Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación Android y Servicio Web Spring para la monitorización de datos obtenidos en un vehículo haciendo uso de la plataforma FIWARE

Autor: Sergio Mellado Contioso

Tutor: María Teresa Ariza Gómez

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación Android y Servicio Web Spring para la monitorización de datos obtenidos en un vehículo haciendo uso de la plataforma FIWARE

Autor:
Sergio Mellado Contioso

Tutor:
María Teresa Ariza Gómez
Profesor titular

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo de Fin de Grado: Aplicación Android y Servicio Web Spring para la monitorización de datos obtenidos en un vehículo haciendo uso de la plataforma FIWARE

Autor: Sergio Mellado Contioso

Tutor: María Teresa Ariza Gómez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Con este Trabajo Fin de Grado llego al final de una de las etapas más importantes y de las que me siento más orgulloso de mi vida. Ha tenido momentos mejores y peores, pero siempre aprendiendo cosas nuevas, también me ha ayudado para crecer a nivel personal y tengo que agradecérselo a cada una de las personas que han estado siempre conmigo de principio a fin en todo este tiempo.

En primer lugar, agradecérselo a toda mi familia, y en especial a mis padres y abuelos que han estado apoyándome y dándome ánimos siempre que lo he necesitado. Todo esto no habría sido posible sin ellos.

En segundo lugar, me gustaría agradecerle a cada uno de los profesores con los que he cursado clases y prácticas en la Escuela. Por vuestras explicaciones y consejos, que me han llevado a estar donde estoy y me ayudan a seguir mejorando día a día en el futuro.

También, agradecerle a mi tutora en este proyecto, María Teresa Ariza Gómez, por su tiempo siempre que lo he necesitado y las facilidades cuando he tenido algún problema.

Por último, agradecer a los compañeros de clase que han estado desde el primer día, por esos momentos de diversión y los días intensos de estudio, se han convertido en unos amigos para toda la vida.

Gracias.

Sergio Mellado Contioso

Sevilla, 2020

Resumen

Hace tiempo parecía muy lejano poder conectar cualquier objeto cotidiano a Internet, e intercambiar información sobre los hábitos de consumo y gustos de las personas. Hoy en día, gracias al Internet de las Cosas (*IoT*), todo eso es posible. IoT es la interacción entre múltiples objetos físicos conectados y con tecnologías de obtención de datos, que permiten comunicarse e intercambiar información entre ellos en tiempo real mediante Internet.

En este proyecto se ha diseñado e implementado un software para la monitorización de la velocidad, posición (latitud y longitud), aceleración (ejes x, y, z del acelerómetro) y la velocidad de rotación (ejes x, y, z del giroscopio) de los vehículos, obtenidos por los sensores de un dispositivo móvil *Android*. Consiste en una aplicación desarrollada en Android, la cual envía todos los valores obtenidos por los sensores del dispositivo *Android* al Context broker. Se ha utilizado *Orion Context Broker*, el cual pertenece a una plataforma desarrollada para el internet de las cosas (*IoT*) llamada *FIWARE*. Se ha optado por utilizar FIWARE ya que es una plataforma de código abierto centrada en el desarrollo de soluciones inteligentes.
Además, se desarrolla un servicio web *Spring,* el cual se encarga de realizar una suscripción al context broker. Mediante esta suscripción, cada vez que se envían nuevos valores al context broker, el servicio web *Spring* recibe notificaciones. El servicio web *Spring* se encarga de realizar las transacciones con la base de datos *MySQL*, donde quedarán almacenados todos los valores obtenidos por la aplicación *Android*.
A través de la aplicación *Android* se permite visualizar todos los datos obtenidos en un vehículo, mediante el acceso al servicio web.

Se ha realizado una aplicación móvil en *Android*, observando que el uso de teléfonos inteligentes se ha disparado en la última década, siendo el sistema operativo *Android* la razón de la mayor parte del crecimiento. Actualmente, Android, el sistema operativo de Google, tiene más de 10 años y da vida casi al 90% de los smartphones que utilizamos a diario en nuestro día a día.

Abstract

A long time ago it seemed very far to be able to connect any everyday object to the Internet, and exchange information about people's consumption habits and tastes. Today, thanks to the Internet of Things (IoT), all of that is possible. IoT is the interaction between multiple physical objects connected and with data collection technologies, which allow communication and exchange of information between them in real time through the Internet.

In this project, software has been designed and implemented to monitor speed, position (latitude and longitude), acceleration (x, y, z axis of the accelerometer) and rotational speed (x, y, z axis of the gyroscope). of vehicles, obtained by the sensors of an Android mobile device. It consists of an application developed on Android, which sends all the values obtained by the sensors of the Android device to the Context broker. Orion Context Broker has been used, which belongs to a platform developed for the Internet of Things (IoT) called FIWARE. FIWARE has been chosen as it is an open source platform focused on the development of smart solutions.
In addition, a Spring web service is developed, which is responsible for subscribing to the context broker. Through this subscription, every time new values are sent to the context broker, the Spring web service receives notifications. The Spring web service is in charge of carrying out the transactions with the MySQL database, where all the values obtained by the Android application will be stored.
Through the Android application, you can view all the data obtained in a vehicle, by accessing the web service.

A mobile application has been made on Android, noting that the use of smartphones has exploded in the last decade, with the Android operating system being the reason for most of the growth. Currently, Android, Google's operating system, has more than 10 years of life and gives life to almost 90% of the smartphones that we use daily in our day to day.

Índice

Agradecimientos **9**

Resumen **11**

Abstract **13**

Índice **14**

Índice de Tablas **17**

Índice de ilustraciones **19**

1 Introducción **22**
1.1 Motivación *22*
1.2 Objetivos *22*
1.3 Antecedentes *23*
1.4 Descripción de la solución *23*
1.4.1 Objetivos específicos 23
1.4.2 Funcionalidades 23
1.4.3 Esquema de la arquitectura 24
1.5 Estructura de la memoria *25*

2 Recursos utilizados **26**
2.1 Recursos Hardware *26*
2.1.1 Ordenador Intel Core i7-3630QM 26
2.1.2 Teléfono móvil Sony Xperia SP 27
2.2 Recursos Software *27*
2.2.1 Máquina Virtual Centos 27
2.2.2 Navegador Web Google Chrome 28
2.2.3 Spring Tool Suite 3 28
2.2.4 Android Studio 28
2.2.5 Xampp 29

3 Tecnologías utilizadas **30**
3.1 Spring *30*
3.1.1 Hibernate 30
3.2 Docker *31*
3.3 MySQL *31*
3.4 Android *31*
3.5 phpMyAdmin *32*
3.6 REST *32*
3.7 JSON *33*
3.8 Mapbox *33*
3.9 SQLite *33*
3.10 MongoDB *34*
3.11 Fiware Orion Context Broker *34*

4 Servicio Web desarrollado con Spring y su conexión con la base de datos **35**
4.1 Clases *38*

4.2 Ficheros de configuración *46*
4.3 Servidor de Base de Datos *49*
4.4 API Rest *51*
4.4.1 Consultas utilizando el método GET 51
4.4.2 Consultas utilizando el método POST 54

5 Aplicación Android MySpeed Go y conexión con el context broker **55**
5.1 Context Broker *55*
5.1.1 Entidades 55
5.1.2 Suscripciones 60
5.2 Aplicación Android MySpeed Go *64*
5.2.1 Clases 64
5.2.2 Ficheros de configuración 72
5.2.3 Recursos 75

6 Interfaz de usuario y funcionalidad **77**

7 Conclusiones y líneas futuras **83**

Anexo A: Instalación de Orion Context Broker en CentOS a través de Docker **85**

Anexo B: Instalación y configuración de Android Studio **88**

Anexo C: Instalación y configuración de Spring Tool Suite **92**

Anexo D: Configuración de la base de datos MYSQL **94**

Anexo E: Puesta en marcha y ejecución del proyecto **99**

Referencias **102**

ÍNDICE DE TABLAS

Tabla 1: API Rest – Consultas de datos mediante GET 53
Tabla 2: API Rest – Gestión de suscripciones mediante GET 54
Tabla 3: Gestión de notificaciones mediante POST 54

ÍNDICE DE ILUSTRACIONES

Ilustración 1: Esquema de la arquitectura del sistema 24
Ilustración 2: Intel(R) Core(TM) i7-3630 26
Ilustración 3: Sony Xperia SP 27
Ilustración 4: Centos 7 28
Ilustración 5: Navegador Google Chrome 28
Ilustración 6: Spring Tool Suite 3 28
Ilustración 7: Android Studio 29
Ilustración 8: Xampp 29
Ilustración 9: Spring 30
Ilustración 10: Hibernate 30
Ilustración 11: Docker 31
Ilustración 12: MySQL 31
Ilustración 13: Android 32
Ilustración 14: phpMyAdmin 32
Ilustración 15: JSON 33
Ilustración 16: Mapbox 33
Ilustración 17: SQLite 34
Ilustración 18: MongoDB 34
Ilustración 19: Orion Context Broker 34
Ilustración 20: Registro – Diagrama de Secuencia 35
Ilustración 21: Inicio de sesión – Diagrama de Secuencia 36
Ilustración 22: Asignar Permisos – Diagrama de Secuencia 36
Ilustración 23: Monitorización de datos – Diagrama de Secuencia 37
Ilustración 24: Visualización de datos – Diagrama de Secuencia 37
Ilustración 25: Clase POJO Attribute 40
Ilustración 26: Clase POJO Metadata 40
Ilustración 27: Clase POJO Notification 41
Ilustración 28: Clase POJO Entity 42
Ilustración 29: Clase Application 43
Ilustración 30: Método notificaciones de la clase SensorController 44
Ilustración 31: Fichero de configuración application.properties 47
Ilustración 32: Fichero de configuración hibernate.cfg.xml 47
Ilustración 33: Fichero Usuario.hbm.xml 48
Ilustración 34: Fichero Dato.hbm.xml 48

Ilustración 35: Modelo Entidad-Relación de la Base de Datos 49
Ilustración 36: Ejemplo de datos almacenados en la tabla *datos* 51
Ilustración 37: Cabecera de petición HTTP 56
Ilustración 38: Ejemplo de entidades 58
Ilustración 39: Ejemplo de atributo 59
Ilustración 40: Ejemplo de entidad accedida por filtro 59
Ilustración 41: Ejemplo de entidad accedida por filtro más restrictivo 60
Ilustración 42: Ejemplo de suscripción 60
Ilustración 43: Interacciones de las suscripciones 62
Ilustración 44: Configuración de la suscripción 63
Ilustración 45: Cancelación de la suscripción 63
Ilustración 46: Diagrama de casos de uso de la aplicación Android 64
Ilustración 47: Diagrama de clases – Aplicación Android 65
Ilustración 48 : Diagrama de clases – Conexiones Aplicación Android 66
Ilustración 49: Clase MyApplication 67
Ilustración 50: Clase ClientREST **¡Error! Marcador no definido.**
Ilustración 51: Clase ConnectionClass 70
Ilustración 52: Fichero AndroidManifest.xml 73
Ilustración 53: Fichero build.gradle (Project) 74
Ilustración 54: Fichero build.gradle (Module) 74
Ilustración 55: Directorio /res 75
Ilustración 56: Directorio /res/layout 76
Ilustración 57: Pantalla Principal – Aplicación Android 78
Ilustración 58: Pantalla de Registro – Aplicación Android 78
Ilustración 59: Pantalla Ajustes Principales – Aplicación Android 79
Ilustración 60: Pantalla Selección de rol – Aplicación Android 79
Ilustración 61: Pantalla Ajustes CB – Aplicación Android 80
Ilustración 62: Pantalla Permisos – Aplicación Android 80
Ilustración 63: Pantalla Monitorización -Aplicación Android 81
Ilustración 64: Pantalla Visualización SQLite – Aplicación Android 81
Ilustración 65: Gráfica Velocidad/Hora – Aplicación Android 82
Ilustración 66: Pantalla Visualización – Aplicación Android 82
Ilustración 67: Mapa con la ruta – Aplicación Android 82
Ilustración 68: Listas Acelerómetro y Giroscopio – Aplicación Android 82
Ilustración 69: Ejecución Context Broker 86
Ilustración 70: Detención del Context Broker 87
Ilustración 71: Instalación Android Studio – Paso 1 88
Ilustración 72: Instalación Android Studio – Paso 2 89

Ilustración 73: Instalación Android Studio – Paso 3 89
Ilustración 74: Instalación Android Studio – Paso 4 89
Ilustración 75: Configuración Android Studio – Paso 1 90
Ilustración 76: Configuración Android Studio – Paso 2 90
Ilustración 77: Configuración Android Studio – Paso 3 91
Ilustración 78: Configuración Android Studio – Paso 4 91
Ilustración 79: Fichero de instalación STS 92
Ilustración 80: Descarga de MySQL Community 94
Ilustración 81: Instalación MySQL Community finalizada 95
Ilustración 82: Descarga de Xampp 96
Ilustración 83: Panel de Control de Xampp 96
Ilustración 84: Opciones de MySQL en el panel de control 97
Ilustración 85: Inicio de sesión en phpMyAdmin 97
Ilustración 86: Visualización de las tablas de la base de datos 98
Ilustración 87: Lanzamiento máquina virtual CentOS 99
Ilustración 88: Importar proyecto en STS 100
Ilustración 89: Selección dispositivos Android 101

1 INTRODUCCIÓN

En la vida no existe nada que temer, solo cosas que comprender

- Marie Curie -

1.1 Motivación

El presente trabajo de fin de grado, del grado de Ingeniería de las Tecnologías de Telecomunicación con el propósito de dar soluciones a problemas del día a día, se ha enfocado a trabajar con el llamado Internet de las Cosas (IoT). Para ello se utilizan numerosos conceptos aprendidos durante el grado.

En este proyecto se ha hecho uso de una aplicación Android, ya que el uso de teléfonos inteligentes se ha disparado en la última década y la mayor parte de ese crecimiento se debe al sistema operativo Android.

Se ha optado por el desarrollo de una aplicación móvil Android, ya que el mundo actual se encuentra en constante evolución y desarrollo en el ámbito de los dispositivos móviles, su continuo desarrollo y el incremento del uso en el día a día de cada ser humano han logrado que se conviertan en objetos casi necesarios para vivir. Cada vez el rango de edades que utiliza un dispositivo móvil es más amplio, desde niños hasta ancianos. En la actualidad, ha tomado mucha importancia el llevar nuestro dispositivo a cualquier lugar, ya sea, por ejemplo, para consultar un mapa, el tiempo atmosférico o comunicarnos entre nosotros.

Se ha utilizado una tecnología como es el Fiware Orion Context Broker, desarrollado por Telefónica, y que permite administrar todo el ciclo de vida de la información de contexto, incluyendo las actualizaciones, consultas, registros y suscripciones.

Otro motivo para la realización de este proyecto ha sido poder trabajar y tener la posibilidad de adentrarme más en el aprendizaje del Internet de las cosas (IoT), el cual está teniendo un gran crecimiento en los últimos años y pienso que en pocos años estará más implantado en la sociedad.

1.2 Objetivos

El objetivo es la realización de un proyecto capaz de obtener los valores de los sensores de un dispositivo móvil Android, como son la velocidad, posición (latitud y longitud), aceleración (ejes x, y, z del acelerómetro) y velocidad rotacional (ejes x, y, z del giroscopio) mientras nos movemos en un vehículo. Enviar al Orion Context Broker, en el momento que los obtiene la aplicación Android, cada uno de esos valores. Por último, el servicio web Spring se suscribe al Orion Context Broker para recibir los valores y almacenarlos en una base de datos MySQL.

Además, desde la aplicación Android, se pueden visualizar cada uno de estos valores almacenados en la base de datos.

1.3 Antecedentes

Existen aplicaciones móviles para recoger valores de sensores en los dispositivos móviles, estas aplicaciones están disponibles para los usuarios de Android a través de la *Google Play Store*. Pero no existe ninguna con las funcionalidades desarrolladas en este proyecto, que permita monitorizar y almacenar los datos mientras los visualizamos en tiempo real y además utilizar para ello la tecnología de Orion Context Broker.

Para la parte de este proyecto referente al servicio web se ha tomado como base lo realizado por Pablo Bermejo Pérez en su Trabajo Fin de Grado [1].

1.4 Descripción de la solución

En este apartado se describirá la solución que se ha implementado para poder alcanzar los objetivos indicados.

1.4.1 Objetivos específicos

- **Obtención de datos de los sensores del dispositivo móvil**

  Desde la aplicación móvil desarrollada en Android se obtienen los diversos valores como son la velocidad, posición (latitud y longitud), aceleración (ejes x, y, z del acelerómetro) y velocidad rotacional (ejes x, y, z del giroscopio).

- **Envío de datos al Orion Context Broker**

  El usuario introduce desde los ajustes de la aplicación un intervalo de tiempo para el refresco de los datos tomados en cada instante.

  Los datos se obtienen cada intervalo de tiempo desde los sensores del dispositivo móvil Android. Este dispositivo en el mismo instante que los obtiene, los envía al context broker.

- **Almacenamiento de datos**

  El context broker establece una comunicación con un servicio web, el cual se suscribe al context broker, recibe los datos y los guarda en una base de datos MySQL.

- **Visualización de datos**

  Desde la aplicación móvil Android también es posible visualizar los valores de los datos almacenados en la base de datos MySQL. Para ello se establecerá una comunicación con el servicio web y así, poder acceder a la base de datos para realizar la consulta que se desea. Luego solo tiene que devolver el resultado a la aplicación Android, y esta se encargará de mostrarlos por pantalla de diversas maneras.

- **Registro e inicio de sesión de usuarios**

  Para acceder a la aplicación Android y poder usar todas sus funcionalidades es necesario estar registrado previamente en la aplicación y luego iniciar sesión desde la pantalla Principal de la aplicación. Toda la información de los usuarios se almacena también en la base de datos MySQL, estableciendo el acceso a ella mediante el servicio web.

1.4.2 Funcionalidades

A continuación, se muestran todas las funcionalidades que ofrece el sistema a cada usuario que haga uso de él.

- Obtención de la velocidad del vehículo, posición (latitud y longitud), aceleración (ejes x, y, z del acelerómetro) y velocidad rotacional (ejes x, y, z del giroscopio).
- Aviso leve por pantalla en caso de estar cerca de la velocidad límite establecida.
- Aviso por pantalla en caso de superar la velocidad límite establecida.
- Suscripción a los datos de un vehículo obtenidos a través del dispositivo móvil.
- Consulta de los datos almacenados después de establecer la suscripción.

- Consulta de vehículos totales suscritos.
- Cancelación de la suscripción de un vehículo indicando su matrícula.
- Consulta de datos en la base de datos SQLite de Android.
- Representación y visualización mediante gráficas, mapas o listas en la aplicación móvil de datos almacenados.
- Registro de nuevo usuario en la aplicación Android.
- Inicio de sesión con un usuario ya registrado en la aplicación Android.

1.4.3 Esquema de la arquitectura

En la Ilustración 1 se muestra el esquema de la arquitectura del sistema. En él aparecen cada una de las conexiones e interacciones entre las tecnologías utilizadas.

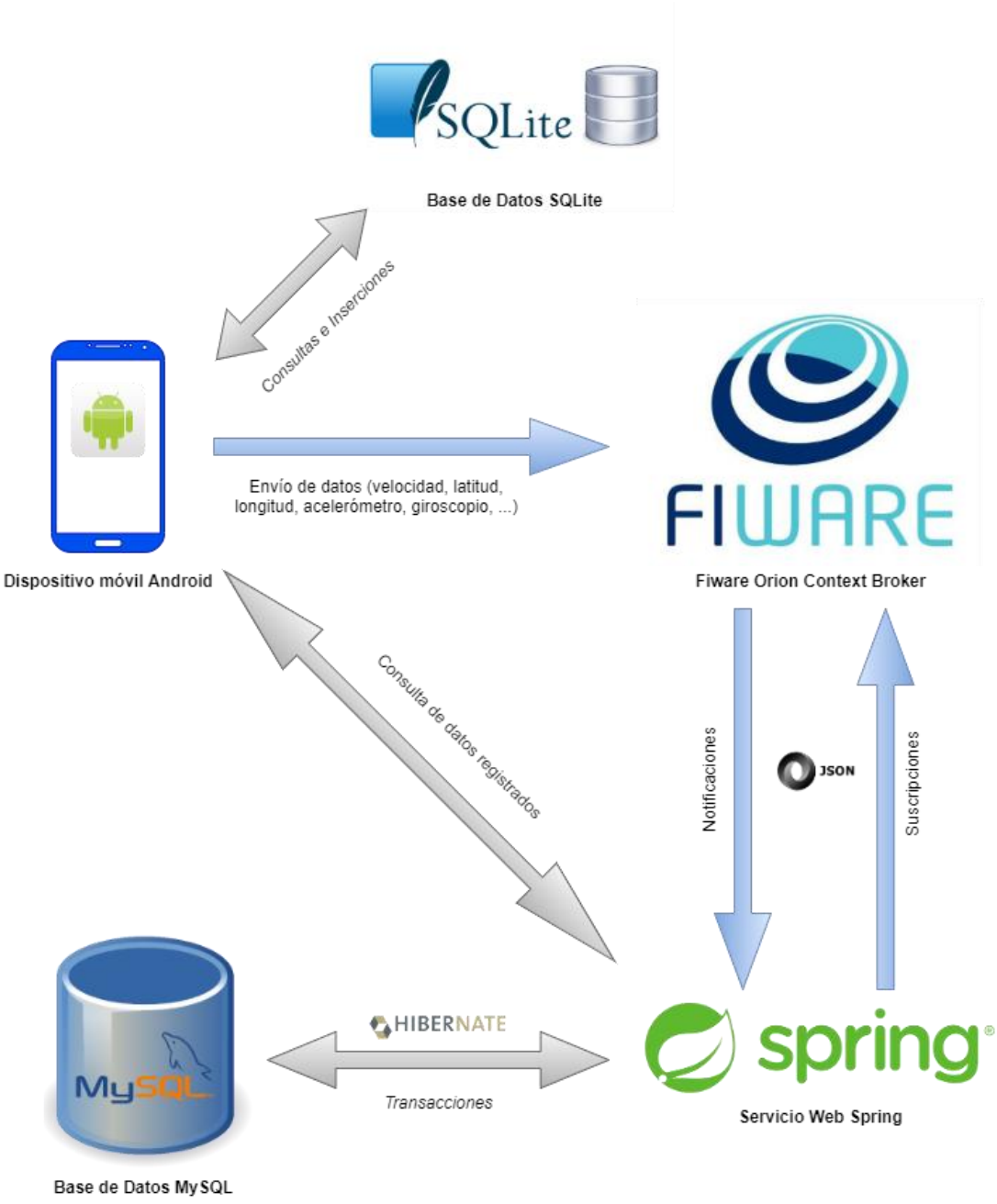


Ilustración 1: Esquema de la arquitectura del sistema

1.5 Estructura de la memoria

1. Introducción: Se presenta el problema y se describen los objetivos y funcionalidades que se han adoptado.
2. Recursos utilizados: En este apartado se muestran los recursos software y hardware utilizados para el proyecto.
3. Tecnologías utilizadas: Se describen todas las tecnologías que se utilizan en todo el proyecto
4. Smartphone y Context Broker: Se muestra y analiza la aplicación Android que se ha llevado a cabo y el Orion Context Broker.
5. Servidor Web y Base de Datos: Se describe como se ha montado el servidor web utilizando Spring y la base de datos MySQL.
6. Interfaz de usuario y funcionalidad: Se describen todas las características que muestra cada pantalla de la aplicación Android para su uso por el usuario.
7. Conclusiones y líneas futuras: Ideas finales obtenidas tras realizar el proyecto y posibles mejoras para el futuro.
    - Anexo A: Instalación de Orion Context Broker en CentOS a través de Docker.
    - Anexo B: Instalación y configuración de Android Studio.
    - Anexo C: Instalación y configuración de Spring Tool Suite.
    - Anexo D: Configuración de la base de datos MySQL.
    - Anexo E: Puesta en funcionamiento y ejecución del sistema.

2 RECURSOS UTILIZADOS

Vivir es enfrentar un problema tras otro. La forma en que lo encaras hace la diferencia.

- Benjamin Franklin -

En este capítulo se van a nombrar y describir cada uno de los recursos hardware y software que han sido necesarios para la realización del proyecto.

2.1 Recursos Hardware

2.1.1 Ordenador Intel Core i7-3630QM

Se ha empleado un ordenador portátil con las siguientes características:

- Procesador: Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz (8CPUs), ~2.4GHz.
- Memoria RAM: 4,00 GB.
- Sistema Operativo: Windows 10 Pro.
- Tarjeta gráfica: AMD Radeon HD 7600M Series.
- Almacenamiento: 640GB HDD + 256GB SSD.

Ilustración 2: Intel(R) Core(TM) i7-3630

En dicho ordenador se aloja el context broker, la base de datos y el servidor web.

2.1.2 Teléfono móvil Sony Xperia SP

Para poder obtener los datos de velocidad, acelerómetro, giroscopio y ubicación se utilizan los sensores que ofrece el Smartphone Sony Xperia SP. Este dispositivo cuenta con conexión Wi-Fi para poder conectarse a la red y transmitir los valores obtenidos.

Sistema operativo: Android.

Tamaño de pantalla: 4,6''.

Procesador: Qualcomm Snapdragon S4 Pro MSM8960T, doble núcleo Krait a 1.7 GHz., GPU Adreno 320.

Memoria RAM: 1GB.

Almacenamiento: 8GB.

Batería: 2370 mAh.

Conectividad: NFC, 4G, WiFi, Bluetooth 4.0, AGPS.

Ilustración 3: Sony Xperia SP

2.2 Recursos Software

2.2.1 Máquina Virtual Centos

La máquina virtual utilizada en este proyecto es Centos 7 (Core), con kernel Linux 3.14.1-lt y virtualizada mediante VMware Workstation 15 player.

Esta es una copia del sistema operativo empleado en los equipos de la sala del laboratorio de telemática de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla.

Ilustración 4: Centos 7

2.2.2 Navegador Web Google Chrome

Es un navegador de código cerrado creado por la compañía Google INC. Es el más rápido, y tiene como objetivo ser seguro, estable y práctico, siempre ofreciendo la mayor comodidad para el usuario a la hora de navegar. Tiene una gran capacidad para procesar los códigos de JavaScript. Está disponible de forma gratuita.

Para administrar la base de datos a través de phpmyadmin se utiliza el navegador web Google Chrome.

Ilustración 5: Navegador Google Chrome

2.2.3 Spring Tool Suite 3

Es un entorno de desarrollo que se basa en Eclipse y personalizado para desarrollar, ejecutar y depurar aplicaciones Spring. Se incluye intregraciones para Git o Maven entre otras.

Es totalmente gratuito, de código abierto.

Mediante Spring Tool Suite 3 se desarrolla el servidor web basado en el uso de Spring con Hibernate.

Ilustración 6: Spring Tool Suite 3

2.2.4 Android Studio

Es un entorno de desarrollo integrado para la plataforma Android. Basado en el software IntelliJ IDEA de JetBrains y es gratuito. Tiene algunas diferencias con Eclipse como la utilización de Gradle para gestionar y automatizar la construcción de proyectos.

