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

Aplicación y Servicio web para la gestión de información de sensores usando Fiware y Spring

Autor: Pablo Bermejo Pérez

Tutor: María Teresa Ariza Gómez

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

Sevilla, 2019

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

Aplicación y Servicio web para la gestión de información de sensores usando Fiware y Spring

Autor:
Pablo Bermejo Pérez

Tutor:
María Teresa Ariza Gómez
Profesor titular

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo de Fin de Grado: Aplicación y Servicio web para la gestión de información de sensores usando Fiware y Spring

Autor: Pablo Bermejo Pérez

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, 2019

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

En primer lugar, quisiera agradecer a mi tutora, Dra. doña María Teresa Ariza Gómez, por su colaboración y sus indicaciones en el Proyecto y, sobre todo, por el trato tan cercano que me ha brindado.

También me gustaría agradecer al profesor Dr. don Antonio Jesús Sierra Collado por haberme proporcionado parte de los componentes necesarios para la construcción del dispositivo de ayuda a la conducción.

Por último, agradecer a mis padres y a mi hermana por el apoyo moral y psicológico que me han dado durante la realización de este Trabajo de Fin de Grado.

Pablo Bermejo Pérez

Sevilla, 2019

Resumen

El *Internet of Things* es un concepto que en la actualidad está gozando de una gran inversión en investigación y desarrollo. La idea de que los objetos de la vida cotidiana estén conectados y sean capaces de sentir y comunicarse abre paso a infinitas posibilidades, ejerciendo un fuerte impacto tanto en la vida de las personas como en los negocios. Se prevé que el internet del futuro estará repleto de dispositivos con sensores y procesadores incorporados, superando en número a los convencionales (ordenadores, teléfonos móviles, etc.), y se estiman unas cifras de más de 50.000 millones de dispositivos para el año 2020 [1].

El Proyecto que en este Trabajo se expone es una continuación del proyecto iniciado por Luis Martínez Ruiz en 2018 para su Trabajo de Fin de Grado, el cual se enfocó en la creación de un dispositivo IoT dotado de un sensor GPS y de un acelerómetro con giroscopio que, conectados a una Raspberry Pi, son capaces de transmitir la información que miden a un *context broker*[1].

En el presente Proyecto se ha mejorado el dispositivo inicial de manera que: pueda funcionar de forma autónoma e incorporarse en un vehículo midiendo parámetros como su posición, su velocidad o su aceleración y dirección; que pueda consultar a un servidor de internet la velocidad máxima de la vía por la que circula el vehículo en tiempo real, y que avise al conductor con señales luminosas y acústicas de que se está superando la velocidad máxima de la vía, o de que se está superando la velocidad a la que se activaría un radar de tráfico.

No obstante, aparte de implementar las mejoras mencionadas en el anterior párrafo, este Proyecto se centra en el desarrollo de una aplicación web, bautizada como *Sensorapp*, con la que el usuario podrá suscribirse, mediante un número de matrícula, a la información que el context broker recopile sobre su vehículo, así como consultar los datos recogidos y almacenados, tras la suscripción, en una *base de datos*.

[1] Servidor que hace las veces de "interfaz" entre los dispositivos inteligentes y las aplicaciones de usuario.

Abstract

The Internet of Things is a concept that is currently enjoying a large investment in research and development. The idea of daily life objects being connected and capable of feeling and communicating opens up infinite possibilities, exerting a strong impact both on the lives of people and on businesses. It is expected that the Internet of the future will be full of devices with built-in sensors and processors, outnumbering the conventional ones (computers, mobile phones, etc.), and an amount of 50,000 million devices is estimated for the year 2020 [1].

The Project that is exposed in this Thesis is a continuation of the project initiated by Luis Martínez Ruiz in 2018 for his Bachelor's Degree Thesis, which focused on the creation of an IoT device equipped with a GPS sensor and an accelerometer with gyroscope which, connected to a Raspberry Pi, are capable of transmitting the information they measure to a context broker.

In the present Project, the initial device has been improved so that: it can operate autonomously and be incorporated into a vehicle in order to measure parameters such as its position, its speed or its acceleration and direction; it can query the maximum speed of the road through which the vehicle circulates in real time, from an internet server; it warns the driver, with light and acoustic signals, that the vehicle is exceeding the maximum speed of the road, or that it is exceeding the speed at which a speed radar would be activated.

However, apart from implementing the improvements mentioned in the previous paragraph, this Project focuses on the development of a web application, named Sensorapp, which allows the user to subscribe, through a vehicle registration number, to the information that the context broker collected from their vehicle, as well as query the data collected and stored, after the subscription, in a database.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

1 INTRODUCCIÓN

If your business is not on the internet, then your business will be out of business.

- Bill Gates -

1.1 Motivación

Este Proyecto se ha realizado con la finalidad de diseñar un sistema de ayuda a la conducción, en el contexto del *vehículo conectado*[2]. Este sistema, permitiría a un usuario recopilar y consultar los datos de circulación que recogiese un dispositivo integrado en su vehículo. Además, este dispositivo sería capaz de informar a al conductor, mediante señales luminosas y acústicas, de la velocidad a la que circula con respecto a la velocidad máxima permitida de la vía en la que se encuentra. Para desarrollarlo, se han empleado una serie de tecnologías, con objeto también de indagar en el funcionamiento de las mismas y en las posibilidades que ofrecen.

Una de esas tecnologías es el Fiware Orion Context Broker, desarrollado por Telefónica para el proyecto europeo FIWARE, que implementa una API REST para la comunicación y una estructura de datos basada en JSON, la cual está organizada en entidades que contienen una serie de atributos. Lo más interesante de este context broker es que ofrece la posibilidad de suscribirse a una determinada entidad, y establecer una condición para sus atributos bajo la cual se enviará una notificación, filtrando y reduciendo así la enorme cantidad de datos generada por el internet de las cosas.

Otra de esas tecnologías es Spring, un *framework*[3] basado en Eclipse para el desarrollo de aplicaciones Java, y con el que se ha desarrollado la aplicación web de este Trabajo. Además, Spring es capaz de integrar otras tecnologías de gran interés, como por ejemplo Hibernate, con la que se puede establecer una comunicación con cualquier base de datos, tenga el lenguaje que tenga, mediante la abstracción del lenguaje relacional a un lenguaje orientado a objetos.

1.2 Objetivos

El objetivo del Trabajo es la realización de un Proyecto que implemente un Sistema formado por: dispositivos IoT de ayuda a la conducción, dotados de sensores GPS y acelerómetro/ giroscopio y diseñados para incorporarse en vehículos automóviles; un context broker que reúna la información que envían los dispositivos; una aplicación web con la que el usuario pueda suscribirse a un determinado vehículo y pueda consultar los datos recogidos tras la suscripción, y una base de datos en la que la aplicación web almacena los datos que envía el context broker, mediante notificaciones, en base a las distintas suscripciones de los usuarios.

[2] Vehículo dotado de acceso a internet y, generalmente, de conexión por satélite.
[3] Entorno de trabajo.

1.3 Antecedentes

El proyecto realizado en 2018 por Luis Martínez Ruiz para su Trabajo de Fin de Grado sentó las bases de este Proyecto. En él se seleccionaron e integraron los dos sensores y la Raspberry Pi para conformar el dispositivo de medición y envío de datos. Además, se programaron los dos programas (uno por sensor) que extraen los datos de ambos sensores y los envían al context broker. En este Proyecto se ha continuado desarrollando ese dispositivo y se ha reutilizado el código de dichos programas, ampliándolo para que funcione de acuerdo a los objetivos propuestos.

Aquel Proyecto ya utilizaba Fiware Orion como context broker, sin embargo, no se hacía uso en él del sistema de suscripciones/notificaciones. La obtención de los datos del context broker se llevaba a cabo mediante peticiones GET efectuadas por Freeboard, tecnología que fue utilizada entonces como entorno gráfico de interfaz de usuario. En este Proyecto se ha dejado de utilizar Freeboard y se ha programado (mediante Spring Boot), como sustituto, una aplicación web que es capaz de suscribirse a las entidades del context broker, recibir sus notificaciones, almacenar los datos contenidos en las mismas en una base de datos, y ofrecer una interfaz gráfica con la que el usuario puede consultar y suscribirse a los datos.

1.4 Descripción de la solución

En esta sección se describe de manera general la solución implementada para alcanzar los objetivos propuestos.

1.4.1 Objetivos específicos

- **Obtención y envío de datos de los sensores**

  Como sensor GPS se ha utilizado el sensor u-blox NEO-6M-0-001. Se ha conectado a la Raspberry Pi y se ha instalado GPSD, un demonio[4] que recoge los datos del sensor por la interfaz serie UART. Para realizar y enviar las mediciones, se ha desarrollado un programa en lenguaje Python que, haciendo uso de la librería "gps", pide los datos al demonio GPSD y los envía al context broker. Como acelerómetro y giroscopio se ha empleado el sensor MPU-6050. Este se ha conectado mediante interfaz I2C a la Raspberry Pi, y sus datos se recogen y se envían gracias a otro programa desarrollado en Python. Ambos programas se ejecutan en la Raspberry Pi.

- **Obtención de velocidad máxima de la vía**

  Para ello, el programa mencionado en el apartado anterior para el sensor GPS, envía a un servidor la posición y el tipo de vehículo en cuestión, y le solicita la velocidad máxima de la vía por la que circula en ese momento.

- **Alertas de velocidad**

  En función de la velocidad máxima devuelta por el servidor, el programa calcula si se supera o no la velocidad máxima de la vía o la velocidad de activación de un supuesto radar de tráfico ubicado en dicha vía y, dependiendo del resultado, emite unas señales luminosas y acústicas u otras.

- **Suscripción a los datos de un vehículo y recepción de las notificaciones del context broker**

  Los usuarios se pueden suscribir, por medio de una aplicación web, a la información de un vehículo determinado, especificando su matrícula. La aplicación web enviará la suscripción al context broker y este enviará a la misma las notificaciones, cuya información será almacenada en una base de datos.

- **Almacenamiento y consulta de los datos**

  Se ha utilizado una base de datos PostgreSQL, donde la aplicación web almacena los datos que le llegan de las notificaciones del context broker.

[4] Programa no interactivo que se ejecuta en segundo plano y que proporciona algún tipo de servicio.

1.4.2 Funcionalidades

De cara al usuario final, el Sistema diseñado ofrece las siguientes funcionalidades:

- ✓ Medición de la posición, velocidad y otros parámetros.
- ✓ Alertas en caso de superar la velocidad máxima de la vía.
- ✓ Suscripción a los datos de un vehículo dado por su número de matrícula.
- ✓ Consulta de los datos recopilados durante una suscripción.
- ✓ Consulta de los vehículos suscritos.
- ✓ Consulta de los distintos tipos de vehículos existentes y sus velocidades máximas permitidas.
- ✓ Consulta de estadísticas de viaje para un trayecto concreto.
- ✓ Suscripción a todos los vehículos registrados en el context broker.
- ✓ Cancelar una suscripción dada por su número de identificación de suscripción.

1.4.3 Esquema de la arquitectura

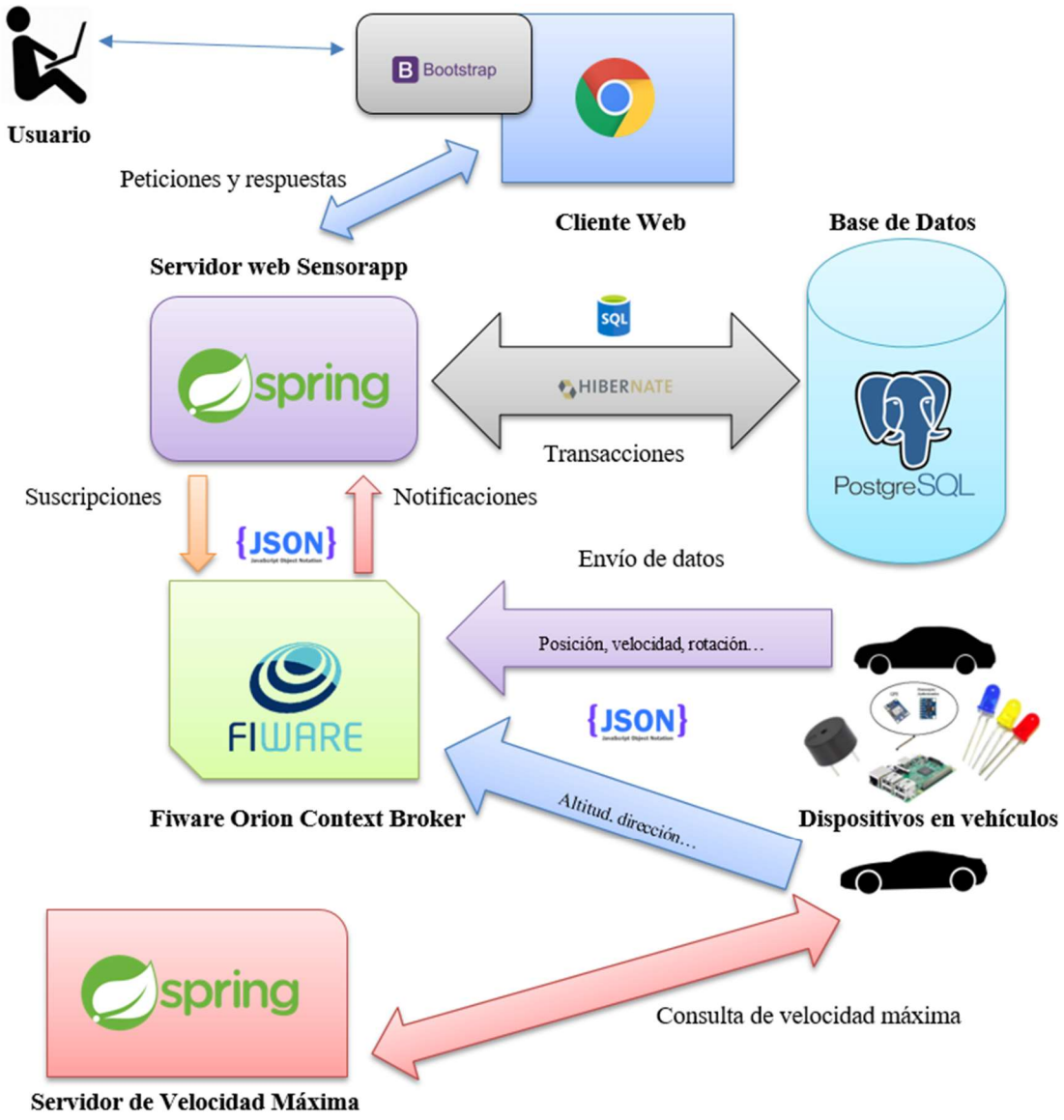

Ilustración 1: Arquitectura del Sistema

1.5 Estructura de la memoria

Con el fin de facilitar la lectura del documento, se proporciona en este apartado un resumen de cada capítulo:

1. **Introducción:** Presentación del problema en su contexto y solución adoptada.
2. **Recursos utilizados:** Recursos software y hardware empleados para alojar la aplicación web, el servidor de base de datos y el context broker, así como para implementar los sensores y su comunicación con el resto del Sistema.
3. **Tecnologías utilizadas:** Resumen de las principales tecnologías empleadas para desarrollar el Proyecto.
4. **Aplicación desarrollada: Dispositivo y Context Broker:** Análisis y características del dispositivo diseñado y del Context Broker Fiware Orion.
5. **Aplicación desarrollada: Servidor Web y Base de Datos:** Estructura de la aplicación web, y análisis de la implementación y de las tecnologías Spring, Hibernate y Bootstrap utilizadas en el Proyecto.
6. **Interfaz de usuario y funcionalidad:** Descripción de las funcionalidades que ofrece la aplicación web de cara al usuario.
7. **Conclusiones y líneas futuras:** Ideas y características que podrían implementarse para mejorar el Sistema en próximas versiones del mismo.

Además de los nueve capítulos expuestos, se proporciona una serie de anexos que servirán como manual para reproducir el Sistema paso a paso:

- **Anexo A:** Instalación de Orion Context Broker utilizando Docker.
- **Anexo B:** Configuración y conexión de sensores con Raspberry Pi.
- **Anexo C:** Instalación y configuración de STS y de Hibernate Tools.
- **Anexo D:** Configuración del servidor de Base de Datos PostgreSQL.
- **Anexo E:** Configuración de accesibilidad del servidor web y el context broker.
- **Anexo F:** Puesta en marcha del Sistema.

2 RECURSOS UTILIZADOS

I do not fear computers. I fear lack of them.

- Isaac Asimov -

Este capítulo se dedicará a exponer los recursos utilizados en el proyecto, tanto hardware como software, detallando sus elementos y características más relevantes.

2.1. Recursos Hardware

2.1.1 Ordenador de escritorio Intel Core i5-4690k

Se ha utilizado un ordenador de escritorio para alojar la aplicación web, el context broker y la base de datos. Sus características son las siguientes:

Ilustración 2: Intel® Core™ i5-4690k

- Procesador: Intel® Core™ i5-4690k CPU @ 3.50GHz (4 CPUs).
- Placa Base: Gigabyte GA-Z97P-D3.
- Memoria RAM: 16 GB.
- Tarjeta gráfica: NVIDIA GeForce GTX 660.
- Almacenamiento: 1TB HDD + 256GB SSD.
- Sistema Operativo: Windows 10 Pro.

2.1.2 Raspberry Pi Model B+

Como núcleo de procesamiento y comunicación del dispositivo IoT, se ha escogido una Raspberry Pi Model B+. A ella se conectarán los sensores, los leds, el buzzer y un adaptador Wi-Fi, ya que este modelo no cuenta con una interfaz Wi-Fi instalada de serie y será necesaria la conexión inalámbrica.

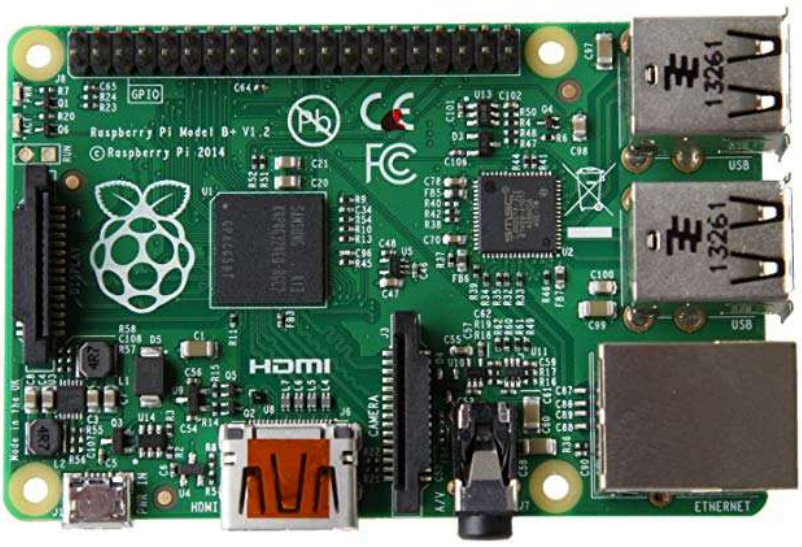

Ilustración 3:Raspberry Pi Model B+

- Procesador:
  - Chipset Broadcom BCM2835.
  - ARM1176JZF-S 700 MHz.
- GPU:
  - Broadcom VideoCore IV 250 MHz.
  - OpenGL ES 2.0.
- RAM: 1GB.
- Conectividad:
  - Ethernet 10/100 Mbit/s (8P8C).
  - Capacidad para adaptador Wi-Fi USB.
  - Salida de vídeo: HDMI Rev. 1.3 y 1.4.
  - Salida de audio: Jack de 3,5 mm.
  - USB 4 x Conector USB 2.0.
  - Conector GPIO:
    - 40 pines de 2,54 mm en 2 tiras de 20 pines.
    - 27 pines GPIO, así como 3,3 V, +5 V y GND y comunicación serie.
  - Conector de 15 pines para cámara MIPI con interfaz en serie (CSI).
  - Conector de 15 pines para display MIPI con interfaz en serie (DSI).
  - Ranura para tarjeta de memoria Micro SD [2].

2.1.3 Adaptador Wi-Fi USB Belkin N150

Adaptador Wi-Fi necesario para la conexión a Internet de la Raspberry Pi.

Ilustración 4: Adaptador Wi-Fi Belkin N150

- N150 - 802.11n/b/g.
- Velocidad: 150 Mbps.
- Dimensiones: 15 mm x 18 mm (ancho x alto).
- Peso: aproximadamente 9 gramos.
- Encriptación: WPA2, WPA, WEP 64/128-bit.
- Compatible con XP (SP2 y superior), Windows Vista (32 y 64bit) y Windows 7 (32 y 64bit).
- WPS (Wi-Fi Protected Setup) [3].

2.1.4 Sensor MPU-6050

El sensor utilizado como giroscopio y acelerómetro es el MPU-6050. Sus características se listan a continuación:

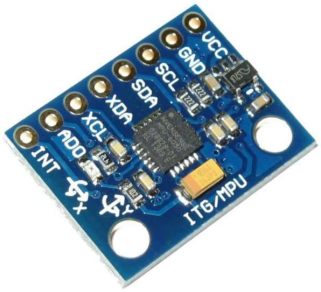

Ilustración 5: Sensor MPU-6050

- Salida digital de 6 ejes.
- Giroscopio con sensibilidad de ±250, ±500, ±1000, y ±2000dps.
- Acelerómetro con sensibilidad de ±2g, ±4g, ±8g y ±16g.
- Algoritmos embebidos para calibración.
- Sensor de temperatura digital.
- Entrada digital de video FSYNC.
- Interrupciones programables.
- Voltaje de alimentación: 2.37 a 3.46V.
- Voltaje lógico: 1.8V±5% o VDD.
- 10000g tolerancia de aceleración máxima [4].

2.1.5 Sensor u-blox NEO-6M-0-001

Este ha sido el sensor escogido para recoger los datos GPS vía satélite.

