador autogenerado o indicar uno propio y donde debe indicar la dirección MAC. Los dispositivos, cuando hablan con el servidor, se identifican con su dirección MAC, dado que no conocen su identificador en la base de datos. La razón de hacer esto así es que en los dispositivos Android Wear como smartwatches no se dispone de teclado virtual y era la única forma de poder distinguir los dispositivos.

El problema surge cuando Google publica su nueva versión de Android (6.0), ya que elimina por completo la posibilidad de acceder a cierta información sobre el hardware del dispositivo como la dirección MAC. En su lugar, la API devuelve ahora una cadena constante `"02:00:00:00:00:00"`. Este cambio se justifica con la intención de otorgar una mayor privacidad al usuario ya que persigue extinguir el *tracking* de dispositivos. Un extracto de [59]:

```
Access to Hardware Identifier

To provide users with greater data protection, starting in this release,
Android removes programmatic access to the device's local hardware identifier
for apps using the Wi-Fi and Bluetooth APIs. The WifiInfo.getMacAddress() and
theBluetoothAdapter.getAddress() methods now return a constant value
of 02:00:00:00:00:00.

...

Note: When a device running Android 6.0 (API level 23) initiates
a background Wi-Fi or Bluetooth scan, the operation is visible
to external devices as originating from a randomized MAC address.
```

Tras evaluar el problema e investigar posibles soluciones [18] que nos permitan identificar de forma única a un dispositivo, vemos las siguientes opciones:

- **Tracking de las instalaciones**: al instalar la aplicación y arrancarla por primera vez generamos un UUID y este se guarda junto con los datos de la aplicación.
  - Ventajas: es sencillo, no requiere de permisos especiales, no hay dependencia con la API (sujeta a cambios) y es lo que recomiendan desde Google para respetar la privacidad (recomiendan hacer tracking de instalaciones y no de dispositivos).
  - Desventajas: El id muere con la instalación, por lo que habría que actualizarlo si se reinstala la app o casos similares.
- **TelephonyManager.getDeviceId()** [56]: método que ofrece la API de Android y nos devuelve el IMEI, MEID o ESN, es bastante utilizado.
  - Ventajas: id único por dispositivo, sobrevive incluso a *factory resets*.
  - Desventajas: Requiere permisos de teléfono aunque la app no lo use para nada y algunos dispositivos pueden no tener este identificador como es el caso del smartwatch, así que **no sirve**.
- **Número de serie** [54]: desde Android 2.3 existe en la API acceso a un número de serie android.os.Build.SERIAL.
  - Ventajas: las mismas que las de TelephonyManager, y no requiere permisos especiales.
  - Desventajas: los dispositivos pueden no tener este número, solo están obligados en caso de no ser teléfonos, por lo que se complementa con la otra opción, pero no me acaba de convencer.
- **ANDROID_ID** [55]: es un entero de 64 bits generado la primera vez que el dispositivo se enciende y persiste excepto si se hace un reseteo a los valores de fábrica o similar.
  - Ventajas: id único por dispositivo, sencillo.
  - Desventajas: No es 100 % persistente, además con versiones de Android iguales o superiores a 4.2 si el dispositivo tiene más de un usuario, cambia para cada usuario. También se han dado casos, por ejemplo, en que toda una gama de dispositivos del mismo fabricante y modelo salieron con el mismo id.

Finalmente, la solución preferida es la de generar nuestro propio identificador único para cada instalación. Por tanto, hay que realizar los siguientes cambios:

- Generar un UUID al instalar la app.
- El dispositivo debe autoregistrarse en el servidor con su id.
- No usar la MAC como identificador del dispositivo en ninguna parte.

Tanto el campo MAC de un dispositivo en nuestro modelo de datos como la funcionalidad de registrar un dispositivo desde el backoffice pueden mantenerse debido a que no afectan negativamente y pueden ser simplemente ignoradas u ocultadas.

7.5. Sprint 5

7.5.1. Planificación del sprint

En el quinto sprint se ha revisado el Product Backlog y se han escogido los siguientes items:

- H16 - Validar muestra (6 PH)
- H19 - Registrar instalación en dispositivo (4 PH)

Todos los items escogidos hacen un total de 10 PH.

