UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Sistema para el control del posicionamiento en tiempo real y análisis de flotas de vehículos de transporte

Sergio Salcedo Vicente
Tutor: Roberto Latorre Camino

Julio 2021

Sergio Salcedo Vicente
Sistema para el control del posicionamiento en tiempo real y análisis de flotas de vehículos de transporte

Sergio Salcedo Vicente
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Sistema para el control del posicionamiento en tiempo real y análisis de flotas de vehículos de transporte

AUTOR: Sergio Salcedo Vicente
TUTOR: Roberto Latorre Camino

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2021

Resumen (castellano)

Ante la situación que estamos viviendo en los últimos meses, el mundo de la logística está en auge. Cada vez más empresas y ecommerce se están viendo obligados a ofrecer sus servicios de forma online con el fin de llegar a más clientes.

Ante este escenario, es imprescindible que las empresas tengan una correcta digitalización de su negocio y puedan gestionar sus pedidos de la forma más eficiente posible. Para garantizar una correcta entrega de los servicios cumpliendo los tiempos de entrega fijados, es imprescindible realizar una logística impecable. La logística es una parte fundamental de un negocio que lleva mucho tiempo gestionar correctamente. Para ayudar a las empresas a gestionar fácil, rápida y eficientemente sus envíos, permitiendo que puedan centrarse de pleno en el núcleo de su negocio, surge la necesidad de desarrollar este TFG.

Con envíos y rutas de transporte por toda España, es imprescindible disponer de una herramienta para el control y la gestión eficiente de flotas de transportistas que permita localizar a cada uno de ellos de forma precisa, así como gestionar las posibles incidencias que pueden ocurrir durante una ruta de entrega, facilitando además el análisis del comportamiento de los transportistas durante sus servicios y la identificación de las zonas más demandadas de cara a la optimización del proceso en el futuro.

Este TFG consiste en desarrollar un piloto que permita controlar y analizar las flotas de vehículos con el fin de gestionar las incidencias y ayudar a aportar un servicio más rápido para el reparto bajo demanda.

Por otra parte, el sistema integrará una app nativa en Android que se utilizará como mecanismo de recogida y envío de coordenadas para el tracking de cada uno de los transportistas.

Abstract (English)

Given the situation that we are experiencing in the last months, logistics is in the boom. More and more companies are being forced to offer their services through an online channel to target more customers.

Under this scenario, companies should make their business logic online and efficiently manage their orders. To guarantee a proper delivery and meet the time deadlines, it is crucial to make perfect logistics. Logistics is an important part of a business that takes too much effort to manage properly. In order to help companies to manage their orders in a fast, easy, and efficient way, letting them focus on their business core, borns the idea of this TFG.

Having orders and routes all over Spain, it is critical to have a tool to control and manage courier fleets that let you locate every single vehicle with high accuracy and manage some incidences that may occur during delivery. It is also important to analyze how vehicles are performing and identify hot points to optimize the process in the future.

This TFG aims to develop a prototype that lets you control and analyze Courier fleets to manage incidents and help the business make a quicker and efficient delivery on demand.

On the other hand, the system will include a native Android App that will be used as a coordinates tracker to locate the position of every single vehicle registered on the system.

Agradecimientos

A mi familia y amigos por su apoyo y soporte a lo largo de toda la carrera.

A mis compañeros de la escuela que me han ayudado cuando lo necesitaba.

A todos los profesores de la escuela que han contribuido en mi formación y en especial a Roberto Latorre por aceptar la tutelación mi proyecto y guiarme durante el desarrollo del Trabajo de Fin de Grado.

Finalmente, agradecer el compromiso de mi compañero de prácticas a lo largo de toda la carrera, Sergio Torrijos, que ha contribuido en las buenas calificaciones y resultados obtenidos.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Cada vez más negocios se están dando cuenta de la importancia de tener una correcta digitalización de su negocio para poder llegar al mayor número de clientes posible.

La mayoría de las empresas que necesitan realizar envíos bajo demanda como por ejemplo negocios de hostelería, floristerías etc. no disponen de una metodología ágil para gestionar este proceso. Muchos acaban recurriendo a métodos rudimentarios como el uso de Excel para enviar las direcciones, llamar reiteradamente al transportista para saber su ubicación actual y poder asignarle direcciones cerca de su ubicación.

Esto no solo ralentiza los procesos de la empresa, sino que también el rendimiento del transportista ya que debe estar constantemente pendiente del teléfono y parar de trabajar para atender las nuevas directrices.

Además, algunos transportistas infringen las normas del negocio utilizando el vehículo de empresa para uso personal, cuando no deberían. Si a esto se le suma detenciones espontáneas en mitad de la jornada laboral, esto puede incurrir en excesos de velocidad para poder cumplir los tiempos de entrega con sus respectivas sanciones económicas, ocasionando de esta forma un coste adicional que *a priori* el propietario no tiene forma de medir y por tanto de justificar.

Para solventar esta problemática, nace FleetTracker, que es la plataforma propuesta y desarrollada en este TFG. FleetTracker ofrece una solución de control de flotas de vehículos para saber en todo momento su posicionamiento y poder controlar de forma más sencilla nuevas direcciones de entrega entrantes.

Gracias al sistema de control de velocidad se puede saber de forma instantánea la velocidad punta del vehículo y con el sistema de grabaciones, se puede grabar y revisar el posicionamiento exacto del transportista después de su jornada laboral y visualizar estadísticas.

1.2 Objetivos

- Visualización en tiempo real de la ubicación de cada vehículo de la flota.
- Visualización conjunta para tener una vista global del posicionamiento de cada vehículo.
- Posibilidad de grabar el recorrido en tramos y ver el posicionamiento exacto del transportista durante una jornada de trabajo.
- Añadir/eliminar vehículos de la flota y ver el estado y características de cada uno: matrícula, modelo, etc.

Entre las funcionalidades avanzadas del seguimiento destacan:

- Correcta orientación del vehículo en el mapa (bearing).
- Visualización de la ubicación a una tasa aceptable de 15fps aprox.
- Tolerancia a pérdidas de conexión tipo túneles, agotamiento de batería etc.

2 Estado del arte

En este capítulo se describirán las funcionalidades de otras plataformas de control de flotas disponibles en el mercado. Se destacarán las características de cada una de ellas y se justificará la necesidad de crear una herramienta como la presentada en este TFG.

2.1 Análisis de competencia

	Movolytics [1]	Webfleet [2]	Portefy Fleet [3]	FleetTracker
Múltiples Vehículos	Sí	Sí	Sí	Sí
Seguimiento a tiempo real	Básico	Básico	Básico	Avanzado
Historial de movimientos	Sí	Sí	Sí	Sí
Privacidad de los datos	Sí	Sí	Sí	Sí
Interfaz sencilla	No	No	Sí	Sí
Planificación de rutas	Sí	Sí	Sí	No
Dispositivo de Tracking	Localizador GPS	Localizador GPS	Smartphone	Smartphone

Tabla 2-1: Análisis de funcionalidades de diferentes plataformas en el mercado

La tabla 2.1 muestra las características de distintas soluciones en el mercado. Los datos han sido extraídos de diferentes páginas comparativas además de un análisis personal en el que se han destacado las funcionalidades específicas ofrecidas en cada una de las soluciones.

Analizando el cuadro comparativo anterior, se pueden destacar las fortalezas y debilidades en conjunto de todas las soluciones propuestas. Todas ellas ofrecen seguimiento en tiempo real sobre múltiples vehículos.

FleetTracker es la única solución que ofrece un seguimiento a tiempo real avanzado, ya que permite obtener al más mínimo detalle la posición exacta del vehículo con el correcto ángulo de giro y el movimiento de forma continua sin intervalos. Esto permite saber exactamente la trayectoria que un vehículo está siguiendo en todo momento, ya que la posición no se percibirá con saltos.

Todas las soluciones ofrecen un historial de recorridos o grabaciones, que permite grabar el posicionamiento de los vehículos durante las jornadas de trabajo, que posteriormente se pueden consultar para obtener estadísticas y generar informes de

rendimiento. Además, todas ellas se encargan de tratar correctamente los datos personales de las empresas como son las contraseñas, historiales de ubicación etc.

Movolytics y WebFleet tienen bastantes opciones, pero la interfaz de uso no es tan sencilla. Al final, estas plataformas disponen de ciertos menús cuyo uso es eventual. La mera presencia de estos menús dificulta la navegación del usuario para las tareas más importantes. Esto es un gran inconveniente ya que muchos clientes no necesitan de todos los informes y estadísticas avanzadas. Requieren de una versión "Lite" en la que no necesites formación y se pueda usar desde la primera vez que se entra en el sistema. Esto se ha conseguido en las soluciones Portefy Fleet y FleetTracker.

En cuanto al dispositivo de trackeo del vehículo, podemos observar que, en las 2 primeras soluciones, se necesita adquirir un dispositivo Hardware GPS por cada vehículo que cuesta alrededor de 50 y 150€. Si se tiene una flota de 10 vehículos, habría que hacer una inversión de entre 500 a 1500€. Esto supondría un coste extra considerable. Además, hay que estar pendiente de la batería de los dispositivos, ya que en cualquier momento se puede agotar y si no se está pendiente, puede suponer pérdidas de seguimiento hasta que se recarguen las baterías.