Ilustración 6: Sensor u-blox NEO-6M-0-001

- Receptor GPS.
- Alimentación: 2,7V a 3,6V.
- Interfaces: UART, USB, SPI, I2C.
- Características:
  - Cristal RTC.
  - 3 pines configurables.
  - Reloj.
  - Interrupciones externas [5].

2.1.6 LEDs y Buzzer

Para emitir las señales luminosas se han utilizado tres leds, de colores azul, amarillo y rojo. Para las señales acústicas, se ha empleado un transductor electro-acústico de dos pines, que emite un pitido continuo cuando se le aplica tensión.

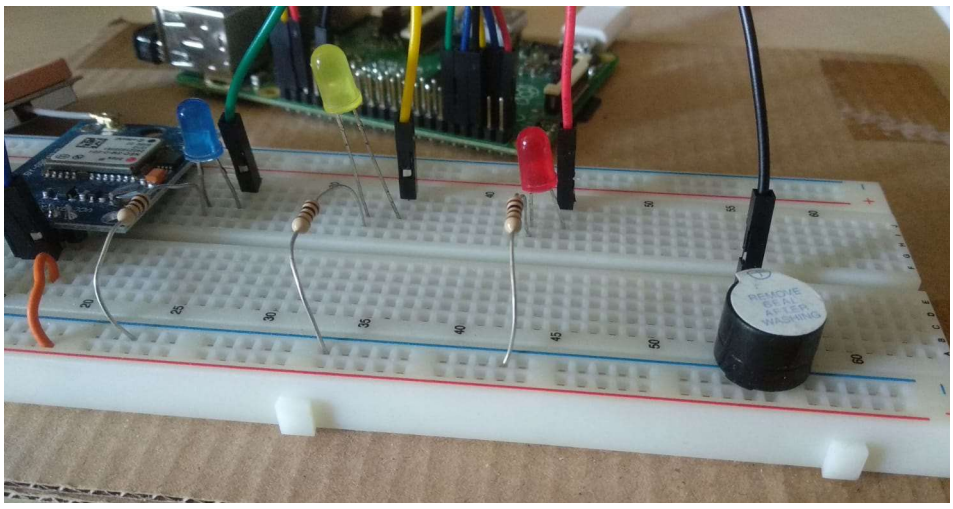

Ilustración 7: LEDs de colores y Buzzer (derecha)

2.2 Recursos Software

2.2.1 Máquina Virtual Ubuntu

Para este proyecto se ha utilizado una máquina virtual Ubuntu 16.04.5 LTS, con kernel Linux 4.15.0-50-generic, la cual es una copia del sistema operativo presente en los equipos de las salas del centro de cálculo de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla. En esta subsección se listan los recursos software utilizados trabajando dentro de la máquina virtual.

2.2.2 Accumulator Server

Se ha utilizado el Accumulator Server para la depuración de las suscripciones y notificaciones del context broker. Esta herramienta ha sido desarrollada por Telefónica I+D para cumplir con ese mismo propósito en el Context Broker Fiware Orion, dentro del proyecto europeo FIWARE. Se trata de un servidor que recibe e imprime por pantalla las notificaciones que le llegan del context broker. Su instalación y uso se verá en el Anexo A: Instalación de Orion Context Broker utilizando Docker.

2.2.3 Navegador Web Google Chrome

Ilustración 8: Google Chrome

Como cliente para acceder a la aplicación web se ha utilizado el navegador Google Chrome, con la extensión JSONView para una mejor visualización de los datos en formato JSON.

3 TECNOLOGÍAS UTILIZADAS

Technology is anything that wasn't around when you were born.

- Alan Kay -

Este capítulo es una introducción a las principales tecnologías utilizadas para la implementación del Sistema.

3.1 Docker y MongoDB

Para alojar el context broker que recopila los datos de los sensores se ha utilizado Docker como contenedor de software. Su instalación se detalla en el Anexo A: Instalación de Orion Context Broker utilizando Docker

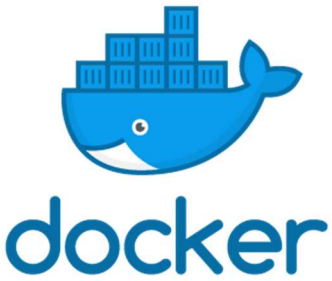

Ilustración 9: Docker

Un contenedor de software es una unidad estándar de software que empaqueta un código y todas sus dependencias, de forma que la aplicación se ejecuta fácilmente y de la misma manera en cualquier entorno (portabilidad). Un contenedor Docker incluye todo lo necesario para ejecutar una aplicación: código, entorno de ejecución, herramientas del sistema, librerías y configuración. El motor Docker es el contenedor de facto en la industria del software, el cual está disponible en varias distribuciones de Linux (CentOS, Debian, Fedora, Oracle Linux, RHEL, SUSE y Ubuntu) y en sistemas operativos Windows Server [6].

Como base de datos en la que se almacenan temporalmente las entidades con los datos recogidos se ha utilizado MongoDB, también dentro del contenedor Docker.

Ilustración 10: MongoDB

MongoDB es una base de datos gratuita no relacional que almacena datos en documentos JSON flexibles, es decir, sus campos pueden variar de un documento a otro y la estructura de los datos puede cambiarse a lo largo del tiempo. El modelo de los documentos se puede mapear a objetos en el código de aplicación, lo que facilita el manejo de los datos [7].

3.2 Fiware Orion Context Broker

Fiware Orion Context Broker ha sido el context broker elegido para recopilar los datos de los sensores GPS y Acelerómetro. Se trata de un proyecto de código abierto desarrollado por el departamento de I+D de Telefónica y que forma parte del proyecto europeo FIWARE. Es una implementación en C++ de la API REST NGSIv2 dentro de FIWARE.

Ilustración 11: Fiware Orion Context Broker

Orion Context Broker permite gestionar al completo el ciclo de vida de la información contextual, incluyendo actualizaciones, peticiones, registros y suscripciones. La información contextual está formada por entidades y sus atributos, los cuales pueden albergar también metadatos (información sobre los datos). Usar Orion Context Broker permite crear elementos contextuales y gestionarlos mediante actualizaciones y peticiones. Además, un usuario puede suscribirse a la información contextual de manera que, cuando ocurra un evento comprendido dentro de una condición, le será enviada una notificación [8].

Su funcionalidad se describirá con más detalles en el Capítulo 4, y su instalación en el Anexo A: Instalación de Orion Context Broker utilizando Docker.

3.3 PostgreSQL

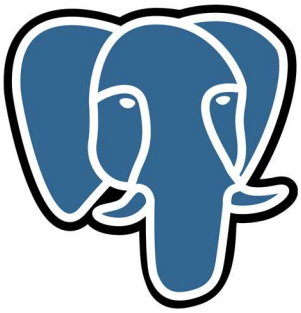

Ilustración 12: PostgreSQL

PostgreSQL es una base de datos relacional, de código abierto, que usa y amplía el lenguaje SQL combinado con otras muchas características que permiten almacenar datos de forma segura y escalable. Sus orígenes se remontan a 1986, donde comienza como parte del proyecto POSTGRES en la Universidad de California en Berkeley, y lleva ya más de 30 años en desarrollo activo. Algunas de sus características son las siguientes [9]:

- Tipos de datos
  - Primitivos: Integer, Numeric, String, Boolean.
  - Estructurados: Date/Time, Array, Range, UUID.
  - Documentos: JSON/JSONB, XML, Key-value (Hstore).
  - Geométricos: Point, Line, Circle, Polygon.
  - Personalizados: Composite, Custom Types.
- Integridad de los datos
  - UNIQUE, NOT NULL
  - Primary Keys
  - Foreign Keys
  - Exclusion Constraints
  - Explicit Locks, Advisory Locks
- Concurrencia
  - Transacciones.
- Seguridad
  - Autenticación: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, Certificado.
  - Control de acceso robusto.
  - Seguridad a nivel de filas y columnas.
- Extensibilidad
  - Procedimientos y funciones almacenadas.
  - Lenguajes procedurales: PL/PGSQL, Perl, Python.
  - Posibilidad de conectar a otra base o flujo de datos con una interfaz SQL estándar.

3.4 Spring

Spring es un framework y un contenedor de *inversión de control*[5] e *inyección de dependencias*[6] para la plataforma de Java. En este Proyecto se ha utilizado Spring para construir la aplicación web Sensorapp. Para su ejecución se ha utilizado Spring Boot, que es un conjunto de herramientas que permiten construir y ejecutar aplicaciones Spring. Las características y funcionalidades de Spring se explicarán con más detalle en el Capítulo 5.

Ilustración 13: Spring

Tanto Spring como Spring Boot están integrados dentro de la aplicación Spring Tool Suite 3. Esta es un entorno de desarrollo basado en Eclipse que facilita el desarrollo de aplicaciones, proporcionando ayudas de lenguaje y de entorno para Spring, y las combina con las existentes para Java, Web y Java EE en Eclipse. Su instalación se detalla en el Anexo C: Instalación y configuración de STS.

3.4.1 Hibernate

Hibernate es una herramienta para la plataforma de Java que permite abstraer la comunicación con una base de datos relacional. Esto se lleva a cabo mediante el *mapeo* o asignación de clases Java a cada una de las tablas de la base de datos, cuyas columnas se asignan a cada uno de los atributos de su correspondiente clase. De esta manera se pueden programar las transacciones con la base de datos mediante un lenguaje orientado a objetos (Java), el cual, gracias al mapeo de clases, se traducirá de forma automática al lenguaje relacional que emplee la base de datos.

Ilustración 14: Hibernate

Hibernate Tools es un set de herramientas para Hibernate integradas en Spring Tool Suite que permite la creación y edición de los ficheros de configuración de Hibernate, necesarios para la comunicación con la base de datos mediante Hibernate. Hibernate Tools es uno de los componentes que conforman el núcleo de JBoss Tools, cuyo proceso de instalación en STS se describe en el Anexo C: Instalación y configuración de STS.

[5] Principio de diseño software en el que el flujo de ejecución del programa no es secuencial, sino que responde a sucesos o solicitudes.
[6] Patrón de diseño de la programación orientada a objetos en el que un objeto o método suministra (inyecta) las dependencias (parámetros de los que depende un objeto) a un objeto dependiente.

3.4.2 Bootstrap

Ilustración 15: Bootsrap

Bootstrap es una librería de componentes front-end[7], de código abierto, para el desarrollo de interfaces gráficas con los lenguajes HTML, CSS y JavaScript. En este Trabajo Bootstrap ha sido utilizado para crear la interfaz gráfica de usuario de la aplicación web Sensorapp. Su implementación se detallará en el Capítulo 5.

[7] Parte de la aplicación visible al usuario final, con la que este puede interactuar.

4 APLICACIÓN DESARROLLADA: DISPOSITIVO Y CONTEXT BROKER

Never trust a computer you can't throw out a window.

- Steve Wozniak –

La arquitectura del Sistema diseñado, tal como se ha adelantado en la subsección 1.4.3: Esquema de la arquitectura, consta de cinco bloques: la aplicación web, la base de datos, el context broker, los dispositivos que contienen los sensores y el servidor de velocidad máxima.

En este capítulo se profundiza en la estructura interna de dos de los bloques constitutivos: el dispositivo integrado en el vehículo y el context broker.

4.1 Dispositivo

El dispositivo diseñado para medir los datos, enviarlos al context broker, consultar la velocidad de la vía y emitir las señales luminosas y acústicas, consta de varios componentes conectados entre sí por medio de una Raspberry Pi, que será la que ejecute los programas que llevan a cabo dichas tareas. Los detalles de la comunicación con los sensores se explican en el Trabajo de Fin de Grado de Luis Martínez Ruiz [2].

4.1.1 Raspberry Pi Model B+

El componente elegido para hacer las veces de procesador lógico e interfaz de comunicaciones del dispositivo IoT, es una Raspberry Pi Model B+, cuyas características se describen en el Capítulo 2. A ella se conectarán los dos sensores y ejecutará los dos programas descritos en las siguientes subsecciones, así como el demonio GPSD. También ha de ser debidamente configurada antes de ejecutar dichos programas. Su configuración y la conexión con el resto de componentes se explica en detalle en el Anexo B: Configuración y conexión de sensores con Raspberry Pi.

4.1.2 Sensor MPU-6050

Este sensor es un acelerómetro y giroscopio, que se conectará mediante interfaz I2C [2] a la Raspberry Pi y del que se extraerán las mediciones de las rotaciones de sus dos ejes:

- Rotación del eje X
- Rotación del eje Y

Para ello se ha diseñado un programa que extrae esos datos del sensor, en forma de variables decimales, y los envía al context broker en el formato apropiado.

4.1.2.1 Programa mpu6050

Este programa ha sido desarrollado en lenguaje Python y al ejecutarlo funciona de la siguiente manera:

1. **Obtención de la dirección IP y el puerto del context broker**

   Esta información está contenida en un fichero de configuración llamado *sensor.conf*, que deberá estar ubicado en el mismo directorio que el programa. La primera línea de este fichero contiene la dirección IP del context broker y la segunda línea, su puerto.

2. **Obtención de la matrícula del vehículo en el que está instalado el dispositivo y el tipo de vehículo del que se trata**

   Estos datos también están guardados dentro del fichero de configuración mencionado. En la tercera línea del mismo estará escrita la matrícula del vehículo y en la cuarta línea su tipo.

3. **Conexión con el context broker**

   La conexión se realiza utilizando la dirección IP y el puerto leídos del fichero *sensor.conf*.

4. **Creación, en el context broker, de una entidad vacía**

   Esta entidad se creará mediante una petición POST al context broker, enviando la matrícula del vehículo como atributo "id" de la entidad y el tipo de vehículo como atributo "type" de la entidad. Para el resto de atributos el valor enviado será 0. Si la entidad ya existe, no provoca ningún cambio en el context broker. La entidad que se debe crear, así como sus distintos atributos y sus tipos de datos, se detalla en la Sección 4.2.1 de este capítulo. El mensaje de creación de entidad es igual en ambos programas y su estructura se muestra en la siguiente ilustración:

```
# Realizamos la conexión con el Context Broker
orion_con = httplib.HTTPConnection(TCP_IP_ORION, TCP_Puerto_ORION)
headers = {"Content-type": "application/json", "Accept": "application/json"}
data = json.loads('{"id" : "'+matricula+'", "type" : "'+tipo+'", '
        + '"latitud" : {"value" : 0, "type" : "Float"}, '
        + '"longitud" : {"value" : 0, "type" : "Float"}, '
        + '"altitud" : {"value" : 0, "type" : "Float"}, '
        + '"velocidad" : {"value" : 0, "type" : "Float"}, '
        + '"direccion" : {"value" : 0, "type" : "Float"}, '
        + '"subida" : {"value" : 0, "type" : "Float"}, '
        + '"rx" : {"value" : 0, "type" : "Float"}, '
        + '"ry" : {"value" : 0, "type" : "Float"}, '
        + '"fecha" : {"value" : 0, "type" : "String"}, '
        + '"hora" : {"value" : 0, "type" : "String"}}')
# Creamos la entidad la primera vez
try:
 orion_con.request('POST', '/v2/entities', json.dumps(data), headers)
 resp = orion_con.getresponse()
 print resp.read() # En caso de que la entidad ya exista, aparecerá un error indicandotelo
except Exception as e:
 print (e)
```

Ilustración 16: Creación de entidad

5. **Lectura de los datos y envío al context broker**

   Una vez leídas las rotaciones de los ejes X e Y, se almacenan en las variables "rx" y "ry", respectivamente. Estas variables se envían a la entidad creada en el context broker, dentro de los atributos "rx" y "ry" de la misma, mediante una petición PATCH. Además, en el mismo mensaje se envían también la fecha y la hora de la medición, dentro de los atributos "fecha" y "hora" de la entidad. La petición PATCH permite actualizar solo algunos atributos de la entidad, a diferencia de PUT, con la que se deben enviar todos los atributos. Ha de usarse, pues, PATCH porque se desconocen los datos de los atributos correspondientes al sensor GPS.

```
# Envío de datos al Context Broker mediante petición PATCH
 fecha=datetime.datetime.now().strftime("%d/%m/%Y")
 hora=datetime.datetime.now().strftime("%X")
 if math.isnan(valueX) or math.isnan(valueY):
  print ("Error de lectura. Intentándolo otra vez...")
 else:
  data = json.loads('{"rx" : {"value" : ' + str(valueX) + ', "type" : "Float", "metadata": {"unidades": {"value": "Grados eje h
        + '"ry" : {"value" : ' + str(valueY) + ', "type" : "Float", "metadata": {"unidades": {"value": "Grados eje vertical"}}}
        + '"fecha" : {"value" : "'+fecha+'", "type" : "String", "metadata": {"unidades": {"value": "Día/Mes/Año"}}}, '
        + '"hora" : {"value" : "'+hora+'", "type" : "String", "metadata": {"unidades": {"value": "Horas:Minutos:Segundos"}}}}')
  headers = {"Content-type": "application/json", "Accept": "application/json"}

 try:
     orion_con.request("PATCH", "/v2/entities/"+matricula+"/attrs?type="+tipo, json.dumps(data), headers)
     time.sleep(1)
     resp = orion_con.getresponse()
     print ("Context Broker: ", resp.status, resp.reason, resp.read())
     orion_con.close()
 except Exception as f:
     print (f)
```

Ilustración 17: Actualización de datos del acelerómetro

6. **Descanso de unos segundos**

   El tiempo de descanso se puede configurar editando el valor de la variable "READ_TIME" del programa.

7. **Vuelta a leer y enviar**

   El programa entra así en un bucle infinito en el que estará constantemente midiendo y enviando datos al context broker.

4.1.3 Sensor NEO-6M-0-001

Este sensor GPS se conectará a la Raspberry Pi por medio de la interfaz de comunicación serie UART [2]. Para extraer los datos del sensor se hace uso de un demonio llamado GPSD, el cual estará instalado en la Raspberry Pi. El programa *neo* leerá los datos recopilados por el GPSD para realizar las mediciones y enviarlas al context broker. En concreto se realizarán las siguientes mediciones, que se guardarán en variables de nombre similar:

- Latitud: variable "latitud"
- Longitud: variable "longitud"
- Altitud: variable "altitud"
- Velocidad: variable "velocidad"
- Dirección: variable "dirección"
- Velocidad de subida: variable "subida"

4.1.3.1 Programa neo

Este programa se ha desarrollado en Python y su funcionamiento es el siguiente:

1. **Asignación y configuración de los pines GPIO de la Raspberry Pi**

   Se asigna un pin a cada led y un pin al buzzer[8] instalados, y se ponen todos a nivel bajo (apagado). La asignación de pines es la siguiente:

   - Led Azul: GPIO 21
   - Led Amarillo: GPIO 20
   - Led Rojo: GPIO 16
   - Buzzer: GPIO 12

[8] Transductor electro-acústico, también llamado zumbador.

2. Obtención de la dirección IP y el puerto del context broker

Esta información está incluida en el fichero de configuración *sensor.conf*, que estará ubicado en el mismo directorio que el programa. La primera línea de este fichero contiene la dirección IP del context broker y la segunda línea, su puerto.

3. Obtención de la matrícula del vehículo en el que está instalado el dispositivo y el tipo de vehículo del que se trata

Estos datos también están guardados dentro del fichero de configuración mencionado. En la tercera línea del mismo estará escrita la matrícula del vehículo y en la cuarta línea su tipo.

4. Obtención de la dirección IP y puerto del servidor de velocidad máxima

Líneas quinta y sexta, respectivamente, del fichero de configuración.

5. Conexión con el context broker

La conexión se lleva a cabo usando la dirección IP y el puerto leídos del fichero *sensor.conf*.

6. Creación, en el context broker, de una entidad vacía

Esta entidad se creará mediante una petición POST al context broker, enviando la matrícula del vehículo como atributo "id" de la entidad y el tipo de vehículo como atributo "type" de la entidad. Para el resto de atributos el valor enviado será 0. Si la entidad ya existe, no provoca ningún cambio en el context broker. La entidad que se debe crear, así como sus distintos atributos y sus tipos de datos, se detalla en la Sección 4.2.1 de este capítulo. El mensaje de creación de entidad es igual en ambos programas y su estructura se muestra en la Ilustración 16.

7. Lectura de los datos

Se leen los datos y se almacena cada uno en su variable correspondiente.

8. Obtención de la velocidad máxima de la vía

Para ello se conecta con el servidor de velocidad máxima, enviando la latitud y la longitud actual, así como el tipo de vehículo. El servidor responderá enviando un número entero que será la velocidad máxima a la que debe circular ese vehículo por esa vía, en kilómetros por hora.

9. Cálculo de la velocidad de activación de un radar de tráfico

Se sigue la denominada "*regla del 7*" utilizada por la DGT [10]: si la velocidad máxima es inferior a 100 km/h, se suma 7 km/h a esa velocidad; en cambio, si es mayor que 100 km/h, la velocidad a la que se activará el radar será un 7% mayor que la velocidad máxima permitida.

```
# Obtenemos velocidad máxima del servidor
try:
 print ("Conectando con servidor de velocidad máxima...")
 vmax_con.request('GET', "/sensor/vmax?latitud="+str(latitud)+"&longitud="+str(longitud)+"&tipo="+tipo)
 rsp = vmax_con.getresponse()
 vmax=int(rsp.read())
except Exception as f:
 print (f)
 print ("No se ha podido conectar con el servidor de velocidad máxima")

# Calculamos la velocidad de activación de un radar (regla del 7, fuente: DGT)
if vmax < 100:
 vradar= vmax+7
else:
 vradar= vmax+vmax*0.07
```

Ilustración 18: Obtención de la velocidad máxima

10. Señalización luminosa y acústica

Se encenderá el led azul si la velocidad del vehículo es inferior a la velocidad máxima permitida; el led amarillo si la velocidad del vehículo es superior, y el led rojo y el buzzer si la velocidad del vehículo es superior a la velocidad de activación de un radar.