7.5.2. Ejecución del sprint

7.5.2.1. Mejorando los clasificadores

Cuando se crean los clasificadores para un usuario siguiendo el proceso de creación del mapa Wi-Fi, se toman un determinado número de muestras para cada lugar de interés. En un principio se tomaban 10 muestras por lugar, pero resultó ser insuficiente, dando como resultado malas predicciones. Se subió a 50, que daba mejores predicciones y seguía siendo bastante rápido, pues es importante no hacer esperar demasiado al usuario. Claro que lo ideal seria tomar muchas más, pues se podría aumentar la efectividad. Se decidió dar la posibilidad al usuario de tomar muestras válidas en cualquier momento para mejorar sus clasificadores.

Básicamente, se le muestra una lista de sus lugares, elige uno, se toma una muestra, se le pone el lugar elegido y se manda al servidor. Cuando el servidor recibe una muestra válida simplemente la guarda en la base de datos y vuelve a generar los clasificadores del usuario utilizando todas las muestras válidas que tenga. Por suerte, la creación de los clasificadores es prácticamente instantánea, al menos para el volumen de muestras con el que se ha trabajado.

7.5.2.2. Registrando la instalación de la aplicación *frontend*

Tras el cambio explicado en 7.4.2.3, se ha optado por no utilizar la dirección MAC para identificar de forma única al dispositivo y en su lugar generar un identificador único universal (UUID).

Se ha creado una clase de utilidad (Installation) con un método estático y sincronizado (getId), que podría verse como un Singleton [37]. Se mantiene una copia del identificador de la instalación en una propiedad estática y se persiste en las *Shared Preferences* de Android.

Cuando se llama al método para obtener el identificador se hace lo siguiente: si la propiedad que mantiene una copia del identificador es nulo o es una cadena vacía, se intenta obtener desde las *Shared Preferences*, en caso de que no exista, se genera un nuevo UUID y se asigna a la propiedad y se persiste en las *Shared Preferences*. Por último, se devuelve el identificador.

La primera vez que un dispositivo arranca la aplicación recién instalada, esta mostrará el paso de registrar el dispositivo, que simplemente mandará el identificador de la instalación al *backend*. Automáticamente el *backend* registrará la nueva instalación y aparecerá en el listado, desde donde podrá ser editada para asignarle un nombre y se le podrá vincular un usuario. Si se registra con éxito, el dispositivo ya no volverá a pedir este paso en futuros inicios de la aplicación.

7.6. Resumen de Sprints

En la tabla 7.1 se detalla en que sprint ha sido implementada cada historia de usuario.

Historia de usuario	**Sprint**
H18 - Tomar muestra de señales Wi-Fi	1
H14 - Crear mapa de señales Wi-Fi	1
H15 - Clasificar muestra	2
H13 - Obtener datos del usuario vinculado desde el dispositivo	2
H17 - Clasificar muestras automáticamente	2
H01 - Login backoffice	3
H02 - Logout backoffice	3
H03 - Registrar dispositivo	3
H07 - Registrar usuario	3
H05 - Consultar dispositivo	3
H09 - Consultar usuario	3
H06 - Editar dispositivo	3
H10 - Editar usuario	3
H04 - Listar dispositivos	4
H08 - Listar usuarios	4
H11 - Vincular usuario a un dispositivo	4
H12 - Desvincular usuario de un dispositivo	4
H16 - Validar muestra	5
H19 - Registrar instalación en dispositivo	5

Tabla 7.1: Resumen de Sprints.

7.7. Verificación y validación

En este apartado se explica qué pruebas se han realizado para verificar y validar el correcto funcionamiento del sistema.

Destacar que la empresa no tiene una política estricta sobre las pruebas del software. No se requiere que se realicen pruebas automáticas a la hora de desarrollar un proyecto, aunque el desarrollador es libre de hacerlas. Evidentemente, sí deberán hacerse las pruebas manuales necesarias para poder validar que el sistema hace lo que se espera de él.

Debido a la ajustada planificación que se ha tenido, se ha optado por no aplicar técnicas como el *TDD (Test-Driven Development)* y solo se han realizado unas pocas pruebas en los puntos en los que se haya considerado oportuno.