Esto no ocurre en las 2 últimas soluciones, donde el dispositivo de trackeo es el propio dispositivo móvil del trabajador. De esta forma, no hay que realizar ninguna inversión extra, ya que al tratarse de smartphones, la batería siempre se puede monitorizar e incluso llevar un cargador conectado al mechero del coche para no agotar la batería durante una jornada de trabajo.

En cuanto a la planificación de rutas, FleetTracker es la única plataforma que no ofrece dicha característica. No obstante, si hacemos una comparativa general de todo el cuadro, vemos que si combinamos las características de Portefy Fleet y FleetTracker obtenemos una solución de seguimiento integral con todas las características.

Este es otro de los objetivos del proyecto; tomar el TFG como piloto para poder realizar una combinación de ambas plataformas en un futuro y conseguir una solución que presente todas las características necesarias para un control total.

2.2 Problemas de las empresas

Bajo mi propia experiencia, tras el análisis de los procesos logísticos de pequeños comercios durante los últimos meses, he detectado que en los envíos al ser la última parte de la cadena de un pedido o servicio, no se le presta la atención necesaria y eso desencadena en pérdidas de rendimiento que a la larga repercuten negativamente en la evolución del negocio.

Las empresas con reparto propio que no utilizan ningún software de seguimiento de flotas, optimización ni estadísticas acaban teniendo un coste más alto de lo esperado por no optimizar y gestionar bien los procedimientos.

Esto ocurre por varias razones entre las que se pueden destacar las siguientes:

- **Creación de rutas ineficiente**: El procedimiento que se suele emplear en algunas empresas con reparto propio, es leer las direcciones de todos los envíos que hayan

entrado el día anterior, crear una hoja de ruta (optimizando las direcciones como se pueda, sin garantía alguna) y entregársela al transportista para que le vaya realizando durante el día.

- **Consumo excesivo de combustible**: Debido a que las rutas no están optimizadas, esto genera un consumo de combustible extra que se podría ahorrar con el simple hecho de realizar una correcta optimización.

Por otra parte, pueden existir ocasiones en las que se tenga que desviar a un transportista a realizar una recogida en una dirección extra. Si en un momento determinado se encuentran en ruta 10 vehículos, al no tener la ubicación exacta de los mismos, el encargado de tráfico debería llamar 1 a 1 a todos los transportistas confirmando su ubicación y elegir al que más cerca se encuentra en el momento para que atienda dicha solicitud. Todo esto bajo condiciones ideales en las que todos los transportistas están disponibles y cogen el teléfono. Si no fuese el caso, (situación muy probable), habría que llamar repetidamente hasta poder comunicarse con ellos. Esto provoca una pérdida de tiempo que se puede ahorrar con un simple vistazo al mapa de seguimiento de FleetTracker y llamar exclusivamente al transportista que se encuentre en ese momento más cerca de la ubicación.

- **Incertidumbre acerca de la actividad del vehículo**: Al no disponer de ningún dispositivo de GPS que monitorice en todo momento la actividad del vehículo, no se puede saber si se está usando el vehículo/vehículos de empresa para los fines establecidos, o se está usando con fines ajenos a la empresa, lo cual repercute en un gasto de combustible adicional, además de desgaste del vehículo.

  Gracias al sistema de trackeo de FleetTracker, esto es posible controlarlo de forma muy sencilla, ya que con el sistema de grabaciones se puede dejar grabando un trayecto durante toda la jornada de trabajo y revisarlo después, para ver exactamente la actividad del vehículo y los lugares más transitados.

- **Control de velocidad**: Algunos transportistas arriesgan más de la cuenta y circulan a velocidades superiores a las máximas permitidas de las carreteras. Esto puede derivar en sanciones económicas por exceso de velocidad, poner en riesgo la seguridad vial, además de un aumento del consumo de gasolina por circular a velocidades muy superiores a las que se debería.

  Con FleetTracker, es posible llevar en todo momento un control de las velocidades y comprobar si el transportista está cumpliendo los límites de velocidad establecidos en el tramo por el que circula en cada momento

Por todo lo expuesto, surge la necesidad de una herramienta para ayudar a los negocios a solventar esta problemática como la propuesta del TFG, FleetTracker.

En los próximos apartados, se describirá con más detalle cada una de las partes que componen esta herramienta, el stack de tecnología empleado y una serie de pruebas y resultados del sistema funcionando en entornos simulados y reales.

3 Diseño

En esta sección, se muestran las diferentes interfaces gráficas de usuario que componen el sistema de tracking. Por un lado, se describirán las vistas de la App en Android encargada de la transmisión de posicionamiento y por otro el diseño de las pantallas de la WebApp encargada de la recepción del posicionamiento, control y visualización de los vehículos que componen la flota.

3.1 App para la geolocalización de los vehículos

3.1.1 Log-in

Lo primero que se visualiza al abrir la aplicación es la pantalla de Login, donde se debe introducir la matrícula del vehículo que queremos realizar seguimiento.

La App realiza una petición http al endpoint de iniciar sesión que devolverá OK en el caso de que la matrícula sea correcta. En el caso de que el Login sea correcto, en la respuesta se incorporará un token JWT que será necesario introducir en los headers de futuras peticiones para poder consumir los recursos de la API REST.

Figura 3-1: Pantalla de inicio de sesión de la App

3.1.2 Home

Una vez que se complete el Login inicial, en la App se mostrará la vista Home que tiene los controles necesarios para poder realizar la tarea del seguimiento a tiempo real.

Cabe destacar que existen dos modos diferentes. Por una parte, tenemos el modo "Live", que obtiene las coordenadas físicas del dispositivo con intervalos de 1 segundo. Esta frecuencia de actualización es suficiente para realizar un seguimiento de gran calidad y se puede configurar para que sea mayor o menor. Hay que tener en cuenta que el teléfono siempre intentará obtener el posicionamiento con la frecuencia establecida, pero en función de la ubicación y los satélites GPS que existan, podrán surgir ocasiones donde no se pueda cumplir el intervalo establecido. No obstante, FleetTracker está optimizado para dar servicio incluso con este tipo de pérdidas.

Por otra parte, se dispone de un Modo "Debug". Este modo es muy útil para probar el correcto funcionamiento de la transmisión de coordenadas sin necesidad de salir de casa.

Simplemente se selecciona una ruta preestablecida del ComboBox y se pulsa sobre "Empezar transmisión" para comenzar la emisión de coordenadas.

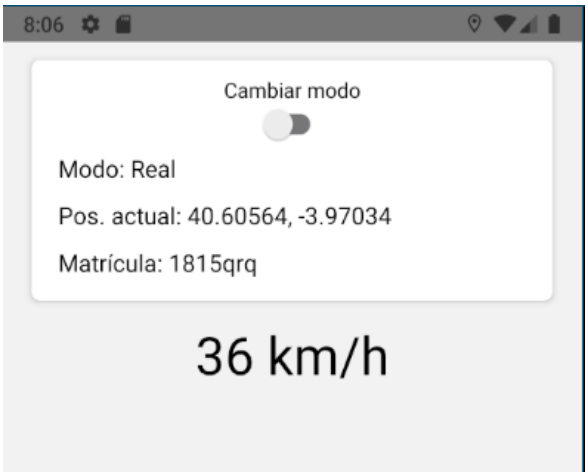

Figura 3-2: Modo Real

Figura 3-3: Modo debug

3.1.3. Selector de itinerarios

Como se comentó previamente, el modo Debug dispone de una serie de rutas precalculadas que permiten probar el correcto funcionamiento de la transmisión. Estas rutas están compuestas por Polylines que es una forma de codificar un array de coordenadas latitud longitud, con la intención de tener un formato más compacto. Se explicará más detalladamente en el capítulo 4.4.3.

Se pueden incluir rutas propias introduciendo el Polyline del itinerario correspondiente.

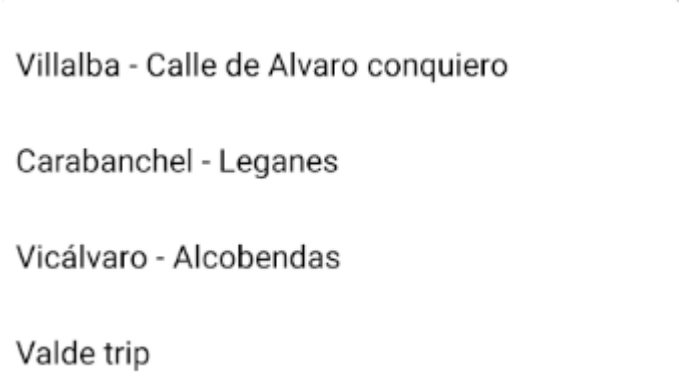

Figura 3-4: Distintos itinerarios en modo Debug

3.2 WebApp para el control de flotas de vehículos

3.2.1 Log-in

Nada más introducir la URL de la web en el navegador, nos aparecerá la pantalla de Login. Se deberán introducir las credenciales para poder acceder a la vista principal.

Al igual que en la App, se realiza una petición al server que comprobará si las credenciales son correctas. En cuyo caso otorgará acceso al sistema y se devolverá un JWT para realizar próximas peticiones a la API.

Por motivos de seguridad, la sesión se mantiene abierta durante unas horas y si después de este tiempo se quiere volver a acceder, se deberán proporcionar nuevamente las credenciales de acceso.