Se ha utilizado para desarrollar la aplicación Android encargada de recoger los valores de los sensores, de la comunicación con el Context Broker instalado en la maquina virtual y con el servidor web que accede a la base de datos [2].

Ilustración 7: Android Studio

2.2.5 Xampp

XAMPP es una distribución de Apache completamente gratuita y fácil de instalar que contiene varios módulos. El paquete de instalación de XAMPP es muy fácil de instalar y usar.

Se puede instalar de foma local en el sistema operativo, en este trabajo se ha utilizado el módulo de phpMyAdmin donde se gestiona la base de datos MySQL desde el navegador Google Chrome [3].

Ilustración 8: Xampp

3 TECNOLOGÍAS UTILIZADAS

La ciencia se compone de errores, que, a su vez, son los pasos hacia la verdad.

- Julio Verne -

En este capítulo se muestran y analizan cada una de las tecnologías utilizadas para llevar a cabo este proyecto.

3.1 Spring

Es un framework de código abierto para la creación de aplicaciones Java con soporte para Groovy y Kotlin.

Posee estructura modular y gran flexibilidad para implementar diferentes tipos de arquitectura según necesite la aplicación. Se ha utilizado la tecnología Spring para realizar el desarrollo del servidor web en este proyecto, porque ofrece facilidad para el desarrollo y despliegue [4].

Ilustración 9: Spring

3.1.1 Hibernate

Herramienta de mapeo objeto-relacional para la plataforma Java que facilita el mapeo de atributos entre la base de datos relacional y el modelo de objetos de una aplicación, a través de archivos declarativos (XML) o anotaciones en los beans que permiten establecer estas relaciones.

Busca solucionar el problema de la diferencia entre los dos modelos de datos que coexisten en la misma aplicación, el empleado en la memoria del ordenador y el usado en las bases de datos [5].

Ilustración 10: Hibernate

3.2 Docker

Proyecto que automatize el despliegue de aplicaciones dentro de contenedores de software, a la vez que proporciona una capa de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. Utiliza características de aislamiento del kernel Linux, como cgroups y namespaces, que permiten que contenedores independientes se ejecuten en solo una instancia de Linux. Es de código abierto [6].

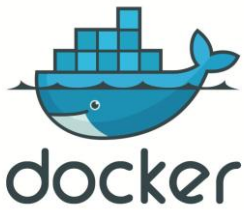


Ilustración 11: Docker

3.3 MySQL

MySQL es un sistema gestor de bases de datos relacionales potente y versátil para satisfacer la mayoría de proyectos en la web.

Es el sistema gestor de base de datos más popular de Internet. De código abierto, con un conjunto de funcionalidades muy amplio y disponible en la mayoría de proyectos y servidores. Además, es multiplataforma, es decir, permite instalarlo en Windows, Linux y Mac [7].

Ilustración 12: MySQL

3.4 Android

Android es un sistema operativo pensado para teléfonos móviles con pantalla táctil, aunque ya se utiliza también en tabletas, televisores, relojes inteligentes y automóviles.

Basado en Linux es un núcleo de sistema operativo libre, gratuito y multiplataforma. El sistema operativo proporciona las interfaces necesarias para desarrollar aplicaciones que puedan acceder a funciones del teléfono, como son el GPS, llamadas, …, de forma muy sencilla mediante Java.

Para el almacenamiento utiliza la base de datos SQLite.

Destaca por la libertad que ofrece a los usuarios, fabricantes y desarrolladores porque no hay que pagar nada para programar en este sistema o incluirlo en un teléfono.

Ilustración 13: Android

3.5 phpMyAdmin

Herramienta escrita en PHP para manejar la administración de MySQL mediante páginas web, utilizando un navegador web [8].

Las funciones que puede realizar actualmente son:

- Crear y eliminar bases de datos.
- Crear, eliminar y alterar tablas.
- Borrar, editar y añadir campos.
- Ejecutar sentencia SQL.
- Administrar claves en campos.
- Administrar privilegios.
- Exportar datos en varios formatos.

Ilustración 14: phpMyAdmin

3.6 REST

Interfaz para conectar varios sistemas basados en el protocolo HTTP y nos sirve para obtener y generar datos y realizar operaciones, retornando los datos en formatos como XML o JSON, siendo este último el más utilizado por ser más legible y ligero.

Es la tecnología utilizada para el desarrollo del servidor del proyecto.

REST se apoya en HTTP, utilizando los mismos verbos:

- GET
- POST
- PUT
- DELETE

3.7 JSON

Es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript. Se utiliza para transmitir datos en aplicaciones web.

Considerado un lenguaje independiente de formato de datos, su especificación se describe en RFC4627.

El servidor REST del proyecto utiliza JSON en vez de XML, ya que posee mayor sencillez frente a XML.

Ilustración 15: JSON

3.8 Mapbox

Plataforma de datos de ubicación para aplicaciones móviles y web. Sus datos son tomados de bases de datos abiertas (OpenStreetMap y la NASA) y de bases de datos de propietarios (DigitalGlobe).

Es de código abierto, sus API, SDK y datos de mapas de actualización en vivo dan a los desarrolladores la posibilidad de construir mejores experiencias de navegación [9].

Ilustración 16: Mapbox

3.9 SQLite

Es una biblioteca escrita en lenguaje C que se encarga de implementar un motor de base de datos SQL pequeño, rápido, autónomo, muy confiable y numerosas funciones.

Estable y multiplataforma. Tiene su código fuente en dominio público y gratuito para que se pueda utilizar en cualquier proyecto.

Se encuentra integrado en todos los teléfonos móviles e incluido en numerosas aplicaciones de gran uso diario [10].

Ilustración 17: SQLite

3.10 MongoDB

Base de datos NoSQL, distribuida, basada en documentos de uso general, ha sido diseñada para desarrolladores de aplicaciones modernas y para la nube.

No guarda los datos en tablas como se hace en las bases de datos relacionales, las guarda en estructuras de datos BSON, muy similar a JSON, para que la integración con las aplicaciones sea más sencilla y rápida. Es multiplataforma [11].

Ilustración 18: MongoDB

3.11 Fiware Orion Context Broker

El Orion Context Broker es un componente obligatorio de cualquier plataforma desarrollada con FIWARE. Aporta una función fundamental en cualquier sistema inteligente, administrar información de contexto, consultarla y actualizarla.

Publica la información de contexto por entidades, como pueden ser los sensores, así la información de contexto que se ha publicado se encuentra disponible para más entidades que la necesiten.

Orion Context Broker es un servidor que implementa una API basada en el modelo de información NGSI, permite varias operaciones:

- Registrar aplicaciones de proveedores de contexto.
- Actualizar información de contexto.
- Ser notificado cuando haya cambios en la información de contexto.
- Consultar información de contexto.

Siempre se encuentra escuchando en un puerto, de forma general suele ser el 1026 y utiliza la base de datos MongoDB. En esta base de datos se almacena el estado actual de las entidades, no almacena información histórica de los cambios producidos en las entidades [12].

Ilustración 19: Orion Context Broker

4 SERVICIO WEB DESARROLLADO CON SPRING Y SU CONEXIÓN CON LA BASE DE DATOS

El trabajo te da significado y propósito, y la vida está vacía sin ambos.

- Stephen Hawking -

En este capítulo y en el siguiente se describen todas las partes de la que consta el sistema que se ha diseñado.

En total consta de 4 partes, las cuales son, la aplicación móvil construida en Android, la base de datos MySQL administrada mediante phpMyAdmin y Xampp, el context broker en una máquina virtual con el sistema operativo CentOS 7 y el servicio web mediante Spring e Hibernate.

Para comprender mejor el funcionamiento general de la aplicación se muestran a continuación varios diagramas de secuencia, donde se puede ver de forma clara las interacciones entre las distintas partes que forman el sistema.

En el primer diagrama de secuencia se muestra el registro de un nuevo usuario (Ilustración 20).

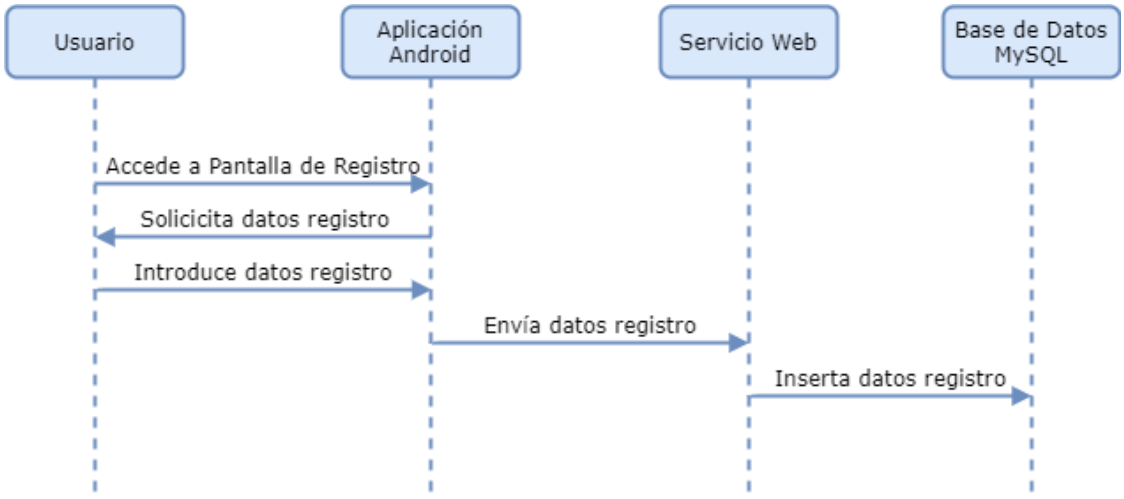


Ilustración 20: Registro – Diagrama de Secuencia

En la Ilustración 21 se observa el inicio de sesión de un usuario que ya se encuentra registrado en el sistema.

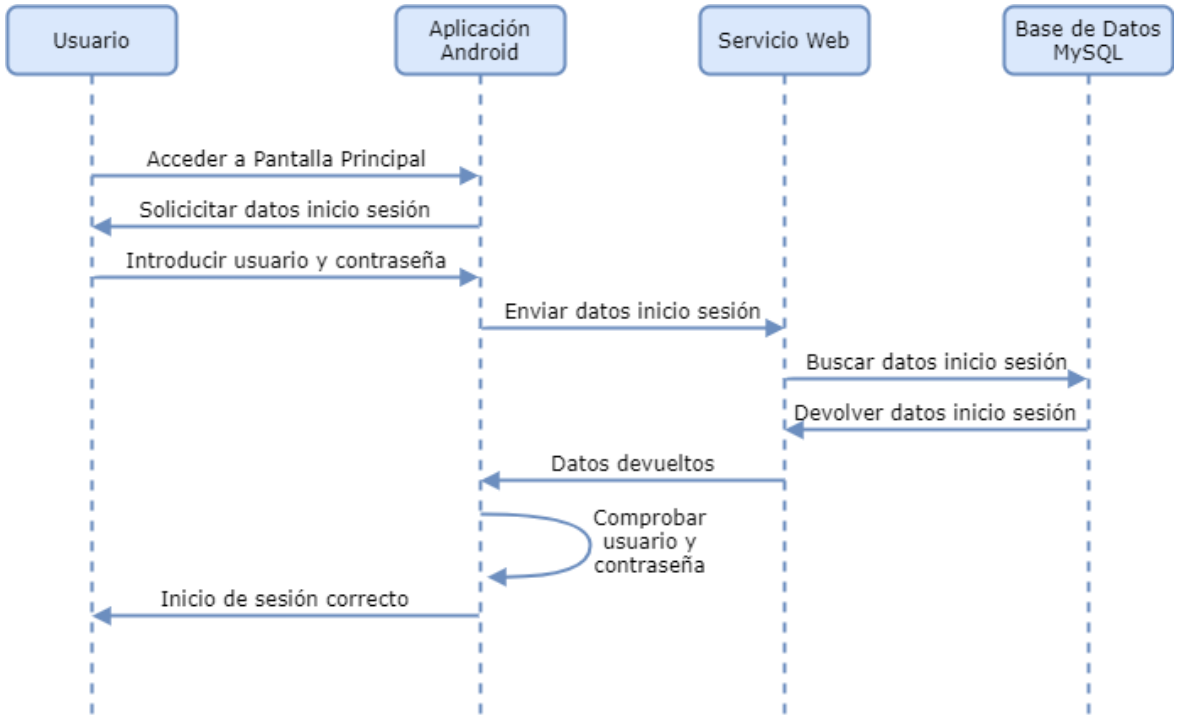


Ilustración 21: Inicio de sesión – Diagrama de Secuencia

En la Ilustración 22 se muestra como un usuario asigna permisos de visualización a otro usuario que se encuentre registrado en el sistema. Así este usuario podrá visualizar los datos almacenados correspondientes al usuario que le ha dado permiso

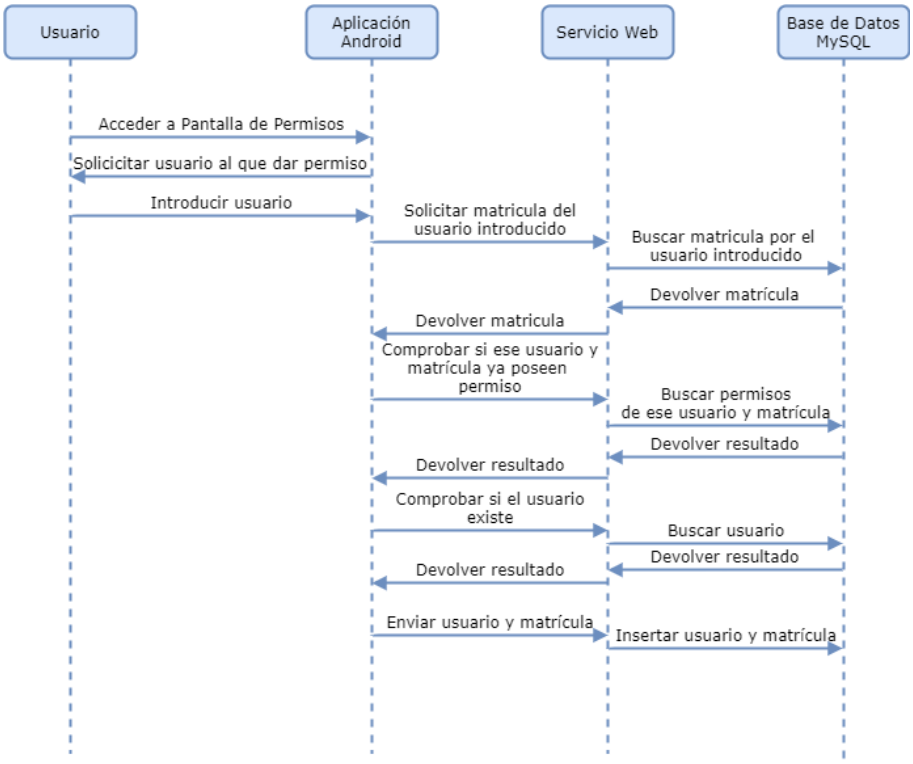


Ilustración 22: Asignar Permisos – Diagrama de Secuencia

En este diagrama de secuencia se observa la monitorización de los datos recogidos desde la aplicación Android (Ilustración 23).

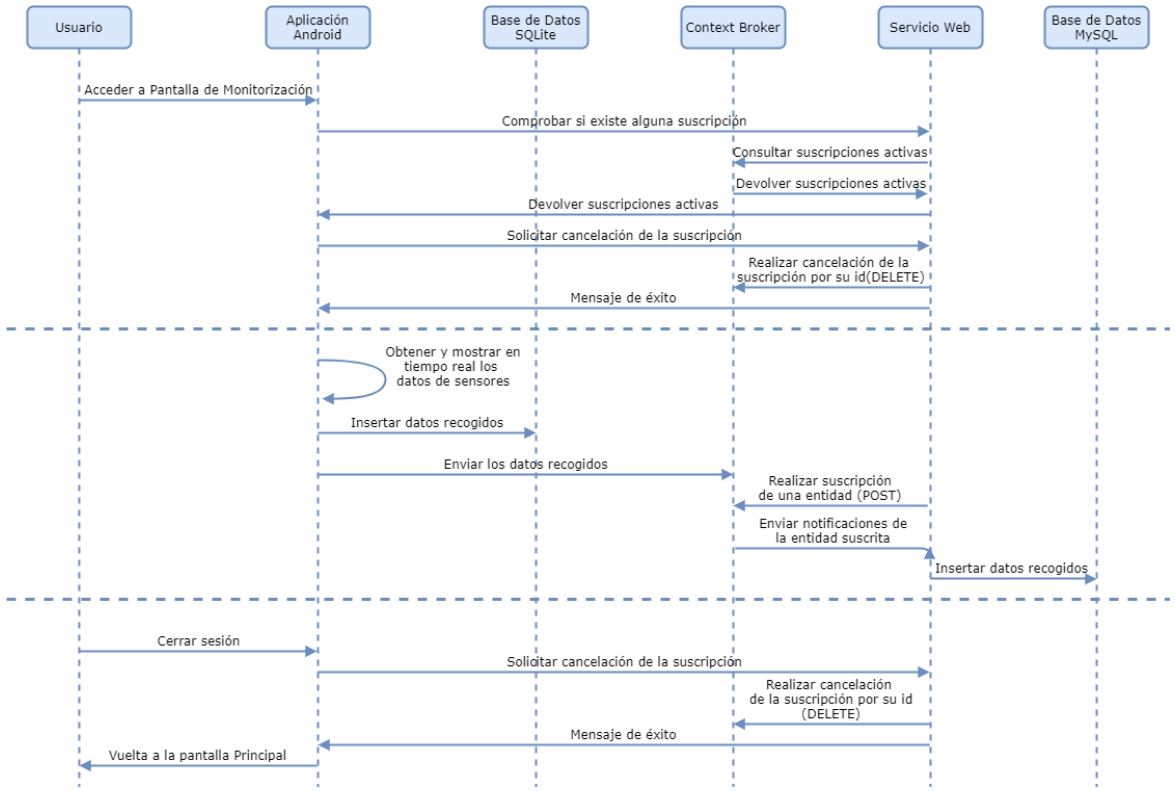


Ilustración 23: Monitorización de datos – Diagrama de Secuencia

En la Ilustración 24 se puede observar la visualización de los datos monitorizados y guardados en la base de datos MySQL.

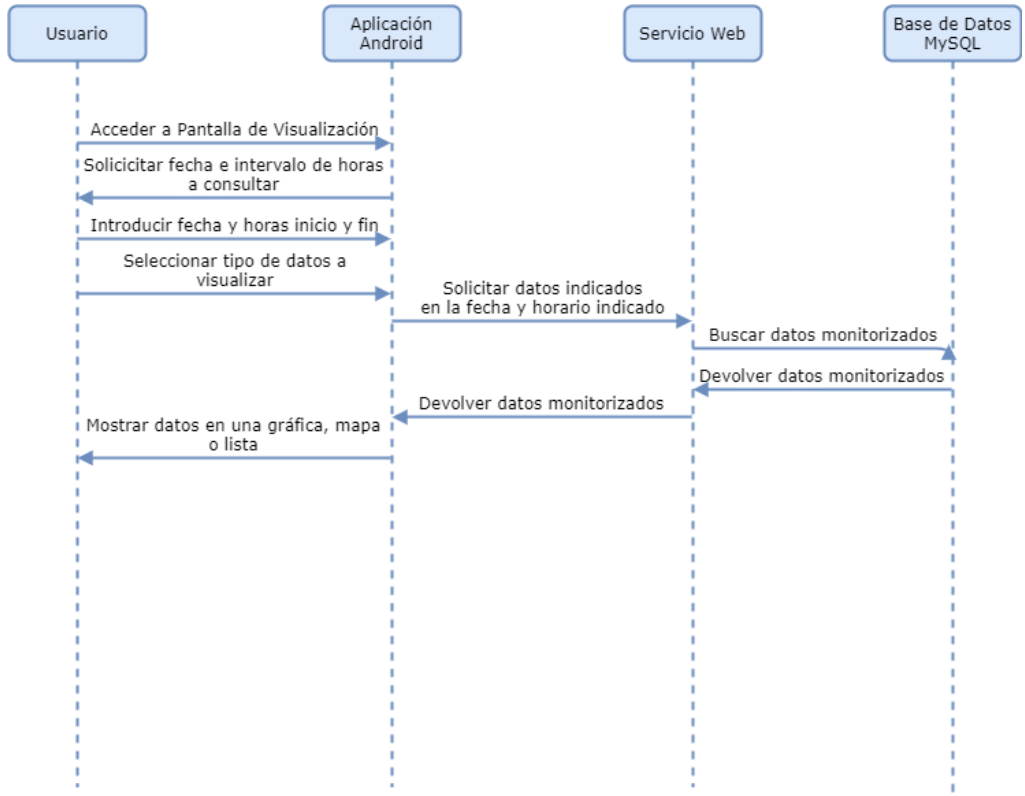


Ilustración 24: Visualización de datos – Diagrama de Secuencia

A continuación, se tratarán las partes referentes al servicio web Spring y la base de datos MySQL.

Las demás partes del sistema diseñado se tratan en el siguiente capítulo.

4.1 Clases

En este apartado se muestran cada una de las clases utilizadas en el servicio web.

Todas estas clases se encuentran en el directorio */valores/src/main/java/monitorización/valores*

Clases POJO utilizadas en el servicio web

Una clase POJO es un objeto Java común, el cual no está vinculado por ninguna restricción especial que no sea forzada por la especificación de lenguaje Java y que no requiera de ninguna ruta de clase. A través de las clases POJO se proporciona mayor legibilidad y reutilización de los programas.

Básicamente una clase POJO define una entidad, en este proyecto se pueden dividir las clases POJO en dos grupos, las que se utilizan para la comunicación con el context broker o con la base de datos MySQL.

Para la comunicación con la base de datos:

Se ha realizado una clase POJO por cada tabla de la base de datos, los atributos de estas clases deben coincidir con todos los atributos de cada una de las tablas, y además ser del mismo tipo.

- **Usuario**

  Esta clase implementa la tabla "usuarios" de la base de datos, esta clase posee 4 atributos:

  - Private String usuario: Hace referencia al nombre introducido por el usuario.
  - Private String pass: Referencia a la contraseña introducida por el usuario.
  - Private String tipo: Indica el tipo de vehículo introducido por usuario.
  - Private String matricula: Indica el valor de la matrícula del vehículo introducido por el usuario.

- **Permiso**

  Implementa la tabla "permisos" de la base de datos, posee 3 atributos:

  - Private long id: Informa el id de cada registro, es un número que indica el orden de inserción de cada registro en esta tabla de la base de datos.
  - Private String usuario: Referencia al usuario.
  - Private String matricula: Hace referencia a una matrícula de un vehículo registrado.

- **Tipo**

  Mediante esta clase se hace referencia a la tabla "tipos" de la base de datos, se compone de 2 atributos:

  - Private String tipo: Indica el tipo de vehículo.
  - Private String vmax: Indica la velocidad máxima por cada tipo de vehículo.

- **Dato**

  Esta clase implementa la tabla "datos" de la base de datos, esta clase es la que posee mayor número de atributos, con un total de 13:

  - Private long id: Informa el id de cada registro, el cual es un número que sigue el orden de inserción de cada registro en la tabla de la base de datos.
  - Private String matricula: Indica la matrícula del vehículo que se está monitorizando.
  - Private double latitud: Referencia el valor de la latitud en un momento concreto.
  - Private double longitud: Referencia el valor de la longitud en un momento concreto.
  - Private double velocidad: Muestra la velocidad del vehículo registrada en ese momento.
  - Private String fecha: La fecha en la que se ha realizado el registro.
  - Private String hora: La hora en la que se ha realizado el registro.
  - Private double acelerometro_x: Valor del eje x del acelerómetro en el momento del registro.
  - Private double acelerometro_y: Valor del eje y del acelerómetro en el momento del registro.
  - Private double acelerometro_z: Valor del eje z del acelerómetro en el momento del registro.
  - Private double giroscopio_x: Valor del eje x del giroscopio en el momento del registro.
  - Private double giroscopio_y: Valor del eje x del giroscopio en el momento del registro.
  - Private double giroscopio_z: Valor del eje x del giroscopio en el momento del registro.

Para la comunicación con el context broker:

Para recibir las notificaciones del context broker se utilizan estas clases, estas notificaciones recibidas se guardan como objetos.

Las entidades y atributos del context broker llegan al servicio web en formato JSON y mediante la anotación @JsonProperty, que relaciona el nombre de las propiedades del flujo de datos JSON a cada atributo en Java, consigue convertirlos a objetos que se permitan leer y tratar su información de forma sencilla.

Se implementan una serie de métodos para ello que siguen los estándares de las clases POJO.

- **Attribute**

  Las entidades del context broker poseen atributos, esta clase se implementa para que esos atributos pertenezcan a uno de los campos establecidos:

  - Private Metadata metadata: De tipo Metadata y hace referencia al campo metadata que aparece en el JSON procedente del context broker.
  - Private String type: Campo de tipo String que posee el tipo del dato que se guarda en el campo value.
  - Private T value: Este campo de tipo T, es el tipo genérico de Java ya que en ese momento no se conoce con exactitud el tipo de dato que se obtendrá del context broker, posee el valor del atributo.

En la Ilustración 25 se muestra la clase Attribute del servicio web, donde se pueden observar cada uno de los campos descritos anteriormente.

```java
package monitorizacion.valores;

import com.fasterxml.jackson.annotation.JsonProperty;

//Clase POJO para las notificaciones procedentes del context broker
public class Attribute<T>{

    @JsonProperty("metadata")
    private Metadata metadata;
    @JsonProperty("type")
    private String type;
    @JsonProperty("value")
    private T value;

    public Attribute(){}

    public Attribute(Metadata metadata, String type, T value){
    this.metadata=metadata;
    this.type=type;
    this.value=value;
    }

    public Metadata getMetadata(){
    return metadata;
    }
    public void setMetadata(Metadata metadata){
        this.metadata=metadata;
```

Ilustración 25: Clase POJO Attribute

- **Metadata**

  Esta clase, la cual aparece en la Ilustración 26, posee un objeto de tipo "Attribute", este tipo de objeto se ha implementado en el proyecto mediante la clase Attribute, mostrada anteriormente. Esta clase al seguir el estándar de una clase POJO, posee un getter ("getUnidades") y un setter ("setUnidades"). Además de un constructor por defecto y otro que va asignando el valor del parámetro pasado al atributo que le corresponde.

```java
package monitorizacion.valores;

import com.fasterxml.jackson.annotation.JsonProperty;

//Clase POJO para los metadatos del context broker
public class Metadata{

    @JsonProperty("unidades")
    private Attribute unidades;

    public Metadata(){}

    public Metadata(Attribute unidades){
    this.unidades=unidades;
    }

    public Attribute getUnidades(){
    return unidades;
    }
    public void setUnidades(Attribute unidades){
        this.unidades=unidades;
    }

}
```

Ilustración 26: Clase POJO Metadata

- Private Attribute unidades: En este campo se muestra la unidad usada para medir cada una de las medidas que va tomando la aplicación.