7.7.1. Pruebas unitarias

Las pruebas unitarias consisten en probar las partes del código de forma aislada. Deben ser pruebas pequeñas, rápidas y automáticas. Para poder aislar el código se utilizan técnicas como los *mocks* que reemplazan las dependencias del código a probar por unas que podemos controlar el resultado o comportamiento que van a tener.

En este proyecto se han realizado unas pocas pruebas unitarias, sobretodo en el backend y cubriendo las funciones relacionadas con la creación y uso de los clasificadores de muestras.

7.7.2. Pruebas de integración

Las pruebas de integración consisten en probar todas las parte del código de forma conjunta. Esto incluye también usar componentes externos como bases de datos. Aunque son pruebas automáticas, son más lentas a la hora de ser ejecutadas pero aseguran que los componentes se integran de forma correcta.

En este proyecto se han realizado unas pocas pruebas de integración, sobre todo en el backend y cubriendo los servicios encargados de la comunicación y los repositorios que trabajan con la base de datos. Esto ha permitido comprobar que el sistema funciona correctamente cuando se integra con el resto de componentes.

7.7.3. Pruebas de aceptación

Las pruebas de aceptación consisten en probar que el sistema realiza lo que se desea, es decir, validan los requisitos del usuario. Pueden ser manuales o automáticas, pero debido a la complejidad de automatizar pruebas sobre interfaces gráficas suelen acabar siendo manuales.

En este proyecto se han realizado pruebas de aceptación manuales que validen los requisitos del sistema. Se han realizado pruebas para cada uno de los escenarios descritos en las historias de usuario, es decir, comprobando que el sistema responde bien frente a casos excepcionales como pérdidas de conexión, introducción de datos no válidos, etc. Se ha comprobado que los resultados son los esperados en cada caso y que se cumplen todos los objetivos del proyecto.

7.7.4. Otras pruebas

Se han realizado diversas pruebas manuales para evaluar la efectividad de los clasificadores con diferentes configuraciones, con el fin de utilizar la configuración más óptima.

En un principio se tomaban 10 muestras por lugar a la hora de crear los clasificadores debido a que no se quería hacer esperar demasiado al usuario. Tras ver que la tasa de aciertos en las clasificaciones no era suficientemente buena (aproximadamente de un 55 %), se probó con 20, 50 y 100 muestras por lugar.

Las pruebas se realizaron en un escenario con 4 lugares (habitaciones diferentes), clasificando 5 muestras por lugar. Los resultados se muestran en la tabla 7.2.

Creación de los clasificadores		Clasificación de muestras		
Muestras (por lugar/total)	Tiempo (por lugar)	Aciertos	Fallos	Tasa de acierto
10/40	∼11 segundos	11	9	55 %
20/80	∼23 segundos	13	7	65 %
50/200	∼58 segundos	17	3	85 %
100/400	∼122 segundos	19	1	95 %

Tabla 7.2: Resultados de las pruebas.

A medida que se aumenta el número de muestras, aumenta también la tasa de acierto y el tiempo de espera en la toma de muestras. Al final, tras valorar los resultados se optó por dejarlo en 50 muestras por lugar, ya que ofrece una tasa de acierto lo suficientemente alta (aproximadamente de un 85 %) y la toma se realiza en apenas un minuto por lugar.

Aún así, la efectividad siempre va a ir ligada a la calidad del escenario en cuestión: número de puntos de acceso disponibles y la distribución de estos, distancia entre los lugares a clasificar, etc.

Capítulo 8

Despliegue

Índice del capítulo

Este capítulo explica en qué consiste el despliegue del sistema informático y cómo se ha llevado a cabo en este proyecto.

8.1. Instalación clásica

El sistema informático desarrollado está formado por cinco componentes: el frontend, el backend, el backoffice, la base de datos (elasticsearch) y el broker MQTT (apache apollo).

El frontend (aplicación móvil) debe instalarse en los dispositivos Android en los que desee ser usada. La instalación puede hacerse de forma manual, mediante el archivo APK o de forma automática a través de Google Play. Al ser este un proyecto con un público cerrado no se ha publicado la aplicación en Google Play, así que solo puede ser instalada mediante el archivo APK. Antes de compilar la aplicación hay que especificar en el archivo config.xml (el archivo donde se definen valores constantes de configuración que se importan en el código) el host, usuario y contraseña del broker para poder establecer las comunicaciones con el backend.