Figura 3-5: Pantalla de Login en la WebApp de gestión de flotas

3.2.2 Home

Una vez que se haya ingresado correctamente al sistema, se mostrará la vista Home. Esta vista contiene los controles necesarios para gestionar y visualizar los vehículos de nuestra flota.

La primera vez que se accede a la página, no disponemos de ningún vehículo. Por lo que se puede añadir uno pulsando sobre "Añadir vehículo". Saldrá un Modal que pedirá introducir la matrícula del vehículo del cual realizar seguimiento. También servirá como Identificador.

En la parte superior, encontramos 2 labels que muestran los vehículos totales en la flota, y cuántos de ellos están online (operativos) en ese momento. Un vehículo está de servicio cuando tiene la App abierta o en segundo plano. Una vez que se cierra, la web detecta dicho evento y se marca como Offline. De esta forma el encargado de tráfico puede visualizar la actividad de los vehículos.

En la parte superior derecha, hay un botón llamado "Vista global". Si se pulsa sobre este botón, se puede obtener la ubicación de todos los vehículos online de la flota de forma instantánea. Si se pulsa nuevamente, se ocultará la posición de los vehículos.

En el panel principal, se dispone de una sección con los vehículos de la flota, su estado y la posibilidad de visualizar estadísticas y el posicionamiento a tiempo real de dicho vehículo en el caso de pulsar el icono del ojo.

Si se pulsa sobre el icono de la papelera, se mostrará un modal emergente que pedirá confirmación al usuario para eliminar dicho vehículo.

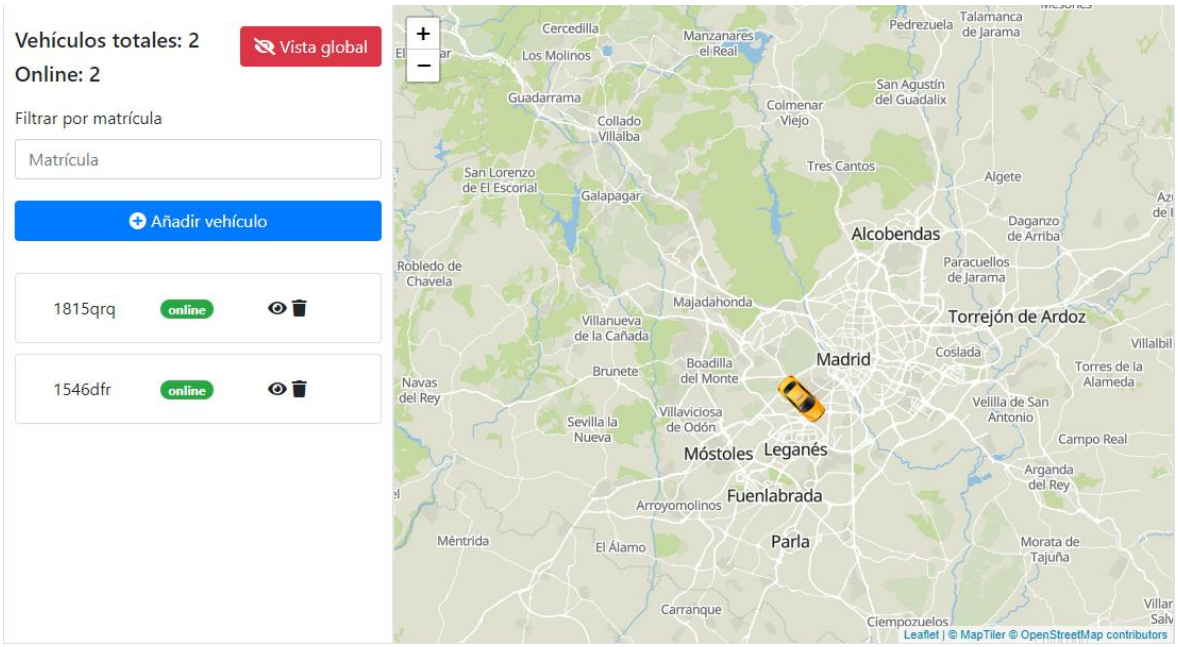

Figura 3-6: Vista global habilitada de los vehículos

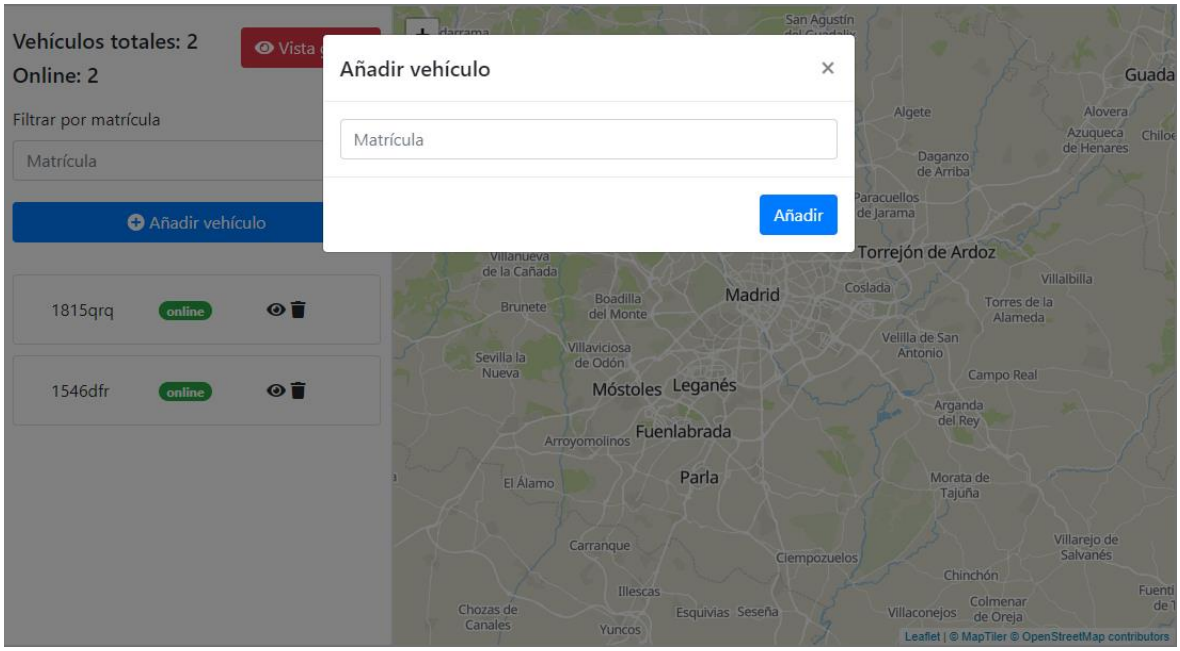

Figura 3-7: Añadir un nuevo vehículo a la flota

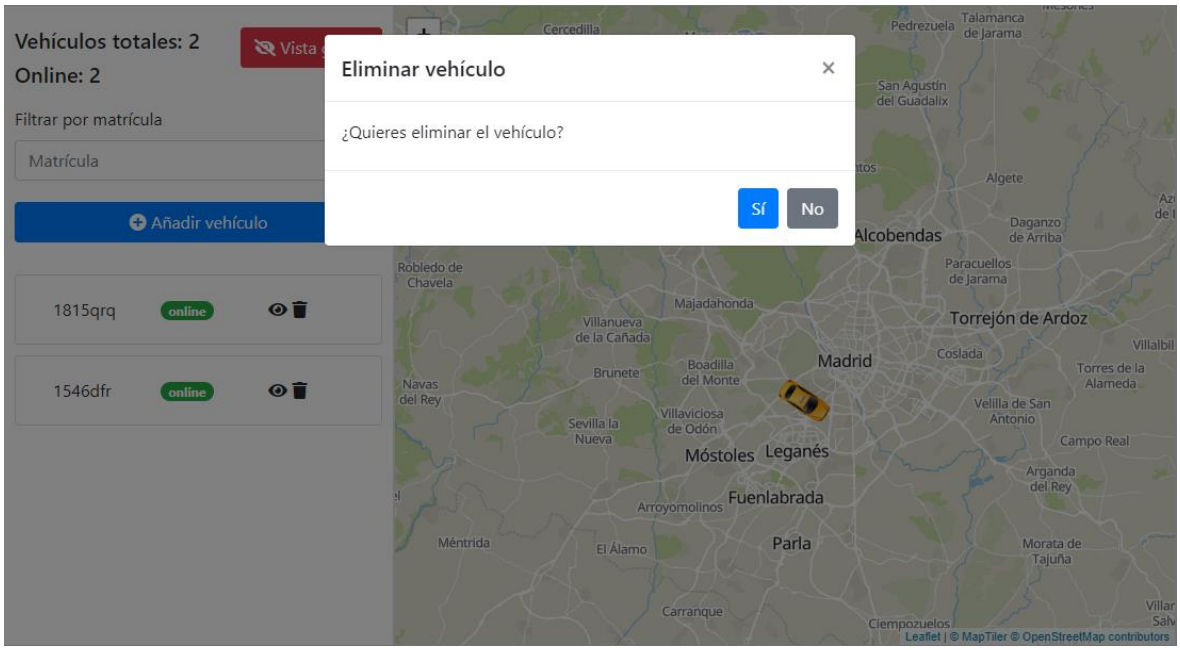

Figura 3-8: Eliminar un vehículo existente de la flota

3.2.3 Seguimiento a tiempo real

Al pulsar sobre el icono del ojo en cualquiera de los vehículos listados en la vista home, accedemos a la pantalla del seguimiento a tiempo real donde podemos ver su ubicación exacta en el mapa.