- **Notification**

  Como ya se nombró con anterioridad, desde el servicio web se reciben las notificaciones procedentes del context broker, para poderlas tratar correctamente y recibirlas como objetos se ha realizado esta clase que se muestra en la Ilustración 27, llamada "Notification".

  En esta clase se definen dos atributos, siguiendo el formato JSON que poseen las notificaciones que se reciben desde el context broker:

  - Private List<Entity> data: Es una lista de las entidades que han sido informadas en la notificación recibida.
  - Private String subscriptionId: Contiene el valor del identificador de la suscripción, es de tipo "String".

```
package monitorizacion.valores;

import java.util.List;

//Clase POJO para las notificaciones del context broker
public class Notification{

    @JsonProperty("data")
    private List<Entity> data;
    @JsonProperty("subscriptionId")
    private String subscriptionId;

    public Notification(){}

    public Notification(List<Entity> data, String subscriptionId){
    this.data=data;
    this.subscriptionId=subscriptionId;
    }


    public String getSubscriptionId(){
    return subscriptionId;
    }
    public void setSubscriptionId(String subscriptionId){
        this.subscriptionId=subscriptionId;
    }
```

Ilustración 27: Clase POJO Notification

- **Entity**

  En esta clase, mostrada en la Ilustracion 28, se tratan los atributos que poseen las entidades del context broker, es decir, tiene tantos atributos como atributos tengan las entidades. Las entidades llegan desde el context broker en formato JSON, mediante esta clase se proporciona abstracción y se definen los objetos que poseen esta información de cada entidad.

  A continuación, se muestran todos los atributos que se han definido en la clase "Entity":

  - Private String id: Indica el identificador de la entidad recibida desde el context broker, es de tipo "String".
  - Private String type: Hace referencia al tipo que aparece en la entidad recibida.
  - Private Attribute latitud: Posee el valor de la latitud en el instante medido.
  - Private Attribute longitud: Posee el valor de la longitud en el instante medido.
  - Private Attribute velocidad: Informa del valor de la velocidad en un instante concreto.
  - Private Attribute fecha: Fecha en la se ha realizado el registro de los datos en la aplicación Android.

- Private Attribute hora: Hora en la que se ha realizado el registro de los datos en la aplicación Android.
- Private Attribute acelerometro_x: Valor del eje x del acelerómetro.
- Private Attribute acelerometro_y: Valor del eje y del acelerómetro.
- Private Attribute acelerometro_z: Valor del eje z del acelerómetro.
- Private Attribute giroscopio_x: Valor del eje x del giroscopio.
- Private Attribute giroscopio_y: Valor del eje y del giroscopio.
- Private Attribute giroscopio_z: Valor del eje z del giroscopio.

```
package monitorizacion.valores;

import com.fasterxml.jackson.annotation.JsonProperty;

//Clase POJO para las entidades del context broker
public class Entity{

    @JsonProperty("id")
    private String id;
    @JsonProperty("type")
    private String type;
    @JsonProperty("latitud")
    private Attribute latitud;
    @JsonProperty("longitud")
    private Attribute longitud;
    @JsonProperty("velocidad")
    private Attribute velocidad;
    @JsonProperty("fecha")
    private Attribute fecha;
    @JsonProperty("hora")
    private Attribute hora;
    @JsonProperty("acelerometro_x")
    private Attribute acelerometro_x;
    @JsonProperty("acelerometro_y")
    private Attribute acelerometro_y;
    @JsonProperty("acelerometro_z")
    private Attribute acelerometro_z;
    @JsonProperty("giroscopio_x")
    private Attribute giroscopio_x;
    @JsonProperty("giroscopio_y")
    private Attribute giroscopio_y;
    @JsonProperty("giroscopio_z")
    private Attribute giroscopio_z;

    public Entity(){}

    public Entity(Attribute fecha, Attribute hora, String
```

Ilustración 28: Clase POJO Entity

Clases utilizadas para controlar el funcionamiento del servicio web

- **Application**

  Esta es una clase de configuración que declara un método "main", como se puede ver en la Ilustración 29, que es el principal de la aplicación, el que se ejecuta en primer lugar nada más ejecutar la aplicación, llamando a la clase que contiene la anotación @RestController, que se encuentra en el controlador ("SensorController") y equivale a la combinación de las anotaciones @Controller y @ResponseBody.

  Se utiliza la anotación @SpringBootApplication que es una combinación de las anotaciones @Configuration, @ComponentScan y @EnableAutoConfiguration, de esta manera se proporciona la misma funcionalidad de estas tres anotaciones, pero con una sola línea de código. Esto es posible a partir de la versión Spring 1.2 en adelante.

```
package monitorizacion.valores;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
    }
}
```

Ilustración 29: Clase Application

- **SensorController**

  Esta es la clase que controla el servicio web, se puede observar que en ella aparece la anotación @RestController que es la combinación de las anotaciones @Controller y @ResponseBody. Los métodos de esta clase implementan todas las funcionalidades posibles del servicio web.

  La anotación @RestController permite crear controladores Restful y se detecta automáticamente a través del escaneo de classpath.

  @RestController simplemente devuelve el objeto y los datos del objeto se escriben directamente en la respuesta como JSON o XML.

  Además, también utiliza las anotaciones @RequestMapping y @PostMapping para asociar una url al controlador.

  Para el caso de @RequestMapping se especifica el método HTTP (GET, POST, …) que se asocia al método java.

  Al comienzo de la clase se obtienen las direcciones y puertos del context broker y la propia aplicación. Mediante la anotación @Value se obtienen sus valores del fichero *aplication.properties.* Estos dos valores serán imprescindibles para poder administrar los distintos mensajes de las suscripciones al context broker.

  - Private String ip_port_webservice: Contiene la dirección IP y Puerto donde se ejecutará el servicio web.
  - Private String ip_port_contextbroker: Posee la dirección IP y Puerto del context broker.

  El método que sigue la notación @PostMapping es el encargado de recibir las notificaciones del context broker.

  - Public void notificaciones: Este método, mostrado en la Ilustración 30, recibe las notificaciones del context broker en formato JSON y las transforma a un objeto "Notification". Crea una instancia a la clase "Database" y utilizando su método "insertarDatos" se van insertando los datos de las notificaciones recibidas en la base de datos.

```
//Recibe notificaciones del context broker e inserta datos en la base de datos
@PostMapping("/sensor/notificaciones")
public void notificaciones(
  @RequestBody Notification notificacion){

    //Introducir los datos de la notificación en la base de datos
    if (notificacion.getData().iterator().hasNext()) {

        DataBase bd= new DataBase();
        bd.insertarDatos(notificacion);
    }
}
```

Ilustración 30: Método notificaciones de la clase SensorController

Los métodos que siguen la notación @RequestMapping son los encargados de gestionar las suscripciones con el context broker y consultar los datos.

En primer lugar, trataremos los que se encargan de gestionar las suscripciones.

- Public void registro: Permite registrar un nuevo usuario. Para ello es necesario conocer el valor de los campos usuario, pass, tipo y matricula introducidos desde la pantalla de nuevo registro de la aplicación Android. Se inserta un nuevo registro con esos datos en la tabla "permisos" de la base de datos.
- Public String suscribir: Este método se encarga de suscribir un vehículo registrado al context broker para que puedan recibir notificaciones cada vez que cambie el valor del campo "hora". Se le pasa como parámetro la matrícula del vehículo para identificar las entidades con ese identificador desde el mensaje de suscripción. Devuelve un mensaje de éxito o error al finalizar.
- Public String suscripciones: Permite consultar las suscripciones existentes en el context broker. Para ello se utiliza la petición GET. No recibe argumentos y devuelve, en formato JSON, todas las suscripciones.
- Public String cancelar_suscripcion: Este método se encarga de cancelar una suscripción a partir de su id. Para ello se utiliza una petición DELETE. Recibe el id de la suscripción como argumento de entrada. Devuelve un mensaje de éxito o error en formato string.

A continuación, se describen todos los métodos con la anotación @RequestMapping que permiten insertar o consultar los datos existentes.

- Public void permisos: Este método se encarga de insertar un usuario y su matrícula en la tabla "permisos" de la base de datos. Esta nueva entrada en la tabla "permisos" permite que el usuario indicado pueda visualizar los datos que ha monitorizado el vehículo que posee la matrícula con la que se ha asociado. Recibe el usuario y la matrícula como argumentos de entrada. No devuelve nada.
- Public void eliminarpermisos: Este método permite eliminar un usuario y su matricula asociada en la tabla "permisos" de la base de datos. Así, al usuario indicado no se le permite visualizar los datos monitorizados por el vehículo que posee la matrícula que tenía asociada. Recibe como argumentos de entrada el usuario y la matrícula y no devuelve nada.
- Public String buscarmatricula: Permite consultar la matrícula de un usuario en la tabla "usuarios" de la base de datos. Recibe como parámetro de entrada el usuario que usará para obtener la matrícula asociada a él en la tabla "usuarios" de la base de datos. Devuelve la matrícula obtenida de la consulta.
- Public String [] buscarmatriculasasociadas: Permite consultar las matrículas asociadas a un usuario en la tabla "permisos" de la base de datos. Recibe el usuario como parámetro de entrada y devuelve una matriz de cadenas con las matrículas asociadas a ese usuario.

- Public List<Usuario> usuario: Este método es el encargado de consultar si existen el usuario y contraseña en la tabla "usuarios" de la base de datos. Recibo como parámetros de entrada el usuario y la contraseña. Devuelve una lista de objetos de tipo "Usuario".
- Public List<Permiso> rol: Permite consultar la tabla "permisos" de la base de datos para conocer si existe la asociación entre un usuario y matrícula pasados por parámetros al método. Devuelve una lista de objetos de tipo "Permiso".
- Public String [] buscarhora: Método para consultar la hora en la tabla "datos" de la base de datos. Recibe como argumentos de entrada la fecha, la matrícula del vehículo y las horas de inicio y fin que establecen el intervalo a buscar. Devuelve una matriz de cadenas con los valores de todas las horas en las que existen registros dentro del intervalo de tiempo indicado.
- Public String [] buscarvelocidad: Método encargado de consultar la velocidad en la tabla "datos" de la base de datos. Recibe como argumentos de entrada la fecha, la matrícula del vehículo y las horas de inicio y fin que establecen el intervalo a buscar. Devuelve una matriz de cadenas con los valores de las velocidades de cada registro existente dentro del intervalo de tiempo indicado.
- Public String [] buscarlatitud: Permite consultar la latitud en la tabla "datos" de la base de datos. Recibe como argumentos de entrada la fecha, la matrícula del vehículo y las horas de inicio y fin que establecen el intervalo a buscar. Devuelve una matriz de cadenas con todos los valores de las latitudes dentro del período de tiempo indicado.
- Public String [] buscarlongitud: Método encargado de consultar la longitud en la tabla "datos" de la base de datos. Recibe como argumentos de entrada la fecha, la matrícula del vehículo y las horas de inicio y fin que establecen el intervalo a buscar. Devuelve una matriz de cadenas con los valores de las longitudes dentro del período de tiempo indicado.
- Public String [] buscaracelerometro: Permite consultar los valores del acelerómetro en la tabla "datos" de la base de datos. Se consultan los valores pertenecientes a los ejes x, y, z del acelerómetro. Recibe como argumentos de entrada la fecha, la matrícula del vehículo y las horas de inicio y fin que establecen el intervalo a buscar. Devuelve una matriz de cadenas con los valores del acelerómetro de cada registro existente dentro del intervalo de tiempo indicado.
- Public String [] buscargiroscopio: Permite consultar los valores del giroscopio en la tabla "datos" de la base de datos. Se consultan los valores pertenecientes a los ejes x, y, z del giroscopio. Recibe como argumentos de entrada la fecha, la matrícula del vehículo y las horas de inicio y fin que establecen el intervalo a buscar. Devuelve una matriz de cadenas con los valores del giroscopio dentro del intervalo de tiempo indicado.
- Public String buscarTipo: Este método se encarga de consultar el tipo de vehículo de un determinado usuario en la tabla "usuarios" de la base de datos. Recibe como argumento de entrada el usuario. Devuelve una cadena con el valor del tipo de vehículo asociado a ese usuario.
- Public String buscarvmax: Este método permite consultar la velocidad máxima permitida para un tipo de vehículo. Se le pasa como parámetro de entrada el tipo de vehículo y devuelve una cadena con la velocidad máxima asociada a ese tipo de vehículo, según se encuentra establecido en la tabla "tipos" de la base de datos.

- **Database**

  El propósito de esta clase es realizar las transacciones del servicio web con la base de datos, para ello se utiliza Hibernate. Contiene métodos para insertar los datos recibidos de cada notificación del context broker, eliminar y consultar los valores de la base de datos.

  En primer lugar, se describen los métodos que se encargar de insertar y eliminar.

  - Public void insertarDatos: Método que se encarga de insertar los datos en la tabla "datos" de la base de datos "monitorizacion". Recibe como argumento de entrada un objeto de tipo

"Notification" con los datos de la notificación recibida del context broker. No devuelve nada.

- Public void insertarUsuario: Este método permite insertar un registro de nuevo usuario en la tabla "usuarios" de la base de datos "monitorizacion". Recibe como argumentos de entrada 4 cadenas que son el usuario, la contraseña, el tipo de vehículo y la matrícula. No devuelve nada.
- Public void insertarPermiso: Se encarga de insertar una nueva entrada en la tabla "permisos" de la base de datos para dar permisos de visualización a un usuario sobre los datos de monitorización de un vehículo. Recibe como parámetros de entrada dos cadenas, una con el usuario y otra con la matrícula. No devuelve nada.
- Public void eliminarPermiso: Este método se encarga de eliminar una entrada de la tabla "permisos" de la base de datos. Recibe como el usuario y la matrícula como parámetros de entrada y no devuelve nada.

A continuación, se describen los métodos encargados de las consultas a la base de datos mediante consultas dinámicas gracias al uso de Hibernate en el servicio web.

Acompaña a Hibernate una API de consultas por criterios que será aplicada en todos estos métodos. Además en la clase se importa "org.hibernate.criterion.Restrictions", que permite limitar el conjunto de resultado mediante la inclusión de restricciones de forma dinámica.

- Public List<Usuario> consultarPorUsuarioMatricula: Este método se encarga de realizar las consultas a la tabla "usuarios" de la base de datos filtrando por el usuario y contraseña que son pasados como argumentos de entrada al método. Realiza una consulta dinámica según las condiciones que se le indiquen, en este caso las condiciones son que coincida con los dos argumentos pasados al método. Devuelve una lista de objetos de tipo "Usuario".
- Public List<Permiso> consultarPermiso: Se encarga de realizar consultas a la tabla "permisos" de la base de datos "monitorizacion" aplicando las condiciones dinámicas indicadas en el método. Recibe dos cadenas, una con el usuario y otra con la matrícula. Devuelve una lista de objetos de tipo "Permiso".
- Public List<Permiso> consultarMatriculas: Este método se encarga de realizar una consulta a la tabla "permisos" de la base de datos con la condición de que coincida el usuario. Recibe como argumento de entrada el usuario y devuelve una lista de objetos de tipo "Permiso".
- Public List<Usuario> consultarPorUsuario: Este método es el encargado de consultar a la tabla "usuarios" de la base de datos con la condición dinámica de que coincida el usuario. Recibe el usuario como argumento de entrada y devuelve una lista de objetos de tipo "Usuario".
- Public List<Dato> consultarDatos: Este método es el encargado de realizar las consultas a la tabla "datos" de la base de datos filtrando dinámicamente por los argumentos de entrada al método, los cuales son la fecha, matrícula, hora de inicio y hora de fin. Devuelve una lista de objetos de tipo "Dato".
- Public String consultarVmax: Realiza las consultas a la tabla "tipos" de la base de datos para obtener la velocidad máxima. A la consulta se le aplica el tipo de vehículo, pasado como parámetro de entrada, como filtro dinámico. Devuelve una cadena con el valor de la velocidad máxima.

4.2 Ficheros de configuración

En este apartado se van a tratar todos los ficheros de configuración que posee el servicio web para su correcto funcionamiento. Estos ficheros se dividirán en dos grupos para su análisis. Todos estos ficheros se sitúan en el directorio *valores/src/main/resources.*

Ficheros de configuración del servicio web

- application.properties: Este es el fichero de configuración del servicio web. En ese fichero se definen una serie de propiedades para que posteriormente se puedan acceder a esos valores desde cualquier clase. En este caso, como se puede observar en la Ilustración 31, se define:
  - ip_port_webservice: Esta propiedad hace referencia a la dirección ip y al puerto donde se ejecuta el servicio web.
  - ip_port_contextbroker: En esta otra propiedad se indica la dirección ip y el puerto donde se está ejecutando el context broker.

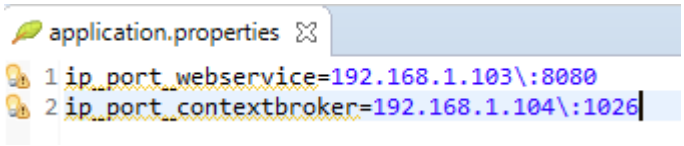


Ilustración 31: Fichero de configuración application.properties

Ficheros de configuración de hibernate

- hibernate.cfg.xml: Se utiliza este fichero, el cual se muestra en la Ilustración 32, para configurar hibernate. Es un fichero XML. Este fichero debe situarse en el paquete raíz de todas las clases del proyecto. Contiene las propiedades de configuración y las clases que se quieren mapear. Se indica el lenguaje SQL que se ha empleado, el nombre de usuario y contraseña del propietario de la base de datos y el driver a utilizar. También se configura la dirección IP y puerto donde se ejecuta el servidor de la base de datos y la ubicación de los ficheros de mapeo de las tablas, asociándolas con sus clases POJO.

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">1234</property>
        <property name="hibernate.connection.url">jdbc:mysql://192.168.1.103/monitorizacion</property>

        <mapping resource="Dato.hbm.xml"></mapping>
        <mapping resource="Usuario.hbm.xml"></mapping>
        <mapping resource="Permisos.hbm.xml"></mapping>
        <mapping resource="Tipos.hbm.xml"></mapping>

    </session-factory>
</hibernate-configuration>
```

Ilustración 32: Fichero de configuración hibernate.cfg.xml

A continuación, se muestran los ficheros encargados del mapeo por cada tabla de la base de datos.

- Usuario.hbm.xml: Este fichero se muestra en la Ilustración 33 y realiza el mapeo de la tabla "usuarios" de la base de datos con la clase "Usuario" y cada uno de sus atributos con las columnas de la tabla anteriormente indicada.

Usuario.hbm.xml

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 03-feb-2019 13:01:42 by Hibernate Tools 3.5.0.Final -->
<hibernate-mapping>
    <class name="monitorizacion.valores.Usuario" table="usuarios">

        <id name="usuario" column="usuario" type="string"/>
        <property name="pass" column="pass" type="string"/>
        <property name="tipo" column="tipo" type="string"/>
        <property name="matricula" column="matricula" type="string"/>
    </class>
</hibernate-mapping>
```

Ilustración 33: Fichero Usuario.hbm.xml

- Permiso.hbm.xml: En este fichero se mapea la tabla "permisos" de la base de datos con la clase "Permiso" y todos sus atributos con las columnas de la tabla.
- Tipo.hbm.xml: Este fichero realiza el mapeo de la tabla "tipos" de la base de datos con la clase "Tipo" y todos sus atributos con cada una de las columnas de la tabla indicada.
- Dato.hbm.xml: En este fichero, mostrado en la Ilustración 34, se mapea la tabla "datos" de la base de datos con la clase "Dato", además de asociar sus atributos con las columnas de esa tabla. Se establece un "id" como primer campo, este campo se mapea como un generador de secuencia, de ese modo va aumentado su valor con cada nuevo registro.

Dato.hbm.xml

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 03-feb-2019 13:01:42 by Hibernate Tools 3.5.0.Final -->
<hibernate-mapping>
    <class name="monitorizacion.valores.Dato" table="datos">
        <id name="id" column="id" type="long">
        <generator class="increment"/>
        </id>