El resto de componentes deben instalarse en un servidor. Pueden instalarse todos en el mismo servidor o distribuirlos entre varios servidores, siempre y cuando se especifique a cada componente como debe comunicarse con el resto.

Primero hay que instalar la base de datos y el broker MQTT ya que son dependencias para las aplicaciones backend y el backoffice. Ambas aplicaciones se han empaquetado en formato JAR y pueden ser ejecutadas sin necesidad de un servidor Tomcat ya que Spring Boot incluye un servidor embebido.

8.2. Instalación con Docker

En este proyecto se ha realizado el despliegue utilizando Docker debido a sus beneficios y con el fin de aprender esta tecnología. En esta sección se explica brevemente que es Docker y como se ha *dockerizado* el sistema.

8.2.1. ¿Qué es Docker?

Docker [9] es una herramienta que permite empaquetar una aplicación junto con sus dependencias en un contenedor que puede ser desplegado de forma trivial en cualquier máquina que disponga del motor Docker.

Los componentes básicos de Docker son las imágenes y los contenedores. Las imágenes son *snapshots* o instantáneas del sistema de ficheros que contienen todo lo necesario para el funcionamiento de la aplicación, mientras que lo contenedores son instancias en ejecución de las imágenes y mantienen su propio estado.

Las imágenes Docker son construidas por capas que contienen los cambios en el sistema de ficheros, lo que permite compartir el resto del sistema de ficheros entre capas, resultando estas muy ligeras. Así mismo, las capas pueden ser compartidas entre distintas imágenes por lo que el uso de disco y las descargas son más eficientes.

Los contenedores Docker, a diferencia de las máquinas virtuales, son ligeros y rápidos dado que comparten el sistema operativo anfitrión y su kernel, lo que permite un uso más eficiente de la memoria RAM. Los contenedores son seguros debido a que virtualizan su propio sistema de ficheros.

8.2.2. Dockerizando el sistema

Para construir las imágenes de cada componente se escribe un fichero `Dockerfile` en el que partiendo de una imagen base se específica instrucción a instrucción los comandos a ejecutar para instalar todo lo necesario. Con el comando `docker build` construimos las imágenes con el nombre deseado.

Una vez tenemos las imágenes construidas se debe ejecutar el comando `docker run` en el que especificaremos las opciones necesarias: mapeado de puertos entre el contenedor y el host, uso de volúmenes, variables de entorno, etc.

Se han utilizado contenedores especiales que solo contienen un volumen de datos para desacoplar los datos y ficheros de configuración de las aplicaciones del contenedor que las ejecuta, así se puede compartir entre varias instancias y facilita el realizar copias de seguridad y evita la pérdida de los datos en caso de eliminar el contenedor en ejecución.

Además, se han creado *shell scripts* para facilitar las tareas de construir las imágenes, crear los contenedores de datos, ejecutar las instancias, pararlas, arrancarlas de nuevo y eliminarlas.

Por último se ha creado un fichero `docker-compose.yml` qué específica como construir y arrancar todos los contenedores anteriores de forma conjunta, así podemos gestionar el ciclo de vida del sistema con el comando `docker-compose`. También se han creado scripts para crear, arrancar, parar y eliminar todo los contenedores de forma conjunta.

Todos los ficheros de Docker y scripts creados pueden consultarse online en el repositorio de GitHub [45]. Los ficheros específicos de Docker se adjuntan en el anexo A.

Capítulo 9

Conclusiones

Índice del capítulo

En este capítulo se exponen las conclusiones, tanto a nivel técnico como a nivel personal, tras haber finalizado el desarrollo del proyecto. También se incluye un apartado en el que se propone el trabajo futuro para el proyecto.

9.1. Conclusiones técnicas

A nivel técnico se ha logrado llevar a cabo el desarrollo del sistema con éxito, cumpliendo los objetivos del proyecto y la funcionalidad esperada del sistema.

El fin de este proyecto era poder valorar la viabilidad de esta tecnología así como su efectividad. Dado que el sistema ha demostrado ser funcional, se puede concluir que el sistema es viable. En cuanto a su efectividad, durante el desarrollo se han hecho ciertas pruebas que han demostrado resultados aceptables con una tasa de acierto en la clasificación de las muestras bastante bueno. Claro está, al final esto va a depender de muchos factores: el entorno concreto, número y distribución de los puntos de acceso Wi-Fi, número de muestras válidas para entrenar los clasificadores, etc.