Además, en la parte superior encontramos 2 labels que nos muestran la matrícula del vehículo, además de la velocidad real del mismo.

El sistema es capaz de obtener la velocidad límite de cada una de las vías públicas por donde se conduzca, por lo que es posible detectar infracciones de límite de velocidad y hacer las medias de velocidad de un itinerario específico.

Si el vehículo supera la velocidad límite de la vía, se marcará en un tono rojo para que el gestor de tráfico se percate de la situación. Del mismo modo, siempre que el vehículo se mantenga dentro del límite de la velocidad máxima permitida, saldrá en una tonalidad verde.

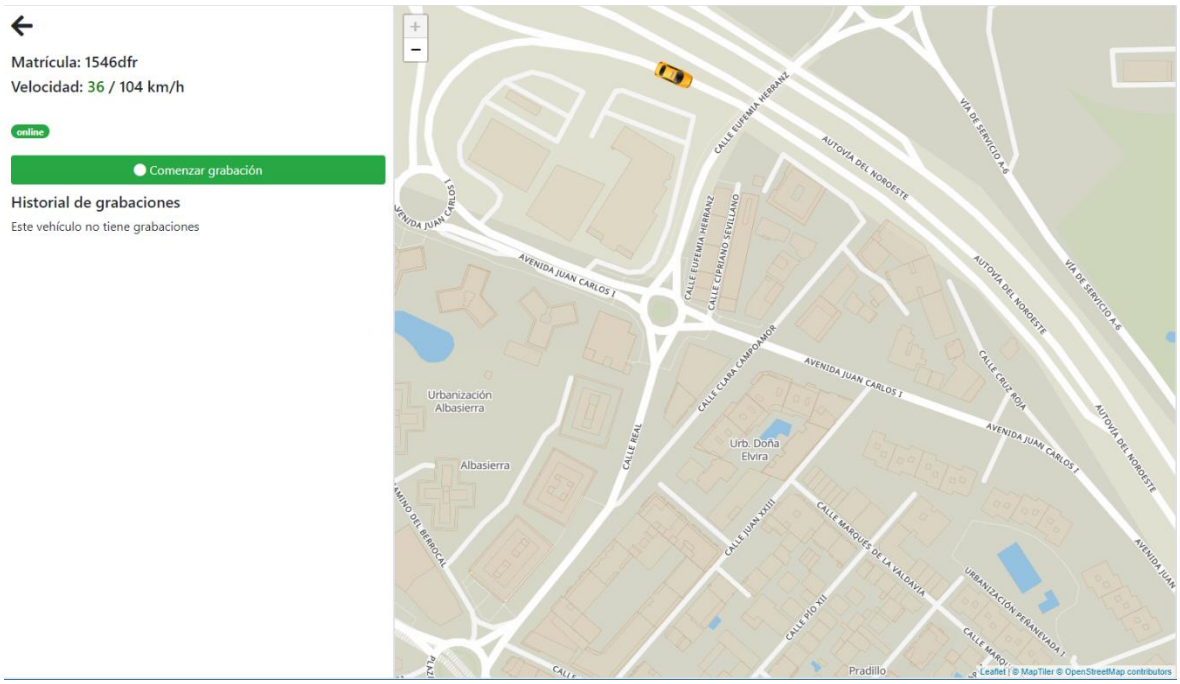

Figura 3-9: Seguimiento a tiempo real dónde el vehículo supera la velocidad máxima permitida

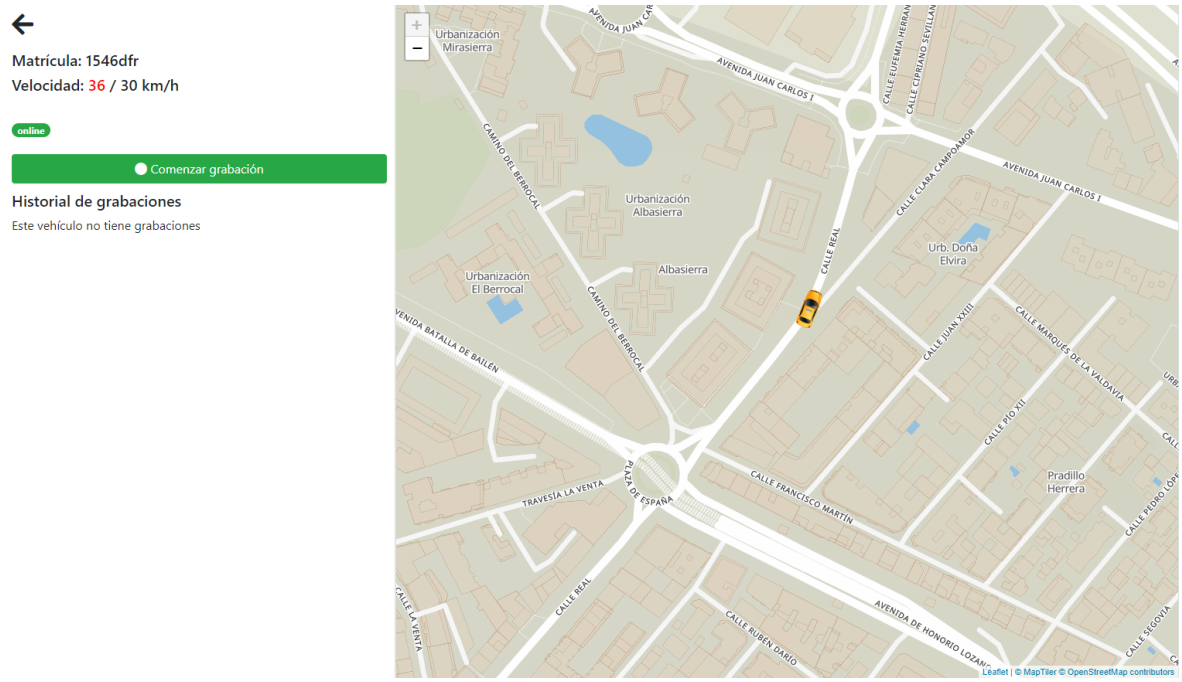

Figura 3-10: Seguimiento de un vehículo específico en tiempo real

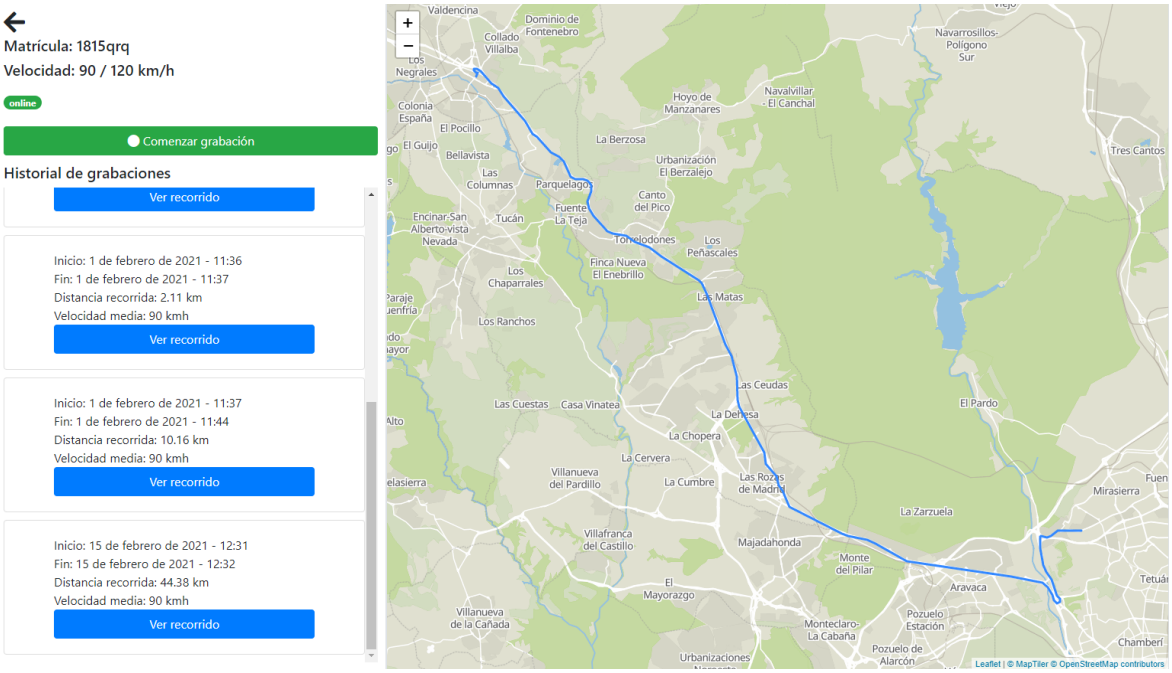

Figura 3-11: Historial de recorridos de un vehículo específico

3.2.4 Estadísticas e historial de recorridos

Sobre la misma vista, disponemos de un botón llamado “Comenzar Grabación”. Si pulsamos sobre este botón, el sistema almacenará la ubicación del transportista en cada instante con una precisión determinada por la frecuencia de emisión de coordenadas de la App.