        <property name="matricula" column="matricula" type="string"/>
        <property name="latitud" column="latitud" type="double"/>
        <property name="longitud" column="longitud" type="double"/>
        <property name="velocidad" column="velocidad" type="double"/>
        <property name="fecha" column="fecha" type="string"/>
        <property name="hora" column="hora" type="string"/>
        <property name="acelerometro_x" column="acelerometro_x" type="double"/>
        <property name="acelerometro_y" column="acelerometro_y" type="double"/>
        <property name="acelerometro_z" column="acelerometro_z" type="double"/>
        <property name="giroscopio_x" column="giroscopio_x" type="double"/>
        <property name="giroscopio_y" column="giroscopio_y" type="double"/>
        <property name="giroscopio_z" column="giroscopio_z" type="double"/>
    </class>
</hibernate-mapping>
```

Ilustración 34: Fichero Dato.hbm.xml

4.3 Servidor de Base de Datos

La base de datos diseñada para este proyecto se compone de cuatro tablas, cada una con sus campos correspondientes y las relaciones necesarias entre ellas.

Se ha creado un fichero *creaTablas.sql* que en primer lugar hace un borrado de la base de datos y las cuatro tablas, en el caso de que existieran con anterioridad, y posteriormente crea la base de datos "monitorizacion" y cada una de las tablas y sus campos asignando las claves primarias y externas que correspondan entre los campos indicados.

En la Ilustración 35 se muestra un modelo Entidad-Relación con cada una de las tablas para que se pueda entender el diseño de la base de datos con más claridad.

Se procede a describir cada una de las tablas mostradas en el modelo Entidad-Relación.

- Tabla "usuarios": Almacena todos los usuarios que se han registrado mediante la aplicación Android. Formada por cuatro columnas de tipo VARCHAR.
    - usuario: Nombre del usuario que se registra en el sistema y mediante el cual se puede logear en la aplicación Android. Es la clave primaria de la tabla y es de tipo VARCHAR.
    - pass: Contraseña asociada al usuario registrado. La utiliza junto al nombre de usuario para poder logearse en la pantalla principal de la aplicación Android. Es de tipo VARCHAR.
    - tipo: Informa del tipo de vehículo asociado al usuario. Se trata de una clave externa que referencia a la columna "tipo" de la tabla "tipos". Esta columna es de tipo VARCHAR.
    - matricula: Contiene la matrícula del vehículo del usuario. Esta columna es de tipo VARCHAR.

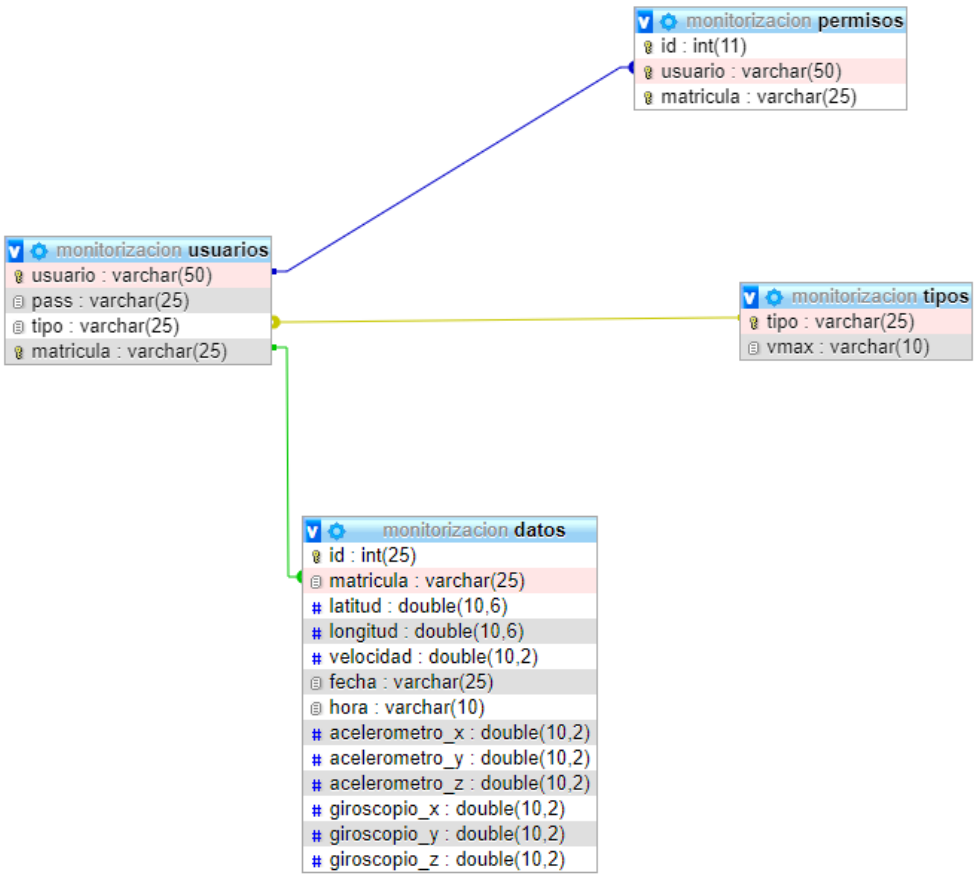


Ilustración 35: Modelo Entidad-Relación de la Base de Datos

- Tabla “permisos”: Contiene todos los valores de la asociación entre usuario y matrícula. Aparecen todos los usuarios y matrículas que se encuentran relacionados entre sí, para que dicho usuario tenga permisos de visualización sobre los datos registrados de la matrícula a la que se encuentre asociada en esta tabla. Esta tabla consta de tres columnas.
    - Id: Identificador de cada registro añadido a esta tabla. Se trata de una secuencia numérica que se va autoincrementando por cada nueva fila que sea añadida en la tabla. Es clave primaria y de tipo INTEGER.
    - usuario: Almacena todos los usuarios y los que poseen permisos de visualización sobre otras matrculas que no son la suya. Es clave primaria y clave externa que referencia a la columna “usuario” de la tabla “usuarios”. Se trata de un campo de tipo VARCHAR.
    - matricula: Informa de las matrículas que se encuentran asociadas a determinados usuarios para la visualización de sus datos recogidos. Es clave primaria y de tipo VARCHAR.

- Tabla “tipos”: Almacena los distintos tipos de vehículos disponibles para registrar en la aplicación y la velocidad máxima asociada a cada tipo. Tabla formada por 2 columnas.
    - tipo: Contiene todos los tipos existentes de vehículos para su selección desde la aplicación. Este campo es la clave primaria de la tabla y es de tipo VARCHAR.
    - vmax: Este campo proporciona los valores de la velocidad máxima asociada a cada tipo de vehículo. Es de tipo VARCHAR.

  Los valores de velocidad máxima asociados a cada tipo de vehículo, se establecen en el fichero *tipos.txt* y se copian a esta tabla. Estos valores se han obtenido de la página de la ITV que tiene como fuente a la DGT [13].

- Tabla “datos”: Esta tabla se encarga de almacenar los valores de todos los datos monitorizados a través de la aplicación mediante los sensores del dispositivo móvil, en la Ilustración 36 se muestran algunos valores introducidos de ejemplo. Se encuentra formada por 13 columnas.
    - id: Identificador de cada registro añadido en esta tabla. Es una secuencia numérica que se va autoincrementando por cada fila añadida. Es la clave primaria de la tabla. Este campo es de tipo INTEGER.
    - matricula: Posee el número de matrícula del vehículo que se está monitorizando o visualizando. Se trata de una clave externa que referencia a la columna “matricula” de la tabla “usuarios”. Es de tipo VARCHAR.
    - latitud: Contiene el valor de la latitud. Es de tipo DOUBLE.
    - longitud: Contiene el valor de longitud. Este campo es de tipo DOUBLE.
    - velocidad: Informa de la velocidad a la que circula el vehículo en cada registro. Es de tipo DOUBLE.
    - fecha: Almacena la fecha en la que se ha realizado la recogida de datos desde los sensores del dispositivo móvil. Es de tipo VARCHAR.
    - hora: Informa de la hora en la que se ha realizado la obtención de los datos desde los sensores del dispotivo móvil. Este campo es de tipo VARCHAR.
    - acelerometro_x: Contiene el valor del eje x del acelerómetro. Es de tipo DOUBLE.
    - acelerometro_y: Contiene el valor del eje y del acelerómetro. Es de tipo DOUBLE.
    - acelerometro_z: Contiene el valor del eje z del acelerómetro. Es de tipo DOUBLE.
    - giroscopio_x: Informa del valor del eje x del acelerómetro: Es de tipo DOUBLE.

- giroscopio_y: Informa del valor del eje y del acelerómetro: Es de tipo DOUBLE.
- giroscopio_z: Informa del valor del eje z del acelerómetro: Es de tipo DOUBLE.

id	matricula	latitud	longitud	velocidad	fecha	hora	acelerometro_x	acelerometro_y	acelerometro_z	giroscopio_x	giroscopio_y	giroscopio_z
1	123	36.728954	-6.434355	0.00	16/03/2020	01:05	0.01	0.01	10.31	0.01	0.42	10.31
2	123	36.728954	-6.434355	0.00	16/03/2020	01:06	0.04	0.04	10.33	0.04	0.41	10.33
3	123	36.728954	-6.434355	0.00	16/03/2020	01:07	0.00	0.00	10.29	0.00	0.42	10.29
4	123	36.728954	-6.434355	0.00	16/03/2020	01:08	0.03	0.03	10.38	0.03	0.45	10.38

Ilustración 36: Ejemplo de datos almacenados en la tabla *datos*

4.4 API Rest

En este apartado se analiza la API Rest utilizada. Los URI y métodos utilizados para hacer uso de las funcionalidades que se han implementado en el servicio web.

Para formar los URI en este documento se han utilizado los corchetes "[]" para indicar que el elemento que aparezca en su interior será sustituido por el nombre o el valor del parámetro indicado.

A continuación, se dividirán en dos grupos principales uno para los métodos GET (Tabla 1 y 2) y otro para los POST (Tabla 3). A su vez el primer grupo se dividirán en otros dos subgrupos para diferenciar más claramente la funcionalidad de cada una de las consultas.

4.4.1 Consultas utilizando el método GET

Dentro de este grupo se hará la diferencia entre las consultas de los datos y la gestión de las suscripciones y notificaciones:

- **Consulta de los datos**

URI	Parámetros	Descripción
/sensor/registro?[parámetro1] =[valor1] &[parámetro2]=[valor2] &[parámetro3]=[valor3] &[parámetro4]=[valor4]	"usuario": Nombre del usuario. "pass": Contraseña asociada al usuario. "tipo": Tipo de vehículo "matricula": Matrícula del vehículo.	Insertar en la tabla "usuarios" de la base de datos un nuevo usuario registrado.
/sensor/permisos?[parámetro1] =[valor1] &[parámetro2]=[valor2]	"usuario": Nombre del usuario. "matricula": Matrícula del vehículo.	Insertar en la tabla "permisos" de la base de datos un nuevo registro que indica la asociación entre usuario y matrícula. Para permitir a ese usuario visualizar los datos de la matrícula.

/sensor/buscarmatricula?[parámetro] =[valor]	"usuario": Nombre del usuario.	Consultar la matrícula de un usuario en la tabla "usuarios" de la base de datos.
/sensor/login?[parámetro1]=[valor1]& [parámetro2]=[valor2]	"usuario": Nombre del usuario. "pass": Contraseña asociada al usuario.	Consultar en la tabla "usuarios" de la base de datos si existe una entrada de ese usuario con esa matrícula.
/sensor/rol?[parámetro1] =[valor1]]&[parámetro2]=[valor2]	"usuario": Nombre del usuario. "matricula": Matrícula del vehículo.	Consultar en la tabla "permisos" de la base de datos si existe una entrada de ese usuario asociado a esa matrícula.
/sensor/buscarhora?[parámetro1] =[valor1] &[parámetro2]=[valor2] &[parámetro3]=[valor3] &[parámetro4]=[valor4]	"fecha": Fecha en la que se ha realizado el registro. "matricula": Matrícula del vehículo. "horaInicio": Campo "hora" y marca el inicio del intervalo de tiempo. "horaFin": Campo "hora" y marca el fin del intervalo de tiempo.	Consultar en la tabla "datos" de la base de datos las horas en las que existen registros dentro del intervalo indicado.
/sensor/buscarvelocidad?[parámetro1] =[valor1] &[parámetro2]=[valor2] &[parámetro3]=[valor3] &[parámetro4]=[valor4]	"fecha": Fecha en la que se ha realizado el registro. "matricula": Matrícula del vehículo. "horaInicio": Campo "hora" y marca el inicio del intervalo de tiempo. "horaFin": Campo "hora" y marca el fin del intervalo de tiempo.	Consultar en la tabla "datos" de la base de datos las velocidades en el intervalo de horas indicado.
/sensor/buscarlatitud?[parámetro1] =[valor1] &[parámetro2]=[valor2] &[parámetro3]=[valor3] &[parámetro4]=[valor4]	"fecha": Fecha en la que se ha realizado el registro. "matricula": Matrícula del vehículo. "horaInicio": Campo "hora" y marca el inicio del intervalo de tiempo. "horaFin": Campo "hora" y marca el fin del intervalo de tiempo.	Consultar en la tabla "datos" de la base de datos los valores de la latitud en el intervalo de horas indicado.
/sensor/buscarlongitud?[parámetro1] =[valor1] &[parámetro2]=[valor2] &[parámetro3]=[valor3] &[parámetro4]=[valor4]	"fecha": Fecha del registro. "matricula": Matrícula del vehículo. "horaInicio": Campo "hora" y marca el inicio del intervalo de tiempo. "horaFin": Campo "hora" y marca el fin del intervalo de tiempo.	Consultar en la tabla "datos" de la base de datos los valores de la longitud en el intervalo de horas indicado.

/sensor/buscaracelerometro?[parámetro1] =[valor1] &[parámetro2]=[valor2] &[parámetro3]=[valor3] &[parámetro4]=[valor4]	"fecha": Fecha en la que se ha realizado el registro. "matricula": Matrícula del vehículo. "horaInicio": Campo "hora" y marca el inicio del intervalo de tiempo. "horaFin": Campo "hora" y marca el fin del intervalo de tiempo.	Consultar en la tabla "datos" de la base de datos los valores de los ejes x, y, z del acelerómetro en el intervalo de horas indicado.
/sensor/buscargiroscopio?[parámetro1] =[valor1] &[parámetro2]=[valor2] &[parámetro3]=[valor3] &[parámetro4]=[valor4]	"fecha": Fecha en la que se ha realizado el registro. "matricula": Matrícula del vehículo. "horaInicio": Campo "hora" y marca el inicio del intervalo de tiempo. "horaFin": Campo "hora" y marca el fin del intervalo de tiempo.	Consultar en la tabla "datos" de la base de datos los valores de los ejes x, y, z del giroscopio en el intervalo de horas indicado.
/sensor/buscartipo?[parámetro] =[valor]	"usuario": Nombre del usuario.	Consultar en la tabla "usuarios" de la base de datos el tipo del vehículo asociado al usuario indicado.
/sensor/buscarvmax?[parámetro] =[valor]	"tipo": Tipo de vehículo	Consultar en la tabla "tipos" de la base de datos la velocidad máxima asociada al tipo de vehículo indicado.
/sensor/eliminarpermisos?[parámetro1] =[valor1] &[parámetro2]=[valor2]	"usuario": Nombre del usuario. "matricula": Matrícula del vehículo.	Elimina de la tabla "permisos" de la base de datos el registro que contiene el usuario y contraseña indicados. Para no permitir a ese usuario visualizar los datos de la matrícula.
/sensor/buscarmatriculasasociadas?[parámetro] =[valor]	"usuario": Nombre del usuario.	Consultar en la tabla "permisos" de la base de datos todas las matrículas que tiene asociadas el usuario indicado.

Tabla 1: API Rest – Consultas de datos mediante GET

- **Gestión de las suscripciones**

URI	Parámetros	Descripción
/sensor/cancelar_suscripcion?[parámetro] =[valor]	"subscriptionId": Se le asigna el valor del "id" de la suscripción existente.	Cancela una suscripción que exista en el context broker, a través del id.
/sensor/suscribir?[parámetro] =[valor]	"matricula": Matrícula del vehículo.	Realiza una suscripción utilizando la matrícula como identificador de la entidad a suscribirse.
/sensor/suscripciones?[parámetro] =[valor]	No Aplica	Consulta las suscripciones existentes en el contex broker.

Tabla 2: API Rest – Gestión de suscripciones mediante GET

4.4.2 Consultas utilizando el método POST

Dentro de este grupo solo se tratarán las consultas referentes a la gestión de las notificaciones.

URI	Parámetros	Descripción
/sensor/notificaciones	No Aplica	Recibe las notificaciones del context broker

Tabla 3: Gestión de notificaciones mediante POST

5 APLICACIÓN ANDROID MYSPEED GO Y CONEXIÓN CON EL CONTEXT BROKER

Poco conocimiento hace que las personas se sientan orgullosas. Mucho conocimiento, que se sientan humildes.

- Leonardo da Vinci -

En este capítulo se describen las otras dos partes del proyecto, el Context Broker y la Aplicación Android.

Se ha utilizado Orion Context Broker, el cual pertenece a una plataforma desarrollada para IoT llamada FIWARE. Aporta una función fundamental en cualquier sistema inteligente, administra, consulta y actualiza información de contexto.

5.1 Context Broker

Se utiliza Orion Context Broker en una máquina virtual VMWare con el sistema operativo CentOS 7.

Este context broker guarda sus datos en formato JSON, por lo tanto, todos los datos que recibe de la aplicación móvil desarrollada en Android están en ese formato, para que puedan ser tratados correctamente.

Se ha configurado para que se acceda a través de la siguiente dirección *http://localhost:1026/v2/.*

Una vez lanzado el context broker se pueden consultar las entidades creadas y las suscripciones existentes.

Para visualizar las entidades se utiliza la sección */v2/entities* y para las suscripciones */v2/subscriptions*.

5.1.1 Entidades

Orion Context Broker comienza en un estado vacío, es decir no posee entidades.

Todos los datos recogidos por los sensores del Smartphone Sony Xperia SP referentes al vehículo, se envían al Orion Context Broker, este los almacena y clasifica en distintas entidades. Las entidades están constituidas por atributos, que representan los datos que se pueden medir en un vehículo.

Orion Context Broker utiliza MongoDB como base de datos para almacenar el estado actual de las entidades, pero no la información histórica de sus cambios.

Cada una de las entidades poseen un EntityId y un EntityType único para poder identificar la entidad de la que se trata. Además, también posee los campos de fecha, hora, velocidad, latitud, longitud, y los valores de los

ejes del acelerómero y giroscopio. En todos estos últimos campos se especifican el tipo, valor y metadata.

Cada entidad posee un identificador único "id" del objeto, su valor es una cadena que contiene el identificador de la identidad. También existe la propiedad "Type" donde se especifica el tipo de identidad del objeto, su valor es una cadena que contiene el nombre del tipo de la entidad.

Los atributos de las entidades se indican mediante propiedades adicionales, como la propiedad "name", cuyo valor es una cadena que posee el nombre del atributo. Su valor se identifica mediante la propiedad "value" donde cualquier tipo JSON puede ser su valor.

Existe una restricción que indica la imposibilidad de utilizar "id" y "type" como nombres de atributos.

Se utiliza la API Restful NGSI v2 [14], mediante la cual se usa un cliente REST para enviar peticiones HTTP.

Dicha API necesita especificar:

- El método HTTP (GET, POST, PUT, DELETE, OPTIONS, HEAD, TRACE, CONNECT).
- La URL, la cual incluye la IP del contex broker y la operación que se desea realizar, por ejemplo *http://localhost:1026/v2/entities*
- Cabecera, indicada en la Ilustración 37.

```
Content-Type: "application/json"
Accept: application/json
```

Ilustración 37: Cabecera de petición HTTP

- Cuerpo de la petición.

En el sistema diseñado las entidades poseen 13 atributos, los cuales son:

- id: Identificador único para cada vehículo, posee el valor de la matrícula. Junto al valor del atributo "type" identifican a la entidad.
- type: Tipo de cada entidad, posee el valor del tipo del vehículo que se está monitorizando (coche, camión, …). Junto al valor del atributo "id" identifican a la entidad.
- acelerometro_x: Valor del eje x del acelerómetro, mide la tasa de cambio de la velocidad de un objeto en ese eje.
- acelerometro_y: Valor del eje y del acelerómetro, mide la tasa de cambio de la velocidad de un objeto en ese eje.
- acelerometro_z: Valor del eje z del acelerómetro, mide la tasa de cambio de la velocidad de un objeto en ese eje.
- fecha: Fecha en la que se han monitorizado los datos, en formato dd/mm/aaaa.
- giroscopio_x: Valor del eje x del giroscopio, mide la tasa de cambio de la velocidad de un objeto en ese eje.
- giroscopio_y: Valor del eje y del giroscopio, mide la tasa de cambio de la velocidad de un objeto en ese eje.
- giroscopio_z: Valor del eje z del giroscopio, mide la tasa de cambio de la velocidad de un objeto en ese eje.

- hora: Hora en la que se han monitorizado los datos, en formato hh:mm.
- latitud: Latitud, valor en grados decimales.
- longitud: Longitud, valor en grados decimales.
- velocidad: Velocidad en kilómetros por hora del vehículo a monitorizar.

Estos atributos, menos en el caso de "id" y "type", tienen una serie de valores en tres campos diferentes:

- value: Valor del atributo.
- type: Especifica el tipo de dato del campo value, en el caso de los atributos "fecha" y "hora" se tratan de strings y para los 9 casos restantes son de tipo double.
- metadata: Los metadatos son datos que describen o hablan sobre otros datos, es decir describen el contenido de la información del atributo que se esté tratando.

  En este caso el campo metadato contiene el atributo "unidades" que contiene información sobre el dato que se está proporcionando en ese atributo, asi como sus unidades de medida. En todos los casos al ser una descripción el campo "type" estará informado como "text".

Para poder visualizar todas las entidades existentes en el Orion Context Broker se realiza una petición GET a la siguiente url: *http://localhost:1026/v2/entities*. Se recibirá una respuesta en formato JSON procedente del context broker con cada una de las entidades que existan en ese momento.

A continuación, en la Ilustración 38 se muestra un ejemplo de una entidad del proyecto, donde se pueden ver cada uno de los atributos descritos y los valores de cada uno de ellos.

```
[
  {
      id: "123",
      type: "Coche",
      acelerometro_x: {
          type: "double",
          value: "0.029419951140880585",
          metadata: {
              unidades: {
                  type: "text",
                  value: "Valor del eje x del acelerometro"
              }
          }
      },
      acelerometro_y: {
          type: "double",
          value: "0.4511059522628784",
          metadata: {
              unidades: {
                  type: "text",
                  value: "Valor del eje y del acelerometro"
              }
          }
      },
      acelerometro_z: {
          type: "double",
          value: "10.375436782836914",
          metadata: {
              unidades: {
                  type: "text",
                  value: "Valor del eje z del acelerometro"
              }
          }
      },
      fecha: {
          type: "string",
          value: "16/03/2020",
          metadata: {
              unidades: {
                  type: "text",
                  value: "dd/mm/aaaa"
              }
          }
      },
      giroscopio_x: {
          type: "double",
          value: "0.029419951140880585",
          metadata: {
              unidades: {
                  type: "text",
                  value: "Valor del eje x del giroscopio"
              }
          }
      },
      giroscopio_y: {
          type: "double",
          value: "0.4511059522628784",
          metadata: {
              unidades: {
                  type: "text",
                  value: "Valor del eje y del giroscopio"
              }
          }
      },
      giroscopio_z: {
          type: "double",
          value: "10.375436782836914",
          metadata: {
              unidades: {
                  type: "text",
                  value: "Valor del eje z del giroscopio"
              }
          }
      },
      hora: {
          type: "string",
          value: "01:08",
          metadata: {
              unidades: {
                  type: "text",
                  value: "hh:mm"
              }
          }
      },
      latitud: {
          type: "double",
          value: "36.7289535",
          metadata: {
              unidades: {
                  type: "text",
                  value: "Latitud del GPS"
              }
          }
      },
      longitud: {
          type: "double",
          value: "-6.4343554",
          metadata: {
              unidades: {
                  type: "text",
                  value: "Longitud del GPS"
              }
          }
      },
      velocidad: {
          type: "double",
          value: "0",
          metadata: {
              unidades: {
                  type: "text",
                  value: "Velocidad en Km/h"
              }
          }
      }
  }
]
```


Ilustración 38: Ejemplo de entidades

También es posible consultar un solo atributo de una entidad concreta, por ejemplo, si se quiere obtener el atributo "velocidad" de la entidad con "id" equivalente a "123" se realiza la siguiente peticición:

http://localhost:1026/v2/entities/123/attrs/velocidad, se obtiene el resultado mostrado en la Ilustración 39.


```
{
    type: "double",
    value: "0",
  ▾ metadata: {
      ▾ unidades: {
            type: "text",
            value: "Velocidad en Km/h"
        }
    }
```


Ilustración 39: Ejemplo de atributo

También es posible indicar en la petición el tipo de vehículo, por ejemplo, con la siguiento petición GET:

http://localhost:1026/v2/entities?type=Coche, con el resultado que aparece en la Ilustración 40.


```
[
  ▾ {
      id: "123",
      type: "Coche",
    ▾ acelerometro_x: {
        type: "double",
        value: "0.029419951140880585",
      ▾ metadata: {
          ▾ unidades: {
                type: "text",
                value: "Valor del eje x del acelerometro"
            }
        }
    },
    ▾ acelerometro_y: {
        type: "double",
        value: "0.4511059522628784",
      ▾ metadata: {
          ▾ unidades: {
```


Ilustración 40: Ejemplo de entidad accedida por filtro

Además, es posible realizar una petición para mostrar solo el campo valor de los atributos especificados, por ejemplo, esta petición devolverá solo el nombre y valor de los atributos velocidad, fecha y hora de las entidades cuyo id y type sean 123 y Coche, respectivamente:

http://localhost:1026/v2/entities/123?type=Coche&options=keyValues&attrs=velocidad,fecha,hora, en la Ilustración 41 se puede ver un posible resultado de la petición anterior.

```
{
    id: "123",
    type: "Coche",
    velocidad: "0",
    fecha: "16/03/2020",
    hora: "01:08"
}
```


Ilustración 41: Ejemplo de entidad accedida por filtro más restrictivo

Por último, para eliminar una entidad existente basta con enviar una petición DELETE donde se indique el "id" de la entidad a eliminar. Por ejemplo, para eliminar la entidad con el "id" igual a "123" se realiza la siguiente petición utilizanco *curl*:

```
$ curl -X DELETE 'http://localhost:1026/v2/entities/123'
```

También se puede eliminar entidades mediante el valor del atributo "type":

```
$ curl -X DELETE 'http://localhost:1026/v2/entities/123?type=Coche'
```

5.1.2 Suscripciones

Además de las entidades que se acaban de mostrar en el apartado anterior, Orion Context Broker también posee suscripciones a cierta información, para que cuando cambie el valor del campo "hora" de la entidad, la aplicación reciba una notificación asíncrona.

Las suscripciones se van almacenando en una lista y pueden ser consultadas mediante peticiones GET al context broker siguiendo esta estructura: *http://localhost:1026/v2/subscriptions*, a esta petición se le responderá devolviendo una lista con todas las suscripciones existentes en el context broker en ese momento, similar a la que aparece en la Ilustración 42.


```
[
  ▾ {
      id: "5e6ec2be204db94f11cfe8c5",
      description: "A subscription to get info about GPS",
      status: "active",
    ▾ subject: {
      ▾ entities: [
        ▾ {
            id: "123",
            typePattern: ".*"
          }
        ],
      ▾ condition: {
        ▾ attrs: [
            "hora"
          ]
        }
      },
    ▾ notification: {
        timesSent: 4,
        lastNotification: "2020-03-16T00:08:02.00Z",
        attrs: [ ],
        onlyChangedAttrs: false,
        attrsFormat: "normalized",
      ▾ http: {
          url: http://192.168.1.103:8080/sensor/notificaciones
        },
        lastSuccess: "2020-03-16T00:08:03.00Z",
        lastSuccessCode: 200
      },
      throttling: 5
    }
]
```


Ilustración 42: Ejemplo de suscripción

Como se observa en la ilustración anterior, cada suscripción consta de una serie de campos que se describen a continuación:

- id: Es un número único identificativo para cada suscripción, es generado de forma automática cuando se crea la suscripción.
- description: Texto que muestra una breve descripción de la suscripción creada.
- status: Muestra el estado de la suscripción. Tiene la posibilidad de estar en diversos estados, los cuales son:
    - active: En este estado la suscripción envía notificaciones de forma constante.
    - oneshot: En este estado se envía una suscripción y directamente pasa al estado "inactive" [15].
    - inactive: Permanece en este estado hasta que vuelve a cambiar a otro.
- subject: Campos que determinan envíos de notificaciones de la suscripción.
    - entities: En este campo aparece la entidad a la que va referida esta suscripción.
        - id: Este campo "id" admite solo el valor de un identificador, que en el caso de este proyecto es el valor de la matrícula que se utiliza como id en la entidad asociada.
        - typePattern: Admite cualquier expresión regular, como por ejemplo con el valor ".*" [16].
    - condition: En este campo se indica las condiciones a cumplir para el envio de notificaciones.
        - attrs: Incluye los valores de los atributos que, si son modificados, lanzan un nuevo envío de notificación. En el caso de que esté vacío se tiene en cuenta las modificaciones en cualquiera de los atributos que posea la entidad.
- notification: Tiene numerosos parámetros como son:
    - timesSent: Indica el número de notificaciones que han sido enviadas desde que se creó la suscripción.
    - lastNotification: Indica la fecha y hora de la última notificación enviada.
    - attrs: En este parámetro se indican los atributos que se quieren notificar. Si se encuentra vacío se notifican todos.
    - attrsFormat: En este campo se indica el formato asociado al mensaje de notificación. Pueden ser: "normalized" que será igual que el formato de las entidades, "legacy" posee compatibilidad con NGSIv1 y "keyValues" de tipo clave valor.
    - http: Contiene el valor de la url a la que se envían las notificaciones.
    - lastSuccess: Muestra la fecha y hora del último envío de notificación realizado de forma correcta.
    - lastSuccessCode: Muestra el código de estado HTTP del último envío de notificación realizado.
- throttling: El mínimo intervalo de tiempo que debe existir entre el envío de dos notificaciones. Se mide en segundos.

Una vez descrita la suscripción y los campos que posee se describe cómo añadir y eliminar una suscripción de una de las entidades del context broker.

Cualquier usuario de la aplicación al enviar los datos recogidos por los sensores de su móvil al Orion Context Broker automáticamente está realizando una suscripción a la entidad cuyo campo "id" posea el valor de la matrícula del usuario.

Dicha suscripción se realiza mediante una petición POST a la url *http://localhost:1026/v2/subscriptions* desde el servicio web al context broker con los valores necesarios para configurar la suscripción para que sean notificados solo los campos que se deseen.

A continuación, se muestra en la Ilustración 43 la interacción de las suscripciones con cada uno de los actores, los actores básicos son:

- Orion Context Broker: De todos los componentes, es el principal, actúa como controlador de las suscripciones realizadas
- Productor de Contexto: Se encarga de generar y actualizar contexto. En este proyecto es la aplicación Android.
- Consumidor de Contexto: Se encarga de explotar la información del contexto. En este proyecto es el servicio web Spring.

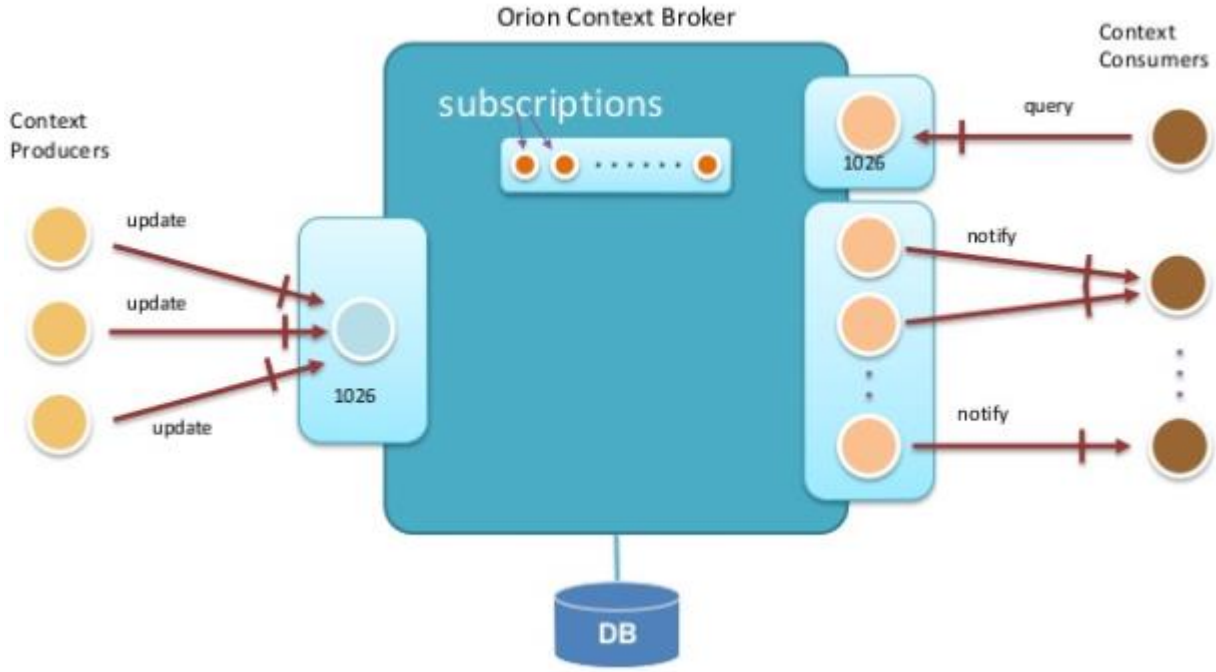


Ilustración 43: Interacciones de las suscripciones

A continuación, en la Ilustración 44 se muestra como han sido configuradas las suscripciones desde el servicio web Spring.