9.2. Conclusiones personales

A nivel personal, puedo decir que tanto la estancia en prácticas en la empresa como el desarrollo del proyecto han resultado ser experiencias de lo más gratificantes y enriquecedoras.

Puedo decir que durante el desarrollo este proyecto he puesto en práctica los conocimientos adquiridos en el grado de Ingeniería Informática a la vez que he adquirido nuevos conocimientos.

Lo que más me ha gustado es el haber trabajado con una variedad tan amplía de tecnologías y herramientas, de las que gran parte eran totalmente desconocidas antes de empezar el proyecto.

9.3. Trabajo futuro

Como trabajo futuro, lo primero que debe hacerse es realizar pruebas reales, esto incluye: entornos, usuarios y aplicaciones. Con ello, podrá valorarse realmente la efectividad del sistema.

Dado que el alcance del proyecto no cubre todo lo que se podría esperar de un sistema completo de localización, sobretodo en la parte posterior de análisis de los datos obtenidos, queda esto pendiente como trabajo futuro. Siempre sujeto a que la empresa quiera dar continuidad al proyecto.

Bibliografía

[1] Android sdk. https://en.wikipedia.org/wiki/Android_software_development#Android_SDK. [Wikipedia] Visitado el 04/07/2016.

[2] Apache lucene. https://en.wikipedia.org/wiki/Lucene. [Wikipedia] Visitado el 04/07/2016.

[3] Aprendizaje automático. https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico. [Wikipedia] Visitado el 04/07/2016.

[4] Arquitectura de software. https://es.wikipedia.org/wiki/Arquitectura_de_software. [Wikipedia] Visitado el 04/07/2016.

[5] Bootstrap. http://getbootstrap.com/. Visitado el 04/07/2016.

[6] Builder pattern. https://en.wikipedia.org/wiki/Builder_pattern. [Wikipedia] Visitado el 04/07/2016.

[7] Css. hoja de estilos en cascada. https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada. [Wikipedia] Visitado el 04/07/2016.

[8] Diagrama de gantt. https://es.wikipedia.org/wiki/Diagrama_de_Gantt. [Wikipedia] Visitado el 28/06/2016.

[9] Docker. https://www.docker.com/. Visitado el 04/07/2016.

[10] Elasticsearch. https://www.elastic.co/products/elasticsearch. Visitado el 04/07/2016.

[11] Enterprise integration patterns. https://en.wikipedia.org/wiki/Enterprise_Integration_Patterns. [Wikipedia] Visitado el 04/07/2016.

[12] Git. https://git-scm.com/. [Git] Visitado el 04/07/2016.

[13] Github. https://github.com/. Visitado el 04/07/2016.

[14] Gradle. https://gradle.org/. Visitado el 04/07/2016.

[15] Herramienta case. https://es.wikipedia.org/wiki/Herramienta_CASE. [Wikipedia] Visitado el 04/07/2016.

[16] Historias de usuario. https://es.wikipedia.org/wiki/Historias_de_usuario. [Wikipedia] Visitado el 28/06/2016.

[17] Html5. https://es.wikipedia.org/wiki/HTML5. [Wikipedia] Visitado el 04/07/2016.

[18] Identifying app installations. http://android-developers.blogspot.com.es/2011/03/identifying-app-installations.html. [Android Developers Blog] Visitado el 02/07/2016.

[19] Invest. https://en.wikipedia.org/wiki/INVEST_(mnemonic). [Wikipedia] Visitado el 28/06/2016.

[20] Java. https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n). [Wikipedia] Visitado el 04/07/2016.

[21] Javascript. https://es.wikipedia.org/wiki/JavaScript. [Wikipedia] Visitado el 04/07/2016.

[22] Javaserver pages. https://en.wikipedia.org/wiki/JavaServer_Pages. [Wikipedia] Visitado el 04/07/2016.

[23] jquery. https://jquery.com/. Visitado el 04/07/2016.

[24] Junit. http://junit.org/junit4/. Visitado el 04/07/2016.