Las grabaciones son persistentes a pérdidas de conexión. De tal forma que, si en cualquier momento se cierra la página u ocurre cualquier problema con la red, en el momento que volvamos a acceder a la grabación, veremos que continúa grabando el recorrido.

Cuando se desee finalizar la grabación de un vehículo, se pulsa sobre el botón de “Detener grabación”, que detendrá la grabación y la guardará en un historial que se puede consultar cuando se desee.

En el historial de grabaciones, podemos visualizar todos los recorridos realizados por el vehículo y datos como la fecha de inicio y fin de la grabación, la distancia recorrida en km., y la velocidad media con la que se realizó el trayecto. Además de la ruta representada en el mapa mediante una Polyline.

4 Desarrollo

Una vez introducidas y explicadas las diferentes partes del sistema, vamos a entrar en profundidad en cada una de las partes que lo componen y cómo se comunican entre sí para ofrecer una solución integral para el control de flotas.

4.1 Stack de tecnología

4.1.1 React Native [4]

Es un framework para desarrollar aplicaciones con target en iOS y Android. El código es puramente Javascript. Tiene una extensión del lenguaje llamada JSX que sirve para diseñar las interfaces de usuario pudiendo mezclar la lógica de renderizado con las vistas que componen el diseño de la App.

Su lema es “Learn once, write anywhere”, por lo que permite hacer target a ambas plataformas con el mismo código base.

El código Javascript de las Apps, se comprime en un bundle que se comunica con el código o componentes nativas del SO en el que se ejecuta. De esta forma, cuando se detecta un evento, el código nativo propaga dicho evento al bundle donde se puede definir la acción a realizar.

4.1.2 Socket.IO [5]

Esta ha sido sin duda, la herramienta clave para poder hacer realidad la funcionalidad clave de la aplicación: Poder transmitir a un rate elevado la información necesaria para poder renderizar un posicionamiento de los vehículos a tiempo real, sin la sensación de que existan pérdidas de conexión o saltos de ubicación inesperados.

La idea de funcionamiento de esta biblioteca es muy parecida a los WebSockets. Socket.io implementa un canal de comunicación bidireccional entre cliente y servidor, de forma que ambos pueden enviar y recibir mensajes simultáneamente. Cada mensaje enviado va vinculado con un evento, de tal forma que cada vez que se recibe un mensaje con un evento determinado, se puede hacer una acción específica para dicho evento.

Esto es muy útil ya que, en este caso, para poder ofrecer la mejor experiencia posible al usuario necesitamos una comunicación fluida entre ambas partes que no se podría conseguir con un esquema clásico de peticiones http. Ya que en este caso la comunicación sería basada en peticiones Cliente->Servidor->Cliente y estaríamos ante un rendimiento mucho menor debido a que el delay en este caso sería mucho mayor. Por otra parte, se produciría una sobrecarga en el servidor por la excesiva cantidad de peticiones realizadas en el caso de que tener múltiples usuarios concurrentes.

4.1.3 NodeJS [6]

NodeJS es un entorno de ejecución para Javascript. Es el entorno sobre el que se ha construido la lógica BackEnd del sistema y con el que se controlan las diferentes bibliotecas que lo conforman para la gestión de la información.

4.1.4 Mongoose [7]

Mongoose es una biblioteca Javascript para el modelado de datos en una base de datos no relacional. En este caso MongoDB. Mongoose asocia un documento MongoDB a través de la definición del esquema del modelo. De esta forma, se puede acceder a la información del documento utilizando consultas sobre el esquema definido sobre los datos del documento.

Es la biblioteca que se ha utilizado para mantener la consistencia de los datos del sistema y poder realizar consultas sobre los mismos.

4.2 Diagrama de comunicación

En esta sección veremos cómo se comunican entre sí los diferentes módulos del sistema FleetTracker.

4.2.1 Comunicación App-Servidor

Dentro de este grupo se incluyen todos los eventos para poder realizar las acciones elementales dentro del sistema. Caben destacar las siguientes:

- **Log-in:** Necesario para autenticar un vehículo del cual realizar el seguimiento a tiempo real.

- **Log-out:** Cuando el transportista cierra la app de trackeo, la app captura dicho evento para notificar al sistema que ha dejado de prestar servicio que pone el estado del vehículo offline.

- **Update de coordenadas:** En las grabaciones de recorridos, es necesario actualizar constantemente las coordenadas del vehículo para poder tener constancia de la posición del mismo durante el período de tiempo que dura la grabación. Como hemos comentado antes, si esta solución se aborda mediante un esquema de peticiones http, puede llegar a ser muy ineficiente ya que, si se quiere actualizar constantemente la posición, puede conllevar una sobrecarga en el sistema si no se gestiona correctamente.

Una primera aproximación para abordar este problema sería propagar las coordenadas mediante Socket.IO para seguidamente realizar updates sucesivos en el esquema que almacena el historial de posicionamiento de los vehículos. El problema de esta solución es que no estamos sobrecargando la red, pero sí el servidor de bases de datos, ya

que se estarían realizando updates con mucha frecuencia, creando de esta forma una saturación.

La solución por la que se ha optado es crear chunks de coordenadas que se van almacenando en un buffer local de la App. Estos chunks se van propagando al sistema en un intervalo de tiempo determinado. De esta forma, se puede comunicar toda la información a un rate de peticiones mucho menor, mejorando de esta forma la saturación del sistema.

4.2.2 Comunicación WebApp-Servidor

Al igual que ocurre con la App, en esta sección se incluyen las acciones principales del sistema, todas ellas bajo el esquema de peticiones http.

- **Listar vehículos**: Obtiene todos los vehículos creados que se renderizan en la web.
- **Obtener grabación específica:** Obtiene la lista de grabaciones con sus respectivas estadísticas de un vehículo determinado.
- **Obtener número de vehículos:** Devuelve un entero con la cantidad de vehículos registrados en el sistema.
- **Obtener vehículos en línea:** Obtiene la cantidad de vehículos que están en línea en ese momento.
- **Obtener Flag de grabación:** Sirve para saber si la grabación de un vehículo específico está o no en curso. Esto es así ya que interesa saber en todo momento el estado de la grabación, aunque se cierre la ventana de la web.
- **Eliminar vehículo:** Elimina un vehículo de la base de datos.
- **Añadir vehículo**: Añade un nuevo vehículo al sistema.
- **Guardar grabación:** Crea y guarda una Polyline a partir de los chunks de coordenadas temporales.
- **Habilitar grabación**: Habilita el flag de grabar para poder realizar grabaciones.

4.2.3 Comunicación de la posición en diferido con Socket.IO

Esta es la parte del conjunto de canales de comunicación que permite la transmisión de coordenadas a tiempo real desde la App de Trackeo hasta la WebApp.

Como se comentó en apartados anteriores, el flujo continuo de coordenadas se realiza mediante la biblioteca Socket.IO que crea una canal Full-Dúplex entre los 2 terminales que se quieren comunicar entre sí (Estos son la App de trackeo y la WebApp).

Antes de explicar con más detalle los flujos de comunicación, es necesario introducir algunos conceptos importantes de Socket.IO.

Servidor: El servidor de Socket.IO se encarga de gestionar los eventos disparados por los clientes que intervienen en la comunicación, además de emitir eventos o mensajes a otros clientes o salas. Por otra parte, dispone de atributos para controlar la cantidad de usuarios conectados, logs etc.

Sala: Una Sala es un elemento en el que los sockets pueden unirse y salir. La idea principal detrás de las salas es crear un canal en el que se puedan transmitir datos en diferido a cualquier cliente que se una a dicha sala. Esta es la idea base de un streaming en diferido. (Un servidor emitiendo a varios clientes).

En nuestro caso, existe una sala por cada vehículo del que se quiera realizar seguimiento. De esta forma, múltiples instancias del navegador que estén ejecutando la WebApp podrán acceder simultáneamente al posicionamiento de este o varios vehículos.

Una vez vistos los conceptos base, podemos introducir las diferentes rutinas que componen el canal de comunicación del streaming de posicionamiento.

En primer lugar, cuando la App comienza la transmisión de coordenadas, emite un evento llamado “join-room”. Este evento es recibido por el servidor de Socket.IO que crea dicha sala (si no estuviese ya creada), y suscribe al cliente en dicha sala.

En este momento ya tenemos una sala en la que se puede transmitir y recibir información.

A continuación, estamos listos para emitir coordenadas a dicha sala. La App tiene configurado un intervalo para transmitir las coordenadas cada 250 ms (siempre y cuando estemos en el modo debug), tiempo más que suficiente para mantener una transmisión adecuada. En cambio, en el modo real, el intervalo óptimo es de 1 segundo, pudiendo ser mayor dependiendo de la información devuelta en cada momento por el GPS.

Nota: Si en la web se renderizase directamente la posición recibida por la App, en condiciones ideales, sin tener en cuenta el delay de la transmisión de coordenadas, estaríamos ante 1frame/1000ms * 1000ms/segundo = 1FPS.