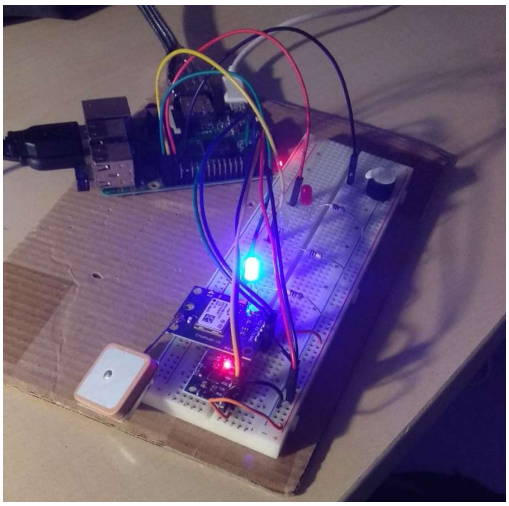

Ilustración 19: Señalización luminosa

11. Envío de datos al context broker

Los atributos de la entidad del context broker se llamarán igual que las variables del programa. Estas variables se envían a la entidad creada en el context broker, dentro de sus atributos correspondientes, mediante una petición PATCH. Además, en el mismo mensaje se envían también la fecha y la hora de la medición, dentro de los atributos "fecha" y "hora" de la entidad. La petición PATCH permite actualizar solo algunos atributos de la entidad, a diferencia de PUT, con la que se deben enviar todos los atributos. Ha de usarse, pues, PATCH porque se desconocen los datos de los atributos correspondientes al giroscopio.

```
data = json.loads('{"latitud" : {"value" : ' + str(latitud) + ', "type" : "Float", "metadata": {"unidades": {"value": "Grados N
        + '"longitud" : {"value" : ' + str(longitud) + ', "type" : "Float", "metadata": {"unidades": {"value": "Grados Este"}}},
        + '"altitud" : {"value" : ' + str(altitud) + ', "type" : "Float", "metadata": {"unidades": {"value": "Metros WGS84"}}},
        + '"velocidad" : {"value" : ' + str(velocidad) + ', "type" : "Float", "metadata": {"unidades": {"value": "Kilómetros por
        + '"direccion" : {"value" : ' + str(direccion) + ', "type" : "Float", "metadata": {"unidades": {"value": "Grados desde e
        + '"subida" : {"value" : ' + str(subida) + ', "type" : "Float", "metadata": {"unidades": {"value": "Metros por minuto"}}
        + '"fecha" : {"value" : "'+fecha+'", "type" : "String", "metadata": {"unidades": {"value": "Día/Mes/Año"}}, '
        + '"hora" : {"value" : "'+hora+'", "type" : "String", "metadata": {"unidades": {"value": "Horas:Minutos:Segundos"}}}}')
headers = {"Content-type": "application/json", "Accept": "application/json"}
try:
 orion_con.request("PATCH", "/v2/entities/"+matricula+"/attrs?type="+tipo, json.dumps(data), headers)
except Exception as g:
 print (g)
 print ("No se han podido enviar los datos")
```

Ilustración 20: Actualización de datos GPS

12. Descanso de unos segundos

El tiempo de descanso se puede configurar editando el valor de la variable "READ_TIME" del programa.

13. Vuelta a la lectura de los datos

El programa entra así en un bucle infinito en el que estará constantemente midiendo y enviando datos al context broker.

4.2 Context Broker

En el context broker Fiware Orion los datos se almacenan en formato JSON y se organizan en distintas secciones, accesibles desde distintas URL. Las secciones que se han utilizado para este Proyecto son */v2/entities*, que contiene las entidades creadas para cada vehículo, y */v2/subscriptions*, que registra cada subscripción que los usuarios dan de alta.

El context broker se ha configurado para que sea accesible desde la dirección http://sensorapp.duckdns.org:1026/v2/.

A continuación, se analizan las características de cada sección del context broker.

4.2.1 Entidades

Los datos medidos por los dispositivos incorporados en los vehículos se envían como elementos de contexto al context broker, el cual los almacena y clasifica según la *entidad* a la que correspondan. Una entidad es una representación virtual de un vehículo, de manera que habrá una entidad por cada vehículo del que se reciban datos. Las entidades están formadas por una serie de *atributos*, que representan cada uno de los datos que pueden ser medidos en un vehículo, así como datos identificativos del vehículo, como su matrícula y tipo. Para el Sistema diseñado, las entidades implementadas tendrán un total de 12 atributos:

- id: Atributo que, junto con el atributo type, identifica unívocamente a la entidad. Su valor será el de la matrícula del vehículo.
- type: Tipo de entidad. Junto con el id permite identificar una entidad unívocamente (puede haber dos entidades con mismo id, pero distinto valor de type). Su valor será el del tipo de vehículo en cuestión (Ej.: turismo, ciclomotor, camión, etc.).
- latitud: Latitud, en grados, de la posición actual del vehículo.
- longitud: Longitud, en grados, de la ubicación del vehículo.
- altitud: Altura a la que se encuentra el vehículo en metros WGS84 [11].
- velocidad: Velocidad del vehículo en kilómetros por hora.
- direccion: Dirección en la que circula el vehículo en grados desde el norte.
- subida: Velocidad de ascenso (positivo) o descenso (negativo) del vehículo en metros por minuto.
- rx: Inclinación, en grados, del eje X del giroscopio.
- ry: Inclinación, en grados, del eje Y del giroscopio.
- fecha: Fecha de la medición, en formato día/mes/año.
- hora: Hora de la medición, en formato horas:minutos:segundos.

Ilustración 21: Context Broker – Entidades y atributos

Cada uno de los atributos anteriores, a excepción de los atributos “id” y “type”, que únicamente contienen un valor, está formado por tres campos:

- type: Tipo de datos que contiene el campo value. Para este Sistema, los atributos “fecha” y “hora” serán de tipo “String” y, los ocho atributos restantes, de tipo “Float”
- value: Valor del atributo, es decir, el valor del dato que contiene el atributo.
- metadata: Metadatos o, lo que es lo mismo, información sobre los datos que contiene el atributo. Contiene atributos en su interior y, por tanto, cada uno de ellos tendrá sus respectivos campos “type” y “value”. En este Proyecto, el campo “metadata” contiene un atributo “unidades” utilizado para informar sobre las unidades en las que está medido cada dato (kilómetros por hora, grados, etc.), recogidas en el campo “value” del atributo. El campo “type” será igual a “Text” (tipo por omisión cuando se introducen cadenas de caracteres alfanuméricos).

Las entidades se pueden consultar enviando una petición GET a la URL http://sensorapp.duckdns.org:1026/v2/entities, y el context broker responderá con un mensaje JSON que contiene una lista de todas las entidades existentes.

Además, enviando peticiones a URL específicas se pueden obtener o actualizar los datos de una o varias entidades o, incluso, de determinados atributos. Esto resulta de gran utilidad y, de hecho, es lo que se ha utilizado para enviar las peticiones PATCH que actualizan los valores de las entidades.

Por ejemplo, accediendo a la URL http://sensorapp.duckdns.org:1026/v2/entities/5678DEF/attrs/altitud, se puede consultar únicamente el atributo “altitud” de la entidad cuyo “id” es igual a “5678DEF” y, si se añade ‘/value’ al final de la misma, sólo devolverá el valor numérico del atributo (ya no es JSON puesto que sólo es un número).

Ilustración 22: Context Broker – Atributo

También se puede especificar un determinado tipo de vehículo. Por ejemplo, un GET a la URL http://sensorapp.duckdns.org:1026/v2/entities?type=ciclomotor devuelve las entidades cuyo tipo de vehículo es "ciclomotor".

Es posible también hacer que el context broker devuelva los atributos en formatos determinados. Se puede hacer que solo se devuelvan los valores de cada atributo, o sólo de los atributos que se especifiquen. Por ejemplo, la siguiente URL devuelve el nombre de atributo y el valor de los atributos "matricula", "latitud", "longitud", "fecha", "hora", "id" y "type" de las entidades de los vehículos cuya matrícula sea "5678DEF" y que sean del tipo "ciclomotor":

http://sensorapp.duckdns.org:1026/v2/entities/5678DEF?type=ciclomotor&options=keyValues&attrs=matricula,latitud,longitud,fecha,hora

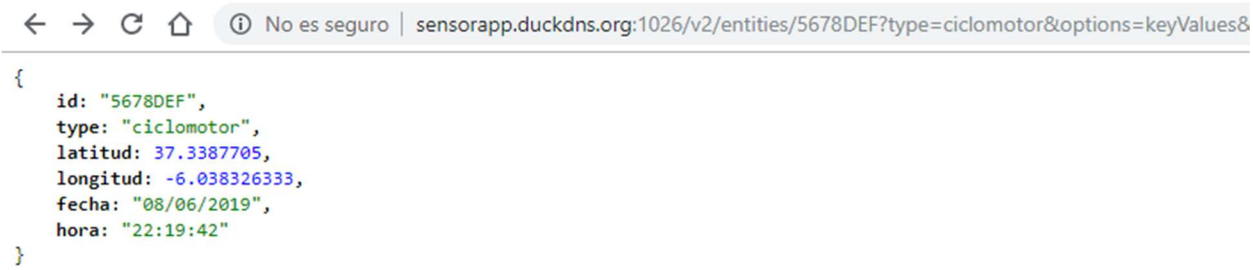

Ilustración 23: Context Broker – Opción keyValues

Para eliminar una entidad basta con enviar una petición DELETE especificando el "id" de la entidad a eliminar. En caso de que hubiese dos entidades con el mismo valor de "id", sería necesario especificar también el valor del atributo "type". El mensaje DELETE puede enviarse, por ejemplo, con el comando *curl*.

```
$ curl localhost:1026/v2/entities/5678DEF?type=ciclomotor -s -S -X DELETE
```

4.2.2 Suscripciones

Las suscripciones enviadas por la aplicación web a petición de los usuarios se almacenan en una lista. Cada suscripción queda registrada en la sección */v2/subscriptions* del context broker, de manera que si se envía un GET a la URL http://sensorapp.duckdns.org:1026/v2/subscriptions, el context broker responderá enviando una lista de todas las suscripciones vigentes.

Cada suscripción tiene una serie de campos:

- id: Número de identificación de la suscripción que se genera de forma automática al crearla.
- description: Texto que describe el propósito de la suscripción.
- status: Estado de la suscripción. Su valor puede ser "active" si se desea que la suscripción envíe notificaciones constantemente, o "oneshot" si se desea que envíe una notificación y, seguidamente, cambie al estado "inactive" hasta que se vuelva a cambiar su estado [12].
- subject: Contiene los campos que determinan el envío de las notificaciones.
    - entities: Entidad o entidades a las que se refiere la suscripción. Contiene dos parámetros:
        - id o idPattern: "id" admite un valor específico de "id" de suscripción (matrícula en este caso), mientras que idPattern admite expresiones regulares, como por ejemplo ".*" (cualquier matrícula) [13].
        - type o typePattern: "type" admite un valor específico de tipo de vehículo y "typePattern" admite expresiones regulares.
    - condition: Es la condición bajo la cual se envía una notificación. Contiene un campo "attrs", el cual incluye una lista de los nombres de los atributos que, al ver modificado su valor, disparan el envío de la notificación. Si la lista está vacía, es decir, no se ha especificado ningún atributo, la notificación se envía al modificarse cualquiera de los atributos de la entidad suscrita [14].
- notification: Parámetros de las notificaciones.
    - timesSent: Número de notificaciones que ha enviado la suscripción desde que ha sido creada.
    - lastNotification: Fecha y hora de envío de la última notificación que se ha enviado.
    - attrs: Lista de atributos que se notifican. Si la lista se deja vacía se notifican todos los atributos de la entidad.
    - attrsFormat: Formato del mensaje de notificación. Admite los valores "keyValues" (idéntico al que se ha comentado en la subsección anterior de este capítulo), "legacy" (compatible con NGSIv1) y "normalized" (igual que el formato de las entidades). En este Proyecto se ha utilizado el formato "normalized".
    - http: Contiene un parámetro "url", cuyo valor es la URL a la que se envían las notificaciones. Como se verá en el próximo capítulo, en este Proyecto las notificaciones se envían a la ruta */sensor/notificaciones* del servidor web.
    - lastSuccess: Fecha y hora del último envío exitoso de notificación.
    - lastSuccessCode: Código de estado HTTP recibido como respuesta a la última notificación enviada.
- throttling: Tiempo mínimo, en segundos, que ha de haber entre dos notificaciones consecutivas.

```
Fiware-Servicepath: /
Content-Length: 490
User-Agent: orion/2.1.0-next libcurl/7.29.0
Host: 192.168.1.45:1028
Accept: application/json
Content-Type: application/json; charset=utf-8
Fiware-Correlator: bcef66a6-4525-11e9-97cd-0242ac120003

{
    "contextResponses": [
        {
            "contextElement": {
                "attributes": [
                    {
                        "name": "matricula",
                        "type": "String",
                        "value": "8298HMY"
                    },
                    {
                        "name": "latitud",
                        "type": "Float",
                        "value": 45643.12334
                    },
                    {
                        "name": "longitud",
                        "type": "Float",
                        "value": -34567.14
                    },
                    {
                        "name": "fecha",
                        "type": "String",
                        "value": "13/03/2019"
                    },
                    {
                        "name": "hora",
                        "type": "String",
                        "value": "01:19:59"
                    }
                ],
                "id": "8298HMY",
                "isPattern": "false",
                "type": "GPS"
            },
            "statusCode": {
                "code": "200",
                "reasonPhrase": "OK"
            }
        }
    ],
    "originator": "localhost",
    "subscriptionId": "5c8849453281b44321e91252"
```

Ilustración 24: Context Broker – Notificación en formato “legacy”

Para suscribirse a una entidad, se ha de enviar un mensaje POST a la URL http://sensorapp.duckdns.org:1026/v2/subscriptions/ que contenga los valores que configuren la suscripción para que notifique sobre los atributos deseados. En la siguiente ilustración se muestra cómo se han configurado las suscripciones en la aplicación web Sensorapp.

```
 // Construimos el mensaje de suscripcion a la matricula dada
JSONObject sub_gps = new JSONObject("{\"description\": \"A subscription to get info about GPS\", "
        + "\"subject\": {\"entities\": [{\"id\": \""+mat+"\", \"typePattern\": \".*\"}], "
        + "\"condition\": {\"attrs\": [\"latitud\", \"longitud\", \"altitud\"]}}, " //Si cambia alguno se envia notificacion
        + "\"notification\": {\"http\": {\"url\": \"http://"+ip_port_sensorapp+"/sensor/notificaciones\"}, "
        + "\"attrs\": [], " //Si no se especifican atributos se notifican todos
        + "\"attrsFormat\": \"normalized\"}, "
        + "\"throttling\": 5}"); //Tiempo de espera tras enviar una notificacion (segundos)

// URL and parameters for the connection, This particulary returns the information passed
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");
```

Ilustración 25:Context Broker – Suscripción

Se puede apreciar cómo el parámetro "id" contiene la matrícula del vehículo que ha introducido el usuario y, en cambio, se acepta cualquier tipo de vehículo (no puede haber dos vehículos de distinto tipo con la misma matrícula). Además, se han especificado como condición los atributos "latitud", "longitud" y "altitud", por lo que sólo se enviará una notificación cuando uno de estos tres atributos cambie.

En este Proyecto también se da la posibilidad al usuario de suscribirse a todos los vehículos presentes en el context broker. Para ello, simplemente hay que cambiar "id" por "idPattern" y asignarle el valor ".*". Un detalle muy importante, que se debe tener en cuenta en este caso, es que no debe haber throttling. La razón de esto es que, si hubiese throttling, cuando se disparase el envío de notificación simultáneamente en varias entidades, sólo se enviaría la notificación de la entidad que haya activado primero la condición, y las demás no se enviarían puesto que tendría que cumplirse el tiempo de espera.

```
 // Construimos el mensaje de suscripcion a todas las matriculas y todos los tipos
JSONObject sub_gps = new JSONObject("{\"description\": \"A subscription to get info about GPS\", "
        + "\"subject\": {\"entities\": [{\"idPattern\": \".*\", \"typePattern\": \".*\"}], "
        + "\"condition\": {\"attrs\": [\"latitud\", \"longitud\", \"altitud\"]}}, " //Si cambia alguno se envia notificacion
        + "\"notification\": {\"http\": {\"url\": \"http://"+ip_port_sensorapp+"/sensor/notificaciones\"}, "
        + "\"attrs\": [], " //Si no se especifican atributos se notifican todos
        + "\"attrsFormat\": \"normalized\"}}"); //En este caso no es posible el throttling

// URL and parameters for the connection, This particulary returns the information passed
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");
```

Ilustración 26: Context Broker – Suscripción a todas las entidades

Para cancelar una suscripción, es necesario enviar una petición DELETE a la URL http://sensorapp.duckdns.org:1026/v2/subscriptions/id_suscripción, siendo "id_suscripción" el número de identificación de la suscripción que se quiere eliminar.

Las notificaciones se envían como mensajes POST a la URL especificada en el campo "url" de la correspondiente suscripción. Cuando se realiza una suscripción a una o varias entidades, instantáneamente se envía una primera notificación, que se conoce como notificación inicial. Esta notificación se envía siempre tras la suscripción, sin necesidad de que se active la condición [15]. Todos los mensajes de notificación contienen una lista con todas las entidades que han activado la condición. Normalmente esta lista sólo contiene la entidad que ha activado la condición de notificación, sin embargo, para la notificación inicial en el caso de suscripción a todos los vehículos, la lista estará formada por todas las entidades del context broker. Tras la notificación inicial, se enviará una notificación por cada entidad que dispare la condición.

5 APLICACIÓN DESARROLLADA: SERVIDOR WEB Y BASE DE DATOS

Everything is designed. Few things are designed well.

- Brian Reed -

En este capítulo se desarrolla la estructura interna del servidor de base de datos, el diseño de la aplicación web, y la API REST implementada. Se detallan las distintas clases y los métodos y atributos que implementan, así como los ficheros de configuración de Hibernate, y los ficheros HTML, que utilizan las librerías de Bootstrap para ofrecer una interfaz gráfica de usuario.

5.1 Clases

En esta sección, se expone cada una de las clases ubicadas en el directorio *sensor/src/main/java/sensor*.

Clases que controlan el funcionamiento de la aplicación

- **Application**

  Clase *@SpringBootApplication* que contiene el método *main* del programa. Este método es el primero al que se llama nada más ejecutar la aplicación. Lo que hace este método es ejecutar la clase etiquetada como *@RestController* que, en este caso, es la clase *SensorControler*.

- **SensorController**

  Esta es la clase que ejerce la Inversión de Control en la aplicación Spring desarrollada. Se trata de una clase que implementa métodos *@RequestMapping y @PostMapping*, que son llamados cuando se recibe una petición GET o POST, respectivamente, a un URI determinado.

  Esta clase cuenta con dos atributos cuyos valores se obtienen, gracias a la anotación @Value, del fichero *application.properties*, el cual se describe en la próxima sección. Estos dos atributos se utilizan para enviar los mensajes de suscripción, cancelación y consulta de las suscripciones al context broker.

  - private String ip_port_sensorapp: Dirección IP y puerto en el que se ejecutará la aplicación web, en formato IP:Puerto.
  - private String ip_port_contextbroker: Dirección IP y puerto en el que se ubica el context broker, en formato IP:Puerto.

Los métodos que contiene esta clase son los que implementan todas las funcionalidades de la aplicación web. Los métodos @RequestMapping que implementan la funcionalidad de consulta de datos son los siguientes:

◊ public List<Dato> datos: Permite consultar la tabla "datos" de la base de datos, en base a los parámetros que se le pasan, por medio de una llamada al método *consultarDatos* de la clase *Database*. Los parámetros que recibe corresponden a las columnas de dicha tabla y son todos opcionales. Devuelve una lista de objetos de tipo "Dato", que es el resultado de la consulta.

```
//Consulta la base de datos de los sensores
@RequestMapping("/sensor/datos")
public List<Dato> datos(
  @RequestParam(value="id", required=false) Long id,
  @RequestParam(value="matricula", required=false) String mat,
  @RequestParam(value="latitud", required=false) Double lat,
  @RequestParam(value="longitud", required=false) Double lon,
  @RequestParam(value="altitud", required=false) Double alt,
  @RequestParam(value="velocidad", required=false) Double vel,
  @RequestParam(value="direccion", required=false) Double dir,
  @RequestParam(value="subida", required=false) Double sub,
  @RequestParam(value="rx", required=false) Double rx,
  @RequestParam(value="ry", required=false) Double ry,
  @RequestParam(value="fecha", required=false) String fecha,
  @RequestParam(value="hora", required=false) String hora) {

    DataBase bd = new DataBase();
    List<Dato> datos = bd.consultarDatos(id, mat, lat, lon, alt, vel, dir, sub, rx, ry, fecha, hora);

return datos;
}
```

Ilustración 27: Sensorapp – Método datos

◊ public List<Vehiculo> vehiculos: Realiza una consulta a la tabla "vehiculos" mediante una llamada al método *consultarVehiculos* de la clase *Database*. Los parámetros que recibe para realizar la consulta son la matrícula y el tipo de vehículo que se busca, ambos opcionales. Devuelve una lista de objetos de tipo "Vehiculo".

◊ public List<Tipo> tipos: Realiza una consulta a la base de datos, en base a los parámetros que se le pasan, por medio de una llamada al método *consultarTipos* de la clase *Database*. Los parámetros que admite son el tipo de vehículo que se desea buscar y su velocidad máxima, ambos opcionales. Devuelve una lista de objetos de tipo "Tipo".