```
 // Construir el mensaje de suscripcion a la matricula dada
JSONObject sub_gps = new JSONObject("{\"description\": \"Suscripcion para obtencion de datos del GPS\", "
        + "\"subject\": {\"entities\": [{\"id\": \""+mat+"\", \"typePattern\": \".*\"}], "
        + "\"condition\": {\"attrs\": [\"hora\"]}}, " //Si cambia alguno se envia notificacion
        + "\"notification\": {\"http\": {\"url\": \"http://"+ip_port_sensor+"/sensor/notificaciones\"}, "
        + "\"attrs\": [], " //Si no se especifican atributos se notifican todos
        + "\"attrsFormat\": \"normalized\"}, "
        + "\"throttling\": 5}"); //Tiempo de espera tras enviar una notificacion (segundos)

//URL y parametros necesarios para la conexion
URL url = new URL("http://"+ip_port_contextbroker+"/v2/subscriptions");
HttpURLConnection httpConnection  = (HttpURLConnection) url.openConnection();
httpConnection.setDoOutput(true);
httpConnection.setRequestMethod("POST");
httpConnection.setRequestProperty("Content-Type", "application/json");
httpConnection.setRequestProperty("Accept", "application/json");

//Escribir el JSON parseado como string para la conexion
DataOutputStream wr = new DataOutputStream(httpConnection.getOutputStream());
wr.write(sub_gps.toString().getBytes());
Integer responseCode = httpConnection.getResponseCode();
```

Ilustración 44: Configuración de la suscripción

Se observa que el mensaje de suscripción a la matrícula se realiza a partir de un Objeto JSON, al cual se le pasan diversos parámetros como son:

- "id": Matrícula del vehículo que se está monitorizando.
- "typePattern": En nuestro proyecto acepta cualquier tipo de vehículo, no tiene restricciones en este campo.
- "attrs": Este campo pertenece a las condiciones a tener en cuenta para enviar una nueva notificación.

  En este proyecto posee el valor de "hora", es decir, se envían notificaciones si cambia el valor del campo "hora".
- "notification": Se indica la url para el envío de las notificaciones.
- "attrs": No se indican atributos, de esta forma se notifican todos.
- "attrsFormat": Con el valor a "normalized".
- "throttling": El tiempo, en segundos, que se espera tras el envio de una notificación.

Cuando se realiza una suscripción a una entidad, de forma instantánea se envía una notificación inicial sin necesidad de que se cumplan las condiciones para ello. Luego una vez se envía esta notificación inicial, se envían las notificaciones de forma normal, cuando las condiciones establecidas vayan cambiando [17].

Por último, también es posible cancelar una suscripción, para ello se envía una petición DELETE a la url: *http://localhost:1026/v2/subscriptions/id_subscription.*

En la Ilustración 45 se muestra como se realiza la cancelación de una suscripción desde el servicio web Spring.

```
//Enviar un mensaje DELETE para cancelar la suscripcion
ResteasyClient client = new ResteasyClientBuilder().build();
ResteasyWebTarget target = client.target("http://"+ip_port_contextbroker+"/v2/subscriptions/"+id);
Response response = target.request().delete();
response.close();
```

Ilustración 45: Cancelación de la suscripción

5.2 Aplicación Android MySpeed Go

Se ha utiliza el dispositivo Sony Xperia SP, smartphone con sistema operativo Android.

Posee una memoria RAM de 1GB, y 8GB de almacenamiento, Además de tener acelerómetro, giroscopio y los sensores necesarios para obtener los valores de la velocidad y ubicación.

Se ha diseñado una aplicación Android encargada de registrar nuevos usuarios, logear usuarios ya registrados y monitorizar y visualizar todos los datos recogidos a través de los sensores del dispositivo móvil.

Esta aplicación Android tendrá comunicación con el Orion Context Broker, el servicio web y la base de datos SQLite de Android.

A continuación, en la Ilustración 46 se muestra un diagrama con los casos de uso de la Aplicación Android.

Ilustración 46: Diagrama de casos de uso de la aplicación Android

5.2.1 Clases

En este apartado se tratan todas las clases que se utilizan para la aplicación Android diseñada, y así lograr un correcto funcionamiento de la misma.

A continación se mostrarán en dos diagramas de clases, cada una de las clases empleadas para la aplicación Android y las relaciones entre ellas.

En el primer diagrama de clases, el cual se muestra en la Ilustración 47, aparecen cada una de las clases utilizadas para el funcionamiento general de la aplicación.

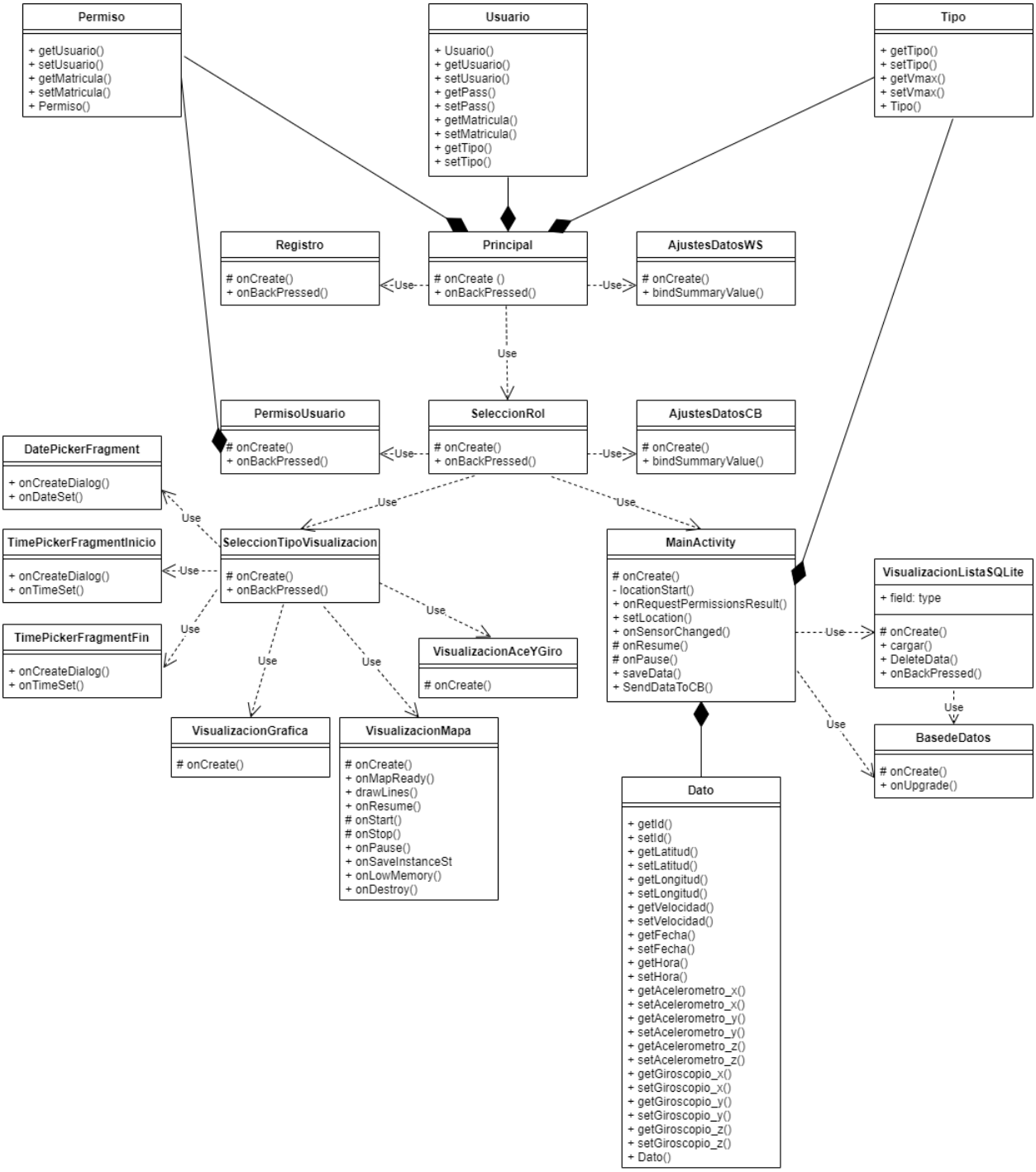


Ilustración 47: Diagrama de clases – Aplicación Android

En el segundo diagrama de clases, mostrado en la Ilustración 48, se observa cada una de las clases encargadas de establecer una conexión con el context broker o el servicio web. Además de las relaciones con las demás clases de la aplicación Android

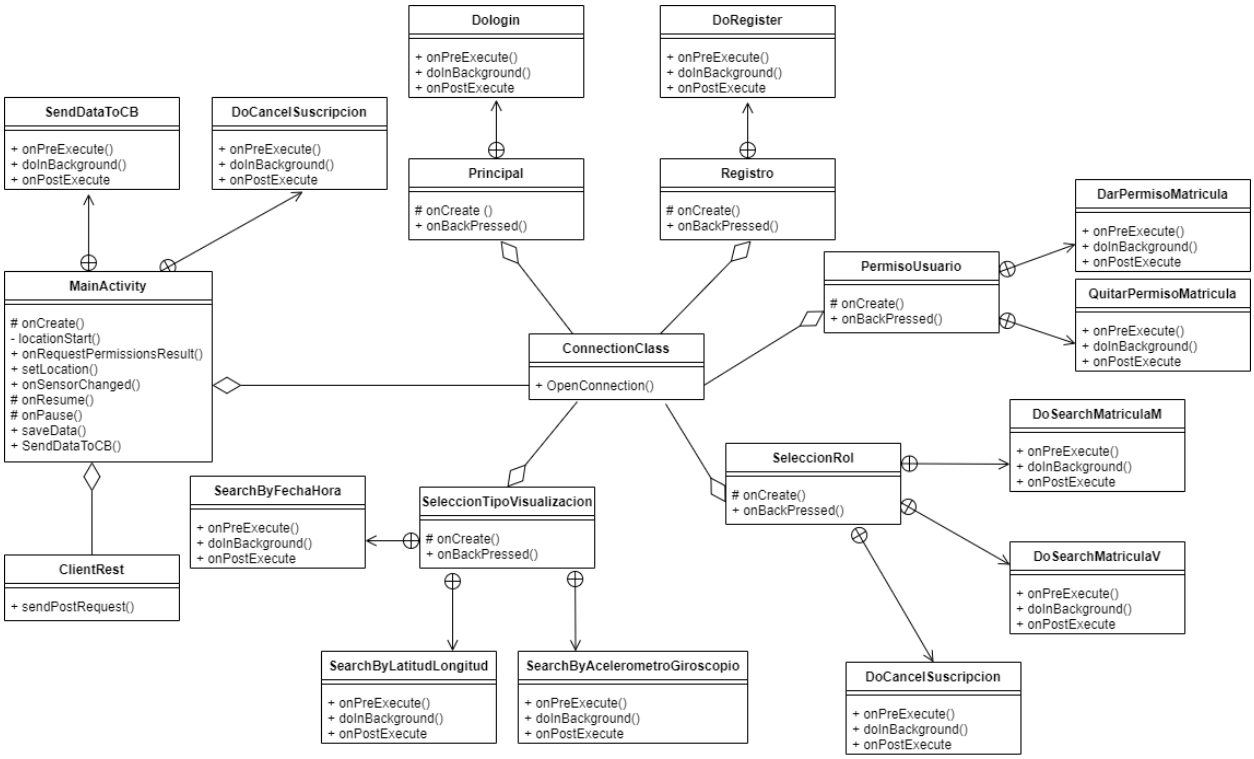


Ilustración 48 : Diagrama de clases – Conexiones Aplicación Android

En primer lugar, se analizarán las clases utilizadas en el Modelo de la aplicación web. Estas clases son las siguientes:

- Usuario: Contiene el usuario y la contraseña que el usuario ha introducido al registrarse, además de la matrícula y el tipo de vehículo. También posee el constructor de la clase, los getters y los setters de los campos.
- Permiso: Esta clase contiene los campos Usuario y Matricula, para la asociación entre las matrículas y los usuarios que tienen permisos para visualizar sus datos recogidos. Además, incluye el constructor de la clase, y los getters y setters de los campos.
- Tipo: Contiene los campos del tipo de vehículo y la velocidad máxima asociada a ese tipo. También posee el contrstructor de la clase y los getters y setters de cada uno de los campos.
- Dato: En esta última se encuentran todos los valores de los campos que se enviarán al context broker, como son el Id, posición (Latitud y Longitud), Velocidad, Fecha, Hora, aceleración (ejes x, y, z del acelerómetro) y la velocidad rotacional (ejes x, y, z del giroscopio). Al igual que las anteriores, contiene el constructor de la clase, los getters y los setters de todos los campos.

Todas las demás clases utilizadas en la aplicación Android se dividen en 4 grupos dependiendo de la funcionalidad que tenga, menos una, la clase MainActivity, encargada de la monitorización de los datos y de mostrarlos en tiempo real. La clase nombrada anteriormente se analizará en primer lugar. Se ha decidido que no forme parte de ninguno de los grupos que aparecen en este apartado porque tiene comunicación con todos los sistemas (servicio web, context broker y base de datos SQLite de Android).

MainActivity

Esta clase se encarga de monitorizar los datos que recoge cada uno de los sensores del dispositivo móvil, además de la fecha y hora en la que se produce cada nuevo registro. También va mostrando en tiempo real todos esos valores para que el usuario pueda verlos.

Se utiliza el método "onLocationChanged" de la clase "Localizacion" que implenta "LocationListener" (android.location.LocationListener) para obtener los valores de la latitud y longitud y guardarlos en un objeto JSON. Además, se utiliza el geocodificador (Android.Location.Geocoder) para obtener la dirección en la que se encuentra el vehículo, utilizando para ello los valores de la latitud y longitud obtenidos.

En el método "onSensorChanged" con un *SensorEvent* como argumento de entrada, se obtienen los valores de los ejes x, y, z del acelerómetro y giroscopio y se guardan en un objeto JSON.

Se ha creado el método "SaveData" desde donde se crea una nueva instancia de la clase "BasedeDatos" para insertar en la base de datos SQLite de Android todos los valores del vehículo que se van obteniendo.

En el método "SendDataToCB" mediante la línea "ClientREST.sendPostRequest(url, petition)" se realiza la petición al context broker indicando su url y el cuerpo de la petición con los valores de la entidad.

Al salir de esta pantalla, se cancela la suscripción existente al context broker mediante una petición al servicio web.

Clases para el funcionamiento principal

Estas son las clases más generales de la aplicación, que no establecen una comunicación directa con la base de datos de Android, el context broker o el servicio web.

- **MyApplication**

```
package com.example.sergio.appvelocidad;

import android.app.Application;
import android.content.Context;

public class MyApplication extends Application {

    private static Context mContext;

    @Override
    public void onCreate() {
        super.onCreate();
        mContext = getApplicationContext();
    }

    public static Context getContext() { return mContext; }
}
```

Ilustración 49: Clase MyApplication

Es la clase de aplicación personalizada que extiende de la "android.app.Application". La clase "Application" de la que extiende, es la clase base de una aplicación Android que contiene todos los componentes, como pueden ser las actividades y servicios. Se instancia antes que cualquier otra clase cuando se crea el proceso para su aplicación o paquete. Se muestra en la Ilustración 49.

- **AjustesDatosWS**

  En esta clase se establecen los ajustes principales de la aplicación Android. Contiene la funcionalidad de la pantalla de ajustes donde se introduce el valor de la dirección ip del servicio web.

- **AjustesDatosCB**

  Esta clase asigna ajustes a la aplicación Android, en ella se establecen los valores de la dirección ip y puerto del context broker y el intervalo de tiempo entre envíos de datos, recogidos por los sensores del dispositivo móvil, al context broker.

- **DatePickerFragment**

  Mediante Android, que proporciona diversos controles para lograr que el usuario pueda elegir una fecha en un cuadro de diálogo en el que aparece un calendario. Este selector ayuda a que cada usuario pueda elegir la fecha que desea, y esta sea válida y esté formateada correctamente [18].

  Permite elegir el dia, el mes y el año. Utiliza una estrategia de persistencia, como es SharedPreferences [19] para poder acceder al valor de la fecha seleccionada desde otra clase.

- **TimePickerFragmentInicio**

  Android proporciona controles para lograr que el usuario pueda elegir una hora en un cuadro de diálogo. Este selector ayuda a que cada usuario pueda elegir la hora de inicio del intervalo que desea, y esta sea válida y siga un formato correcto.

  Permite elegir la hora y los minutos. Utiliza una estrategia de persistencia, como es SharedPreferences [19] para poder acceder al valor de la hora de inicio seleccionada desde otra clase [18].

- **TimePickerFragmentFin**

  El sistema Android proporciona controles para lograr que el usuario pueda elegir una hora en un cuadro de diálogo. Este selector ayuda a que cada usuario pueda elegir la hora de fin del intervalo que desea, y esta sea válida y siga un formato correcto.

  Permite elegir la hora y los minutos. Utiliza una estrategia de persistencia, como es SharedPreferences [19] para poder acceder al valor de la hora de fin seleccionada desde otra clase [18].

- **VisualizacionGrafica**

  En esta clase se establece la funcionalidad para mostrar la gráfica de la velocidad respecto a la hora.

  Para ello el eje x está limitado por la hora de inicio y hora de fin elegidas y aparecen todas las horas que tienen valores registrados dentro de ese intervalo. Mientras en el eje y se muestran los valores de la velocidad que existen en ese intervalo de horas.

  Se obtienen las horas y velocidades a mostrar a partir de SharedPreferences [19]. Para mostrar la gráfica se hace uso de la biblioteca de gráficos para Android “lecho.lib.hellocharts” [20].

- **VisualizacionMapa**

  Esta clase muestra un mapa con la ruta que ha seguido el vehículo durante el día e intervalo de tiempo seleccionado previamente.

  Para obtener el mapa se utiliza Mapbox, un conjunto de herramientas de código abierto para mostrar mapas en las aplicaciones Android. Se ha utlizada la versión 8.4.0 del SDK de Mapbox [21].

  Los valores de las latitudes y longitudes se obtienen mediante SharedPreferences [19] y se monta una

cadena con formato GeoJSON para que se puedan leer esos valores e interpretarlos correctamente en el mapa formando la ruta.

- **VisualizacionAceYGiro**

  Esta clase muestra dos listas con los valores del acelerómetro en una y los del giroscopio en otra.

  Los valores del acelerómetro (eje x, y, z) y giroscopio (eje x, y, z) se obtienen en la clase "SeleccionTipoVisualizacion" mediante la petición al servicio web que se realiza cuando se elige la opción de visualizar el acelerómetro y giróscopio.

  Desde esta clase se accede a esos valores utilizando SharedPreferences [19] y se listan siguiendo el orden de inserción que tuvieron en la tabla "datos" de la base de datos "monitorizacion".

  Se listan mediante un TextView dentro del ScrollView utilizado en su layout. A ese TextView se le asigna los valores listados.

Clases para la comunicación con el Orion Context Broker

En este caso se describen las clases que establecen una relación entre la aplicación Android y el context broker

- **ClientRest**

  Esta clase, la cual se muestra en la Ilustración 50, se encarga mediante la API de Android, de implementar la comunicación cliente servidor. Para ello utiliza la clase HttpURLConnection. Se realiza una petición POST.

  Se sigue el siguiente patrón para implementar dicha comunicación:

  - Se obtiene un nuevo HttpURLConnection y enviando el resultado a HttpURLConnection.
  - Se prepara la solicitud indicando el URI y los encabezados de la solicitud.
  - Cargar un cuerpo de solicitud. Debido a que se incluye el cuerpo de solicitud, las instancias se configuran con "setDoOutput(true)".
  - Leer la respuesta. El cuerpo de la respuesta se lee mediante la secuencia que devuelve "URLConnection.getInputStream()".
  - Una vez leído todo el cuerpo de la respuesta, se desconecta a través de "disconnect()".

  Contiene el método "sendPostRequest" que recibe como argumentos de entrada la URL de la petición y el contenido de la petición. Devuelve una cadena para indicar la respuesta de la petición.

```
public class ClientREST {
    public static String sendPostRequest(final String requestUrl, final String payload) throws IOException {
        final StringBuffer[] jsonString = new StringBuffer[1];

        //Ejecucion asincrona
        AsyncTask.execute(new Runnable() {
            @Override
            public void run() {
                try {
                    //Establecimiento y definicion de la conexion
                    URL url = new URL(requestUrl);
                    HttpURLConnection connection = (HttpURLConnection) url.openConnection();

                    connection.setDoInput(true);
                    connection.setDoOutput(true);
                    connection.setRequestMethod("POST");
                    connection.setRequestProperty("Accept", "application/json");
                    connection.setRequestProperty("Content-Type", "application/json; charset=UTF-8");
                    OutputStreamWriter writer = new OutputStreamWriter(connection.getOutputStream(),  charsetName: "UTF-8");
                    writer.write(payload);
                    writer.close();
                    BufferedReader br = new BufferedReader(new InputStreamReader(connection.getInputStream()));
                    jsonString[0] = new StringBuffer();
                    String line;
                    while ((line = br.readLine()) != null) {
                        jsonString[0].append(line);
                    }
                    br.close();
                    connection.disconnect();
```

Ilustración 50: Clase ClientREST

Clases para la comunicación con el servicio web

Ahora se procede a describir las clases que establecen una relación entre la aplicación Android y el servicio web.

- **Principal**

  Esta clase se encarga de la funcionalidad asociada a la pantalla Principal de la aplicación Android. Establece la comunicación con el servicio web para el envío de una petición con los valores del usuario y contraseña intoducidos por el usuario, para que el servicio web acceda a la base datos y comprueba si existen esos datos en la tabla "usuarios". La aplicación Android recibe un mensaje de respuesta del servicio web con los valores encontrados en la base de datos. En esta clase, también se encarga de obtener la matrícula asociada al usuario en la tabla "usuarios" y las matrículas que el usuario tiene asociadas en la tabla "permisos". Todas estas matrículas se guardan utilizando SharedPreferences, para poder obtener sus valores en otra clase.

- **ConnectionClass**

  Esta clase, mostrada en la Ilustración 51, se encarga mediante la API de Android, de implementar la comunicación cliente servidor. Para ello utiliza la clase HttpURLConnection [22].

  Se sigue el siguiente patrón para implementar dicha comunicación:

  - Se obtiene un nuevo HttpURLConnection y enviando el resultado a HttpURLConnection.
  - Se prepara la solicitud indicando el URI con los parámetros y los encabezados de la solicitud.
  - Cargar un cuerpo de solicitud. Debido a que se incluye el cuerpo de solicitud, las instancias se configuran con "setDoOutput(true)".
  - Leer la respuesta. El cuerpo de la respuesta se lee mediante la secuencia que devuelve "connection.getInputStream()".
  - Una vez leído todo el cuerpo de la respuesta, se desconecta a través de "disconnect()".

```
public class ConnectionClass {

    public StringBuilder OpenConnection(String peticion) {
        StringBuilder sb = new StringBuilder();

        try {
            URL url = new URL(peticion);
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();

            connection.setDoInput(true);
            connection.setDoOutput(true);
            connection.setRequestMethod("POST");
            connection.setRequestProperty("Accept", "application/json");
            connection.setRequestProperty("Content-Type", "application/json; charset=UTF-8");
            BufferedReader br = new BufferedReader(new InputStreamReader(connection.getInputStream()));
            sb = new StringBuilder();
            String line;
            while ((line = br.readLine()) != null) {
                sb.append(line);
            }

            br.close();
            connection.disconnect();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return sb;
    }
}
```

Ilustración 51: Clase ConnectionClass

Se utiliza en el método "openConnection" que recibe como argumentos de entrada la URL de la petición con los parámetros necesarios. Devuelve un "StringBuilder" con los datos devueltos por el servicio web.

- **PermisoUsuario**

  Mediante esta clase se le otorga o se le retira a un usuario el permiso de visualización sobre una matrícula, para que así, un usuario pueda ver los datos registrados por una matrícula, que no es la suya, en la base de datos. Este permiso de visualización lo asigna el usuario con la matrícula a visualizar.

  Para ello realiza peticiones al servicio web, utilizando el método "openConnection" de la clase "ConnectionClass" descrito anteriormente. Primero comprueba si ese usuario existe en el sistema, luego comprueba si el usuario indicado ya posee esos permisos de visualización sobre la matrícula y, por último, se inserta el usuario indicado y la matrícula en la tabla "permisos" de la base de datos. En el caso de que se desee retirar el permiso de visualización a un usuario, simplemente se elimina dicha entrada de la tabla "permisos" de la base de datos.

- **SeleccionRol**

  A través de esta clase se define la funcionalidad de poder elegir a que sección de la aplicación acceder, si a la pantalla de monitorización de datos en tiempo real o a la pantalla de visualización de datos ya registrados.

  Para lograr esto, las peticiones al servicio web se realizan utilizando el método "openConnection" de la clase "ConnectionClass" descrito anteriormente.

  En el caso de acceder al modo de monitorización de datos, se realiza una petición al servicio web para consultar si existen suscripciones realizadas en el context broker con anterioridad que no han sido canceladas, si existe alguna la cancela y crea una nueva. Si no existen suscripciones anteriores, crea la nueva directamente.

- **Registro**

  Esta clase se define para el registro de nuevos usuarios en la aplicación Android. Realiza una petición al servicio web para insertar los datos del nuevo registro en la tabla "usuarios" y otra petición para insertar el usuario y matrícula en la tabla "permisos". Esa inserción en la tabla "permisos" de la base de datos permite que el nuevo usuario tenga permisos de visualización sobre su propia matrícula.

  Se utiliza el método "openConnection" de la clase "ConnectionClass" descrito anteriormente.

- **SeleccionTipoVisualizacion**

  Desde esta clase se realiza la funcionalidad de establecer una fecha y un intervalo de tiempo a consultar, y lanzar las pantallas de visualización de la velocidad, ubicación y acelerómetro y giroscópio.

  En esta clase se crea una nueva instancia de la clase "DatePickerFragment" para permitir al usuario seleccionar la fecha en la que buscar los registros. También crea una instancia de la clase "TimePickerFragmentInicio" para elegir la hora de inicio del intervalo a buscar y otra nueva instancia de la clase "TimePickerFragmentFin" para la hora de fin.

  Para acceder a cualquiera de las tres pantallas de visualización se realizan peticiones al servicio web de donde se obtienen los datos a mostrar. Estos datos se guardan siguiendo una estrategia de persistencia, como es SharedPreferences [19], para su utilización en la siguiente clase.

  Utiliza el método "openConnection" de la clase "ConnectionClass" descrito con anterioridad.

Clases para la comunicación con la base de datos SQLite de Android

Existen muchos sistemas de bases de datos disponibles, pero hay uno de ellos que se ajusta perfectamente a las aplicaciones móviles, SQLite. Esto es debido a que SQLite solo requiere un simple fichero para almacenar los datos, porque la lógica de funcionamiento tiene que ser implementada por la plataforma que quiera insertar o consultar esos datos. En el caso de Android, el SDK Android posee soporte completo para SQLite.

En este apartado se describen las clases que establecen una relación entre la aplicación Android y la base de datos SQLite del propio Android.

- **BasedeDatos**

  En ella se crea la tabla "Tarea" en la base de datos SQLite, donde se guardan todos los datos recogidos por los sensores del dispositivo móvil y posee un identificador que va autoincrementandose con cada nuevo registro en la tabla.

- **VisualizacionListaSQLite**

  Esta clase tiene dos funciones, cargar todos los registros de la tabla "Tarea" de la base de datos SQLite y mostrarlos por pantalla en una lista ordenada por orden de inserción. Y también eliminar todos los registros de la tabla "Tarea".

5.2.2 Ficheros de configuración

En este apartado se describen todos los ficheros de configuración que se utilizan en la aplicación Android.

AndroidManifest.xml

Es un fichero que debe tener todos los proyectos de aplicaciones, y situado en la raíz de la fuente del proyecto. Posee información esencial de la aplicación. Entre otras cosas debe informar de:

- El nombre del paquete de la aplicación.
- Los componentes de la aplicación, en los cuales se incluyen las actividades, servicios, receptores de emisiones y proveedores de contenido
- Los permisos que necesita la aplicación.
- Las funcionalidades de hardware y software que la aplicación requiere y afectan a los dispositivos que pueden instalar la aplicación desde Google Play.

En este caso, al utilizar Android Studio, genera este fichero, el cual se puede observar en la Ilustración 52 con la mayoría de estos elementos esenciales de forma automática.

Muestra el nombre del paquete e ID de la aplicación en el elemento raíz <manifest> con el nombre de paquete "com.example.sergio.appvelocidad" en el atributo "package". Mientras se compila la aplicación, el atributo "package" se utiliza para aplicar ese nombre como espacio de nombres para la clase "R.java" generada en la aplicación (utilizada para acceder a los recursos). También se utiliza para resolver cualquier nombre de clase relativo que se encuentre declarado en el fichero AndroidManifest.

Se utiliza el componente de aplicación <*activity*> para cada subclase de Activity. Si una subclase no se declara en este fichero de configuración, el sistema no puede inciarlo. El nombre de la subclase se debe indicar con el atributo "name".

En la actividad principal se utiliza <*intent-filter*> para indicar los tipos de intents a los que puede responder una actividad. Dentro del elemento <*intent-filter*> se indican dos más:

- <action>: Con el valor del atributo "name" a "android.intent.action.MAIN", esto significa que esta actividad es el punto de entrada a la aplicación, al iniciarse la aplicación se crea esta actividad.

- <category>: En el atributo “name” tiene el valor “android.intent.category.LAUNCHER”, indica que la actividad indicada como el punto de entrada, debe aparecer en el lanzador de la aplicación.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sergio.appvelocidad">
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <application
        android:name=".MyApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="VeloApp"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat.DayNight.NoActionBar"
        android:usesCleartextTraffic="true">
        <activity android:name=".view.Principal">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".MainActivity" />
        <activity android:name=".mostrar"></activity>
        <activity android:name=".SettingsActivity"></activity>
        <activity android:name=".Registro"></activity>
        <activity android:name=".SeleccionRol"></activity>
        <activity android:name=".SettingsPrincipal"></activity>
        <activity android:name=".ModoVisualizacion"></activity>
        <activity android:name=".FechaHoraVisualizacion"></activity>
        <activity android:name=".PermisoUsuario"></activity>
        <activity android:name=".VisualizacionMapa"></activity>
        <activity android:name=".VisualizacionAceYGiro"></activity>
    </application>
</manifest>
```

Ilustración 52: Fichero AndroidManifest.xml

Ficheros de configuración de compilación

Android utiliza “Gradle” para compilar recursos y código fuente de la aplicación y los empaqueta en APK.