Esta frecuencia de actualización no es suficiente para mantener una buena experiencia, ya que, se percibirían saltos del vehículo cada segundo. Para ello, la Web contiene un algoritmo que consigue crear segmentos de coordenadas que son renderizados a

una tasa de FPS, tal que el usuario la perciba como un movimiento continuo. Se explicará con más detalle en próximos apartados.

Además de la transmisión de la posición, se transmite la velocidad en m/s para tener un control de velocidad del vehículo. Para ello, por cada llamada a la rutina de actualización de coordenadas, se calcula la velocidad realizando el cociente de la distancia recorrida en el último segmento de coordenadas y el tiempo empleado.

Para obtener la distancia entre 2 pares de coordenadas latitud, longitud, se puede usar la fórmula del semiverseno, explicada posteriormente.

Para calcular los deltas t, se halla la diferencia entre el UNIX timestamp del primer par y el par actual.

Con el evento 'update-coords', se emiten en el mismo objeto ambos datos, par de coordenadas actual y velocidad empleada entre dicho par y el anterior.

Por otra parte, tenemos el conjunto de eventos entre el servidor de Socket.IO y la WebApp. En primer lugar, de forma similar a la App, tenemos el evento "join-room-web", que une el cliente web a una sala creada (o no), previamente por alguna instancia de ejecución de la App.

Cuando el servidor recibe el evento 'update-coords' de un vehículo específico, éste se encarga de redirigir el objeto contenido en este mensaje a la sala de dicho vehículo mediante el evento 'update_coords_web'. De esta forma, todos los clientes suscritos a dicho canal obtendrán el par de coordenadas actual y la velocidad puntual en dicho instante.

Por último, destacamos el evento "leave-room", en el que se notifica al server que el cliente quiere dejar de recibir coordenadas.

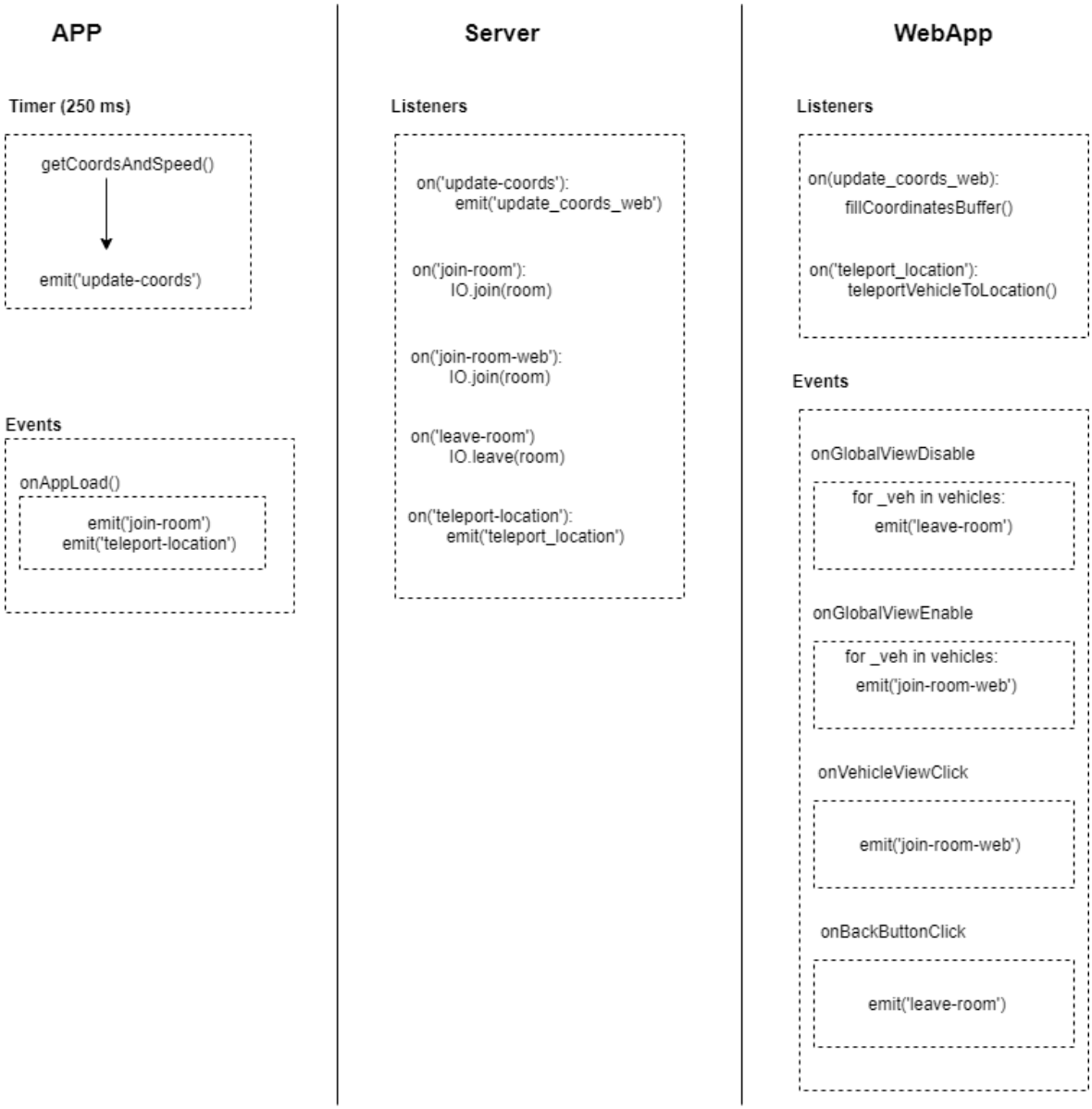

Figura 4-1: Diagrama de streaming de coordenadas por Socket.IO

La figura 4.1 muestra los listeners y los eventos para conseguir la funcionalidad referente a la propagación de coordenadas desde la App hasta la WebApp.

Los listeners son funciones que se ejecutan cuando se recibe un mensaje con un ID específico en Socket.IO.

Los eventos son funciones que se disparan cuando sucede una acción específica tanto en la WebApp como en la App. Cada evento va asociado a un mensaje de Socket.IO que es recibido en el server y en función del ID de dicho mensaje, se realiza una acción específica.

4.3 Base de datos

Con toda la lógica de la aplicación descrita anteriormente, aporta indicios de la necesidad de un motor de almacenamiento para poder consultar y gestionar los datos que fluyen entre las diferentes partes del sistema de información.

4.3.1 Estudiando las opciones

A la hora de elegir un motor de bases de datos, se vienen a la mente multitud de opciones válidas cada una con sus pros y contras. Para esta aplicación específica, hemos optado por el uso de una base de datos No Relacional y específicamente el motor MongoDB [8].

Para justificar el criterio de elección de un motor de base de datos, debemos conocer previamente las características del sistema FleetTracker y elegir la opción que creamos más conveniente para este tipo de sistema.

1. El conjunto de entidades del sistema no es muy grande. Con lo explicado anteriormente, se podría simplificar toda la lógica con 3 entidades; usuarios, grabaciones y vehículos.

2. El sistema no está pensado para almacenar grandes cantidades de datos, ya que se estima que a lo sumo podría haber 1000 usuarios registrados para realizar el control de las flotas.

Teniendo en cuenta estas 2 características, podemos hacer una tabla comparativa entre los 2 tipos de bases de datos comerciales más conocidas (SQL y NoSQL).

NoSQL: Para cada entidad se puede crear un esquema que contenga sus atributos e instanciar en modelos para realizar operaciones de búsqueda o modificación sobre ellos. Crear un esquema supone exclusivamente definir en un fichero la estructura e instanciarlo para hacer búsquedas.

SQL: En primer lugar, debemos diseñar el modelo relacional y pensar como las entidades están relacionadas entre sí. En este caso no es gran problema ya que se puede modelar en 3 entidades. Pero no queda tan fácil ya que por ejemplo dentro del esquema Grabación pensado para almacenar las grabaciones, se necesitan 2 arrays para almacenar coordenadas y el propio historial. Esto en MongoDB se hace fácil ya que permite almacenar la información directamente en objetos JSON muy cómodos de trabajar. En cambio, si queremos almacenar este tipo de arrays de coordenadas, tendríamos que serializarlos en algún atributo de la entidad grabación, contando además, con la deserialización que se debería hacer cada vez que necesitemos acceder a dicha información.

La eficiencia en las búsquedas de bases de datos no relacionales suele empeorar con el aumento de los datos almacenados. En este tipo de escenario, suelen tener mayor rendimiento las bases de datos relacionales gracias a las búsquedas sobre los índices que optimizan el proceso. No obstante, como se ha comentado previamente, ya que estamos en

un tipo de aplicación que no va a almacenar grandes cantidades de datos, el rendimiento no se va a ver afectado por el uso de una base de datos no relacional.

Ante esta comparativa, tiene más sentido usar una base de datos no relacional, y dentro de este grande grupo se ha elegido MongoDB por ser la más conocida y usada en el mercado.

4.3.2 Modelos de la base de datos

Para ayudar a modelar todos los datos en MongoDB usando como lenguaje Javascript, hemos usado una biblioteca llamada Mongoose cuya idea principal es definir objetos con un esquema tipado que se asigna a documentos MongoDB para poder trabajar sobre los mismos con formato JSON.