[25] Marshalling. https://en.wikipedia.org/wiki/Marshalling_(computer_science). [Wikipedia] Visitado el 04/07/2016.

[26] Mockito. http://mockito.org/. Visitado el 04/07/2016.

[27] Model view viewmodel. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel. [Wikipedia] Visitado el 04/07/2016.

[28] Modelo vista controlador. https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador. [Wikipedia] Visitado el 04/07/2016.

[29] Modelo vista presentador. https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93presentador. [Wikipedia] Visitado el 04/07/2016.

[30] Máquina de vectores de soporte. https://es.wikipedia.org/wiki/M%C3%A1quinas_de_vectores_de_soporte. [Wikipedia] Visitado el 04/07/2016.

[31] Ormlite. https://www.sqlite.org/. Visitado el 04/07/2016.

[32] Random forest. https://es.wikipedia.org/wiki/Random_forest. [Wikipedia] Visitado el 04/07/2016.

[33] Reactivex. http://reactivex.io/. Visitado el 04/07/2016.

[34] Red bayesiana. https://es.wikipedia.org/wiki/Red_bayesiana. [Wikipedia] Visitado el 04/07/2016.

[35] Red neuronal artificial. https://es.wikipedia.org/wiki/Red_neuronal_artificial. [Wikipedia] Visitado el 04/07/2016.

[36] Scrum (desarrollo de software). https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software). [Wikipedia] Visitado el 28/06/2016.

[37] Singleton. https://es.wikipedia.org/wiki/Singleton. [Wikipedia] Visitado el 04/07/2016.

[38] Sql. https://es.wikipedia.org/wiki/SQL. [Wikipedia] Visitado el 04/07/2016.

[39] Sqlite. http://ormlite.com/. Visitado el 04/07/2016.

[40] Sublime text. https://www.sublimetext.com/. Visitado el 04/07/2016.

[41] Tdd: Desarrollo guiado por pruebas. https://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas. [Wikipedia] Visitado el 28/06/2016.

[42] Thymeleaf. http://www.thymeleaf.org/. Visitado el 04/07/2016.

[43] Uml. https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado. [Wikipedia] Visitado el 04/07/2016.

[44] Árbol de decisión. https://es.wikipedia.org/wiki/%C3%81rbol_de_decisi%C3%B3n. [Wikipedia] Visitado el 04/07/2016.

[45] adripc64. Sliwdocker. https://github.com/adripc64/SliwDocker. [GitHub] Visitado el 04/07/2016.

[46] Apache. Apache activemq. http://activemq.apache.org/. Visitado el 04/07/2016.

[47] Apache. Apache apollo. https://activemq.apache.org/apollo/. Visitado el 04/07/2016.

[48] Apache. Apache maven. https://maven.apache.org/. Visitado el 04/07/2016.

[49] Atlassian. Comparing workflows. https://www.atlassian.com/git/tutorials/comparing-workflows/. [Git] Visitado el 04/07/2016.

[50] Atlassian. Jira software. https://es.atlassian.com/software/jira. Visitado el 04/07/2016.

[51] Atlassian. Sourcetree. https://www.sourcetreeapp.com/. [Git] Visitado el 04/07/2016.

[52] Paramvir Bahl and Venkata N. Padmanabhan. Radar: An in-building rf-based user location and tracking system. http://research.microsoft.com/en-us/people/padmanab/infocom2000.pdf. [INFOCOM 2000] Visitado el 24/06/2016.

[53] Soluciones Cuatroochenta. Sobre cuatroochenta. http://www.cuatroochenta.com/sobre-cuatroochenta/. Visitado el 24/06/2016.

[54] Android Developers. Build. https://developer.android.com/reference/android/os/Build.html#SERIAL. [Android Reference] Visitado el 02/07/2016.

[55] Android Developers. Settings.secure. https://developer.android.com/reference/android/provider/Settings.Secure.html#ANDROID_ID. [Android Reference] Visitado el 02/07/2016.

[56] Android Developers. Telephonymanager. https://developer.android.com/reference/android/telephony/TelephonyManager.html?hl=es#getDeviceId(). [Android Reference] Visitado el 02/07/2016.

[57] Eclipse. Paho. https://eclipse.org/paho/. Visitado el 04/07/2016.