◊ public String datos_viaje: Con este método se pueden consultar las estadísticas de un trayecto concreto. Ha de recibir: un parámetro "matrícula" con el valor de la matrícula del coche a consultar; un parámetro "vmp", que contiene la velocidad máxima permitida para el trayecto; los parámetros "idInicio" e "idFin, que contienen el número identificador de los datos con los que empieza y acaba el viaje, respectivamente. Si no se especifica el parámetro "vmp", se consulta a la base de datos una velocidad genérica según el tipo de vehículo del que se trate, por medio del método *consultarVmax* de la clase *Database*. En base a la velocidad máxima permitida, se calculan las infracciones cometidas. Devuelve una cadena de texto que contiene la información sobre el viaje.

```
//Informa sobre un viaje (datos comprendidos entre dos ids de la tabla datos, para una matricula dada)
@RequestMapping("/sensor/datos_viaje")
public String datos_viaje(
  @RequestParam(value="idInicio", required=false) Long idInicio,
  @RequestParam(value="idFin", required=false) Long idFin,
  @RequestParam(value="matricula", required=true) String mat,
  @RequestParam(value="vmp", required=false) Double vmp) {

    Dato dato;
    Double vInst=0.0, vAvg=0.0, vMax=0.0, vSum=0.0;
    Double hInst=0.0, hAvg=0.0, hMax=0.0, hSum=0.0;
    Double sInst=0.0, sAvg=0.0, sMax=0.0, sSum=0.0;
    int count=1;
    String mensaje="No hay datos de viaje para los valores introducidos";
    String aviso="";
    DataBase bd = new DataBase();

    //Consultamos la base de datos (si no se especifican idInicio e idFin se consultan todos los datos)
    List<Dato> viaje = bd.consultarViaje(idInicio, idFin, mat);

    //Si no se ha especificado una velocidad maxima para el viaje...
    if (vmp == null) {
        //.. esta sera la maxima permitida para el tipo de vehiculo analizado
        vmp=bd.consultarVmax(mat);
    }

    //Recorremos los datos del viaje
    for(Iterator<Dato> i=viaje.iterator();i.hasNext();count++) {
        dato=i.next();
        //Velocidad media
        vInst=dato.getVelocidad();
        vSum += vInst;
        vAvg=vSum/count;
        //Velocidad maxima alcanzada
        if(vInst>vMax) {
            vMax=vInst;
        }
        //Comprobamos si se ha superado la velocidad maxima permitida
        if(vInst>vmp) {
            aviso+="<br>Velocidad máxima permitida superada el "+dato.getFecha()+" a las "+dato.getHora()
                    +", en "+dato.getLatitud()+" Norte, "+dato.getLongitud()+" Este"+", con "+vInst+" km/h";
        }
        //Altitud media
        hInst=dato.getAltitud();
```

Ilustración 28: Sensorapp – Método datos_viaje

Los métodos @RequestMapping que implementan la gestión de las suscripciones son los siguientes:

- ◊ public String suscribir: Permite suscribir un vehículo en el context broker. Recibe como parámetro la matrícula del vehículo que se quiere suscribir. Devuelve un mensaje de confirmación o fallo en la suscripción.
- ◊ public String suscribir_todo: Permite al usuario suscribir todos los vehículos existentes en el context broker. No recibe ningún parámetro, y devuelve un mensaje de éxito o fracaso de la operación.
- ◊ public String suscripciones: Consulta las suscripciones registradas en el context broker. No admite parámetros y devuelve las suscripciones en formato JSON.
- ◊ public String cancelar_suscripcion: Cancela una suscripción existente en el context broker. Recibe como parámetro el identificador de la suscripción a eliminar y devuelve un mensaje de información sobre el resultado.

```
//Cancela una suscripción dada por su id
@RequestMapping("/sensor/cancelar_suscripcion")
public String cancelar_suscripcion(
    @RequestParam(value="subscriptionId") String id){

    String mensaje = "No se ha podido cancelar la suscripcion: "+id;

    //Enviamos 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();

    if(response.getStatus()==204) {
        mensaje="Se ha cancelado la suscripcion: "+id;
    }

    return mensaje;
}
```

Ilustración 29: Sensorapp – Método cancelar_suscripcion

Para recibir las notificaciones del context broker, se ha programado el siguiente método *@PostMapping*:

- ◊ public void notificaciones: Recibe una notificación del context broker en formato JSON y la transforma en un objeto de tipo "Notification", haciendo uso de la Inyección de Dependencias, propia de Spring. Ese objeto se le pasa como parámetro al método *insertarDatos*, de la clase *DataBase*, que guarda la información que contiene en la base de datos.

Como implementación provisional del servidor de velocidad máxima, se ha programado el siguiente método @RequestMapping:

- ◊ public int vmax: Recibe tres parámetros, que son la latitud, la longitud y el tipo del vehículo que realiza la consulta. Consulta la velocidad máxima a la base de datos, por medio del método *consultarVmaxTipo* de la clase *Database*. Devuelve un entero, que es la velocidad máxima permitida, en kilómetros por hora.

Los siguientes métodos *@RequestMapping* permiten acceder a las distintas vistas de la interfaz gráfica, usando los ficheros HTML. No reciben ningún parámetro y devuelven un objeto de tipo *ModelAndView*.

- ◊ public ModelAndView home
- ◊ public ModelAndView datos_view
- ◊ public ModelAndView vehiculos_view
- ◊ public ModelAndView tipos_view
- ◊ public ModelAndView viajes_view
- ◊ public ModelAndView suscribir_view
- ◊ public ModelAndView cancelar_suscripcion_view

```
//Pagina principal de la aplicacion
@RequestMapping("/")
public ModelAndView home (){

    ModelAndView mav = new ModelAndView("index.html");

return mav;
}

//Interfaz de busqueda de datos
@RequestMapping("/datos")
public ModelAndView datos_view(){

    ModelAndView mav = new ModelAndView("datos.html");

return mav;
}
//Interfaz de busqueda de vehiculos
@RequestMapping("/vehiculos")
public ModelAndView vehiculos_view(){

    ModelAndView mav = new ModelAndView("vehiculos.html");
```

Ilustración 30: Sensorapp – Métodos ModelAndView

- **DataBase**

  Esta clase proporciona los métodos que implementan las transacciones con la base de datos, haciendo uso de Hibernate. El siguiente método permite insertar, en la base de datos, los datos recibidos en una notificación del context broker:

  ◊ public void insertarDatos: Recibe como parámetro un objeto de tipo "Notification" y extrae su información para insertarla en un objeto de tipo "Dato", que se guarda en la base de datos mediante una transacción. Además, se comprueba si el vehículo existe ya o no en la base de datos y, en caso de que no exista, se extrae su matrícula y su tipo y se añade una nueva fila en la tabla "vehiculos".

```
public void insertarDatos(Notification notificacion){

    Dato nuevo=null;
    Entity entity=null;

    //Comenzamos una transacción con la base de datos
    SessionFactory sessionFactory = new org.hibernate.cfg.Configuration().configure().buildSessionFactory();
    Session session = sessionFactory.openSession();
    session.beginTransaction();

    /*Necesitamos un bucle para que guarde todos las entidades de la notificacion inicial*/
    for(Iterator<Entity> i=notificacion.getData().iterator();i.hasNext();) {

        entity=i.next();
        nuevo = new Dato((String) entity.getId(),
                ((Number) entity.getLatitud().getValue()).doubleValue(),
                ((Number) entity.getLongitud().getValue()).doubleValue(),
                ((Number) entity.getAltitud().getValue()).doubleValue(),
                ((Number) entity.getVelocidad().getValue()).doubleValue(),
                ((Number) entity.getDireccion().getValue()).doubleValue(),
                ((Number) entity.getSubida().getValue()).doubleValue(),
                ((Number) entity.getRx().getValue()).doubleValue(),
                ((Number) entity.getRy().getValue()).doubleValue(),
                (String) entity.getFecha().getValue(),
                (String) entity.getHora().getValue()
                );

        //Nos aseguramos de que la Matricula se encuentra en la BD (es clave externa)
        Query q =session.createQuery("FROM Vehiculo V WHERE V.matricula= :mat_id");
        q.setParameter("mat_id", entity.getId());
        List resultados = q.list();
        Iterator iterator = resultados.iterator();
        //Si no se encuentra la matrícula en la BD...
        if (!iterator.hasNext()) {
            //...la introducimos en la BD
            Vehiculo nueva = new Vehiculo(entity.getId(), entity.getType());
            session.save(nueva);
        }

        //Introducimos la nueva información en la BD
        session.save(nuevo);
    }

    session.getTransaction().commit(); //Terminamos transacción
    session.close(); //Cerramos sesión
    sessionFactory.close(); //Cerramos conexiones con BD
}
```

Ilustración 31: Sensorapp – Método insertarDatos

Los siguientes métodos sirven para realizar consultas a la base de datos e implementan las consultas dinámicas que ofrece Hibernate. Las consultas dinámicas permiten añadir restricciones a una consulta a la base de datos en función de una condición que puede ser, por ejemplo, si se ha proporcionado un parámetro o no. De esta manera, a diferencia de SQL, las consultas dejan de ser estáticas y pueden variar y hacerse más restrictivas dependiendo de lo que interese.

◊ public List<Dato> consultarDatos: Este método consulta la tabla "datos" de la base de datos. En función de si los parámetros que se le pasan contienen información o no, se añadirán dichos parámetros como restricciones adicionales a la consulta. Devuelve la lista de filas (lista de objetos de tipo "Dato") resultado de la consulta.

◊ public List<Vehiculo> consultarVehiculos: Funciona de manera similar que el método anterior, pero la consulta se realiza sobre la tabla "vehiculos". Devuelve una lista de objetos de tipo "Vehiculo".

- ◊ public List<Tipo> consultarTipos: Este método utiliza las consultas dinámicas para consultar la tabla "tipos". Devuelve una lista de objetos de tipo "Tipo".
- ◊ public List<Dato> consultarViaje: Realiza una consulta dinámica de los datos comprendidos entre "idInicio" e "idFin", para el vehículo cuya matrícula es igual a "mat". Devuelve una lista de objetos de tipo "Dato", que es el resultado de la consulta.

```
public List<Dato> consultarViaje(Long idInicio, Long idFin, String mat){

    //Comenzamos una transacción con la base de datos
    SessionFactory sessionFactory = new org.hibernate.cfg.Configuration().configure().buildSessionFactory();
    Session session = sessionFactory.openSession();
    session.beginTransaction();

    //Configuramos los parámetros necesarios para las consultas dinámicas
    Criteria cr = session.createCriteria(Dato.class);
    cr.add(Restrictions.eq("matricula", mat));

    //Añadimos las condiciones dinámicas según los parámetros introducidos
    if(idInicio!=null) {
        cr.add(Restrictions.ge("id", idInicio));
    }
    if(idFin!=null) {
        cr.add(Restrictions.le("id", idFin));
    }

    List<Dato> resultados = cr.list();

    session.getTransaction().commit(); //Terminamos transacción
    session.close(); //Cerramos sesión
    sessionFactory.close(); //Cerramos conexiones con BD

    return resultados;
}
```

Ilustración 32: Sensorapp – Método consultarViaje

Por último, se han implementado dos métodos para consultar la velocidad máxima de un vehículo. Se usará uno u otro dependiendo de si se conoce el tipo de vehículo o su matrícula:

- ◊ public Double consultarVmax: Recibe la matrícula de un vehículo como parámetro y busca en la base de datos su velocidad máxima permitida. Devuelve la velocidad en kilómetros por hora.
- ◊ Public int consultarVmaxTipo: Recibe un tipo de vehículo como parámetro, consulta la base de datos, y devuelve la velocidad máxima permitida para ese tipo de vehículo en kilómetros por hora.

```
public Double consultarVmax(String mat){

    Double vmax=0.0;

    //Comenzamos una transacción con la base de datos
    SessionFactory sessionFactory = new org.hibernate.cfg.Configuration().configure().buildSessionFactory();
    Session session = sessionFactory.openSession();
    session.beginTransaction();

    //Realizamos la consulta
    Query q =session.createQuery("FROM Tipo T WHERE T.tipo IN (SELECT V.tipo FROM Vehiculo V WHERE V.matricula= :mat_id)");
    q.setParameter("mat_id", mat);
    List<Tipo> resultados = q.list();
    Iterator iterator = resultados.iterator();

    if (iterator.hasNext()) {
        vmax=((Number) resultados.iterator().next().getVmax()).doubleValue();
    }

    session.getTransaction().commit(); //Terminamos transacción
    session.close(); //Cerramos sesión
    sessionFactory.close(); //Cerramos conexiones con BD

    return vmax;
}
```

Ilustración 33: Sensorapp – Método consultarVmax

Clases POJO para la comunicación con el context broker

Estas clases se utilizan para poder recibir las notificaciones procedentes del context broker, de manera que estas puedan guardarse directamente como objetos. De esta forma se convierten las entidades y atributos del context broker, que llegan en formato JSON (cadena de caracteres), a objetos con los que se puede leer y manipular la información fácilmente. Esto es posible gracias a la anotación *@JsonProperty*, que asocia el nombre de cada propiedad del flujo de datos JSON a su correspondiente atributo en Java.

Los métodos que implementan estas clases son los estándares para una clase POJO, es decir, un getter y un setter por atributo, y dos constructores de clase: uno por defecto y otro que asigna el valor de cada parámetro pasado al constructor a su atributo correspondiente.

- **Notification**

  Esta clase implementa los métodos y atributos necesarios para que se puedan recibir como objetos las notificaciones procedentes del context broker.

  Sus atributos se corresponden con el formato JSON con el que llegan las notificaciones: una lista de entidades y una cadena que contiene el número de identificación de la subscripción que ha generado la notificación. Por tanto, los atributos son los siguientes:

  ◊ private List<Entity> data: Lista de objetos de tipo Entity, que contendrá cada una de las entidades que han sido enviadas en la notificación.

  ◊ private String subscriptionId: Número de identificación de la suscripción.

Notification.java | Entity.java | Attribute.java | Metadata.java

sensor ▸ src/main/java ▸ sensor ▸ Notification ▸

```
1  package sensor;
2
3  import java.util.List;
6
7  //POJO class para las notificaciones del context broker
8  public class Notification{
9
10     @JsonProperty("data")
11     private List<Entity> data;
12     @JsonProperty("subscriptionId")
13     private String subscriptionId;
14
15     public Notification(){}
16
17     public Notification(List<Entity> data, String subscriptionId){
18     this.data=data;
19     this.subscriptionId=subscriptionId;
20     }
21
22
23     public String getSubscriptionId(){
24     return subscriptionId;
25     }
26     public void setSubscriptionId(String subscriptionId){
27         this.subscriptionId=subscriptionId;
28     }
29
30     public List<Entity> getData(){
31     return data;
32     }
33     public void setData(List<Entity> data){
34         this.data=data;
35     }
36 }
```

Ilustración 34: Sensorapp – Clase Notification

- **Entity**

  La clase "Entity" abstrae y define los objetos que contienen la información de las entidades del context broker, conservando la estructura que, en formato JSON, tenían estas. Tiene, por tanto, un atributo por cada uno de los atributos de dichas entidades. Los atributos "id" y "type", al igual que en el context broker, son simplemente cadenas de caracteres que contienen la matrícula y el tipo de vehículo, respectivamente. El resto de atributos, por tanto, serán objetos de tipo "Attribute", pues cada uno constituye un atributo en la entidad del context broker:

- ◊ private String id
- ◊ private String type
- ◊ private Attribute latitud
- ◊ private Attribute longitud
- ◊ private Attribute altitud
- ◊ private Attribute velocidad
- ◊ private Attribute dirección
- ◊ private Attribute subida
- ◊ private Attribute rx
- ◊ private Attribute ry
- ◊ private Attribute fecha
- ◊ private Attribute hora

- **Attribute**

  Esta clase implementa el concepto de atributo de las entidades del context broker. Por tanto, cada atributo de la clase se corresponde con cada uno de los tres campos de los atributos de la entidad:

  - ◊ private String type: Cadena que contiene el tipo de dato del campo value.
  - ◊ private T value: Valor del atributo. Como se desconoce, a priori, qué tipo de dato va a almacenar esta variable (se recuerda que los tipos de datos de los atributos en el context broker podían ser o Float o String), se utiliza el tipo de dato genérico de Java: el tipo < T >.
  - ◊ private Metadata metadata: Objeto de tipo Metadata, que equivale al campo "metadata" en JSON.

Notification.java | Entity.java | Attribute.java | Metadata.java

sensor ▸ src/main/java ▸ sensor ▸ Attribute<T> ▸

```
1  package sensor;
2
3  import com.fasterxml.jackson.annotation.JsonProperty;
4
5  //POJO class para las notificaciones del context broker
6  public class Attribute<T>{
7
8      @JsonProperty("metadata")
9      private Metadata metadata;
10     @JsonProperty("type")
11     private String type;
12     @JsonProperty("value")
13     private T value;
14
15     public Attribute(){}
16
17     public Attribute(Metadata metadata, String type, T value){
18     this.metadata=metadata;
19     this.type=type;
20     this.value=value;
21     }
22
23     public Metadata getMetadata(){
24     return metadata;
25     }
26     public void setMetadata(Metadata metadata){
27         this.metadata=metadata;
28     }
29
30     public String getType(){
```

Ilustración 35: Sensorapp – Clase Attribute

- **Metadata**

  Al igual que en el context broker, esta clase contiene un objeto de tipo atributo, que en esta aplicación se ha implementado con la clase Attribute:

  ◊ private Attribute unidades: Objeto de tipo Attribute (en JSON equivale a un atributo) que informa sobre las unidades en las que se ha medido cada dato en el dispositivo.

Clases POJO para la comunicación con la base de datos

Estas clases se utilizan para abstraer la comunicación con la base de datos relacional mediante Hibernate. Con Hibernate, se utiliza el lenguaje orientado a objetos para realizar transacciones con la base de datos, por lo que se necesita una clase por cada una de las tablas en la base de datos. Cada clase, por tanto, tendrá tantos atributos (y del mismo tipo) como columnas haya en la tabla correspondiente.

Los métodos que implementan estas clases son los estándares para una clase POJO, es decir, un getter y un setter por atributo, y dos constructores de clase: uno por defecto y otro que asigna el valor de cada parámetro pasado al constructor a su atributo correspondiente.

- **Tipo**

  Clase que equivale a la tabla "tipos" de la base de datos. Tiene dos atributos, que se corresponden con las dos columnas de la base de datos:

  ◊ private String tipo

  ◊ private int vmax

- **Vehiculo**

  Clase POJO para la implementación de la tabla "vehiculos". Sus atributos son los siguientes:

  ◊ private String matricula

  ◊ private String tipo

- **Dato**

  Clase que implementa la tabla "datos". Al constructor de esta clase no se le pasa el valor del atributo "id", ya que este es autoincrementado automáticamente por el servidor de base de datos. Tiene un atributo por cada una de las columnas de la tabla "datos":

  ◊ private long id

  ◊ private String matricula

  ◊ private double latitud

  ◊ private double longitud

  ◊ private double altitud

  ◊ private double velocidad

  ◊ private double direccion

  ◊ private double subida

  ◊ private double rx

  ◊ private double ry

  ◊ private String fecha

  ◊ private String hora

5.2 Ficheros de configuración

A continuación, se describen todos los ficheros de configuración que contiene la aplicación web, los cuales están ubicados en la siguiente ruta relativa al directorio *sensor* del proyecto: *src/main/resources*.

Ficheros de configuración de Hibernate

Existen cuatro ficheros de configuración de Hibernate: un fichero de mapeo por cada tabla existente en la base de datos (un total de tres), y uno de configuración global.

- hibernate.cfg.xml: Este es el fichero de configuración global de Hibernate. En él se establecen parámetros como el lenguaje SQL empleado (PostgreSQL en este caso), el driver que se ha de usar, el nombre de usuario y la contraseña del usuario propietario de la base de datos (consultar el Anexo D: Configuración del servidor de Base de Datos PostgreSQL), y la dirección IP y puerto donde se está ejecutando el servidor de base de datos. También se indica la ubicación de los ficheros de mapeo de las tablas, que asocian cada una de ellas con su correspondiente clase 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.PostgreSQLDialect</property>
        <property name="hibernate.connection.driver_class">org.postgresql.Driver</property>
        <property name="hibernate.connection.username">dit</property>
        <property name="hibernate.connection.password">dit</property>
        <property name="hibernate.connection.url">jdbc:postgresql://192.168.1.45:5432/dit</property>

        <mapping resource="Dato.hbm.xml"></mapping>
        <mapping resource="Vehiculo.hbm.xml"></mapping>
        <mapping resource="Tipo.hbm.xml"></mapping>

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

```

Ilustración 36: Sensorapp – Fichero hibernate.cfg.xml

- Dato.hbm.xml: En este fichero se establece la asociación o mapeo de la tabla "datos" con la clase "Dato". También se mapean cada uno de los atributos de esta clase con su correspondiente columna en la tabla "datos". Además, se establece y se mapea el generador de secuencia para que la columna "id" se autoincremente.

Dato.hbm.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="sensor.Dato" table="datos">
        <id name="id" type="long">
        <column name="id"/>
        <generator class="sequence-identity">
        <param name="sequence">datos_id_seq</param>
        </generator>
        </id>

        <property name="matricula" column="matricula" type="string"/>
        <property name="latitud" column="latitud" type="double"/>
        <property name="longitud" column="longitud" type="double"/>
        <property name="altitud" column="altitud" type="double"/>
        <property name="velocidad" column="velocidad" type="double"/>
        <property name="direccion" column="direccion" type="double"/>
        <property name="subida" column="subida" type="double"/>
        <property name="rx" column="rx" type="double"/>
        <property name="ry" column="ry" type="double"/>
        <property name="fecha" column="fecha" type="string"/>
        <property name="hora" column="hora" type="string"/>
    </class>
</hibernate-mapping>

```

Ilustración 37: Sensorapp – Fichero Dato.hbm.xml

- Vehiculo.hbm.xml: En este fichero se mapea la clase "Vehiculo" a la tabla "vehiculos" de la base de datos, y sus atributos a sus correspondientes columnas de la tabla.
- Tipo.hbm.xml: En este fichero se asocia la clase "Tipo" con la tabla "tipos". Sus atributos, de nuevo, se asocian a las columnas de esta tabla.

Fichero de configuración de la aplicación

El fichero de configuración de la aplicación web se llama *application.properties*. Este fichero contiene tres filas. La primera, indica que no se muestren parámetros de depuración al usuario cuando ocurre algún error en la página web. La segunda, indica la dirección IP y el puerto en el que se está ejecutando la aplicación web, y la tercera la dirección IP y puerto en el que se ubica el context broker, ambas en formato IP:Puerto.