Los diferentes esquemas que se han utilizado son los siguientes:

Admin:

En este modelo se encuentran los atributos de los usuarios de la página de gestión de flotas

_id (Schema.Types.ObjectId): ID del usuario que gestiona las flotas

email: Email del usuario

password_hash (String): Hash de la contraseña para el acceso a la plataforma

Recording:

Aquí residen los atributos para almacenar los datos del historial de grabaciones de cada vehículo

_id (Schema.Types.ObjectId): ID de la grabación

vehicleID (String): ID del vehículo al que pertenece la grabación

tmpCoordsArr (Array): Array de coordenadas latitud, longitud que se van almacenando temporalmente en una grabación.

recordings (Array): Array que contiene el historial de grabaciones.

recordings.polyline: Polyline que describe el recorrido de una grabación.

recordings.date_start: Fecha y hora del inicio de la grabación

recordings.date_end: Fecha y hora del final de la grabación

total_distance: Distancia total recorrida en kilómetros.

Vehicle:

Este esquema contiene la definición de los datos de los vehículos presentes en la flota de transporte

_id (Schema.Types.ObjectId): ID del vehículo

matricula (String): Matrícula del vehículo

marca (String): Marca del vehículo

modelo (String): Modelo del vehículo

estado (String): Estado del vehículo (Offline u Online)

4.4 Maps APIs

Para poder visualizar el seguimiento de los diferentes vehículos en la plataforma es necesario un mapa donde poder ubicarlos en todo momento. Existen varias alternativas en el mercado, pero las más conocidas son Google Maps [9], OpenStreetMap [10] y OSRM (Open Source Routing Machine) [11] como software de enrutamiento para itinerarios.

Para elegir una solución, hay que tener en cuenta las necesidades del proyecto y elegir la que más interese.

Para el proyecto se necesita en primer lugar la capacidad de poder situar marcadores en un mapa y poder realizar acciones sobre ellos. Estas acciones son principalmente desplazarlos por el mapa, rotarlos en la dirección correcta, poner Callouts para mostrar la información del vehículo y poder jugar con el Zoom del mapa.

4.4.1 OSRM vs Google Maps

Probablemente, la opción más conocida de mapas es Google Maps. Esta solución ofrece un abanico de APIs para todas las necesidades relacionadas con mapas, como por ejemplo, mapas dinámicos, optimización de rutas, autocompletar direcciones, información de las carreteras, negocios etc.

El principal “inconveniente” de Google Maps, es que las APIs son de pago. No obstante, si el sistema con el que se trabaja no realiza una gran cantidad de peticiones, puede salir un presupuesto adecuado.

El problema viene cuando se tiene un gran volumen de peticiones a las APIs y éstas no se están optimizando cómo se debería. (Ej.: realizar más peticiones de las que se deben, no proteger correctamente las API Keys, que exista algún leak y alguien externo las esté usando de forma fraudulenta incrementando el coste considerablemente, etc.).

Por otra parte, tenemos OSRM, que es un proyecto colaborativo Open Source en el que se tiene acceso a APIs similares a las de Google. Las necesidades del proyecto mencionadas anteriormente pueden ser satisfechas perfectamente con el software de OSRM, por lo que hemos decidido trabajar con dicho software.

Ahora bien, dentro del ecosistema de OSRM, no existe una biblioteca para renderizar mapas como tal. Por lo que hemos usado la biblioteca más usada Open Source llamada Leaflet [12].

Con Leaflet, podemos renderizar mapas en el DOM de la página y combinarlo con OpenStreetMaps y Maptiler [13] para poder crear tiles o (diseños) personalizados que se adapten a las necesidades de cada proyecto.

Además, Leaflet ofrece multitud de plugins o extensiones que se pueden agregar para añadir funcionalidades a los mapas. Las 2 extensiones principales que se han añadido han sido *AnimatedMarker* [14] para desplazar los marcadores y *rotatedMarker* [15] para poder rotarlos hacia la dirección correcta.

Lamentablemente, estos plugins no tenían toda la funcionalidad requerida para poder implementar el proyecto, por lo que se tuvieron que hacer unas modificaciones sobre las extensiones que explicaremos en secciones posteriores.

4.4.2 OSRM para el control de velocidad en vías públicas

Cuando se gestionan flotas de transporte, el control de velocidad de los vehículos es una variable que interesa tenerla controlada para comprobar si hay algunos conductores que están infringiendo las normas de circulación y cerciorarse de ello.

Para poder obtener la velocidad máxima de las vías, nos podemos ayudar de la API de driving. [16]

http://router.project-osrm.org/route/v1/driving/{longitud1},${latitud1};${longitud2},${latitud2}?overview=full&annotations=speed

En la petición incluimos los últimos 2 pares de coordenadas latitud, longitud e incluimos como parámetro el flag speed, para que nos devuelva la velocidad máxima en dicho segmento.

Seguidamente, la velocidad devuelta se convierte a kmh, se compara con la velocidad del vehículo en el último segmento y en función de si sobrepasa o no, se pinta de un color determinado.

Nota: Como se puede observar, el dominio que aloja el servicio de enrutamiento de OSRM es público y accesible gratuitamente para todo el mundo. Si se quiere tener un servidor dedicado exclusivamente para uso personal, se puede descargar una Imagen Docker (disponible también públicamente) y montar dicha imagen en un servidor privado con los países que se quiera mapear.

4.4.3 Trabajando con coordenadas. El formato Polyline

Una Polyline es una forma de codificar un array de coordenadas latitud, longitud en un string con un formato más compacto. Gracias a este formato, se pueden guardar gran cantidad de coordenadas sin ocupar mucho espacio. Lo cual es interesante cuando se quieren almacenar grandes trayectos.

La unión de cada par de coordenadas adyacentes del array conforma un segmento y la unión de todos los segmentos es el recorrido.

Existen 2 algoritmos para la codificación y decodificación de las coordenadas. [17]

En este proyecto, las Polylines se utilizan para representar el historial de grabaciones de los vehículos de la flota.

4.5 Geometría

Para poder calcular las distancias entre diferentes pares de coordenadas y la correcta orientación del vehículo en el mapa, se han utilizado 2 fórmulas que se han transcrito al lenguaje Javascript con el fin de que se puedan aprovechar en el entorno del sistema.

4.5.1 Fórmula del semiverseno para el cálculo de distancias

Para poder realizar el cálculo de distancias entre los diferentes segmentos de coordenadas, hemos usado la fórmula del semiverseno o haversine fórmula [18].

Con la distancia haversine, podemos obtener la distancia que existe entre 2 pares de coordenadas para calcular las velocidades en cada tramo, y con ello poder realizar el cálculo de velocidad en dichos tramos.

Para entender la fórmula, vamos a desglosar los términos.

En primer lugar, hay que introducir el término de ángulo central. Este es el ángulo que forman 2 puntos en la periferia de una circunferencia con respecto al centro de dicha circunferencia.

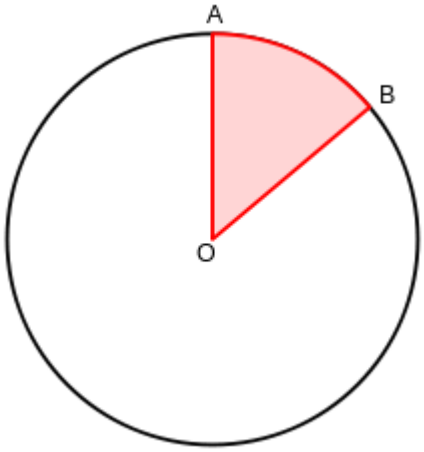

Figura 4-2: Ángulo central entre A y B

La fórmula del semiverseno se define como:

$$hav(\theta) = hav(\alpha_2 - \alpha_1) + \cos(\alpha_1)\cos(\alpha_2)\, hav(\lambda_1 - \lambda_2)$$

Donde θ es el ángulo central entre los 2 puntos sobre los que se quiere calcular la distancia.

Por definición, $\theta = d/r$

Donde d es la distancia esférica entre ambos pares de coordenadas (Parámetro que queremos obtener)

r es el radio de la esfera. En este caso equivale al radio de la tierra que es aproximadamente 6371 km.

Análogamente:
$\alpha_1 \, y \, \alpha_2$ son las latitudes de ambos puntos en radianes
$\lambda_1 \, y \, \lambda_2$ son las longitudes de ambos puntos en radianes

Por otra parte, hav(θ) se define como $sin^2(\theta/r)$. Realizamos la sustitución t = hav(θ)

Despejando, obtenemos la expresión: $sin^2(\theta/r) = t; sin(\theta/r) = \sqrt{t}$

$$sin(d/2r) = \sqrt{t}$$

Finalmente, despejando el parámetro d obtenemos: $d = 2r \, arcsin(\sqrt{t})$

Pero $t = hav(\theta) = hav(\alpha_2 - \alpha_1) + co\,s(\alpha_1)\,co\,s(\alpha_2)\,hav(\lambda_1 - \lambda_2)$

Por lo que la expresión final quedaría como:

$$d = 2rarcsin\left(\sqrt{hav(\alpha_2 - \alpha_1) + co\,s(\alpha_1)\,co\,s(\alpha_2)\,hav(\lambda_1 - \lambda_2)}\right)$$

$$= 2rarcsin\left(\sqrt{sin^2\left(\frac{\alpha_2 - \alpha_1}{2}\right) + cos(\alpha_1)cos(\alpha_2)sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

4.5.2 Fórmula de la orientación (Bearing fórmula)

Otra característica interesante del proyecto es la correcta orientación del vehículo mientras está transitando en las carreteras.