[58] FasterXML. Jackson. https://github.com/FasterXML/jackson. [GitHub] Visitado el 04/07/2016.

[59] Google. Android 6.0 changes. http://developer.android.com/intl/es/about/versions/marshmallow/android-6.0-changes.html#behavior-hardware-id. [Access to hardware identifier] Visitado el 02/07/2016.

[60] Google. Android studio. https://developer.android.com/studio/index.html. Visitado el 04/07/2016.

[61] Google. Android wear. https://developer.android.com/wear/index.html. Visitado el 04/07/2016.

[62] JetBrains. Intellij idea. https://www.jetbrains.com/idea/. Visitado el 04/07/2016.

[63] Daniel Kummer. Gitflow cheatsheet. http://danielkummer.github.io/git-flow-cheatsheet/index.html. [Git] Visitado el 04/07/2016.

[64] No Magic. Magicdraw uml. http://www.nomagic.com/products/magicdraw.html. Visitado el 04/07/2016.

[65] The University of Waikato. Weka. http://www.cs.waikato.ac.nz/ml/weka/. Visitado el 04/07/2016.

[66] orfjackal. Retrolambda. https://github.com/orfjackal/retrolambda. [GitHub] Visitado el 04/07/2016.

[67] Pivotal. Spring boot. http://projects.spring.io/spring-boot/. Visitado el 04/07/2016.

[68] Pivotal. Spring data. http://projects.spring.io/spring-data/. Visitado el 04/07/2016.

[69] Pivotal. Spring framework. http://projects.spring.io/spring-framework/. Visitado el 04/07/2016.

[70] Pivotal. Spring integration. http://projects.spring.io/spring-integration/. Visitado el 04/07/2016.

[71] Pivotal. Spring security. http://projects.spring.io/spring-security/. Visitado el 04/07/2016.

[72] ReactiveX. Rxandroid. https://github.com/ReactiveX/RxAndroid. [GitHub] Visitado el 04/07/2016.

[73] ReactiveX. Rxjava. https://github.com/ReactiveX/RxJava. [GitHub] Visitado el 04/07/2016.

[74] SetBlau. Scrum sprint cycle. http://setblau.com/programacion.php. [Figura] Visitado el 04/07/2016.

[75] Evercoder Software. Moqups. https://moqups.com/. Visitado el 04/07/2016.

Anexo A

Ficheros Docker

En este anexo se incluyen los ficheros específicos de Docker: `Dockerfile` y `docker-compose.yml`. Los scripts no se incluyen ya que son muchos y tampoco aportan detalles importantes, aún así pueden consultarse en el repositorio [45].

A.1. Descripción del repositorio

SliwDocker 1.0

SliwDocker contiene los scripts necesarios para el despliegue de Sliw usando Docker.

Los servicios pueden ser gestionados manualmente o automáticamente, de forma individual o en conjunto.

Importante: si se usan los servicios de forma individual, hay tener en cuenta las dependencias que tienen entre ellos.

Esta versión incluye los siguientes servicios:

- Apache Apollo 1.7.1.
- Elasticsearch 1.7.5.
- Kibana 4.1.
- SliwServer 1.0.
- SliwWebManager 1.0.

Cada servicio tiene su propia carpeta que contiene:

- Opcionalmente, un fichero Dockerfile que define como construir la imagen.
- Opcionalmente, ficheros de configuración extra usados al construir la imagen.
- Opcionalmente, un script build-image.sh para construir la imagen.
- Opcionalmente, un script create-data-volume.sh para crear un contenedor con un volumen para datos.
- Opcionalmente, un script destroy-data-volume.sh para eliminar el contenedor de datos.
- Un script run-instance.sh para crear y arrancar el contenedor.
- Un script start-instance.sh para arrancar el contenedor creado y detenido.
- Un script stop-instance.sh para detener el contenedor creado y en ejecución.
- Un script remove-instance.sh para eliminar el contenedor creado.

La carpeta raíz contiene:

- Un fichero docker-compose.yml que declara como construir y arrancar todos los servicios en conjunto.
- Un script run.sh que crea y arranca todos los servicios.
- Un script start.sh que arranca todos los servicios detenidos.
- Un script stop.sh que detiene todos los servicios en ejecución.
- Un script rm.sh que elimina todos los servicios creados.