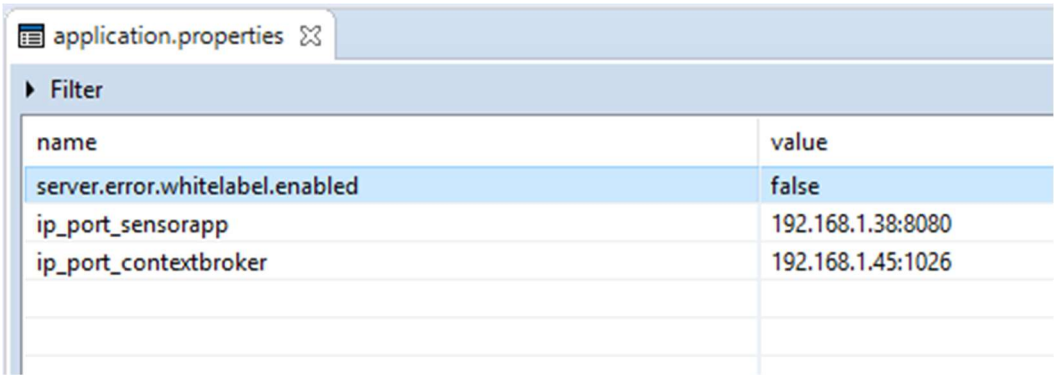

Ilustración 38: Sensorapp – Fichero application.properties

5.3 Ficheros HTML

Los ficheros HTML utilizados para dotar de una interfaz gráfica a la aplicación web, se han construido a partir del "Starter template" que ofrece Bootstrap en el siguiente enlace de su página web: https://getbootstrap.com/docs/4.3/getting-started/introduction/. Este código inicial contiene la ubicación de las librerías de estilos que ofrece Bootstrap. Los ficheros HTML se encuentran en el directorio *sensor/src/main/resources/static* del Proyecto.

- index.html: Este fichero contiene una serie de botones, cada uno de los cuales enlaza a un URI. Los botones se han obtenido de uno de los ejemplos que ofrece Bootstrap en su página web: https://getbootstrap.com/docs/4.3/components/buttons/.
  - Datos: Enlaza a */datos*.
  - Vehículos: Enlaza a */vehiculos*.
  - Tipos: Redirige a */tipos*.
  - Viajes: Redirige a */viajes*.
  - Suscribir: Enlaza con */suscribir*.
  - Suscribir todo: Dirige a */sensor/suscribir_todo*.
  - Suscripciones: Dirige a */sensor/suscripciones*.
  - Cancelar suscripción: Enlaza a */cancelar_suscripcion*.

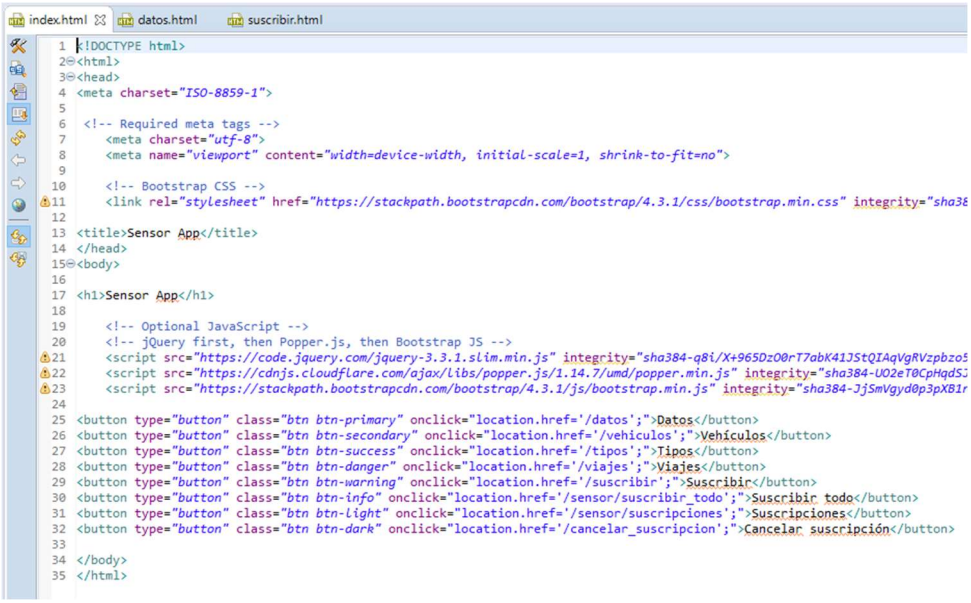

Ilustración 39: Sensorapp – Fichero index.html

Los siguientes ficheros contienen, cada uno, un formulario, un script en lenguaje JavaScript, y dos botones: uno para volver al menú principal (fichero *index.html*) y otro que ejecuta el script. Este script recopila los datos introducidos en el formulario, los almacena en variables, y redirige a un URI construido en base a los valores del formulario (consultar REST API en la siguiente sección). El código del formulario se ha obtenido de la siguiente dirección: https://getbootstrap.com/docs/4.3/components/forms/.

- datos.html: Redirige a */sensor/datos?(&{parámetro}={valor}…)*.
- vehiculos.html: Redirige a */sensor/vehiculos?(&{parámetro}={valor}…)*.
- tipos.html: Redirige a */sensor/tipos?(&{parámetro}={valor}…)*.
- viajes.html: Redirige a */sensor/datos_viaje?matricula={valor}(&{parámetro}={valor}…)*.
- suscribir.html: Enlaza con */sensor/suscribir?matricula={valor}*.
- cancelar_suscripcion.html: Enlaza con */sensor/cancelar_suscripcion?subscriptionId={valor}*.

Ilustración 40: Sensorapp – Fichero vehiculos.html

5.4 REST API

En esta sección se documenta la API REST de la aplicación web, es decir, los distintos URI y los métodos con los que se debe acceder a ellos para hacer uso de las funcionalidades implementadas en la aplicación web. Para los URI, se ha empleado la siguiente notación: los elementos que se encuentren entre llaves deberán ser sustituidos por el nombre de un parámetro o por su valor; todo aquello que esté entre paréntesis será opcional, y los puntos suspensivos indican que se puede repetir lo anterior a continuación.

5.4.1 Consulta de datos

Aquí se documenta la API para los métodos de la aplicación web que permiten consultar los datos recopilados.

Método	URI	Parámetros	Descripción
GET	/sensor/datos?(&{parámetro}={valor}…)	Los de la tabla "datos": "id", "matricula", "latitud", "longitud", "altitud", "velocidad", "direccion", "subida", "rx", "ry", "fecha" y "hora". Todos los parámetros son opcionales.	Consultar datos de circulación de un vehículo.
GET	/sensor/vehiculos?(&{parámetro}={valor}…)	Los de la tabla "vehiculos": "matricula" y "tipo". Todos los parámetros son opcionales.	Consultar vehículos existentes en base de datos.
GET	/sensor/tipos?(&{parámetro}={valor}…)	Los de la tabla "tipos": "matricula" y "velocidadMaxima". Todos los parámetros son opcionales.	Consultar tipos de vehículos y sus velocidades máximas.
GET	/sensor/datos_viaje?matricula={valor} (&{parámetro}={valor}…)	"matricula": matrícula del vehículo. Este parámetro es el único obligatorio. "idInicio" e "idFin": números de identificación de los datos donde comienza y termina el trayecto. "vmp": velocidad máxima permitida para el trayecto.	Consulta las estadísticas e infracciones de un viaje o trayecto concreto.

Tabla 1: REST API – Consulta de datos

5.4.2 Gestión de suscripciones y notificaciones

En esta subsección se describe la API de los métodos que permiten a un usuario suscribirse, consultar las suscripciones, y cancelar la suscripción de uno o todos los vehículos. También se especifica la API para la recepción de notificaciones del context broker.

Método	URI	Parámetros	Descripción
GET	/sensor/suscribir?matricula={valor}	La matrícula del vehículo.	Suscribir un vehículo.
GET	/sensor/suscribir_todo	Ninguno.	Suscribir todos los vehículos del context broker.
GET	/sensor/suscripciones	Ninguno.	Consultar las suscripciones existentes.
GET	/sensor/cancelar_suscripcion?subscriptionId={valor}	Número de identificación de la suscripción a cancelar.	Cancelar una suscripción.
POST	/sensor/notificaciones	Ninguno.	Recibe las notificaciones del context broker.

Tabla 2: REST API – Gestión de suscripciones y notificaciones

5.4.3 Servidor de Velocidad Máxima

API REST para que los dispositivos de ayuda a la conducción puedan obtener la velocidad máxima a la que debe circular el vehículo.

Método	URI	Parámetros	Descripción
GET	/sensor/vmax?latitud={valor}&longitud={valor}&tipo={valor}	“latitud”, “longitud” y “tipo” del vehículo en un instante dado.	Consultar la velocidad máxima a la que se debe circular.

Tabla 3: REST API – Servidor de Velocidad Máxima

5.4.4 Interfaz de usuario

La siguiente tabla muestra la interfaz de programación por la que el usuario puede acceder a las distintas funcionalidades que ofrece la aplicación web.

Método	URI	Parámetros	Descripción
GET	/	Ninguno.	Página principal de Sensorapp.
GET	/datos	Ninguno.	Interfaz de búsqueda de datos.
GET	/vehiculos	Ninguno.	Interfaz de búsqueda de vehículos.
GET	/tipos	Ninguno.	Interfaz de búsqueda de tipos de vehículos.
GET	/viajes	Ninguno.	Interfaz de configuración de viaje.
GET	/suscribir	Ninguno.	Interfaz para la suscripción a un vehículo.
GET	/cancelar_suscripcion	Ninguno.	Interfaz para cancelar una suscripción.

Tabla 4: REST API – Interfaz de usuario

5.5 Servidor de Base de Datos

La Base de Datos, diseñada para albergar los datos de la aplicación web, está formada por tres tablas. Las tablas se han creado utilizando el fichero *creaTablas.sql*. A continuación, se explica la utilidad de cada una de ellas y su contenido. La siguiente ilustración muestra el Modelo Entidad-Asociación bajo el que ha sido diseñada esta Base de Datos.

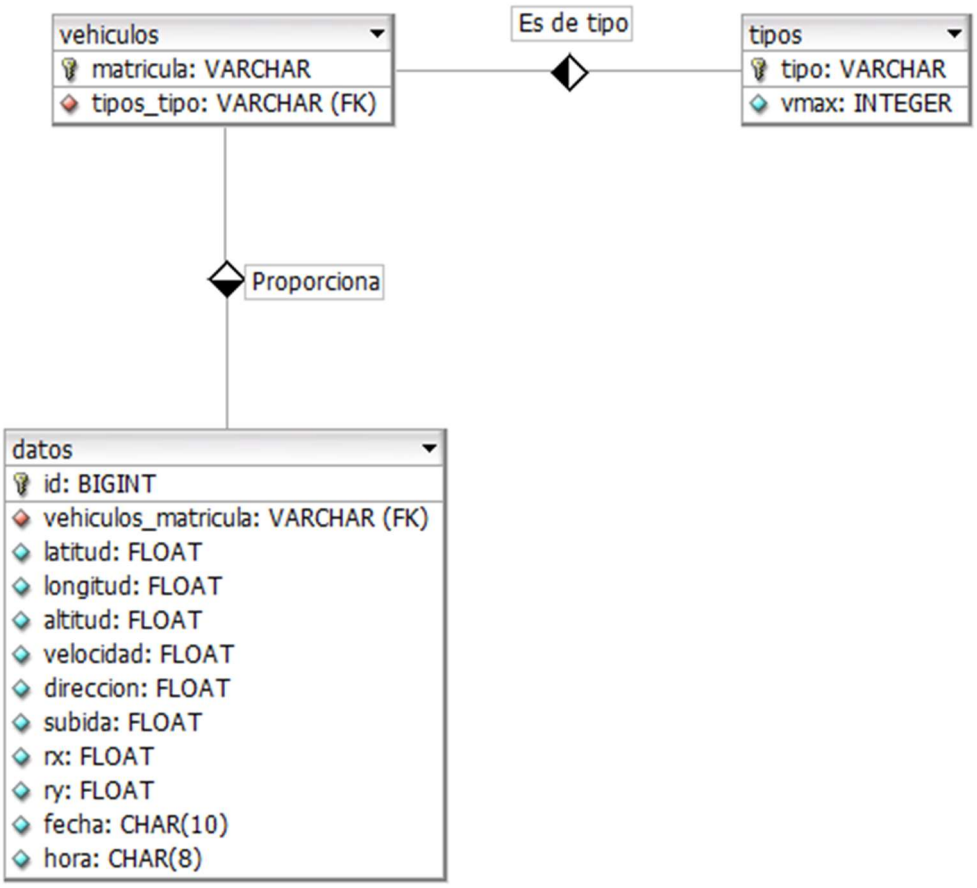

Ilustración 41: Base de Datos – Modelo E-R

- **Tabla "tipos"**

  Contiene todos los tipos de vehículos posibles y la velocidad máxima de cada uno. Sus datos se copian de un fichero llamado *tipos.txt*, cuyo contenido se ha obtenido de la Dirección General de Tráfico [16]. Los valores de esta tabla no se modifican nunca después de haberla creado y copiado los datos del fichero. Está formada por dos columnas:

  - tipo: Tipo de vehículo, es clave primaria de la tabla. El tipo de datos de esta columna es VARCHAR.
  - vmax: Velocidad máxima para un tipo de vehículo. Su tipo de dato es INTEGER.

- **Tabla "vehiculos"**

  Almacena la matrícula y el tipo de los vehículos de los que llegan datos. Formada por dos columnas de tipo VARCHAR:

  - matricula: Contiene las matrículas de los vehículos y es clave primaria.
  - tipo: Tipo de vehículo. Es una clave externa que referencia a la columna "tipo" de la tabla "tipos".

- **Tabla "datos"**

  Contiene una fila por cada dato que llega desde la aplicación web, entendiendo como dato al conjunto de los datos que se han medido para un vehículo en un instante dado. Sus columnas son las siguientes:

  - id: Número de identificación del dato almacenado. Constituye la clave primaria de la tabla y es de tipo BIGSERIAL, por lo que su valor se autoincrementará cada vez que se introduzca una fila en la tabla.
  - matricula: Número de matrícula del vehículo al que corresponde el dato. Es clave externa, referenciando a la columna "matricula" de la tabla "vehiculos".
  - fecha: Fecha de la medición del dato en formato *día/mes/año*. Es de tipo CHAR(10).
  - hora: Hora de la medición, en formato *horas:minutos:segundos*.

  Columnas de tipo FLOAT, que contienen los datos en las mismas unidades que el context broker:

  - latitud
  - longitud
  - altitud
  - velocidad
  - direccion
  - subida
  - rx
  - ry

id	matricula	latitud	longitud	altitud	velocidad	direccion	subida	rx	ry	fecha	hora
53	1234ABC	37.338910725	-6.038051117	53.82	0.2016	343.8362	5.64	5.846124522	-4.157793733	26/04/2019	16:40:11
54	1234ABC	37.338906933	-6.038046419	54.026	0.1944	348.2995	1.74	5.418692009	-5.553317796	26/04/2019	16:40:22
55	1234ABC	37.338891422	-6.038042991	53.216	0.1188	9.9463	-1.56	5.118454845	-7.875431459	26/04/2019	16:40:33
56	1234ABC	37.3388848	-6.038052748	54.212	0.3312	20.0953	12.06	5.117045775	-4.13664452	26/04/2019	16:40:43
57	1234ABC	37.338866771	-6.038056728	52.796	0.1548	205.7175	-0.9	4.989793782	-4.724919194	26/04/2019	16:40:53
58	1234ABC	37.338862031	-6.038051659	54.029	0.6084	1.6175	18.36	7.554064608	-5.086945563	26/04/2019	16:41:03
59	1234ABC	37.338871897	-6.038035906	57.474	1.7532	31.3553	27.66	9.792478185	-6.55704295	26/04/2019	16:41:13
60	1234ABC	37.338891761	-6.038035637	61.113	1.422	233.9246	12.36	4.682479949	-4.718168561	26/04/2019	16:41:23
61	1234ABC	37.338924071	-6.038051871	71.136	2.7036	17.6514	44.58	12.913894451	0.280398918	26/04/2019	16:41:34
62	1234ABC	37.338932134	-6.037987193	65.478	15.1632	91.6845	-15	6.153467298	1.208281958	26/04/2019	16:41:44
63	1234ABC	37.33891104	-6.03724786	64.095	26.2188	90.735	4.38	11.954109605	0.449427416	26/04/2019	16:41:54
64	1234ABC	37.338706529	-6.036895789	71.492	22.7808	198.8254	55.56	0	18.324527437	26/04/2019	16:42:04
65	1234ABC	37.337790095	-6.037113796	74.243	39.672	190.8919	35.04	1.06263761	-6.573780947	26/04/2019	16:42:14
66	1234ABC	37.337187569	-6.037398143	74.669	18.9936	199.2218	24.84	18.452491121	-14.443585244	26/04/2019	16:42:24
67	1234ABC	37.336756838	-6.037823741	81.049	27.45	227.0377	42.72	3.063279281	8.292865979	26/04/2019	16:42:35

Ilustración 42: Base de Datos – Tabla "datos"

5.6 Servidor de Velocidad Máxima

Se ha realizado una implementación provisional del Servidor de Velocidad Máxima dentro de la propia aplicación web. Como se verá en el Capítulo 7, se ha dejado como opción para realizar en un proyecto futuro un servidor que devuelva la velocidad máxima del vehículo en función de la posición en la que se encuentre y, por tanto, de la vía por la que circule. En este Proyecto, simplemente se ha definido la interfaz API REST para ello, y se ha implementado un método sencillo que, usando esa API, devuelve un valor de velocidad máxima estático, según el tipo de vehículo. Este método es el que ya se ha comentado en la Sección 5.1 de este capítulo: el método "vmax" de la clase *SensorController*.

6 INTERFAZ DE USUARIO Y FUNCIONALIDAD

Technology is cool, but you've got to use it as opposed to letting it use you.

- Prince -

La interfaz de usuario de la aplicación web se compone de un menú principal, accesible desde su dirección web (http://sensorapp.duckdns.org/), desde el que se accede, mediante botones, a cada una de las funcionalidades que ofrece, pudiendo volver en cualquier momento a este menú pulsando el botón "Volver". Estas funcionalidades podrían agruparse en dos tipos: consulta de datos y gestión de suscripciones. En este capítulo, se explica en qué consiste cada una de estas funcionalidades y cómo puede un usuario hacer uso de ellas.

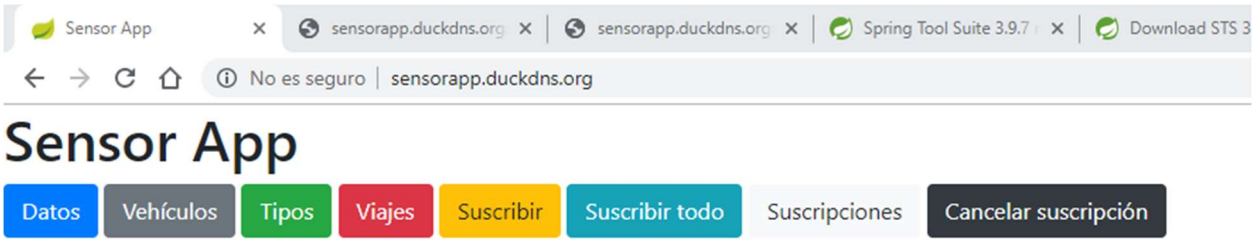

Ilustración 43: Sensorapp – Menú principal

6.1 Gestión de suscripciones

Las funciones para la gestión de suscripciones permiten al usuario suscribirse a la información de uno, varios o todos los vehículos; consultar las suscripciones vigentes en el context broker, y cancelar una suscripción por su número de identificación. Para realizar cada una de estas tareas, la aplicación cuenta con cuatro botones:

- Suscribir: Haciendo click en este botón, el usuario accede a una vista que tiene un cuadro de texto en el que el usuario puede introducir el número de matrícula del vehículo al que se quiere suscribir. Una vez introducida la matrícula, se debe pulsar en "Suscribir" para enviar la suscripción al context broker. Una vez enviada y, en función de la respuesta del context broker, se muestra al usuario un mensaje sobre la validez de la operación.

Ilustración 44: Sensorapp – Suscribir vehículo

- Suscribir todo: Con este botón, un usuario puede realizar una suscripción para extraer datos de todos los vehículos registrados en el context broker. Al pulsar el botón, se envía la suscripción automáticamente al context broker y, posteriormente, se informa al usuario del éxito o fracaso de la operación.
- Suscripciones: Al pulsar este botón, el usuario puede consultar las suscripciones existentes en ese instante en el context broker. Se debe utilizar, en caso de querer eliminar una suscripción, para obtener el identificador de suscripción. Los datos se presentan en formato JSON, tal cual aparecen en el context broker.
- Cancelar suscripción: Al hacer click, se muestra una pantalla con un cuadro de texto, en el que se puede introducir un identificador de suscripción para eliminarla y que deje de enviar notificaciones. Al pulsar en "Cancelar suscripción", una vez introducido el número de identificación, se envía al context broker el mensaje de eliminación. Una vez eliminada, se muestra un mensaje de confirmación. Si no se ha podido eliminar también se informa al usuario.

Ilustración 45: Sensorapp – Cancelar suscripción

6.2 Consulta de datos

Con las funciones de consulta de datos, un usuario puede consultar los datos recopilados tras suscribirse a un vehículo, los vehículos registrados en el context broker, los tipos de vehículos y sus velocidades máximas, y las estadísticas e infracciones en un trayecto dado. Para ello, existen otros cuatro botones:

- Datos: Al pulsar en este botón el usuario accede a una pantalla con varios cuadros de texto. En ellos se puede introducir (opcionalmente) uno o varios criterios de búsqueda de datos. Se puede buscar por: número de identificación de un dato, matrícula del vehículo, latitud, longitud, altitud, dirección, velocidad de subida, rotación de ejes X e Y, fecha y hora de la medición. Al hacer click en "Buscar", se realiza una consulta a la base de datos, y se muestra al usuario, en formato JSON, los datos existentes que cumplen los criterios introducidos, tal como se ilustra a continuación. Si no se introduce ningún criterio de búsqueda, se realiza una consulta de todos los datos existentes en la base de datos.