Al comenzar un nuevo proyecto, Android crea de forma automática algunos de estos ficheros con valores predeterminados.

Se utilizan dos archivos de compilación de nivel superior “build.gradle”, ambos ubicados en el directorio raíz del proyecto.

Además de los repositorios y dependencias que se añaden por defecto al crear el proyecto, en ambos ficheros de compilación se añaden otros repositorios y dependencias necesarias para el correcto funcionamiento de este proyecto.

- build.gradle(Project: appvelocidad): Este fichero de compilación de nivel superior, mostrado en la Ilustración 53, permite añadir opciones de configuración que serán comunes a todos los módulos del proyecto.

```
buildscript {

    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.3.1'


        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
        maven{url "https://jitpack.io"}
        mavenCentral()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Ilustración 53: Fichero build.gradle (Project)

- build.gradle(Module: app): Fichero de compilación de nivel superior de un módulo específico, el cual se observa en la Ilustración 54.

```
}

dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    //noinspection GradleCompatible
    implementation 'com.android.support:appcompat-v7:28.0.0'
    testCompile 'junit:junit:4.12'
    implementation 'com.google.android.gms:play-services-location:9.8.0'
    implementation 'com.android.support:cardview-v7:28.0.0'
    implementation 'com.android.support:design:28.0.0'
    implementation 'com.github.PhilJay:MPAndroidChart:v3.0.0-beta1'
    implementation 'com.github.lecho:hellocharts-library:1.5.8@aar'
    //Para que funcione el servicio post al CB
    implementation 'org.jbundle.util.osgi.wrapped:org.jbundle.util.osgi.wrapped.org.apache.http.client:4.1.2'
    compile 'com.mapbox.mapboxsdk:mapbox-android-sdk:8.4.0'
    implementation 'com.jakewharton.timber:timber:4.7.1'
    implementation 'com.android.support:support-v4:28.+'
    implementation 'com.android.support.constraint:constraint-layout:1.1.2'
    implementation 'com.jakewharton.threetenabp:threetenabp:1.2.1'
}
```

Ilustración 54: Fichero build.gradle (Module)

5.2.3 Recursos

Se pueden proporcionar recursos a la aplicación y aplicarlos haciendo referencia a su ID de recurso. Todos estos ID de recursos se definen en la clase "R" del proyecto que es generada por la herramienta "aapt" automáticamente. Una vez que se está compilando la aplicación, la herramienta "aapt" genera la clase "R" del proyecto y esta clase es la que contiene los ID de los recursos en el directorio *res/*.

El ID de recurso está compuesto por el tipo de recurso (string, drawable y layout) y el nombre del recurso.

Existen dos formas de acceder a un recurso dependiendo si es a través de código o en XML.

- En código: Por ejemplo "R.string.hola", donde "string" es el tipo de recurso y "hola" el nombre del recurso.
- En XML: Por ejemplo "@string/adios", donde "string" es el tipo de recurso y "adios" el nombre del recurso.

En este caso el directorio /res, mostrado en la Ilustración 55, se compone de los siguientes directorios para cado uno de los diferentes tipos de recursos.

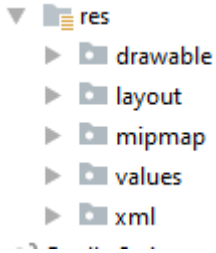


Ilustración 55: Directorio /res

- /res/drawable: Esta carpeta se utiliza para colocar todas las imágenes que se van a utilizar en la aplicación.
- /res/layout: Este directorio, que aparece en la Ilustración 56, contiene los layout asociados a cada una de la clases. Para cada pantalla que se quiera hacer se necesitan ficheros layout distintos y cada layout en un fichero distinto. Las vistas de los layout son accesibles mediante código a través de sus ID de recursos generados en R.java. Un ejemplo de acceso sería:

  ```
  String Matricula = ((EditText) findViewById(R.id.CarReg)).getText().toString();
  ```

  Para asociar el layout a la clase java es necesario indicarlo en la clase mediante:

  ```
  setContentView(R.layout.activity_principal);
  ```

  Y en el layout:

  ```
  tools:context="com.example.sergio.appvelocidad.view.Principal">
  ```

  Todos los ficheros XML layout del proyecto son:

- /res/mipmap: Funciona exactamente igual que la carpeta /res/drawable, pero en esta carpeta vamos a colocar las imágenes que se van a usar dentro de la aplicación, solo se coloca el icono de la aplicación.
- /res/values: Este directorio contiene todas las cadenas de texto, dimensiones, colores y estilos usados en la aplicación, todos ellos en ficheros XML.
- /res/xml: En este directorio se encuentran los ficheros "preferences.xml" y "preferencesprincipal.xml" que son los utilizados como layout para establecer la vista de las pantallas de ajustes de la aplicación.

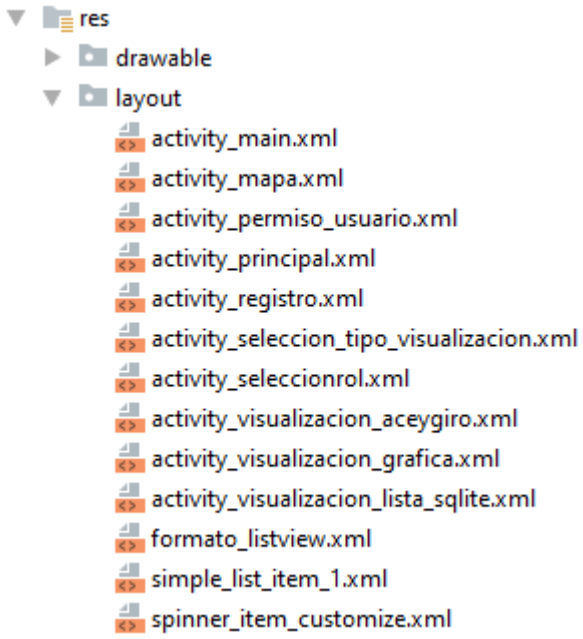


Ilustración 56: Directorio /res/layout

6 INTERFAZ DE USUARIO Y FUNCIONALIDAD

No hay nada imposible para aquel que lo intenta.

- Alejandro Magno -

En este capítulo se analiza cada una de las pantallas de la aplicación Android y las funcionalidades que ofrece al usuario. Se pueden agrupar las funcionalidades en dos grupos, el encargado de realizar las consultas y las inserciones y el encargado de gestionar cada una de las suscripciones al context broker.

A continuación, se muestran y describen cada una de las pantallas que muestra la aplicación Android.

- **Pantalla Principal**

  En la pantalla principal de la aplicación Android, la cual se muestra en la Ilustración 57, se permite:

  - Iniciar sesión con un usuario ya registrado con anterioridad, para ello se indica el nombre del usuario, su contraseña y se pulsta en "INICIAR SESIÓN".
  - Registrar un nuevo usuario que nos lleva a la pantalla de Registro.
  - Acceder a los ajustes principales de la aplicación, donde se establece la dirección IP del servicio web.

- **Pantalla de Registro**

  La pantalla de registro se muestra en la ilustración 58, donde se introducen los datos del nuevo usuario en la aplicación.

  Se rellenan los campos del nombre de usuario, contraseña, matrícula y tipo de vehículo.

  Después se pulsa el botón "CREAR CUENTA", para crear ese nuevo usuario introduciendo sus datos en la base de datos MySQL y nos lleva de vuelta a la pantalla Principal de la aplicación.

  Al pulsar en el botón "INICIAR SESIÓN" se vuelve a la pantalla Principal sin registrar ningún nuevo usuario.

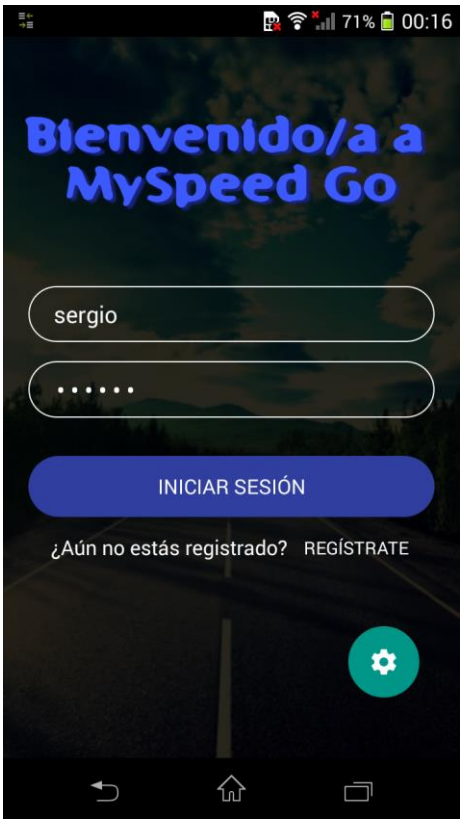


Ilustración 57: Pantalla Principal – Aplicación Android

Ilustración 58: Pantalla de Registro – Aplicación Android

- **Pantalla de Ajustes Principales**

  La ilustración 59 muestra la pantalla de Ajustes Principales de la aplicación Android.

  Se accede a ella tras pulsar el botón situado en la esquina inferior izquierda de la pantalla Principal.

  En ella se establece la dirección IP del servicio web.

- **Pantalla de Selección de Rol**

  Una vez hemos iniciado sesión con nuestro usuario, nos aparece esta pantalla, mostrada en la Ilustración 60, para poder seleccionar nuestro rol dentro de la aplicación.

  Al pulsar en "--Vacío--" aparecerá un desplegable con todas las matriculas disponibles para ese usuario, seleccionamos una y pulsamos en "MONITORIZACIÓN DE DATOS" para acceder a la pantalla de envío de datos al context broker y visualización en tiempo real de los datos recogidos por los sensores del dispositivo, o pulsamos en "VISUALIZACIÓN DE DATOS" para acceder a la pantalla de visualización de los datos guardados en la base de datos MySQL.

  Además, posee 3 botones en la parte inferior de la pantalla. El primero de ellos, más a la izquierda de la pantalla nos lleva la pantalla encargada de otorgar permisos a un usuario para visualizar matrículas registradas en el sistema que no son la suya, el que se encuentra situado en el centro cierra la sesión de usuario iniciada y nos lleva a la pantalla Principal de nuevo, y el situado a la derecha nos lleva a los ajustes del context broker.

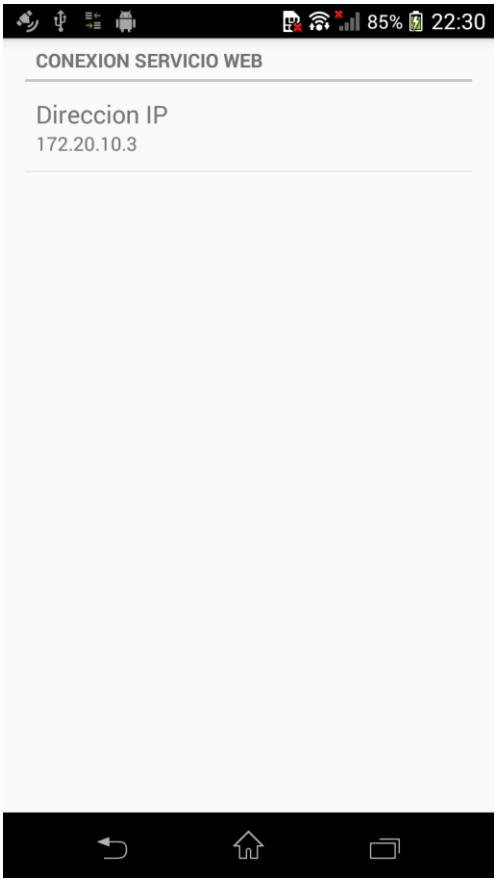


Ilustración 59: Pantalla Ajustes Principales – Aplicación Android

Ilustración 60: Pantalla Selección de rol – Aplicación Android

- **Pantalla para el permiso de visualización de matrículas**

  Desde esta pantalla, que se observa en la Ilustración 62, el usuario que ha iniciado sesión puede dar o quitar permiso a otro usuario registrado en el sistema para poder ver, o no, sus datos monitorizados.

  Pulsando en "DAR PERMISO" se añade el permiso de visualización al usuario indicado en el campo y lleva de vuelta a la pantalla previa, la encargada de seleccionar el rol.

  Pulsando en "QUITAR PERMISO" se elimina el permiso de visualización al usuario indicado en el campo y lleva de vuelta a la pantalla previa, la encargada de seleccionar el rol.

  En la esquina inferior izquierda aparece un botón con el icono de una flecha, una vez sea pulsado, vuelve a la pantalla anterior, la pantalla de selección de rol, sin añadir nuevos permisos de visualización a ningún usuario.

- **Pantalla de Ajustes del Context Broker**

  En la ilustración 61 se puede observar la pantalla de ajustes para la comunicación desde la aplicación con el context broker, a ella se accede tras pulsar el botón de ajustes situado en la esquina derecha de la pantalla de Selección de Rol.

  En esta pantalla se permite establecer tres parámetros:

  - El intervalo de tiempo de refresco para enviar los datos al context broker.
  - La dirección IP del context broker para que sea posible establecer la comunicación de la aplicación con el context broker.
  - El puerto del context broker.

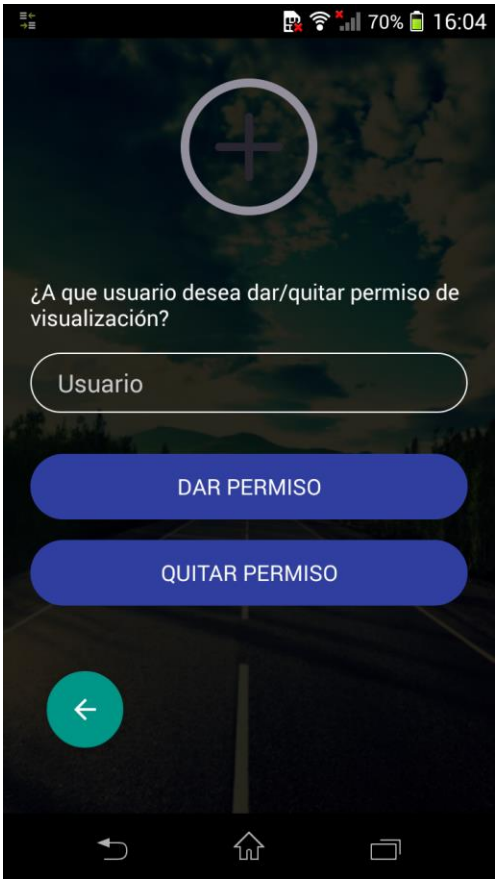


Ilustración 62: Pantalla Permisos – Aplicación Android

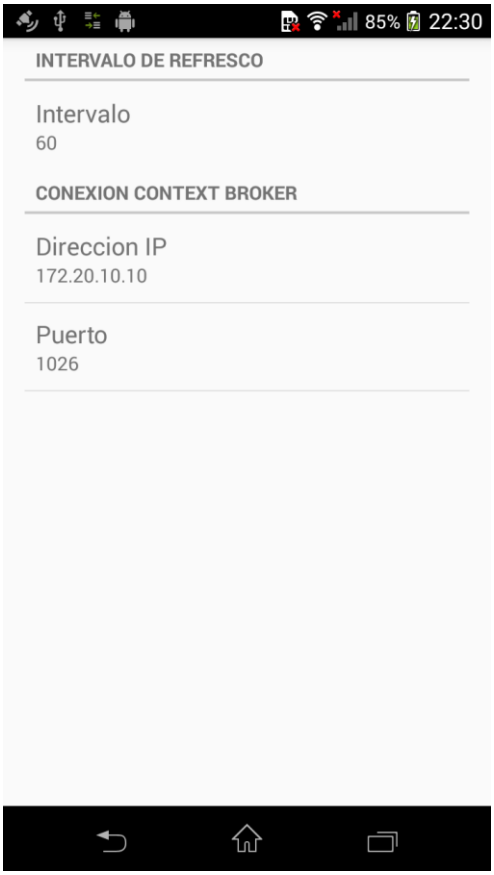


Ilustración 61: Pantalla Ajustes CB – Aplicación Android

- **Pantalla de Monitorización**

  Al pulsar en el botón "MONITORIZACÓN DE DATOS" en la pantalla de Selección de Rol, nos traslada a esta pantalla, mostrada en la Ilustración 63, donde se van actualizando los datos en el intervalo de tiempo que hayamos indicado anteriormente en la ventana de ajustes del context broker. Los datos que se muestran son tratados por el context broker y guardados en MySQL a través del servicio web, además de añadirse a la base de datos SQLite de Android.

  En la parte superior podemos visualizar dos velocidades, la que se encuentra a la izquierda es la velocidad que lleva nuestro vehículo en ese momento actualizándose en el intervalo de tiempo establecido en la pantalla de ajustes, y situada a la derecha, aparece la velocidad limite establecida para el tipo de vehículo con el que se encuentre circulando en ese momento. Si la velocidad a la que circulamos es la velocidad límite menos 20 km/h o mayor pero no superior a la velocidad límite, el recuadro cambia a naranja, como señal de aviso que se encuentra próxima a la establecida como límite, y si supera a la velocidad límite cambia a rojo.

  En la parte inferior aparecen 3 botones, el que se encuentra situado a la izquierda nos muestra los valores guardados en la base de datos SQLite de Android, el situado en el centro de la pantalla permite salir de la sesión del usuario y volver a la pantalla principal de la aplicación, y el que se encuentra a la derecha nos muestra la pantalla de ajustes del context broker.

- **Pantalla para mostrar los valores de la base de datos SQLite**

  Esta pantalla, la cual se puede observar en la Ilustración 64, permite visualizar todos los valores recogidos por los sensores del vehículo que se esta monitorizando y guardados en la base de datos SQLite de Android.

  Además, en la esquina inferior derecha aparece un botón con el icono de una papelera, que al pulsar elimina todo el contenido de la base de datos SQLite y se refresca la lista, apareciendo vacía.

- **Pantalla de Visualización**

  Al pulsar en "VISUALIZACIÓN DE DATOS" en la pantalla de Selección de Rol, nos lleva a esta pantalla, mostrada en la Ilustración 66, en ella podemos ver los datos tratados por el context broker y el servicio web y almacenados en la base de datos MySQL.

  Se establece la fecha y un rango de horas, luego podemos ver 3 tipos de datos guardados:

  Al pulsar en "VELOCIDAD" nos lleva a una pantalla con una gráfica donde se muestra la velocidad que el vehículo con la matrícula asociada indicada a cada hora dentro del rango de horas seleccionado.

  Al pulsar en "MAPA DEL RECORRIDO" nos lleva a una pantalla donde se muestra un mapa con el recorrido del vehículo en ese rango de horas de ese día.

  Al pulsar en "ACELEROMETRO Y GIROSCOPIO" nos lleva a una pantalla donde se muestran dos listas con los valores del acelerómetro y giroscopio en ese rango de horas de ese día.

- Por último, se observan las pantallas de visualización que muestran la velocidad frente a la hora (Ilustración 65), la ruta recorrida por el vehículo representada en un mapa (Ilustración 67) y los valores de todos los ejes del acelerómetro y giroscopio en forma de lista (Ilustración 68).

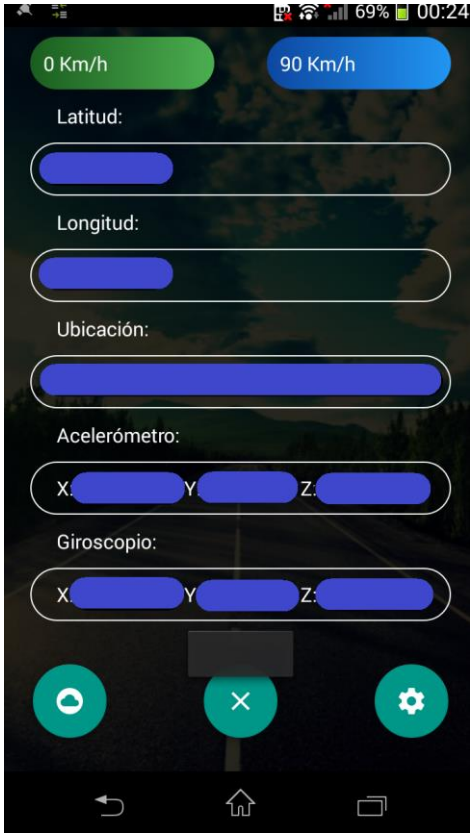


Ilustración 63: Pantalla Monitorización - Aplicación Android

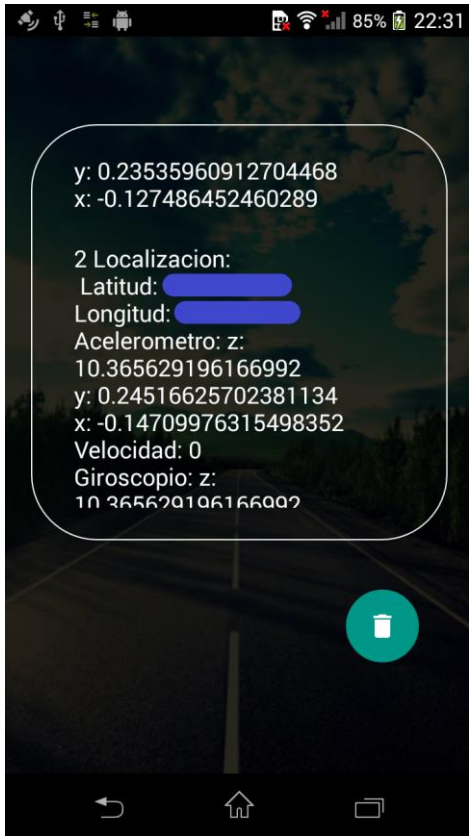


Ilustración 64: Pantalla Visualización SQLite – Aplicación Android

Ilustración 66: Pantalla Visualización – Aplicación Android

Ilustración 65: Gráfica Velocidad/Hora – Aplicación Android

Ilustración 67: Mapa con la ruta – Aplicación Android

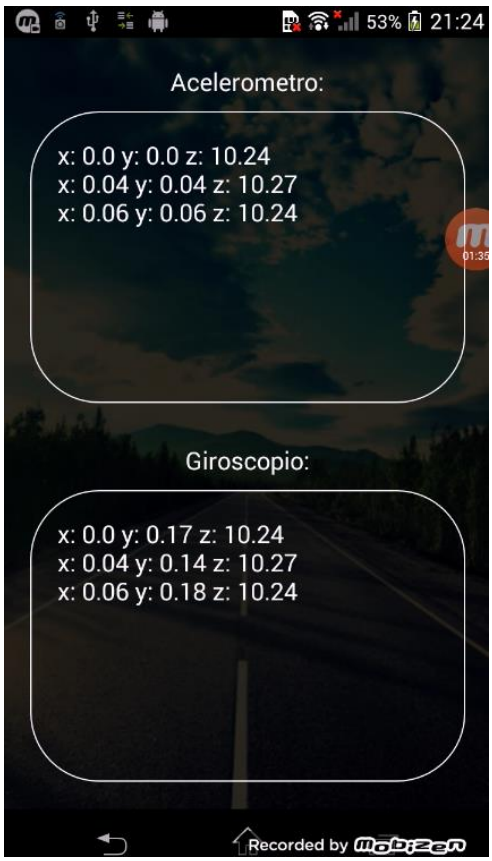


Ilustración 68: Listas Acelerómetro y Giroscopio – Aplicación Android

7 CONCLUSIONES Y LÍNEAS FUTURAS

La experiencia te iluminará, el maestro sólo te puede mostrar el camino.

– Proverbio egipcio -

Como resultado del proyecto realizado, es posible indicar que ha sido muy positivo para aprender y obtener conocimientos más específicos sobre todas las tecnologías utilizadas. Estas tecnologías son muy importantes hoy en día para hacernos la vida más sencilla y por tanto se encuentran en pleno crecimiento, en este caso haciendo más énfasis en el entorno de las ciudades inteligentes con el uso de Fiware.

Entre todos los conocimientos adquiridos a lo largo del desarrollo de este proyecto, hay que destacar el gran peso que han tenido la utilización de Android Studio, el cual, es el entorno de desarrollo integrado (IDE) empleado de forma oficial para el desarrollo de las aplicaciones Android, y el uso de Spring en el desarrollo de las distintas clases java del servicio web. Se tenían conocimientos previos de estas materias por diversas asignaturas cursadas en el grado, pero se trataban de conocimientos básicos, ha sido necesario dedicarle una parte del tiempo a profundizar en todo lo que nos ofrecen estas tecnologías para su uso en el proyecto de la mejor forma posible.

El sistema que se ha desarrollado en este proyecto posee, además de todas las funcionalidades mostradas, un gran número de mejoras. Este proyecto se trata de una gran base, sobre la cual se seguirán realizando mejoras en un futuro para que la experiencia del usuario con la aplicación Android sea cada vez mejor.

Una de esas posibles mejoras sería en la situación de que no se posea conexión a internet, poder seguir almacenando datos en la base de datos SQLite de Android. Para ello se tendrían que mantener esos datos en una cola, esperando a establecer una conexión a internet. En el momento que se posea conexión en el sistema, antes de seguir enviando nuevos datos a la base de datos, enviar todos los que se encuentren a la espera de ser enviados. Es decir, cuando se establece la conexión antes de enviar nuevos datos se comprobaría siempre si existen datos a la espera, una vez se compruebe esto y el resultado sea que la cola esta vacía, se pueden seguir almacenando los nuevos datos recogidos por los sensores.

Otra mejora, y en mi opinión la más importante de todas, es la implementación de seguridad en el sistema. De esta forma se evitarían muchos problemas, desde accesos no permitidos a la aplicación hasta adulterar los datos para conseguir fines maliciosos. También sería útil para proteger la información de cada usuario, ya que se guardan los nombres de usuarios con su contraseña y la matrícula del coche que posee, por ello lo más conveniente sería la implementación de mecanismos para la correcta autenticación. Todas las medidas de seguridad serían necesarias en el sistema completo, formado por aplicación Android, context broker y servicio web, para ello sería

imprescindible la implementación del cifrado en la comunicación entre ellos.

Además de las anteriores mejoras, también sería interesante el poder suscribirse más de un usuario al mismo vehículo a la vez. Mientras viajan en el mismo vehículo, y estar monitorizando los datos de los sensores al mismo tiempo.

ANEXO A: INSTALACIÓN DE ORION CONTEXT BROKER EN CENTOS A TRAVÉS DE DOCKER

En este primer anexo se muestra como instalar y ejecutar Orion Context Broker a través de Docker en una maquina virtual con CentOS 7.

Docker es una tecnología de creación de contenedores que permite crear y usar contenedores de Linux.

La tecnología Docker usa el kernel de Linux y las funciones de este, como Cgroups y namespaces, para segregar procesos y así poder ejecutarse de manera independiente. El propósito de los contenedores es esta independencia: la capacidad de ejecutar varios procesos y aplicaciones por separado haciendo un mejor uso de su infraestructura y, a la vez, mantener la seguridad que tendría con sistemas separados.

Las herramientas desarolladas a partir de los contenedores de Linux, hace a Docker fácil de usar y único, otorgan a los usuarios acceso a las aplicaciones, control sobre las versiones y su distribución y capacidad de implementar de forma rápida.

Docker tiene numerosas ventajas como la modularidad, control de versiones de imágenes y capas, restauración e implementación rápida.

La tecnología Docker es un enfoque más granular y controlable, basado en microservicios, que da prioridad la eficiencia.

Para poder instalar Docker en CentOS 7 se necesitan los siguientes prerequisitos:

- Versión mantenida/compatible de CentOS, Docker no acepta versiones obsoletas.
- Cuenta de usuario con privilegios de superusuario.
- Acceso a la terminal
- Instalador de paquetes de software yum

A.1 Instalación de Docker

Ejecutar los siguientes comandos:

```
$ yum check-update
$ yum install -y yum-utils device-mapper-persistent-data lvm2
$ yum -config-manager –add-repo https://download.docker.com/linux/centos/docker-ce.repo
$ yum install docker
```

Administrar el servicio Docker y comprobar que se ha instalado correctamente:

```
$ systemctl start docker
$ systemctl enable docker
$ systemctl status docker
```

A.2 Ejecución del Context Broker

Para ejecutar el Context Broker en la terminal de la máquina virtual CentOS, se realizan estos comandos:

```
$ sudo docker pull mongo:3.6
$ sudo docker pull fiware/orion
$ sudo docker network create fiware_default
$ sudo docker run -d --name=mongo-db --network=fiware_default \
  --expose=27017 mongo:3.6 --bind_ip_all --smallfiles
$ sudo docker run -d --name fiware-orion -h orion --network=fiware_default \
  -p 1026:1026 fiware/orion -dbhost mongo-db
```

Una vez ejecutados todos estos comandos, comprobamos si el Context Broker se ha iniciado correctamente mediante:

```
$ curl -X GET \
  'http://localhost:1026/version'
```

Debe aparecer un resultado similar al de Ilustración 69.

```
[root@192 ~]# curl -X GET \
> 'http://localhost:1026/version'
{
"orion" : {
  "version" : "2.3.0-next",
  "uptime" : "13 d, 2 h, 50 m, 15 s",
  "git_hash" : "163a46841c32decb63eaf6b377eae17e11e2bbc1",
  "compile_time" : "Fri Feb 14 10:49:11 UTC 2020",
  "compiled_by" : "root",
  "compiled_in" : "95e0a899e2e7",
  "release_date" : "Fri Feb 14 10:49:11 UTC 2020",
  "doc" : "https://fiware-orion.rtfd.io/"
}
}
```

Ilustración 69: Ejecución Context Broker

A.3 Detención del Context Broker

Para parar la ejecución del context broker realizamos los siguientes comandos:

```
$ sudo docker stop fiware-orion
$ sudo docker rm fiware-orion
$ sudo docker stop mongo-db
$ sudo docker rm mongo-db
$ sudo docker network rm fiware_default
```

Tras ejecutar los comandos anteriores se comprueba que la conexión se rechaza, como se observa en la Ilustración 70.

```
[root@192 ~]# sudo docker stop fiware-orion
fiware-orion
[root@192 ~]# sudo docker rm fiware-orion
fiware-orion
[root@192 ~]# sudo docker stop mongo-db
mongo-db
[root@192 ~]# sudo docker rm mongo-db
mongo-db
[root@192 ~]# sudo docker network rm fiware_default
fiware_default
[root@192 ~]# curl -X GET 'http://localhost:1026/version'
curl: (7) Failed connect to localhost:1026; Connection refused
[root@192 ~]#
```

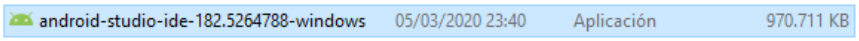
Ilustración 70: Detención del Context Broker

ANEXO B: INSTALACIÓN Y CONFIGURACIÓN DE ANDROID STUDIO

En este anexo se tratará la instalación y configuración de Android Studio en su versión 3.3.1 en el sistema operativo Windows 10.

B.1 Instalación de Android Studio

1. Acceder a la url *https://developer.android.com/studio* y descargar el fichero ejecutable de la versión.
2. Una vez descargado el fichero ejecutable, se lanza y aparece el instalador de Android Studio:



3. En los pasos de la instalación, seleccionar:
    a. Android Virtual Device como componente a instalar (Ilustración 71).

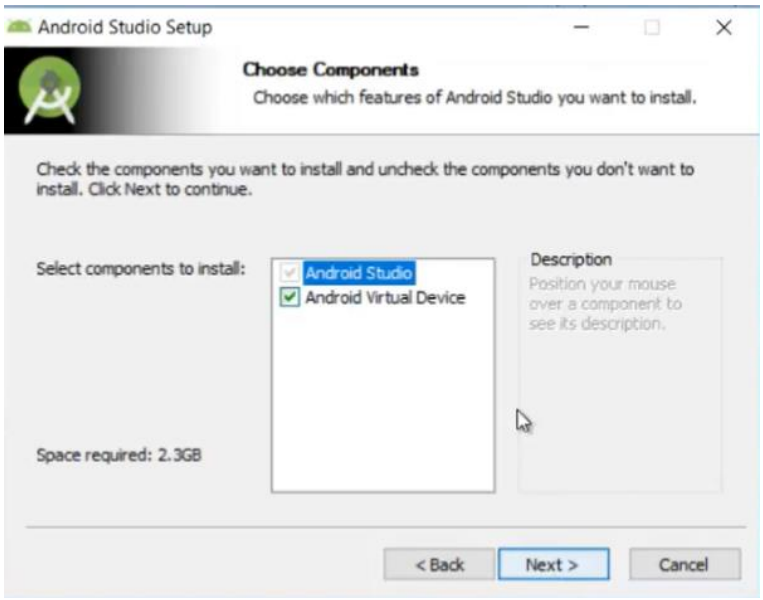


Ilustración 71: Instalación Android Studio – Paso 1

b. El directorio donde se instalará Android Studio (Ilustración 72).

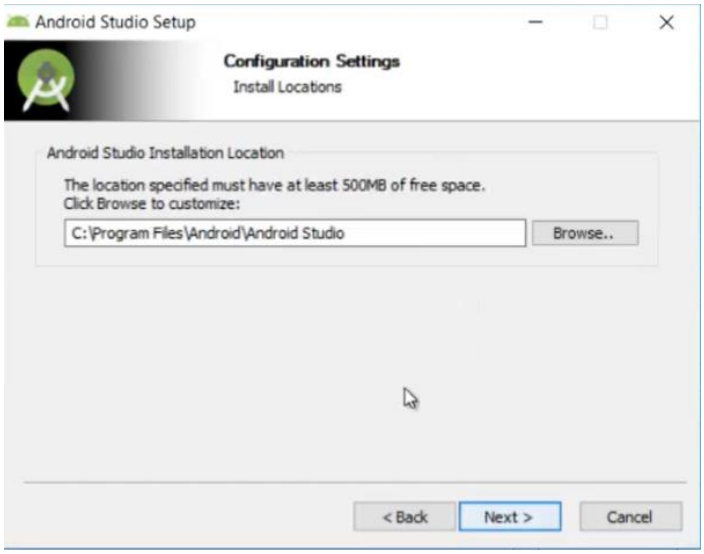


Ilustración 72: Instalación Android Studio – Paso 2

c. La carpeta del menú Inicio (Ilustración 73).

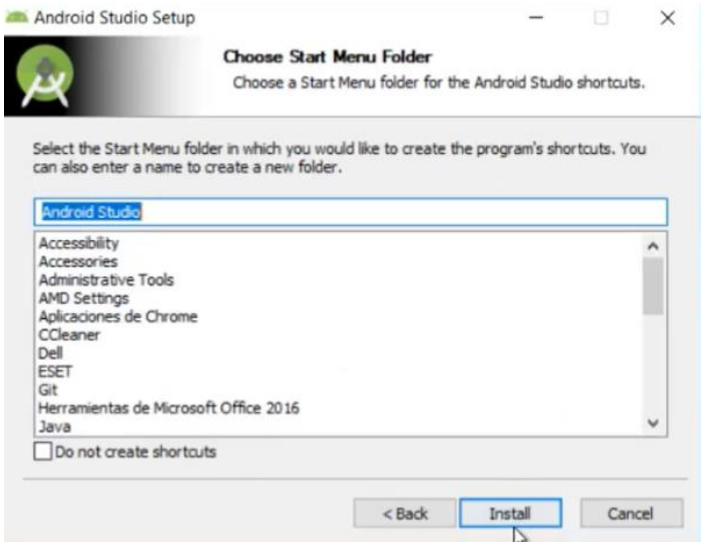


Ilustración 73: Instalación Android Studio – Paso 3

4. Tras esperar el tiempo de la instalación, aparece una pantalla, como se muestra en la Ilustracion 74, donde se marca la opción *Do not import settings.*

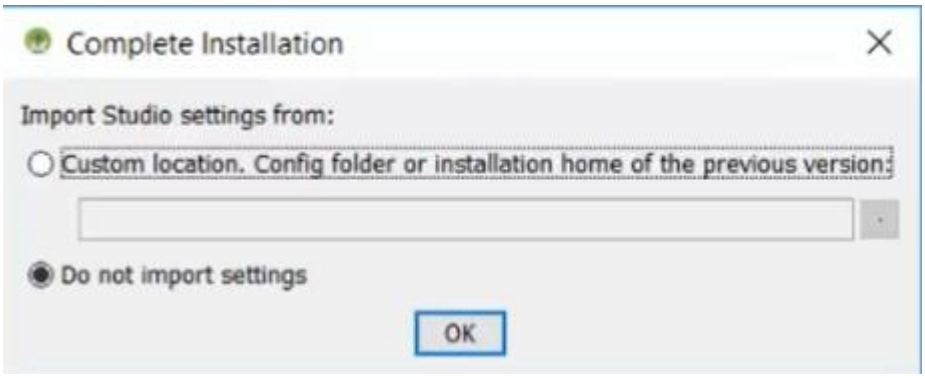


Ilustración 74: Instalación Android Studio – Paso 4

B.2 Configuración de Android Studio

1. Si la instalación se ha realizado correctamente, se inicia automáticamente el programa Android Studio, en la primera ventana que aparece (Ilustración 75) se elige el tipo de configuración. Aparecen las opciones *Standard* y *Custom*, en este caso se elige *Standard*.

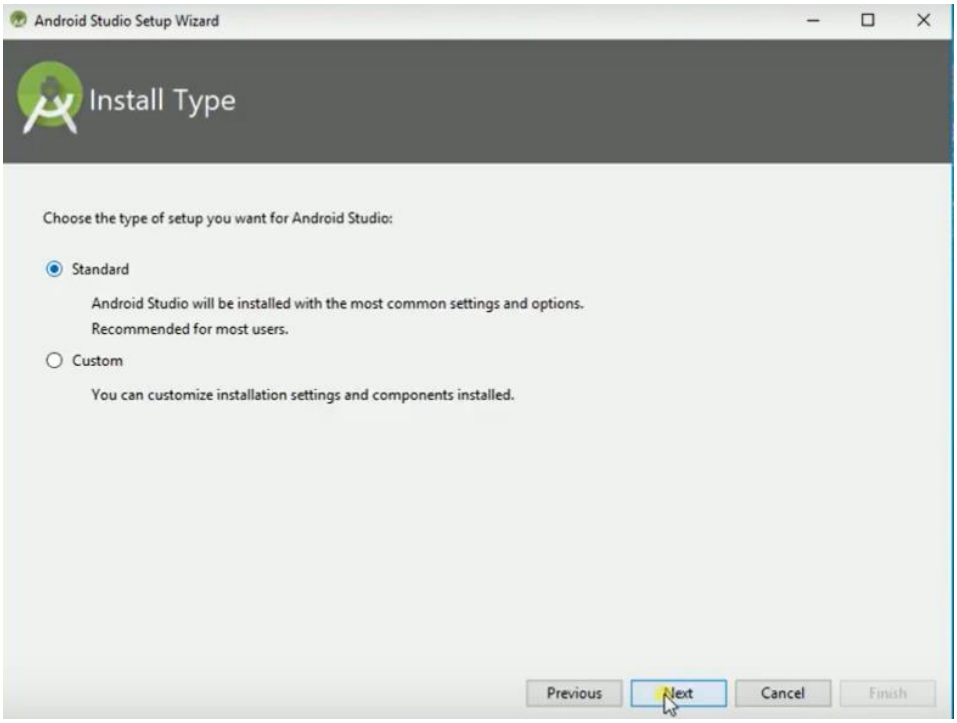


Ilustración 75: Configuración Android Studio – Paso 1

2. Al pulsar en *Siguiente*, aparece la elección del tema (Ilustración 76).

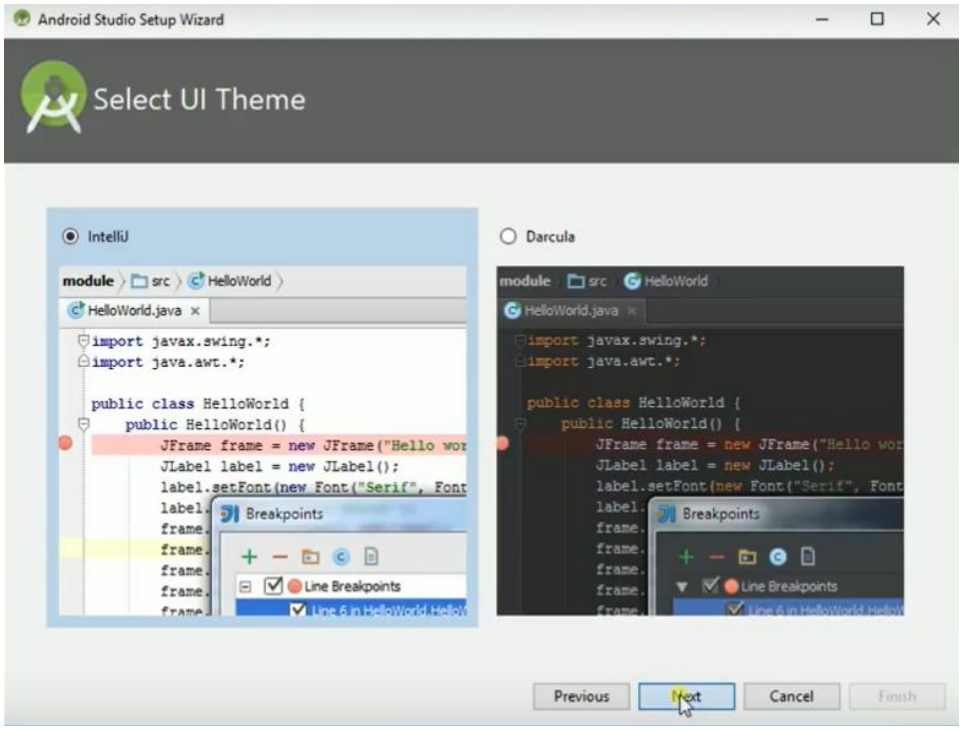


Ilustración 76: Configuración Android Studio – Paso 2

3. Por último, aparece una ventana con la verificación de la configuración establecida y comienza la descarga de los componentes necesarios (Ilustración 77).

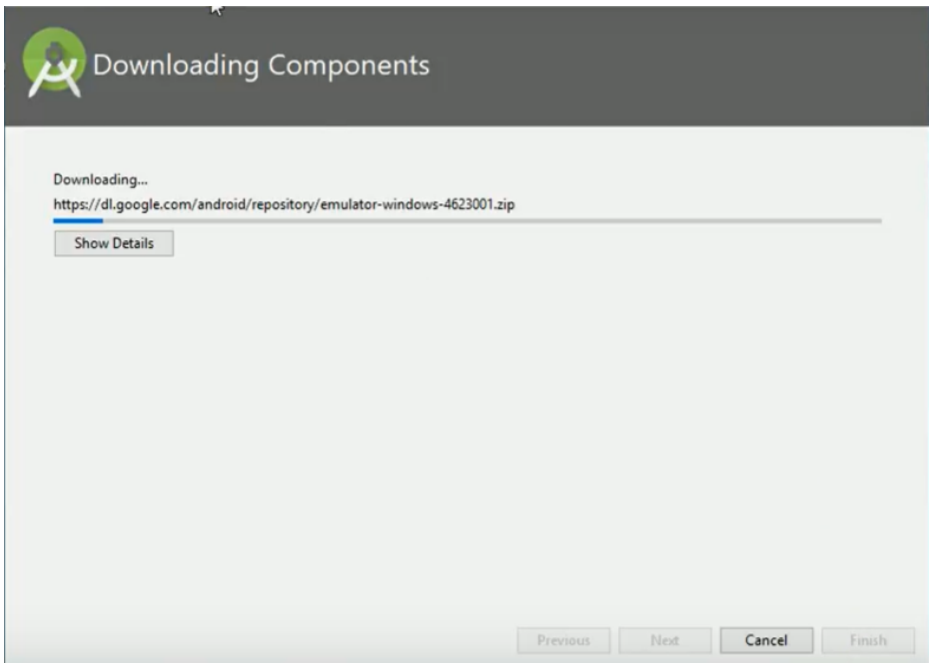


Ilustración 77: Configuración Android Studio – Paso 3

4. Por último, aparece una ventana como la que se observa en la Ilustración 78, en la que ya se puede iniciar un nuevo proyecto, importarlo, …

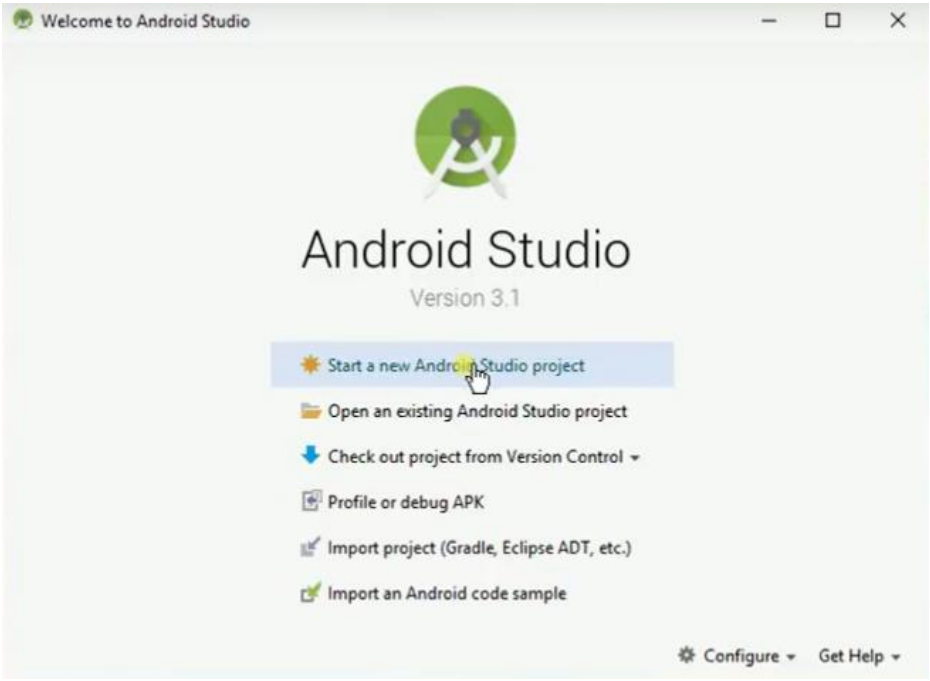


Ilustración 78: Configuración Android Studio – Paso 4

ANEXO C: INSTALACIÓN Y CONFIGURACIÓN DE SPRING TOOL SUITE

En este anexo se muestran los pasos a seguir para la instalación y configuración de Spring Tool Suite versión 3.9.7 en el sistema operativo Window 10.

C.1 Instalación de STS

1. Acceder a la siguiente url de Spring Tools:

   https://spring.io/tools

2. Seleccionar el sistema operativo y formato que se desea y pulsar en descargar
3. En este caso es un versión portable, al descomprimir el fichero descargado se crea el directorio *sts-bundle*, que contiene la carpeta *sts-3.9.7.RELEASE* y esta a su vez el ejecutable del programa (Ilustración 79).

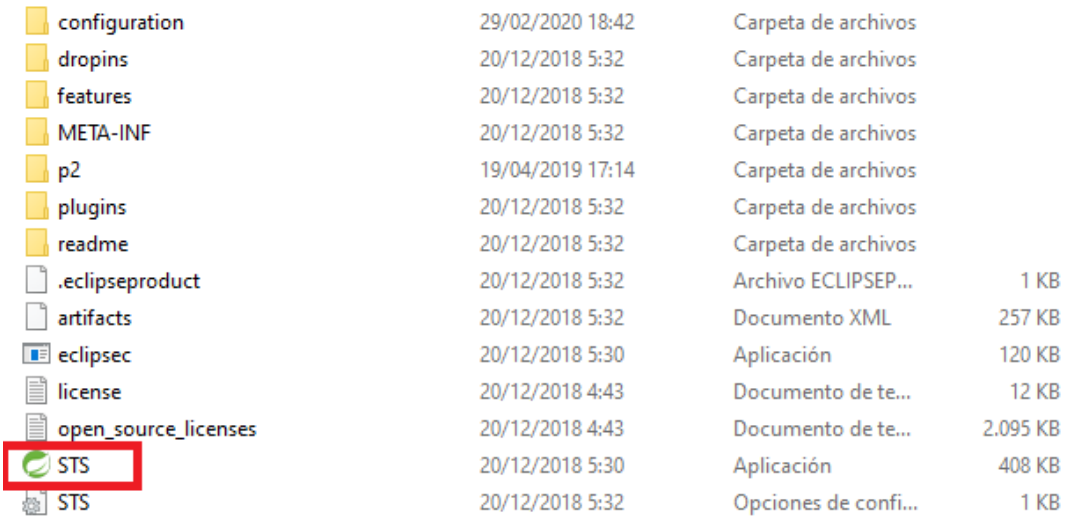


Ilustración 79: Fichero de instalación STS

4. Al lanzar el ejecutable, la primera vez, pide que se seleccione un directorio de trabajo para el proyecto.

C.2 Instalación y Configuración de Hibernate Tools

Cuando ya se ha instalado STS, se puede instalar y configurar Hibernate Tools, la cual es una potente herramienta de persistencia para mapear clases en una base de datos relacional.

1. Se accede desde dentro del entorno, pulsando en la pestaña *Help* y luego en la opción Eclipse *Marketplace…*
2. Aparece una nueva ventana donde se realiza una búsqueda por *Hibernate Tools*. De todos los resultados que aparecen se escoge el que se llama *Jboss Tools*, ya que este paquete contiene Hibernate Tools entre otros.
3. Una vez instalado, se reinicia el equipo para aplicar cambios.
4. Ya solo falta aplicar la perspectiva de Hibernate para crear y editar los ficheros que lo utilizan. Se accede a *Window→Perspective→OpenPerspective→Other…* y ahí seleccionar la opción *Hibernate*.

ANEXO D: CONFIGURACIÓN DE LA BASE DE DATOS MYSQL

D.1 Instalación de MySQL Community

Se instala la versión 8.0.19.0 de MySQL Community en el sistema operativo Windows 10.

1. Descargar desde la url *https://dev.mysql.com/downloads/installer/,* como aparece en la Ilustración 80, el paquete de Windows para instalar MySQL Community en nuestro sistema.

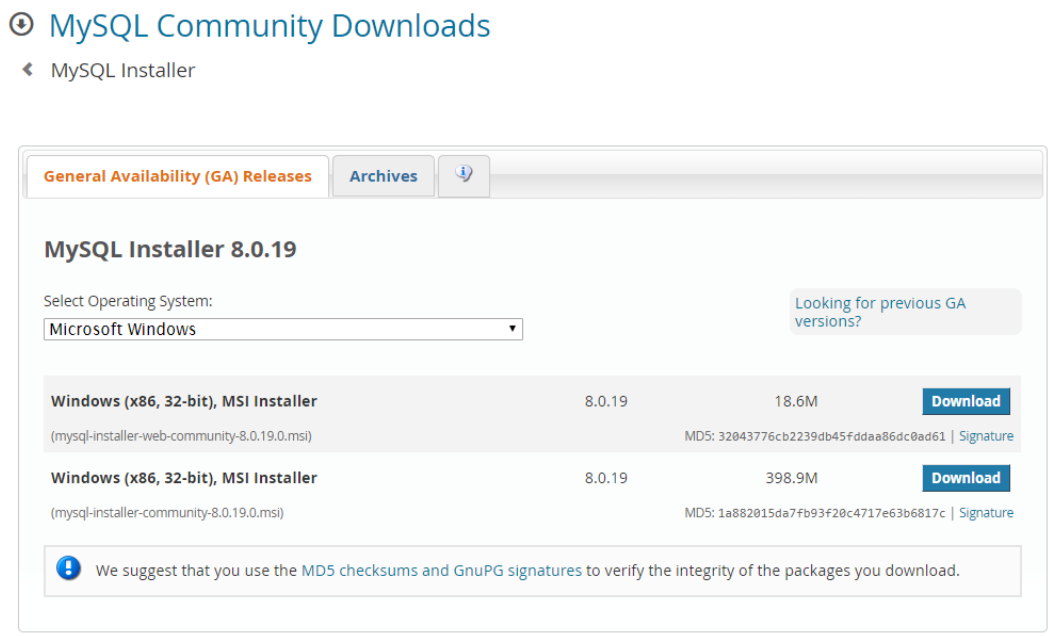


Ilustración 80: Descarga de MySQL Community

2. Lanzar el instalador, y seguir los siguientes pasos:
   a. Aceptar los términos de licencia.
   b. Seleccionar *Developer Default* como el tipo de configuración.
   c. Chequear los requerimientos y descargarlos en caso de que sean necesarios para seguir adelante en la instalación.
   d. Aparece la ventana de instalación donde se instalarán los productos de MySQL.
   e. Configurar el producto, en este caso se configura MySQL Server siguiendo estos pasos:
      i. Establecer el tipo y las redes, donde se selecciona *Development Machine* como tipo de configuración, el puerto 3306 y la opción TCP/IP marcada.
      ii. Establecer la contraseña para la cuenta root de MySQL.
      iii. La ventana de servicio de Windows mantenerla con los valores por defecto.

iv. Seleccionar *Execute* en la ventana de aplicar configuración y una vez sean aplicados todos los cambios aparece un mensaje en el sistema notificando que el estado del servicio MySQL ha cambiado a *Running*.

f. Configurar la conexión al servidor:

i. Seleccionar la opción MySQL Server

ii. Introducir el usuario root y la contraseña que se configuro de MySQL Server en los pasos anteriores.

iii. Pulsar en *Check* para comprobar que la conexión se establece correctamente.

iv. Seleccionar *Execute* en la ventana de aplicar configuración.

g. Al realizar todos estos pasos, tal y como se muestra en la Ilustración 81, aparece la ventana final del instalador con el mensaje de Instalación completa, pulsamos en *Finish*.

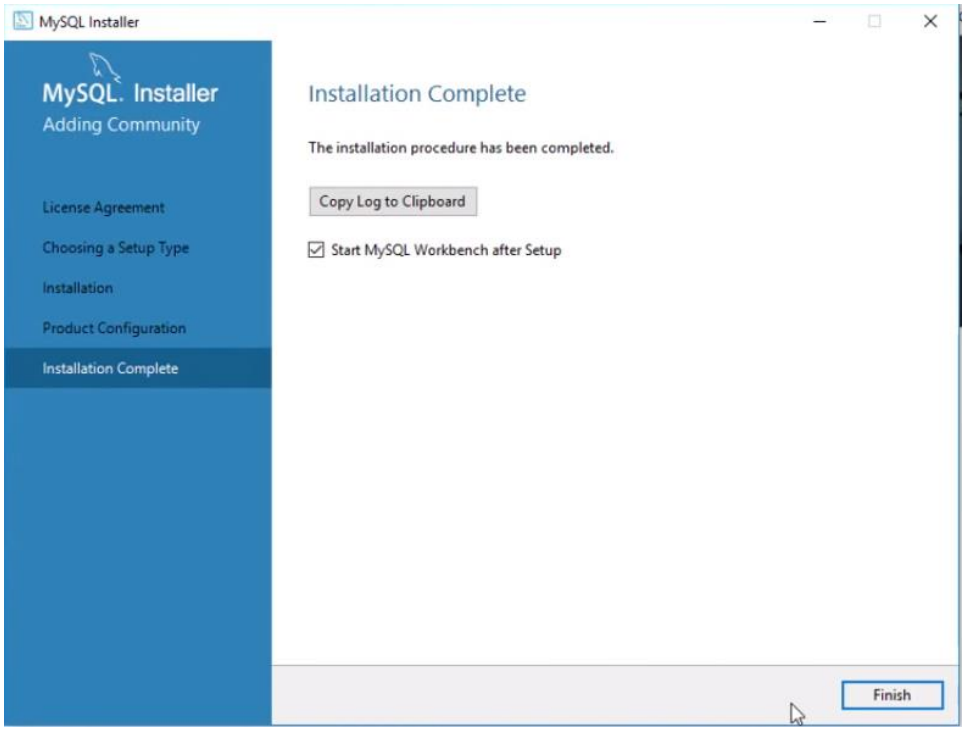


Ilustración 81: Instalación MySQL Community finalizada

D.2 Instalación de XAMPP

XAMPP es una distribución de Apache que incluye softwares libres, en nuestro caso se utiliza MySQL.

Para su instalación se siguen los siguientes pasos:

1. Descargar Xampp (Ilustración 82) de la siguiente url *https://www.apachefriends.org/es/download.html*

Ilustración 82: Descarga de Xampp

2. Ejecutar el ejecutable descargado, en la primera ventana que aparece seleccionamos todos los componentes disponibles, ya deberían estar marcados por defecto.
3. Seleccionar la carpeta contenedora, pulsamos varias veces en Siguiente y comienza la instalación.
4. Si aparece un mensaje emergente de Firewall de Windows Defender seleccionamos *Permitir acceso*.

5. Una vez haya finalizado la instalación pulsamos en Finish y se lanza Xampp automáticamente. Seleccionamos el idioma y aparece el panel de control, mostrado en la Ilustración 83, desde donde podemos iniciar, parar y administrar todos los servicios.

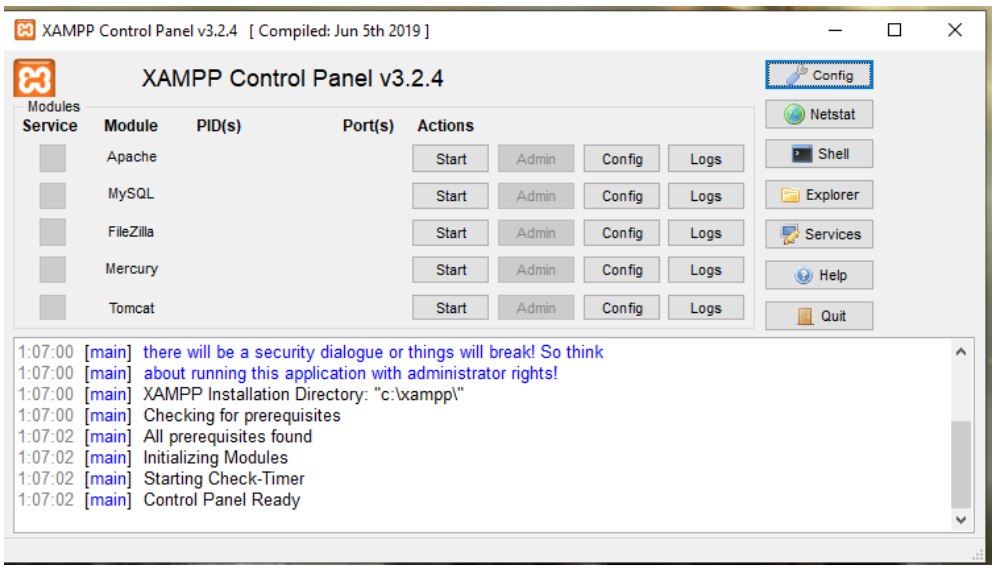


Ilustración 83: Panel de Control de Xampp

D.3 Configuración de phpMyAdmin

Para manejar la administración de MySQL se utiliza la herramienta phpMyAdmin, para ello se siguen estos pasos:

1. En el fichero *config.inc.php* situado en *c:/xampp/phpmyadmin,* modificar el parámetro *$cfg['Servers'][$i]['auth_type']* de *config* a *cookie*.