Si se desea utilizar un servicio de forma individual, el orden de ejecución de los scripts es:

- build-image.sh ->create-data-volume.sh ->run-instance.sh
- stop-instance.sh <->start-instance.sh
- remove-instance.sh ->destroy-data-volume.sh

Si se desea utilizar todos los servicios en conjunto, el orden de ejecución de los scripts es:

- run.sh
- stop.sh <->start.sh
- rm.sh

A.2. Dockerfile

A.2.1. Apache Apollo

```
FROM java:8
MAINTAINER Adrian Puertas Cabedo <al259348@uji.es>

RUN apt-get update -y && \
apt-get install --no-install-recommends -y -q curl

ENV APOLLO_VERSION 1.7.1
ENV APOLLO_NAME apache-apollo-$APOLLO_VERSION
ENV APOLLO_TAR $APOLLO_NAME-unix-distro.tar.gz
ENV APOLLO_HOME /opt/$APOLLO_NAME
ENV APOLLO_INSTANCE /opt/apollo-instance

RUN curl -s -O http://archive.apache.org/dist/activemq/activemq-apollo/$APOLLO_VERSION/$APOL
tar xzf $APOLLO_TAR -C /opt/ && \
rm $APOLLO_TAR && \
$APOLLO_HOME/bin/apollo create /opt/apollo-instance

COPY ./etc/apollo.xml $APOLLO_INSTANCE/etc/apollo.xml

EXPOSE 61613 61680

WORKDIR $APOLLO_INSTANCE

CMD ["./bin/apollo-broker", "run"]
```

A.2.2. Elasticsearch

```
FROM elasticsearch:1.7.5
MAINTAINER Adrian Puertas Cabedo <al259348@uji.es>

RUN bin/plugin -i elasticsearch/marvel/latest
```

A.2.3. Aplicación backend

```
FROM java:8
MAINTAINER Adrian Puertas Cabedo <al259348@uji.es>

VOLUME /tmp
ADD sliwserver-1.0-SNAPSHOT.jar app.jar
RUN sh -c 'touch /app.jar'
```

```
ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/app.jar"]
```

A.2.4. Aplicación backoffice

```
FROM java:8
MAINTAINER Adrian Puertas Cabedo <al259348@uji.es>

VOLUME /tmp
ADD sliwwebmanager-0.0.1-SNAPSHOT.jar app.jar
RUN sh -c 'touch /app.jar'

ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/app.jar"]
```

A.3. docker-compose.yml

```
version: '2'
services:
  apache-apollo-data:
    build: ./apache-apollo
    image: sliw/apache-apollo:1.7.1
    volumes:
      - /opt/apollo-instance/data
      - /opt/apollo-instance/etc
      - /opt/apollo-instance/log
    entrypoint: /bin/true
  apache-apollo-instance:
    build: ./apache-apollo
    image: sliw/apache-apollo:1.7.1
    volumes_from:
      - apache-apollo-data
    ports:
      - "61613:61613"
      - "61680:61680"
  elasticsearch-data:
    build: ./elasticsearch
    image: sliw/elasticsearch:1.7.5
    volumes:
      - /usr/share/elasticsearch/config
      - /usr/share/elasticsearch/data
    entrypoint: /bin/true
  elasticsearch-instance:
    build: ./elasticsearch
    image: sliw/elasticsearch:1.7.5
    volumes_from:
```

```
    - elasticsearch-data
  ports:
    - "9200:9200"
    - "9300:9300"
kibana-instance:
  image: kibana:4.1
  ports:
    - "5601:5601"
  links:
    - elasticsearch-instance:elasticsearch
  depends_on:
    - elasticsearch-instance
webmanager-instance:
  build: ./sliwwebmanager
  image: sliw/webmanager:1.0
  ports:
    - "80:8080"
  links:
    - elasticsearch-instance:elasticsearch
  depends_on:
    - elasticsearch-instance
server-instance:
  build: ./sliwserver
  image: sliw/server:1.0
  links:
    - elasticsearch-instance:elasticsearch
    - apache-apollo-instance:apache-apollo
  depends_on:
    - elasticsearch-instance
    - apache-apollo-instance
```