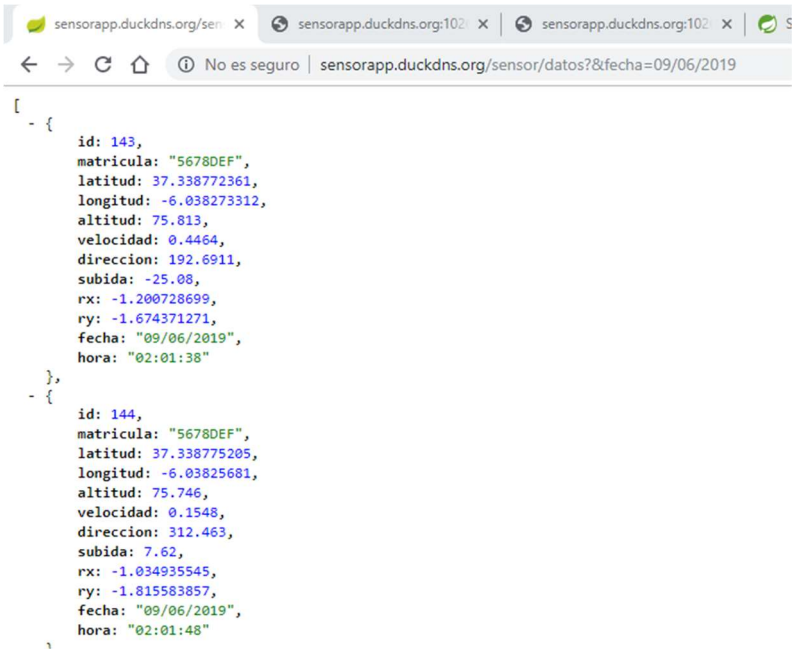

Ilustración 46: Sensorapp – Consulta de datos por fecha

- Vehículos: Cuando el usuario hace click en este botón, accede a una vista con dos cuadros de texto. Se pueden introducir opcionalmente la matrícula y/o el tipo de vehículo a buscar. Al pulsar "Buscar" se consultan los vehículos según los criterios introducidos o, si no se especifica ninguno de los dos, se consultan todos los vehículos existentes en la base de datos.

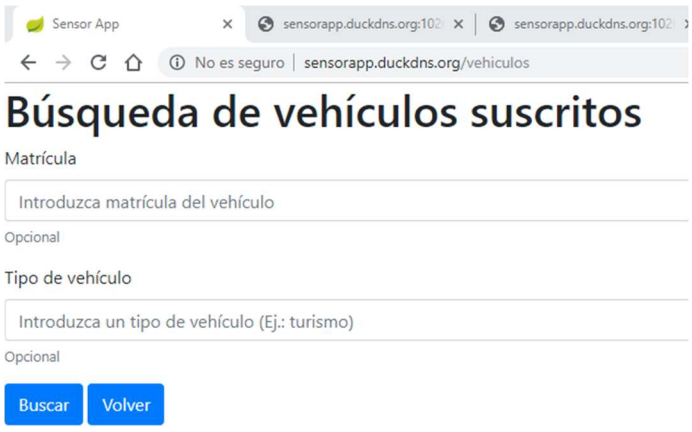

Ilustración 47: Sensorapp – Buscar vehículos

- Tipos: Con este botón un usuario puede consultar todos los tipos de vehículos posibles y sus respectivas velocidades máximas permitidas. Si el usuario selecciona esta función se le mostrará una pantalla con dos cuadros de texto para realizar una búsqueda de tipos de vehículo. Se puede buscar por tipo de vehículo y/o por velocidad máxima permitida para un tipo de vehículo. Si no se rellenan, se buscan todos los tipos de vehículos.
- Viajes: Con este botón, el usuario puede consultar las estadísticas e infracciones de un trayecto o viaje concreto. Cuando se pulsa se accede a una vista en la que se ha de introducir la matrícula del vehículo y, opcionalmente, una velocidad máxima (por omisión se toma de la tabla "tipos" de la base de datos) y un número de identificación de los datos de inicio y fin del viaje. Cuando se pulsa en "Consultar, se muestran estadísticas del viaje como la velocidad máxima alcanzada, la velocidad media del trayecto, la altitud media, etc. Además, se muestran los puntos en los que se ha superado la velocidad máxima especificada, así como la fecha y hora de la infracción.

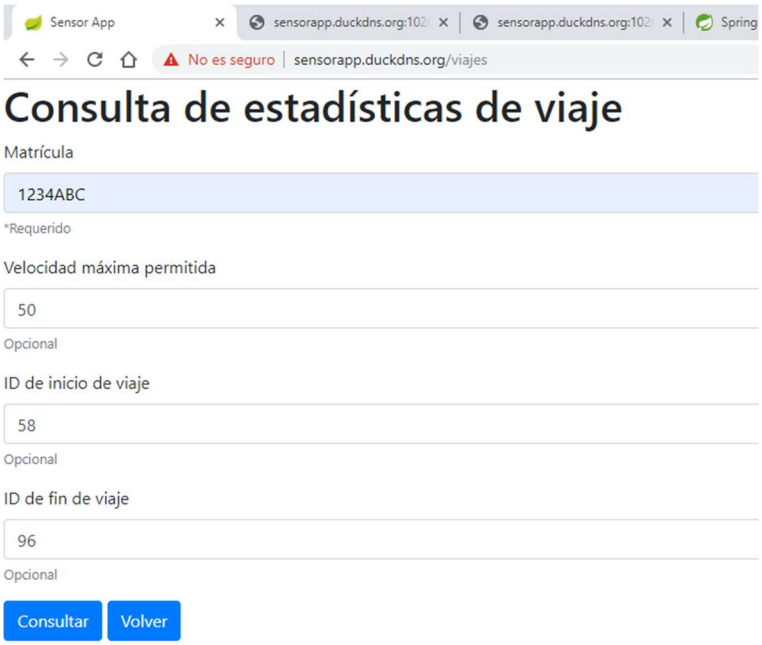

Ilustración 48: Sensorapp – Viajes

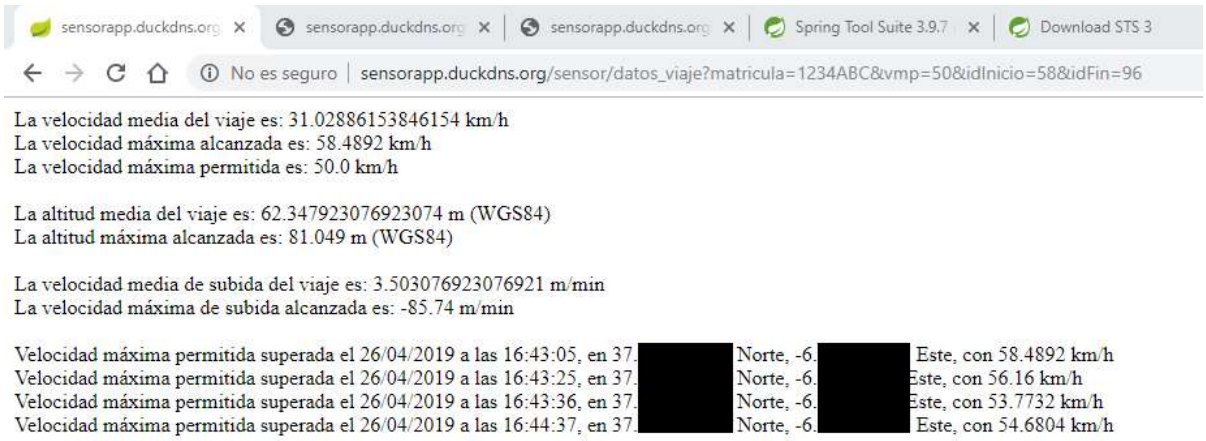

Ilustración 49: Sensorapp – Estadísticas de viaje

7 CONCLUSIONES Y LÍNEAS FUTURAS

Imagination is more important than knowledge. For knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution.

– Albert Einstein -

La aplicación web desarrollada ofrece, además de las funcionalidades vistas en este Trabajo, la posibilidad de añadir un gran número de mejoras. No obstante, hay aspectos de la implementación que han quedado incompletos.

Uno de estos aspectos, y de vital importancia, es la implementación de la seguridad, tanto del context broker como de la página web de Sensorapp. Al no presentar ninguna medida de seguridad, cualquier persona que envíe peticiones a la página web y al context broker, podría, por ejemplo, consultar datos para los que no tiene autorización, o modificar los datos de un determinado vehículo para falsear, con fines maliciosos, su información de circulación. Con el fin de proteger la integridad, confidencialidad, autenticidad y disponibilidad de los datos, es necesaria la implementación de mecanismos que garanticen la autenticación de los usuarios y el cifrado de las comunicaciones.

Otro de los aspectos que han quedado incompletos en este Proyecto, y que supondría una mejora sustancial en la utilidad del Sistema, es el desarrollo de un servidor, al que los dispositivos de ayuda a la conducción puedan consultar la velocidad máxima permitida en la vía por la que circula el vehículo, en tiempo real. Esto ofrecería una funcionalidad mucho más completa en el ámbito de la ayuda a la conducción, puesto que el conductor sería consciente en todo momento de la velocidad que lleva con respecto al límite permitido. De esta manera, podrían evitarse accidentes de tráfico y multas por exceso de velocidad.

Además, una funcionalidad muy interesante que se podría desarrollar para este Sistema, es la detección y localización de accidentes de tráfico, haciendo uso de los sensores integrados en el vehículo conectado. Gracias al acelerómetro que posee el dispositivo de ayuda a la conducción, podrían detectarse cambios bruscos en la aceleración y rotación del vehículo, con lo que se podría saber, por ejemplo, si el vehículo ha chocado o ha volcado. Además de esto, con el sensor GPS se podría ubicar el accidente en tiempo y lugar, y detectar, por ejemplo, si el vehículo ha salido de la vía o ha caído debido al accidente. Utilizando esta información, el Sistema podría alertar de forma automática a los servicios de emergencia, favoreciendo una rápida atención de las víctimas.

También, se podría emplear el dispositivo desarrollado para obtener datos sobre la topografía y geodesia del terreno. Esto es posible gracias a la información sobre la altitud y la posición que proporciona el GPS, unido a los datos sobre la rotación del vehículo que proporciona el giroscopio.

Por último, se podría mejorar la interfaz gráfica de la aplicación web, añadiendo un mapa interactivo en el que se vea reflejada la posición del vehículo en todo momento, o las rutas seguidas y sus datos y estadísticas.

ANEXO A: INSTALACIÓN DE ORION CONTEXT BROKER UTILIZANDO DOCKER

En este anexo se explica paso a paso cómo instalar y desplegar Orion Context Broker en un Docker, utilizando MongoDB como base de datos del context broker.

Para instalar Docker, es necesario contar con una máquina que posea 4GB de RAM, 3GB de espacio en disco y Linux Kernel versión 3.10 o superior [17]. En este caso, se ha utilizado una copia, en forma de máquina virtual, del sistema operativo de las salas del centro de cálculo de la ETSI: Ubuntu 16.04.5 LTS y Kernel Linux 4.15.0-47-generic. Además, es necesario tener instalado previamente el programa curl.

A.1 Instalación de Docker

Ejecutar los siguientes comandos:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
$ sudo apt-get update
$ sudo apt-get install docker-ce
```

Para comprobar que se ha instalado correctamente, ejecutar:

```
$ docker --version
```

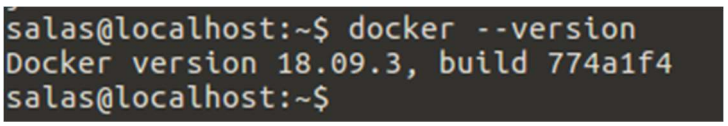

Ilustración 50: Docker – Instalación

A.2 Ejecución del Context Broker

Para arrancar el context broker, basta con teclear los siguientes comandos en un terminal:

```
$ 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
```

Para comprobar que se ha ejecutado correctamente:

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

Una vez arrancado el context broker, éste debería estar plenamente operativo para funcionar en conjunto con el resto del sistema.

```
salas@localhost:~$ curl -X GET \
>   'http://localhost:1026/version'
{
"orion" : {
  "version" : "2.2.0-next",
  "uptime" : "0 d, 0 h, 0 m, 3 s",
  "git_hash" : "cf90a534e6573505181b079a2d752fdf1ca17e12",
  "compile_time" : "Thu Apr 25 09:39:43 UTC 2019",
  "compiled_by" : "root",
  "compiled_in" : "76c06c7e1983",
  "release_date" : "Thu Apr 25 09:39:43 UTC 2019",
  "doc" : "https://fiware-orion.rtfd.io/"
}
}
salas@localhost:~$
```

Ilustración 51: Fiware Orion – Ejecución

Para detener la ejecución del context broker, se deben introducir 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 detener el context broker y comprobar la ejecución (tal como se ha hecho más arriba), no se debería obtener respuesta del context broker (conexión rechazada).

A.3 Depuración de la memoria

Existe un bug en Docker por el cual, a pesar de borrar el contenedor, la red y base de datos creadas para el context broker cada vez que se detiene su ejecución, los datos no se eliminan del disco [18]. De esta manera, poco a poco y tras cada ejecución el sistema operativo se irá quedando sin memoria de disco, por lo que será necesario eliminar manualmente dichos datos cuando se precise (normalmente el sistema operativo avisa cuando se está quedando sin espacio de almacenamiento). Para ello, basta con reinstalar Docker, borrando previamente la carpeta que contiene todos sus datos:

```
$ sudo apt-get remove docker-ce
$ sudo rm -rf /var/lib/docker
$ sudo apt-get install docker-ce$
```

A.4 Instalación del Accumulator Server

Una herramienta muy útil que proporciona Fiware es el Accumulator Server. Este servidor escucha las notificaciones que le llegan y las imprime por la salida estándar. Con él se puede testear los mensajes de notificación del context broker y comprobar el formato de los mismos.

Para instalar el Accumulator Server es suficiente con descargar el código Python desde la dirección:

https://github.com/telefonicaid/fiware-orion/blob/master/scripts/accumulator-server.py

Antes de ejecutarlo hay que asegurarse de tener instalada la librería flask de Python:

```
$ sudo apt-get install python-flask
```

Para ejecutar el Accumulator Server:

```
$ ./accumulator-server.py --port 1028 --url /accumulate --host 192.168.1.45 --pretty-print -v
```

Una vez en ejecución, si se desea que la aplicación notifique al Accumulator Server, basta con introducir como URL de notificación de la suscripción la siguiente URL:

http://IP_Accumulator_Server:1028/accumulate

La siguiente ilustración muestra una notificación recibida por el Accumulator Server, proveniente del context broker. En ella se pueden apreciar los atributos recibidos en la notificación, así como el ID de la suscripción que ha enviado dicha notificación.

```
172.18.0.3 - - [13/Mar/2019 01:19:45] "POST /accumulate HTTP/1.1" 200 -
POST http://192.168.1.45:1028/accumulate
Fiware-Servicepath: /
Content-Length: 386
User-Agent: orion/2.1.0-next libcurl/7.29.0
Ngsiv2-Attrsformat: normalized
Host: 192.168.1.45:1028
Accept: application/json
Content-Type: application/json; charset=utf-8
Fiware-Correlator: bcef66a6-4525-11e9-97cd-0242ac120003

{
    "data": [
        {
            "fecha": {
                "metadata": {},
                "type": "String",
                "value": "13/03/2019"
            },
            "hora": {
                "metadata": {},
                "type": "String",
                "value": "01:19:59"
            },
            "id": "8298HMY",
            "latitud": {
                "metadata": {},
                "type": "Float",
                "value": 45643.12334
            },
            "longitud": {
                "metadata": {},
                "type": "Float",
                "value": -34567.14
            },
            "matricula": {
                "metadata": {},
                "type": "String",
                "value": "8298HMY"
            },
            "type": "GPS"
        }
    ],
    "subscriptionId": "5c884bf83281b44321e91253"
}
=====================================
```

Ilustración 52: Fiware Orion – Accumulator Server

ANEXO B: CONFIGURACIÓN Y CONEXIÓN DE SENSORES CON RASPBERRY PI

En esta sección se explicará de forma detallada la conexión de los sensores GPS y acelerómetro, y de los leds y buzzer, con la Raspberry Pi, tanto a nivel de circuito como de software. Además, se explicará paso a paso cómo configurar la Raspberry Pi para que ejecute los programas de lectura y envío de datos en el arranque de la misma, así como su conexión Wi-Fi a una red móvil y su alimentación dentro del vehículo.

Antes de comenzar, es necesario que la Raspberry Pi tenga instalado el sistema operativo Raspbian, a ser posible su última versión. La instalación se debe realizar siguiendo las instrucciones existentes en la página web de Raspberry Pi [19]. Además, para su conexión a internet desde el vehículo, en el modelo usado de Raspberry Pi, es necesario un adaptador Wi-Fi USB de características similares al descrito en la subsección 2.1.3 del Capítulo 2: Recursos utilizados.

La conexión y configuración de los dos sensores se ha realizado tal cual indica Luis Martínez Ruiz en su TFG [2], por lo que el mismo es perfectamente válido como referencia para todo el proceso, sin embargo dicho proceso se detallará con la misma precisión en este anexo.

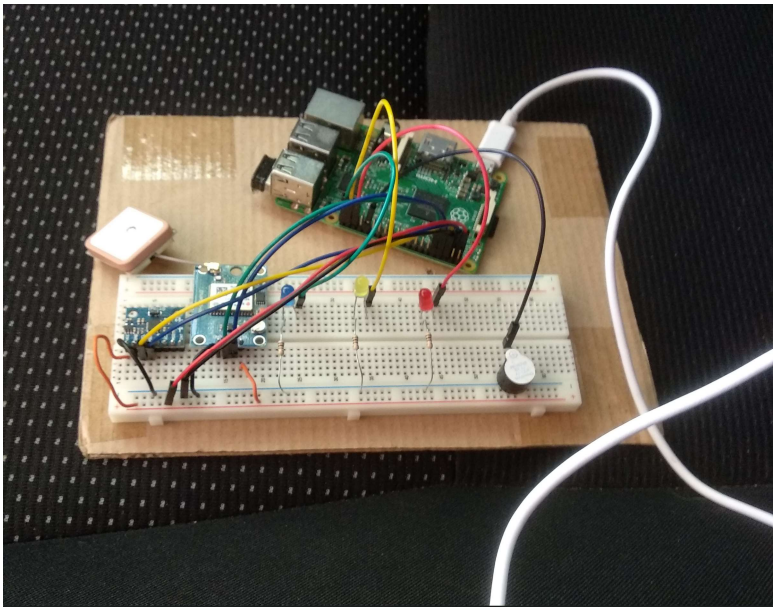

Ilustración 53: Montaje del sistema (interior del vehículo)

B.1 Sensor MPU-6050

En este apartado se detalla la conexión y configuración del sensor Giroscopio/Acelerómetro [20].

- **CONEXIÓN**

  Para la conexión del sensor con la Raspberry Pi se necesitan 4 cables macho/hembra y una placa de pruebas. La relación de pines del sensor con la Raspberry Pi viene dada por la siguiente tabla:

MPU-6050	Raspberry Pi
VCC	Pin 1 (3.3V)
SDA	Pin 3 (SDA)
SCL	Pin 5 (SCL)
GND	Pin 6 (GND)

Tabla 5: MPU-6050 – Conexión

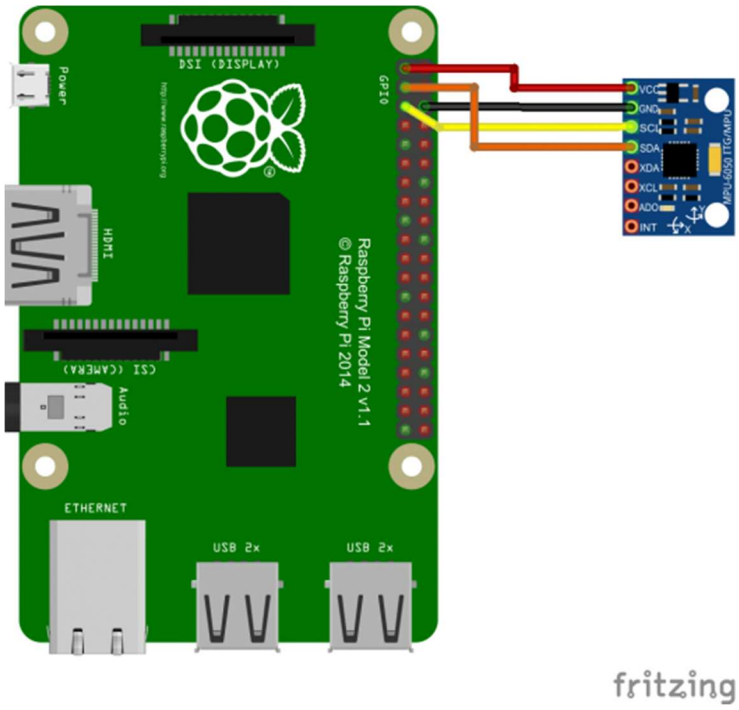

Ilustración 54: MPU-6050 – Conexión

- **CONFIGURACIÓN**

  Una vez montado el circuito, se debe configurar la Raspberry Pi para poder recibir la información del sensor por los pines a los que ha sido conectado.

  1. Para habilitar SPI e I2C hay que ir al menú de configuración de Raspbian:

     ```
     $ sudo raspi-config
     ```

     Una vez en el menú ir a *"5 Interfacing Options"* y dentro de esta activar SPI e I2C.

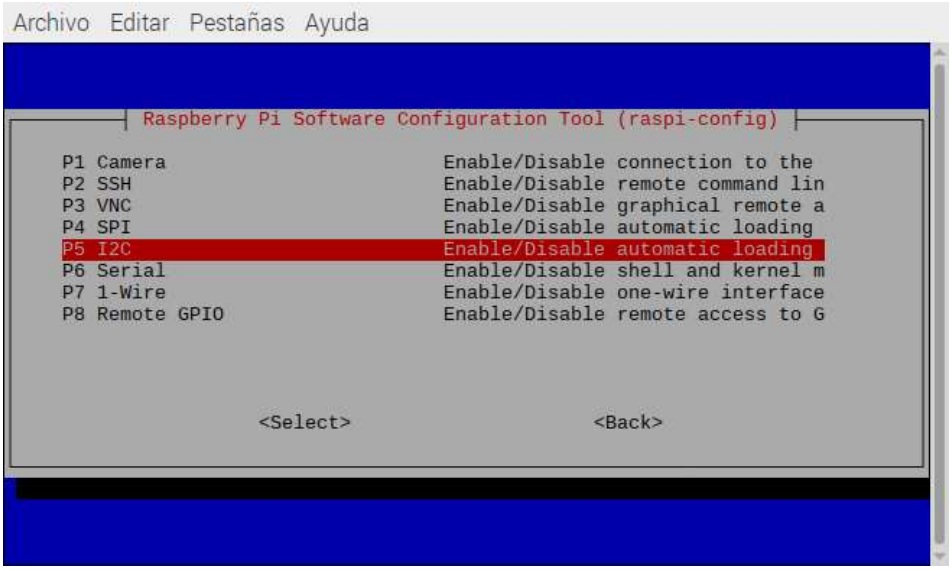

Ilustración 55: MPU-6050 – Configuración I2C

2. Después hay que editar el fichero */etc/modules* y añadir las siguientes líneas:

```
i2c-bcm2708
i2c-dev
```

3. Tras esto, deben instalarse las herramientas necesarias para leer los datos por I2C:

```
$ sudo apt-get install i2c-tools python-smbus
```

4. Por último, se puede comprobar que la conexión se ha configurado correctamente con el siguiente comando:

```
$ sudo i2cdetect -y 1
```

La salida generada por el comando debería ser igual a la mostrada en la siguiente ilustración:

Ilustración 56: MPU-6050 – Puerto I2C

Si todo ha sido configurado correctamente, la Raspberry Pi estaría lista para recoger los datos del sensor y se podría ejecutar el programa *mpu6050* para la lectura y envío de los datos al context broker.

B.2 Sensor u-blox NEO-6M-0-001

En este apartado se describe cómo realizar la conexión y configuración del sensor GPS [21] [22] [23].

- **CONEXIÓN**

  Para la conexión del sensor con la Raspberry Pi se necesitan 4 cables macho/hembra y una placa de pruebas. La relación de pines del sensor con la Raspberry Pi viene dada por la siguiente tabla:

NEO-6M-0-001	Raspberry Pi
VCC	Pin 1 (3.3V)
TX	Pin 10 (RXD0)
RX	Pin 8 (TXD0)
GND	Pin 6 (GND)

Tabla 6: NEO-6M-0-001 – Conexión

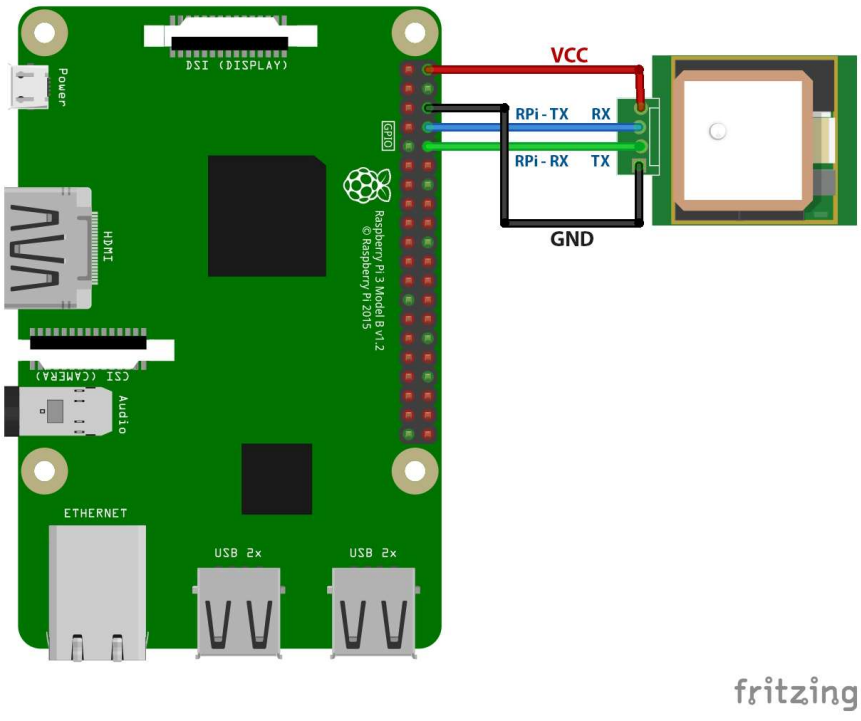

Ilustración 57: NEO-6M-0-001 – Conexión

- **CONFIGURACIÓN**

  Una vez montado el circuito, se debe configurar la Raspberry Pi para poder recibir la información del sensor por los pines a los que ha sido conectado.

1. Modificar el archivo */boot/cmdline.txt*, sustituyendo su contenido por el siguiente:

```
dwc_otg.lpm_enable=0 console=tty1 root=PARTUUID=32c518e3-02 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait quiet splash plymouth.ignore-serial-consoles
```

2. Añadir la siguiente línea al fichero */boot/config.txt*:

```
enable_uart=1
```

3. Instalar las dependencias necesarias:

```
$ sudo apt-get install gpsd gpsd-clients python-gps
```

4. Modificar el fichero /etc/default/gpsd, añadiendo las opciones que siguen:

```
# Default settings for the gpsd init script and the hotplug wrapper.

# Start the gpsd daemon automatically at boot time
START_DAEMON="true"

# Use USB hotplugging to add new USB devices automatically to the daemon
USBAUTO="true"

# Devices gpsd should collect to at boot time.
# They need to be read/writeable, either by user gpsd or the group dialout.
DEVICES=""

# Other options you want to pass to gpsd
GPSD_OPTIONS="/dev/ttyAMA0"
GPSD_SOCKET="/var/run/gpsd.sock"
```

5. Reiniciar la Raspberry Pi para que se aplique la configuración realizada.
6. Después de reiniciar, y a partir de ahora cada vez que se arranque la Raspberry Pi, se ejecutará el demonio gpsd, que extrae los datos del sensor por el puerto serie UART y permite que sean usados por el programa *neo* de recogida y envío de datos al context broker. Para comprobar que los datos están siendo recibidos correctamente, ejecutar en el terminal el comando:

```
$ cgps -s
```

Es posible que el sensor GPS tarde unos minutos en conectar con el satélite, mientras tanto el anterior comando mostrará en el campo "*Status"* el valor *"NO FIX"*. Una vez el sensor haya conectado con el satélite se empezarán a mostrar los datos y la salida será similar a la siguiente ilustración:

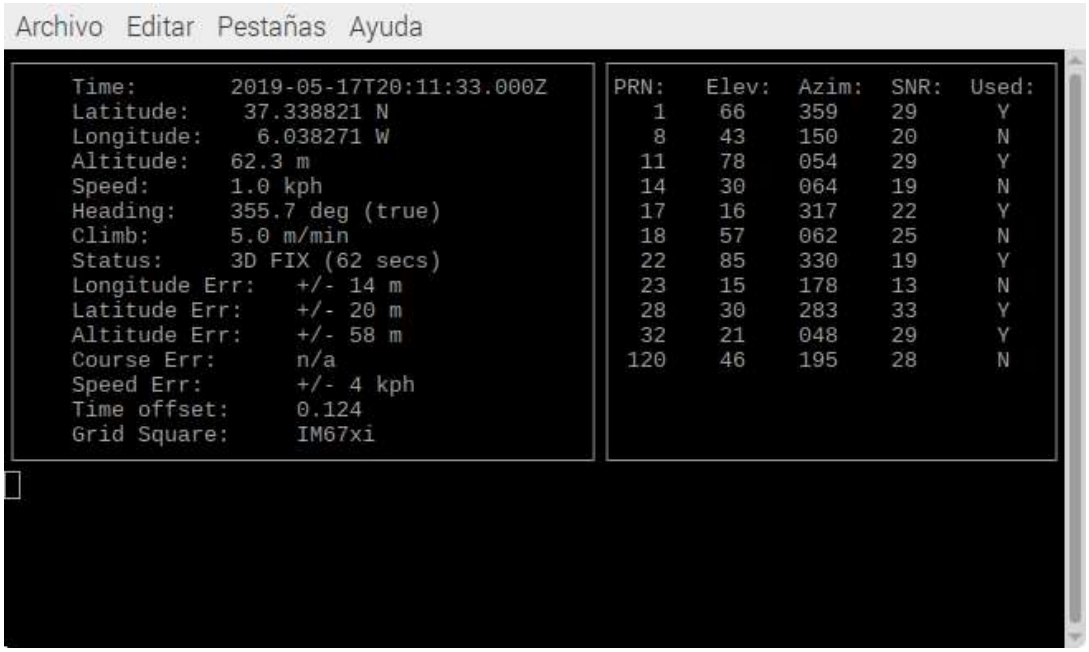

Ilustración 58: NEO-6M-0-001 – gpsd

B.3 LEDs y Buzzer

Para la adición al sistema de las diferentes señales luminosas y acústicas, se necesitan 6 cables macho/hembra, 3 leds de colores (azul, amarillo y rojo), un buzzer y 3 resistencias de 100Ω. La conexión del circuito se realizará conectando cada terminal de la columna izquierda con el correspondiente terminal de la columna derecha de la siguiente tabla:

Ánodo del led Azul	Pin 40 de la Raspberry Pi (GPIO21)
Cátodo del led Azul	Resistencia 1
Ánodo del led Amarillo	Pin 38 de la Raspberry Pi (GPIO20)
Cátodo del led Amarillo	Resistencia 2
Ánodo del led Rojo	Pin 36 de la Raspberry Pi (GPIO16)
Cátodo del led Rojo	Resistencia 3
Pin positivo del Buzzer	Pin 32 de la Raspberry Pi (GPIO12)
Pin negativo del Buzzer	Pin 6 de la Raspberry Pi (GND)
Resistencias 1, 2 y 3	Pin 6 de la Raspberry Pi (GND)

Tabla 7: LEDs y Buzzer – Conexión

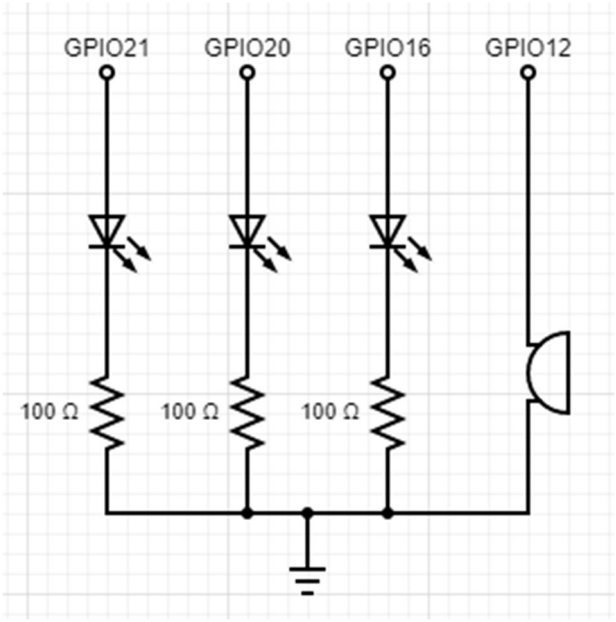

Ilustración 59: LEDs y Buzzer – Conexión

B.4 Configuración de arranque

A continuación se detallará cómo se ha configurado la Raspberry Pi para que ejecute en el arranque los programas de lectura y envío de datos de cada sensor mediante el método "autostart" [24].

1. Crear el directorio *autostart* dentro del directorio oculto */home/pi/.config*

```
$ mkdir /home/pi/.config/autostart
```

2. Crear dentro de este directorio los ficheros *mpu6050.desktop* y *neo.desktop* y asignarles permisos de ejecución:

```
$ touch /home/pi/.config/autostart/mpu6050.desktop
$ touch /home/pi/.config/autostart/neo.desktop
$ chmod +x /home/pi/.config/autostart/mpu6050.desktop
$ chmod +x /home/pi/.config/autostart/neo.desktop
```

3. En el fichero *mpu6050.desktop* escribir lo siguiente (completando la ruta del programa):

```
[Desktop Entry]
Type=Application
Name=mpu6050
Exec=python <ruta_programa_mpu6050>
```

4. Dentro del fichero *neo.desktop*, introducir lo siguiente (completando la ruta del programa):

```
[Desktop Entry]
Type=Application
Name=neo
Exec=python <ruta_programa_neo>
```

5. Reiniciar la Raspberry Pi

```
$ sudo reboot
```

Tras reiniciar, y a partir de este momento cada vez que se arranque la Raspberry Pi, los programas de recogida y envío de datos de los sensores deberían estar ejecutándose al cargar el entorno gráfico. Para comprobarlo:

```
$ ps ax | grep python
```

Y la salida del comando debería ser similar a la siguiente ilustración:

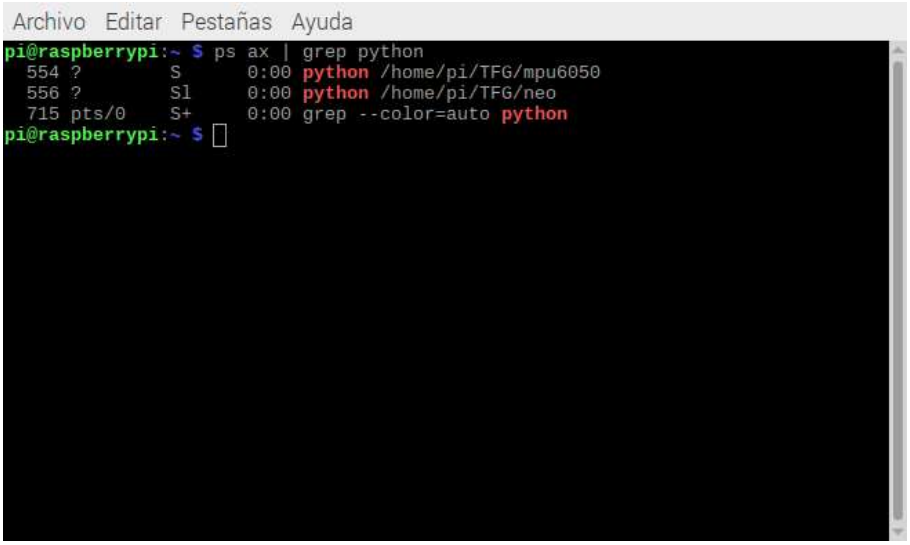

Ilustración 60: Prueba de ejecución en arranque

B.5 Conexión con red Wi-Fi móvil

Como última sección de este anexo, se explica cómo conectar la Raspberry Pi a la red Wi-Fi creada por un teléfono móvil (por ejemplo, el iPhone 5s) para enviar los datos a Internet a través de la red móvil.

Lo primero será activar el punto de acceso Wi-Fi en el teléfono móvil (previamente conectado a una red móvil y con acceso a Internet). Esto puede hacerse desde el menú *Ajustes/Compartir Internet* en iOS, o en *Ajustes/Redes e Internet/Compartir conexión/Punto de acceso Wi-Fi* en Android.

Ilustración 61: iOS – Compartir Internet

En esta ventana se pueden configurar el nombre del punto de acceso (SSID) y la contraseña del mismo. Una vez configurados los parámetros, activar el punto de acceso Wi-Fi para crear la red a la que se conectará la Raspberry Pi.

Una vez activado el punto de acceso desde el teléfono móvil, acceder al menú de configuración de Raspbian desde la consola de comandos de la Raspberry Pi:

```
$ sudo raspi-config
```

En el menú, desplazarse hasta la opción "*2 Network Options"* y dentro de ella elegir "*N2 Wi-fi"*. Se solicitará el SSID de la red y la contraseña, debiéndose introducir las configuradas en el teléfono móvil.

Ahora la Raspberry ha quedado conectada a la red Wi-Fi del teléfono y el sistema estaría listo para ser transportado en el vehículo, junto con el móvil que provee la conexión a Internet, y así poder enviar los datos de circulación y recibir la velocidad máxima de la vía desde el servidor.

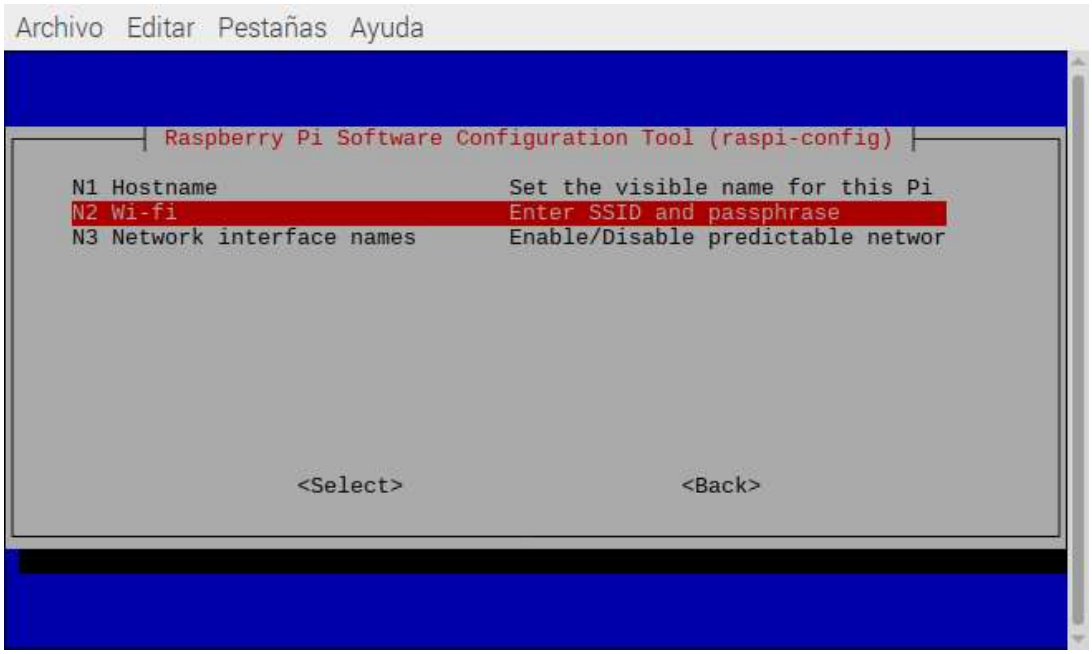

Ilustración 62: Raspbian – Conectar a red Wi-Fi

ANEXO C: INSTALACIÓN Y CONFIGURACIÓN DE STS

A continuación, se detallarán los pasos a seguir para la instalación de STS versión 3.9, junto con Hibernate Tools, en Windows 10.

C.1 Instalación de STS

1. Dirigirse a la página de Spring Tools, la URL es la siguiente:

   https://spring.io/tools3/sts/all

2. Descargar STS para Windows como se indica en la ilustración y extraer el contenido del fichero comprimido.

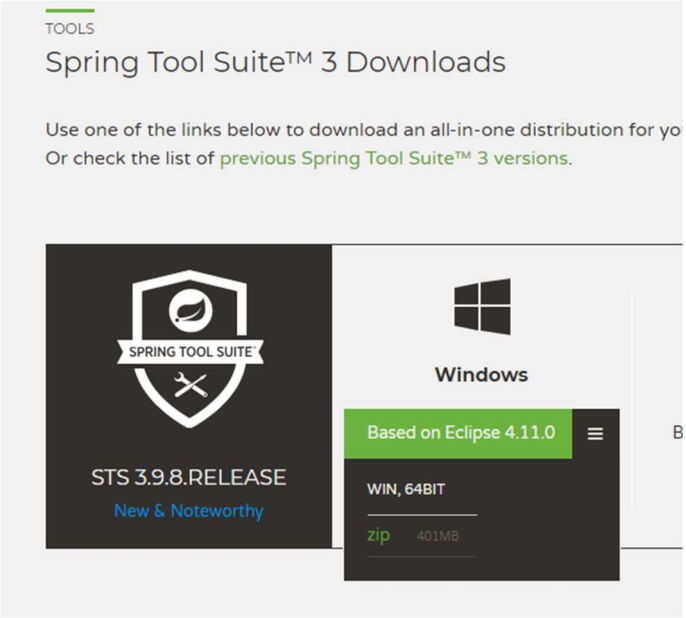

Ilustración 63: STS – Descarga

3. Se trata de una versión portable, por tanto, no necesita instalación. El fichero ejecutable del programa se encuentra en la carpeta *RELEASE* dentro del directorio descomprimido. Al ejecutarlo por primera vez pedirá seleccionar un directorio de trabajo.

C.2 Instalación de Hibernate Tools

Una vez instalado STS se puede proceder con la instalación de Hibernate Tools dentro del entorno:

1. En la interfaz de STS ir a la pestaña *Help* en la barra de menú superior. Dentro de esta pestaña seleccionar *Eclipse Marketplace…* y se abrirá una nueva ventana.

2. Dentro de la ventana de Eclipse Marketplace, en el cuadro de búsqueda, buscar: “hibernate”. En la lista de resultados, seleccionar instalar JBoss Tools. Este es un paquete de herramientas que contiene, entre otras, Hibernate Tools y, por tanto, al instalarlo se está instalando Hibernate en el entorno STS.

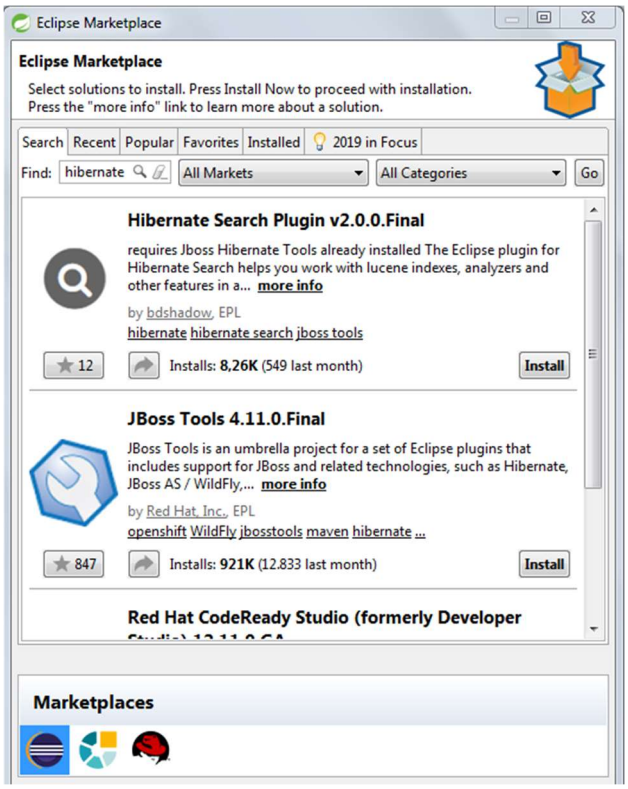

Ilustración 64: STS – Instalación de Hibernate Tools

3. Reiniciar el entorno STS tras instalar JBoss Tools (la misma aplicación lo pedirá) para que se apliquen los cambios. En este punto ya se podría usar la perspectiva Hibernate dentro de STS para crear y editar los ficheros usados por esta tecnología. Para ello, acceder a *Window/Perspective/Open Perspective/Other…* en la barra de menú superior y seleccionar la perspectiva Hibernate.

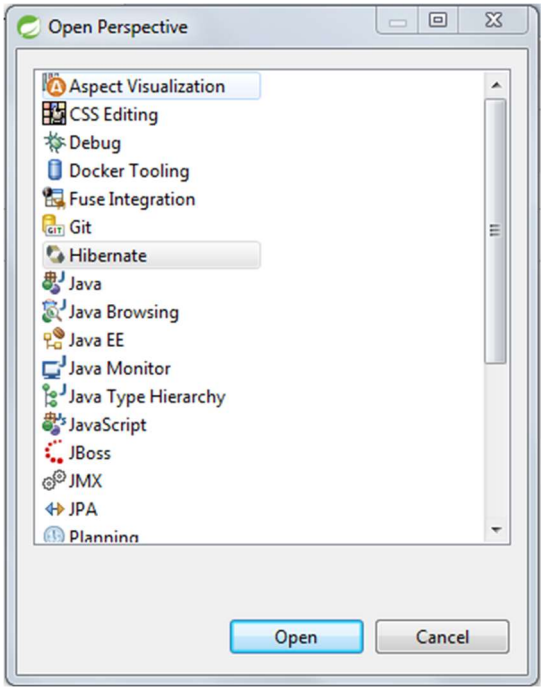

Ilustración 65: STS – Perspectivas

ANEXO D: CONFIGURACIÓN DEL SERVIDOR DE BASE DE DATOS POSTGRESQL

En este anexo se explica cómo configurar el servidor de base de datos PostgreSQL en Ubuntu para que sea accesible desde la aplicación Web. También se describirá cómo se ha creado el usuario y la base de datos a la que se accede mediante Hibernate.

Para instalar PostgreSQL, ejecutar los siguientes comandos:

```
$ sudo apt-get update
$ sudo apt-get -y install postgresql
```

Al instalar *postgresql*, se crea dentro del S.O. Linux el usuario *postgres* que, además, es superusuario del servidor PostgreSQL.

Lo primero que se debe hacer, tras instalar PostgreSQL, es crear un usuario y una base de datos. Para la aplicación de este proyecto se ha creado el usuario *dit*, propietario de la base de datos *dit*.