```
/* Authentication type and info */
$cfg['Servers'][$i]['auth_type'] = 'cookie';
```

2. Desde el panel de control de Xampp se inicia Apache y MySQL.
3. Pulsar *Admin* en MySQL (Ilustración 84) y lanza una nueva ventana en el navegador con la pagina de inicio de sesión de phpMyAdmin.

Ilustración 84: Opciones de MySQL en el panel de control

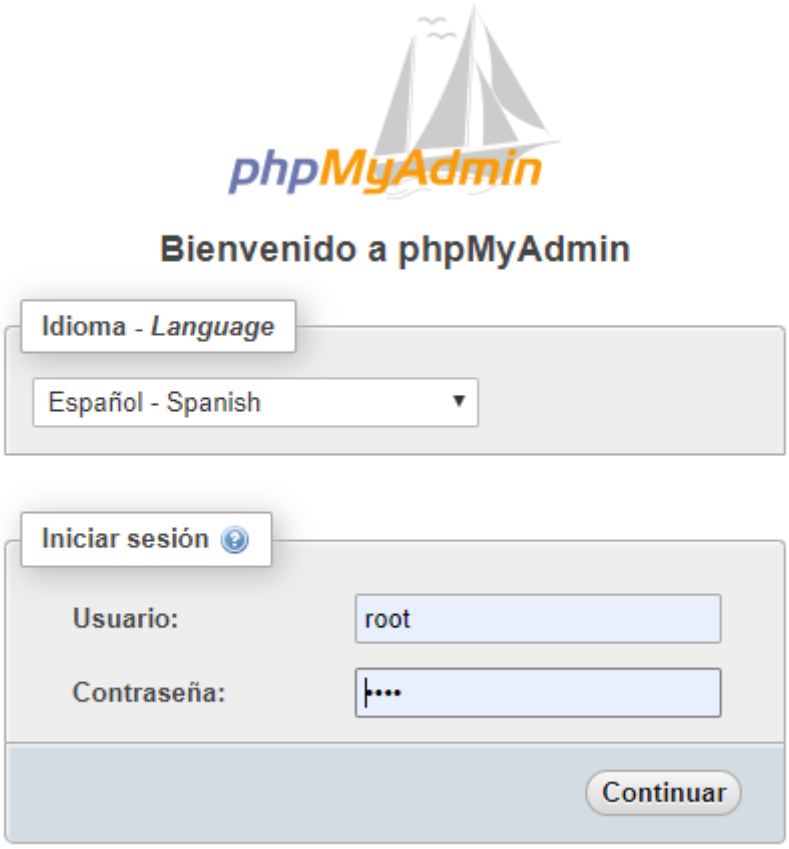


Ilustración 85: Inicio de sesión en phpMyAdmin

4. Intoducir el usuario y contraseña (Ilustración 85) y ya se puede acceder para crear, borrar y administrar bases de datos, tablas y campos, como se observa en la Ilustración 86.

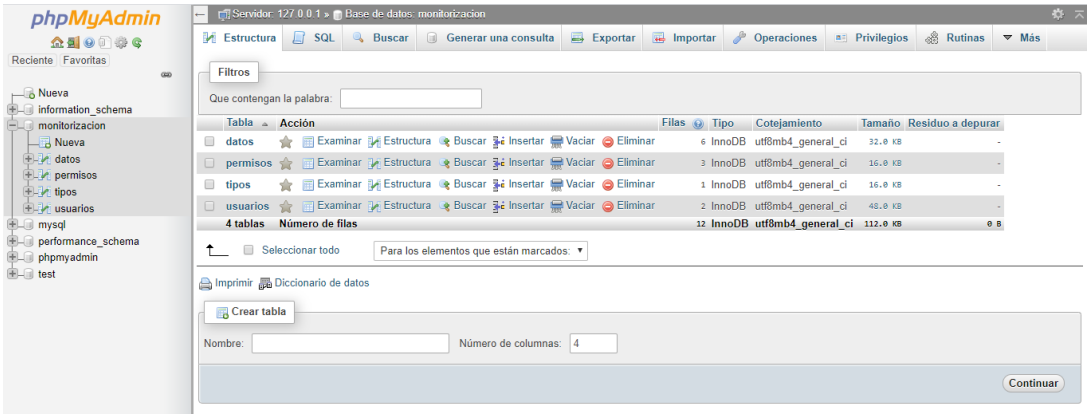


Ilustración 86: Visualización de las tablas de la base de datos

ANEXO E: PUESTA EN MARCHA Y EJECUCIÓN DEL PROYECTO

Para poner en marcha el context broker, se necesita una máquina virtual y el uso de docker. Se siguen los siguientes pasos:

- Descargar e instalar VMware Workstation 15 Player.
- Descargar la máquina virtual CentOS del departamento de ingeniería telemática de la Universidad de Sevilla.
- Arrancar la máquina virtual CentOS descargada desde VMware Workstation 15 Player, como aparece en la Ilustración 87.

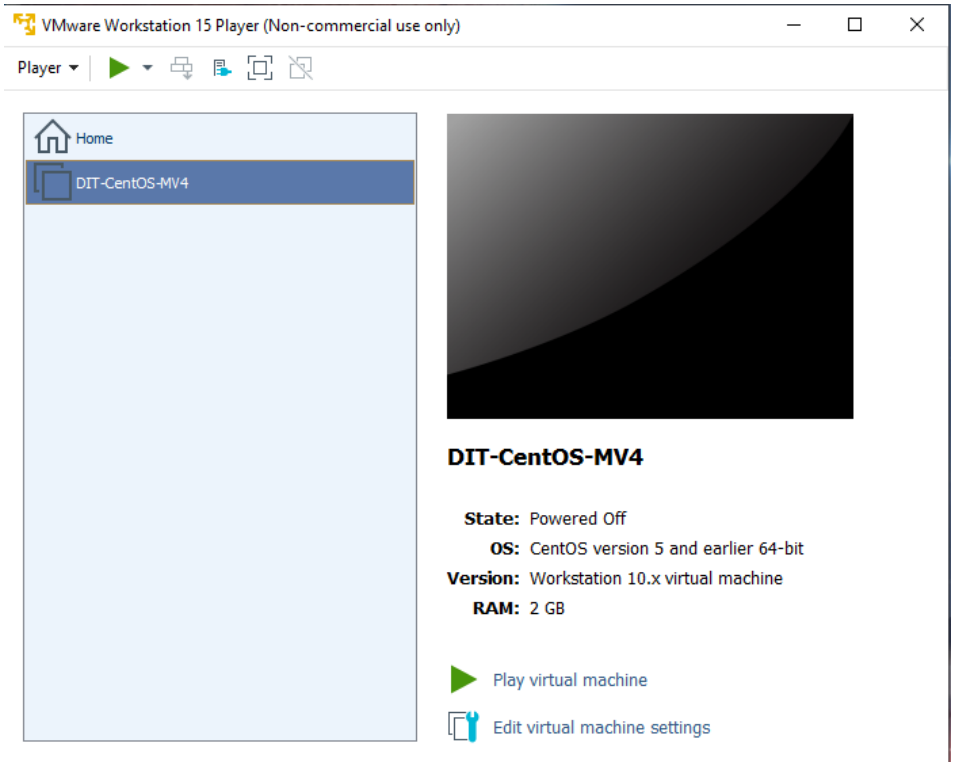


Ilustración 87: Lanzamiento máquina virtual CentOS

- Instalar Docker en la máquina virtual CentOS como se indica en el apartado "A.1 Instalación de Docker" del Anexo A.
- Ejecutar Orion Context Broker en la máquina virtual CentOS, para ello es necesario hacer uso de docker, tal y como se indica en el apartado "A.2 Ejecución del Context Broker" del anexo A.

Una vez seguidos todos estos pasos ya tendríamos el context broker ejecutándose en la máquina virtual. Lo siguiente sería la configuración de la base de datos MySQL. Para ello es necesario hacer uso de XAMPP y phpMyAdmin como se indica en el "Anexo D: Configuración de la base de datos MySQL". Por último, en phpMyAdmin, se ejecuta el fichero *creaTablas.sql* para crear la base de datos "monitorización" y cada una de las tablas necesarias para almacenar todos los valores de la aplicación.

A continuación, hay que ejecutar el servicio web Spring, para ello se siguen los siguientes pasos:

- Se instala y configura Spring Tool Suite como se indica en el "Anexo C: Instalación y configuración de Spring Tool Suite".
- Para importar el proyecto del servicio web, se arranca Spring Tool Suite y pulsando en *File→Open Projects from File System…* aparece una ventana como la de la Ilustración 88.
- Seleccionamos la ruta donde se encuentre el proyecto y finalizamos.
- Asignamos las ip de nuestro equipo y de la maquina virtual del context broker en el fichero *src/main/resources/application.properties* y *src/main/resources/hibernate.cfg.xml.*
- Por último, pulsamos botón derecho sobre el proyecto en la pestaña de "Project Explorer" y seleccionamos *Run As→Spring Boot App.*

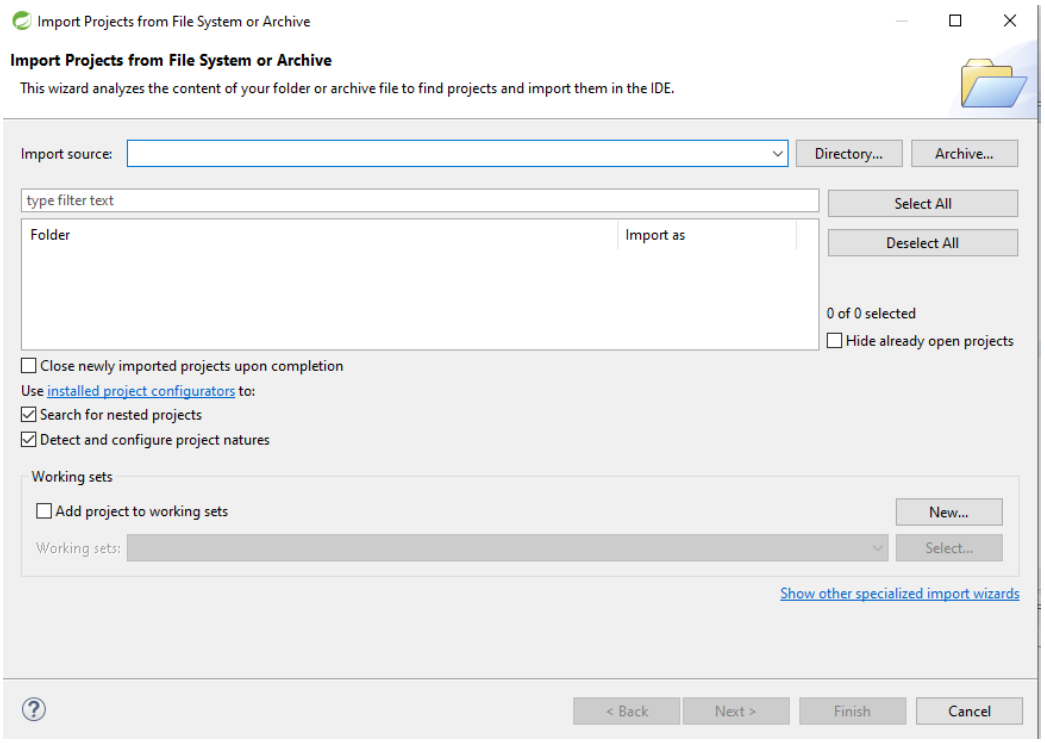


Ilustración 88: Importar proyecto en STS

Para la aplicación Android es necesario lanzarla al dispositivo móvil mediante Android Studio, para ello se siguen los siguientes pasos:

- Descargar, instalar y configurar Android Studio como se indica en el "Anexo B: Instalación y configuración de Android Studio".
- Una vez se tiene arrancado Android Studio, hay que importar el proyecto para ello se pulsa en la opción *Open an existing an Android Studio Project,* se indica el directorio y elegimos el proyecto a importar.
- Para lanzar la aplicación hay dos opciones, en un simulador que proporciona Android Studio o en un dispositivo móvil Android conectado mediante una conexión usb al pc. En el caso del dispositivo móvil, previamente hay que activar la opción de desarrollador en el dispositivo. Como se puede observar en la Ilustración 89, una vez se haya pulsado pulsado en *Run→Run 'app',* aparecen los dispositivos conectados y los simuladores disponibles. Se elige una de las opciones y pulsamos en *OK.*

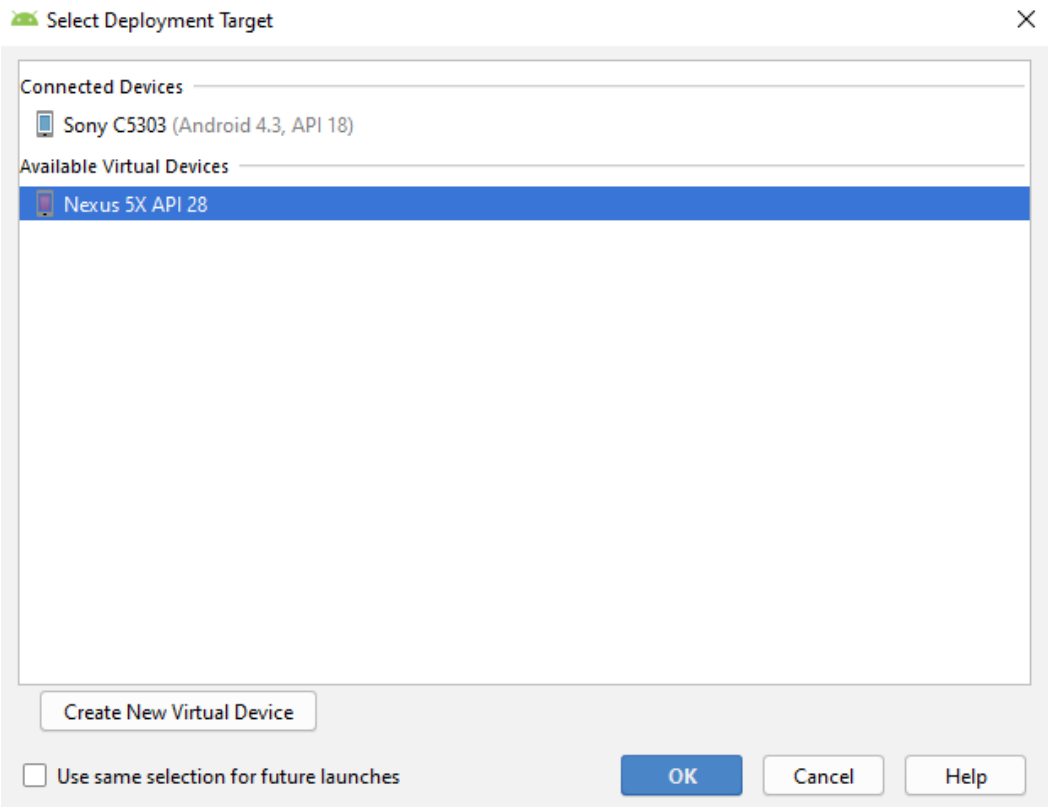


Ilustración 89: Selección dispositivos Android

REFERENCIAS

[1] P. Bermejo Pérez, *Aplicación y Servicio Web para la gestión de información de sensores usando Fiware y Spring*, Sevilla: Universidad de Sevilla, 2019.

[2] Android Studio, <<Android Studio>> [En línea]. Disponible: https://developer.android.com/studio/intro.

[3] Xampp, <<¿Qué es XAMPP?>> [En línea]. Disponible: https://www.apachefriends.org/es/index.html.

[4] Spring, <<Why Spring>> [En línea]. Disponible: https://spring.io/why-spring.

[5] Hibernate, <<Hibernate – Everything data>> [En línea]. Disponible: https://hibernate.org/.

[6] Docker, <<What is a Container?>> [En línea]. Disponible: https://www.docker.com/resources/what-container.

[7] MySQL, <<General Information>> [En línea]. Disponible: https://dev.mysql.com/doc/refman/8.0/en/introduction.html.

[8] phpMyAdmin, <<phpMyAdmin - About>> [En línea]. Disponible: https://www.phpmyadmin.net/.

[9] Mapbox, <<How Mapbox works>> [En línea]. Disponible: https://docs.mapbox.com/help/how-mapbox-works/.

[10] SQLite, <<What Is SQLite?>> [En línea]. Disponible: https://www.sqlite.org/index.html.

[11] MongoDB, <<What is MongoDB?>> [En línea]. Disponible: https://www.mongodb.com/what-is-mongodb.

[12] Fiware Orion Context Broker, <<Orion Context Broker Quick Start Guide>> [En línea]. Disponible: https://fiware-orion.readthedocs.io/en/master/quick_start_guide/index.html.

[13] Dirección General de Tráfico, <<Límites de velocidad por carretera en España>> [En línea]. Disponible: https://itv.com.es/limites-de-velocidad-por-carretera.

[14] Fiware Orion Context Broker, <<Fiware NGSI APIv2 Walkthrough>> [En línea]. Disponible: https://fiware-orion.readthedocs.io/en/master/user/walkthrough_apiv2/index.html.

[15] Fiware Orion Context Broker, <<Oneshoot Subscription>> [En línea]. Disponible: https://fiware-orion.readthedocs.io/en/master/user/oneshot_subscription/index.html.

[16] Fiware Orion Context Broker, <<Using Regular Expressions in Payloads>> [En línea]. Disponible: https://fiware-orion.readthedocs.io/en/1.4.0/user/regex_in_payload/index.html.

[17] Fiware Orion Context Broker, <<Initial Notification>> [En línea]. Disponible: https://fiware-orion.readthedocs.io/en/master/user/initial_notification/index.html.

[18] Android, <<Pickers>> [En línea]. Disponible: https://developer.android.com/guide/topics/ui/controls/pickers.

[19] Android, <<SharedPreferences>> [En línea]. Disponible: https://developer.android.com/reference/android/content/SharedPreferences.

[20] Android, <<HelloCharts for Android>> [En línea]. Disponible: https://github.com/lecho/hellocharts-android.

[21] Mapbox, <<Draw a GeoJSON line>> [En línea]. Disponible: https://docs.mapbox.com/android/maps/examples/draw-a-geojson-line/.

[22] Android, <<HttpURLConnection>> [En línea]. Disponible: https://developer.android.com/reference/java/net/HttpURLConnection.