```
$ sudo useardd dit
$ sudo passwd dit
$ sudo passwd postgres

$ su - postgres
$ psql
postgres=> \password postgres
postgres=> CREATE USER dit CREATEDB;
postgres=> \q
$ exit

$ su - dit
$ psql -U dit -d postgres
dit=> CREATE DATABASE dit;
dit=> \q
$ psql
```

```
dit@localhost: /home/salas
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
dit@localhost:/home/salas$ su dit
Contraseña:
dit@localhost:/home/salas$ psql
psql (9.5.17)
Type "help" for help.

dit=> \l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 dit       | dit      | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
 postgres  | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
 template0 | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 | postgres=CTc/postgres+
           |          |          |             |             | =c/postgres
(4 rows)

dit=>
```

Ilustración 66: PostgreSQL – Bases de Datos

A continuación, es necesario modificar los ficheros de configuración *pg_hba.conf* y *postgresql.conf*, presentes en el directorio */etc/postgresql/9.5/main*.

1. Abrir el fichero *pg_hba.conf* con privilegios de superusuario (para poder modificarlo) y editar la línea debajo de "# IPv4 local connections:" cambiando la dirección IP desde la que se permitirán las conexiones al servidor de base de datos. En este caso, el servidor web, que será el que tenga que conectar con la base de datos, estará ubicado en su misma subred. El contenido del fichero debería ser similar al de la siguiente ilustración:

```
# DO NOT DISABLE!
# If you change this first entry you will need to make sure that the
# database superuser can access the database using some other method.
# Noninteractive access to all databases is required during automatic
# maintenance (custom daily cronjobs, replication, and similar tasks).
#
# Database administrative login by Unix domain socket
local   all             postgres                                peer

# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all   192.168.1.0/24          md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local   replication     postgres                                peer
#host    replication     postgres        127.0.0.1/32            md5
#host    replication     postgres        ::1/128                 md5
```

Ilustración 67: PostgreSQL – Fichero pg_hba.conf

2. Abrir el fichero *postgresql.conf*, también con privilegios de superusuario, y editar el parámetro *listen*_addresses cambiando su valor de " 'localhost' " a " '*' " para permitir conexiones. El puerto usado en este caso ha sido el estándar, es decir, 5432, pero si se desea cambiarlo sólo hay que modificar el parámetro *port*.

```
#------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------

# - Connection Settings -

listen_addresses = '*'    # what IP address(es) to listen on;
          # comma-separated list of addresses;
          # defaults to 'localhost'; use '*' for all
          # (change requires restart)
port = 5432       # (change requires restart)
max_connections = 100     # (change requires restart)
#superuser_reserved_connections = 3 # (change requires restart)
```

Ilustración 68: PostgreSQL – Fichero postgresql.conf

Ahora el servidor de base de datos ya es accesible desde la aplicación web, ubicada en la misma subred, para poder realizar las consultas necesarias con Hibernate.

ANEXO E: CONFIGURACIÓN DE ACCESIBILIDAD DEL SERVIDOR WEB Y EL CONTEXT BROKER

En este anexo se verá cómo configurar los accesos desde Internet al servidor de la aplicación web y al Context Broker, haciendo uso de las técnicas DNAT y DNS.

E.1 DNAT

Para dar acceso desde el exterior tanto a la aplicación web como al context broker, se ha optado por hacer DNAT en el router de la subred privada en la que están hospedados ambos servidores. Para configurarlo, se debe acceder a su interfaz gráfica mediante un navegador web, normalmente mediante la dirección IP más baja de la subred privada, en este caso: 192.168.1.1/24. Una vez dentro acceder a *Configuración Avanzada* y en la sección *NAT* seleccionar *Virtual Servers*. Desde este menú se pueden añadir distintos servicios, pudiendo configurar para cada uno el rango de puertos destino de los paquetes externos entrantes por la interfaz seleccionada (en este caso la interfaz pública del router: ppp0.1), y la IP y el rango de puertos a los que serán redirigidos.

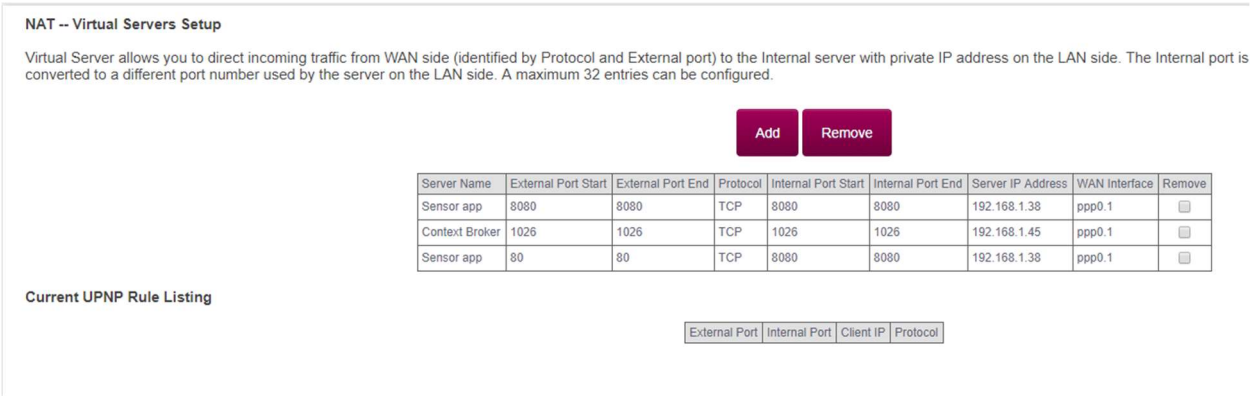

Ilustración 69: Router – Virtual Servers

E.2 DNS

Para dar un nombre de dominio a la aplicación web se ha utilizado un proveedor gratuito llamado DuckDNS. Este proporciona una aplicación que toma la dirección IP pública del router (normalmente cambiante) de forma automática y provee de servicios DNS que re direccionan a la misma [25]. Para poder configurar un nombre de dominio es necesario seguir los siguientes pasos:

1. Ir a la página web de DuckDNS y descargar su aplicación, eligiendo la opción "windows-gui". La URL es la siguiente:

   https://www.duckdns.org/

2. Registrarse en la misma web y una vez se ha entrado en la cuenta de usuario, registrar un nombre de dominio para la aplicación web. Para este proyecto, el nombre escogido ha sido *Sensorapp*. El nombre de dominio completo será igual al nombre escogido más el sufijo que añade el servidor DuckDNS: *sensorapp.duckdns.org*.

Ilustración 70: DuckDNS – Web

3. En la aplicación de escritorio, una vez instalada y ejecutándose, hacer click con el botón derecho del ratón sobre el icono de la misma en la barra inferior de tareas y seleccionar *DuckDNS Settings*. En *Domain* introducir el nombre escogido (sin sufijo) y en *Token* copiar el token generado para la cuenta de usuario creada. También se puede seleccionar un intervalo de tiempo para el que se actualizará la dirección IP pública a la que se re direcciona.

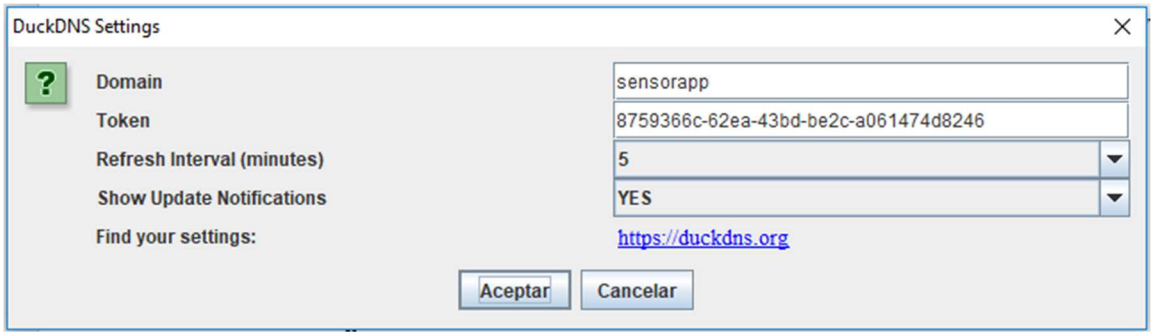

Ilustración 71: DuckDNS – Aplicación

ANEXO F: PUESTA EN MARCHA DEL SISTEMA

A continuación, se explica paso a paso la puesta en funcionamiento del Sistema completo. Todo el código necesario se encuentra en el repositorio https://github.com/pabberper/sensorapp.git.

F.1 Sensores y Raspberry Pi

Para la puesta a punto del dispositivo de ayuda a la conducción, se han de seguir los siguientes pasos:

1. Lo primero que se debe hacer es conectar y configurar los sensores, los leds y el buzzer, siguiendo los pasos del Anexo B: Configuración y conexión de sensores con Raspberry Pi.
2. Tras esto, incluir el código de los programas de control de los sensores en la Raspberry Pi. Los ficheros de código se encuentran dentro del archivo comprimido *sensores.zip*, incluido en el repositorio. Modificar el contenido del fichero *sensor.conf*, cuya estructura se especifica en la Sección 4.1 del Capítulo 4.
3. Después, configurar el arranque de la Raspberry Pi, tal como se indica en la sección B.4 del Anexo B: Configuración y conexión de sensores con Raspberry Pi.
4. Configurar la conexión con la red Wi-Fi móvil como se explica en la Sección B.5 del mismo anexo.
5. Por último, reiniciar la Raspberry Pi. Ahora el dispositivo estaría preparado para recoger y enviar los datos al context broker.

F.2 Context Broker y Base de Datos

Para el arranque del Context Broker Fiware Orion y el Servidor de Base de Datos PostgreSQL, se deben llevar a cabo las siguientes operaciones en Ubuntu:

1. Instalar y ejecutar Fiware Orion Context Broker, siguiendo los pasos detallados en el Anexo A: Instalación de Orion Context Broker utilizando Docker. Una vez instalado y ejecutado, está listo para funcionar en conjunto con el resto del Sistema.
2. Instalar PostgreSQL y crear el usuario *dit*, propietario de la base de datos *dit*. Configurar el acceso remoto a la base de datos, modificando los ficheros de configuración de PostgreSQL. Todo este procedimiento se detalla en el Anexo D: Configuración del servidor de Base de Datos PostgreSQL.
3. Una vez instalado y configurado PostgreSQL, se deben crear las tablas necesarias para almacenar los datos de los usuarios. Para ello, se proporcionan dos ficheros incluidos en el archivo comprimido *creaTablas.zip*, que puede descargarse del repositorio de GitHub. Una vez descomprimidos los dos ficheros (han de estar en el mismo directorio), ejecutar los siguientes comandos:

```
$ sudo service postgresql start
$ su dit
$ cd <directorio donde se han extraído los ficheros>
$ psql
dit=> \i creaTablas.sql
```

Hecho esto, tanto el context broker como la base de datos estarían listos para interactuar con la aplicación web.

F.3 Aplicación Web

Para importar el código de la aplicación web desde el repositorio de GitHub, es necesario abrir Spring Tool Suite y seguir el siguiente procedimiento:

1. Hacer click en "Window", "Show View" y, después, en "Other".
2. En la ventana que se abre, desplegar "Git", seleccionar "Git Repositories" y pulsar "Open".
3. En la vista que se ha desplegado, hacer click en el icono que se ilustra a continuación para clonar el repositorio.

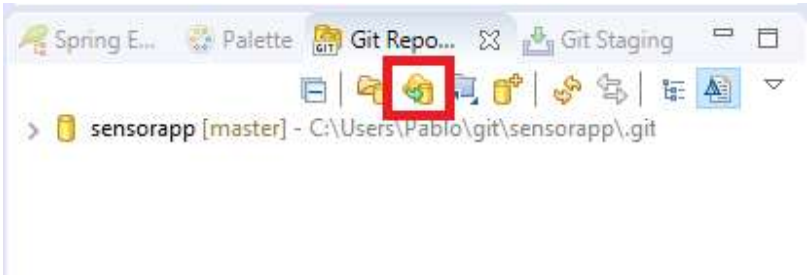

Ilustración 72: STS – Botón de clonado en vista Git

4. En el campo "URI" introducir la URL del repositorio (https://github.com/pabberper/sensorapp.git) y hacer click en "Next" y, después, click en "Next" de nuevo.

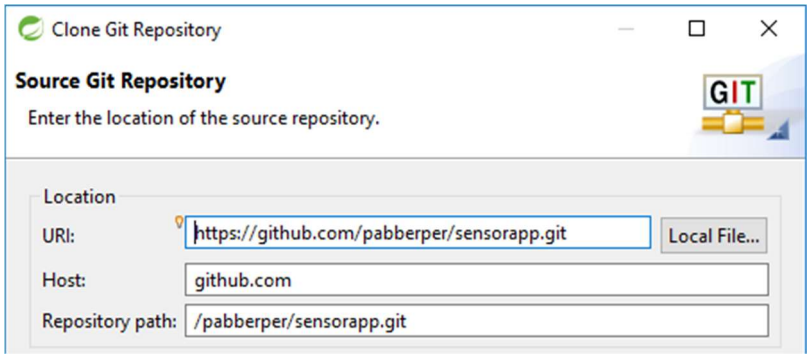

Ilustración 73: STS – Clonar repositorio

5. Seleccionar un directorio en el que guardar el repositorio local y pulsar "Finish".
6. Hacer click en "File" y, en el menú que se despliega, hacer click en "Open Projects from File System".
7. En la nueva ventana, hacer click en "Directory…" y seleccionar el directorio *sensor*, el cual se encuentra dentro del repositorio local que se ha clonado. Por último, pulsar el botón "Finish" para importar el proyecto.

Antes de ejecutar el servidor, es necesario modificar el fichero *application.properties*, especificando las direcciones IP y puertos del context broker, y de la propia aplicación web. Esto es necesario para el correcto funcionamiento de la aplicación (consultar Capítulo 5). El fichero se encuentra en la ruta *src/main/resources*, dentro del proyecto importado.

Para ejecutar el servidor, basta con hacer click derecho en el proyecto importado (se encontrará ahora en el "Package Explorer") y seleccionar "Run As" y, dentro del menú desplegable, "Spring Boot App".

REFERENCIAS

[1] A. Vidacs, *Sensor Networks and Applications,* Budapest: BME, 2018.

[2] L. Martínez Ruiz, *Aplicación web con Freeboard para la recolección de datos de sensores de acelerómetro y gps mediante Orion Context Broker de Fiware,* Sevilla: Universidad de Sevilla, 2018.

[3] Belkin, «Micro adaptador USB Wi-Fi N150,» [En línea]. Available: https://www.belkin.com/es/p/P-F7D1102/.

[4] Orlando, «MPU6050 Arduino, Acelerómetro y Giroscopio - HETPRO/TUTORIALES,» [En línea]. Available: https://hetpro-store.com/TUTORIALES/modulo-acelerometro-y-giroscopio-mpu6050-i2c-twi/.

[5] u-blox, «NEO-6_DataSheet_(GPS.G6-HW-09005).pdf,» [En línea]. Available: https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf.

[6] Docker, «What is a Container? | Docker,» [En línea]. Available: https://www.docker.com/resources/what-container.

[7] MongoDB, «What Is MongoDB? | MongoDB,» [En línea]. Available: https://www.mongodb.com/what-is-mongodb.

[8] T. I+D, «Home - Fiware-Orion,» [En línea]. Available: https://fiware-orion.readthedocs.io/en/master/index.html.

[9] PostgreSQL, «PostgreSQL: About,» [En línea]. Available: https://www.postgresql.org/about/.

[10] D. G. d. Tráfico, «La DGT pone en marcha nuevas medidas para la gestión de la velocidad.,» [En línea]. Available: http://www.dgt.es/es/prensa/notas-de-prensa/2015/20150219-La-DGT-pone-en-marcha-nuevas-medidas-para-la-gestion-de-la-velocidad.shtml.

[11] E. S. Raymond, «gpsd_json,» [En línea]. Available: http://www.catb.org/gpsd/gpsd_json.html.

[12] T. I+D, «Oneshot Subscription - Fiware-Orion,» [En línea]. Available: https://fiware-orion.readthedocs.io/en/master/user/oneshot_subscription/index.html.

[13] T. I+D, «Using regular expressions in payloads - Fiware-Orion,» [En línea]. Available: https://fiware-orion.readthedocs.io/en/master/user/regex_in_payload/index.html.

[14] T. I+D, «API Walkthrough - Fiware-Orion,» [En línea]. Available: https://fiware-orion.readthedocs.io/en/master/user/walkthrough_apiv2/index.html.

[15] T. I+D, «Initial notification - Fiware-Orion,» [En línea]. Available: https://fiware-orion.readthedocs.io/en/master/user/initial_notification/index.html.

[16] D. G. d. Tráfico, «ANEXO II: DEFINICIONES Y CATEGORÍAS DE LOS VEHÍCULOS,» [En línea]. Available: http://www.interior.gob.es/documents/642012/1931881/02.+ANEXO+II.pdf/2b8a13e6-44bf-

410a-8faa-a23098bb3706.

[17] Docker, «UCP System requirements | Docker Documentation,» [En línea]. Available: https://docs.docker.com/v17.09/datacenter/ucp/2.1/guides/admin/install/system-requirements/.

[18] N. Waisbrot, «diskspace - Why is docker image eating up my disk space that is not used by docker - Stack Overflow,» [En línea]. Available: https://stackoverflow.com/questions/27853571/why-is-docker-image-eating-up-my-disk-space-that-is-not-used-by-docker.

[19] R. Pi, «Installing operating system images - Raspberry Pi Documentation,» [En línea]. Available: https://www.raspberrypi.org/documentation/installation/installing-images/.

[20] R. Pi, «Measuring Rotation and acceleration with the Raspberry Pi,» [En línea]. Available: https://tutorials-raspberrypi.com/measuring-rotation-and-acceleration-raspberry-pi/.

[21] sspence, «Raspberry Pi & the Neo 6M GPS: 5 Steps,» [En línea]. Available: https://www.instructables.com/id/Raspberry-Pi-the-Neo-6M-GPS/.

[22] D. TheMan, «Connecting u-blox NEO-6M GPS to Raspberry Pi | Big Dan the Blogging Man,» [En línea]. Available: https://bigdanzblog.wordpress.com/2015/01/18/connecting-u-blox-neo-6m-gps-to-raspberry-pi/.

[23] M. Shop, «Localizador GPS con Raspberry Pi y Ublox NEO 6M | Minitronica.com,» [En línea]. Available: https://www.minitronica.com/tracker-gps-raspberry-pi-ublox-neo/.

[24] S. Hymel, «How to Run a Raspberry Pi Program on Startup - learn.sparkfun.com,» [En línea]. Available: https://learn.sparkfun.com/tutorials/how-to-run-a-raspberry-pi-program-on-startup/all.

[25] DuckDNS, «Duck DNS - about,» [En línea]. Available: https://www.duckdns.org/about.jsp.