La fórmula de la orientación [19] obtiene el ángulo definido por 2 pares de coordenadas en una esfera. Este ángulo se aplica al vehículo para que tenga la correcta orientación dentro del mapa de Leaflet.

Para calcular el bearing, necesitamos el uso de 2 fórmulas:

$$S = cos\phi_B \, sin \, \Delta L$$

Donde ΔL es la diferencia de las longitudes (fin - inicio)
Y ϕ_B es la latitud del punto final

Por otra parte, tenemos:

$$C = cos\ \phi_A\ sin\ \phi_B - sin\ \phi_A\ cos\ \phi_B\ cos\ \Delta L$$

Siguiendo la misma nomenclatura que la anterior, la orientación en radianes se obtiene calculando la arcotangente de 2 parámetros entre S y C:

$$\beta = atan2(S, C)$$

El ángulo obtenido hay que convertirlo en grados ya que la biblioteca de Leaflet trabaja con esta unidad.

Leaflet trabaja con valores entre 0 y 360º, y ya que la arcotangente puede devolver valores entre -1 y 1, la conversión en grados puede tener valores negativos, por lo que habría que obtener el ángulo en positivo. Para esto se suma una revolución y se haya el módulo 360 para obtener el ángulo comprendido entre los límites sobre los que trabaja Leaflet.

$$\beta = \left(\left(\frac{\beta \times 180}{\pi}\right) + 360\right) mod\ 360$$

Y estas 2 fórmulas son fáciles de programar en Javascript, véase el Anexo F.

4.6 Adaptación de las bibliotecas de Leaflet

Las 2 operaciones principales para realizar sobre el mapa son desplazar el marcador o marcadores de los vehículos por el mapa y rotarlos en el ángulo correcto para que estén orientados en la dirección en la que se está conduciendo.

Como se comentó anteriormente, para poder realizar dichas acciones sobre Leaflet, se han instalado 2 plugins llamados AnimatedMarker y RotatedMarker.

Los plugins no tenían toda la funcionalidad necesaria para el proyecto, por lo que fueron necesarias algunas modificaciones y extensiones para poder hacerlo compatible y funcional.

4.6.1 Pausa/Reanudación del vehículo

La primera modificación necesaria es la posibilidad de detener el desplazamiento del marcador en caso de no tener suficientes coordenadas disponibles.

En el streaming de coordenadas, cada vez que la App emite un par de coordenadas nuevas, esta información se va guardando en un buffer temporal en el que la extensión de AnimatedMarker va leyendo para animar el movimiento del vehículo.

No olvidemos que la comunicación de posicionamiento es mediante Socket.IO que envía cada 250ms el posicionamiento a la página. Si existe algún corte de comunicación o las coordenadas disponibles se están renderizando a un ritmo mayor que las recibidas, llegará un punto en el que no existan más coordenadas y haya que detener el vehículo para realizar un buffering durante un tiempo determinado.

Para ello hemos añadido una función que se llama recursivamente hasta que se supere un cierto umbral que indica que el buffer está lo suficientemente lleno para reanudar el desplazamiento.

```
bufferCoords: function() {

  var len = this._latlngs.length;
  var self = this;

  if ((len - this._i) < this._bufferThreshold) {

    this._coordsChecker = setTimeout(function () {
      self.bufferCoords();
    }, 500);

  } else {
    clearTimeout(this._coordsChecker);
    this.animate();
  }
},
```

Figura 4-3: Función recursiva para la comprobación del llenado del buffer

Una vez controlada esta situación, el movimiento del vehículo se puede pausar y continuar según el estado de los datos. Como si de un vídeo en streaming se tratase.

4.6.2 Velocidad de desplazamiento

El plugin permite mover el vehículo a una velocidad constante que se fija en función de un valor predefinido al inicializar el plugin.

El inconveniente de esto es que cada tramo de las carreteras tiene una velocidad distinta, por lo que la velocidad del vehículo va a fluctuar con el tiempo y con ello la velocidad de desplazamiento en el Mapa de Leaflet.

Por ello, para poder ofrecer una velocidad de desplazamiento variable, es necesario desarrollar otra modificación.

En el plugin inicial, la velocidad viene definida por:

```
        this._latlngs[this._i-
1].distanceTo(this._latlngs[this._i])/moveSpeed*this.options.interval
```

Donde podemos ver que se divide la distancia del último segmento de coordenadas entre distance * interval.

Por defecto distance = 15 e interval = 1000ms. Por lo que la velocidad de desplazamiento es constante e igual a d/ 15 (m/s).

Por lo que todos los segmentos se recorrerán en el mismo intervalo (15 segundos).

Esto provocaría que, en tramos pequeños, el coche se moviese a una velocidad muy pequeña, y en tramos largos, se moviera a una velocidad mucho mayor, y en ninguno de los casos esta velocidad de desplazamiento se corresponde con la realidad.

Por esta razón, es necesario introducir una variable que fluctúa en función de la velocidad real del vehículo y adapte correctamente la velocidad de desplazamiento del marcador.

Para ello, simplemente multiplicamos en el denominador por una variable llamada moveSpeed y que cambiará según la velocidad recibida del último segmento, quedando la fórmula de la siguiente forma:

```
1.  speed = moveSpeed ? this._latlngs[this._i - 1].distanceTo(this._latlngs[this._i]) /
2.  moveSpeed * this.options.interval :
3.          this._latlngs[this._i - 1].distanceTo(this._latlngs[this._i]) /
4.  this.options.distance * this.options.interval;
5.
```

4.6.3 Orientación del marcador

Una vez definida la velocidad de desplazamiento del último segmento, queda definir la orientación en dicho segmento. Para definir este ángulo, nos podemos ayudar de la fórmula de la orientación, definida y explicada anteriormente.

Tomando como argumentos los 2 pares de coordenadas del segmento la fórmula nos devuelve el ángulo de orientación normalizado entre 0 y 360º. Ángulo que directamente podemos pasar al plugin rotatedMarker para que realice la correcta orientación del vehículo.

```
1.  var angle = bearing(this._latlngs[this._i].lat, this._latlngs[this._i].lng,
    this._latlngs[this._i + 1].lat, this._latlngs[this._i + 1].lng);
2.
3.  this.setRotationAngle(angle);
```

4.6.4 Métodos añadidos

Para poder actualizar la información inicialmente no soportada por el plugin, ha sido necesario la adición de algunos métodos para poder hacerlo posible. Estos métodos son los siguientes:

- **setBufferThreshold**: Establece el umbral del buffer de coordenadas. Cuanto mayor sea este valor, más tiempo tardará en llenarse y consecuentemente, el delay entre el posicionamiento real del vehículo y el que se muestra en el mapa será mayor. Lo ideal es fijar un valor lo suficientemente grande para que no se detenga el streaming, pero no tarde mucho en llenarse.

- **addCoords**: Añade un nuevo par de coordenadas latitud, longitud al buffer.

- **addSpeed**: Añade la velocidad referente al último segmento de coordenadas.

- **teleport**: Sitúa el marcador en unas coordenadas específicas. Este método es necesario para poder ubicar al vehículo inicialmente en el mapa, antes de comenzar cualquier desplazamiento.

- **start**: Inicializa el plugin de desplazamiento. Es necesario añadirlo ya que, de lo contrario, no se podría comenzar cuando se quisiera.

4.7 Seguridad

Dada la gran cantidad de flujos de comunicación entre las diferentes componentes del sistema. Es importante asegurar que el acceso al sistema es seguro y nadie pueda suplantar nuestra identidad, visualizar los datos con los que estamos trabajando etc.

Es por esto por lo que se han tomado medidas especiales de seguridad para hacer de FleetTracker un sistema seguro y confiable.

4.7.1 Comunicación REST cifrada

Quizás la parte más importante es cifrar los datos que viajan entre ambas aplicaciones y el servidor. Sabemos que por la red viaja información como las credenciales de acceso a la plataforma, posicionamiento de vehículos, estados etc. Es importante mantener esta información privada y que únicamente sea conocida por los agentes que participan en la comunicación.

Esto se puede solucionar añadiendo un certificado SSL en el servidor en el que se esté ejecutando la lógica del BackEnd.

Si el certificado ha sido emitido por una entidad confiable por el navegador en el que el usuario trabaja con la WebApp, la comunicación en este canal está cifrada y sería inaccesible al resto de personas.

Lo mismo ocurre con la APP, cuando se realizan peticiones a un Endpoint REST, si la biblioteca con la que se realizan las peticiones confía en la cadena de certificados, éstas se realizarán de forma segura.

4.7.2. Autenticación y acceso a la API REST

Otro aspecto vital en la seguridad es la correcta restricción de acceso a la API REST.

El esquema de comunicación es mediante peticiones http cuyas direcciones son públicamente accesibles, por lo que es necesario tomar medidas para que únicamente puedan acceder al contenido aquellos usuarios autenticados previamente en el sistema.

Para ello se ha utilizado la biblioteca JSON Web Tokens [20]. JWT define un esquema seguro para transmitir información entre 2 entidades. La información que viaja puede ser verificada y confiable porque está firmada digitalmente.

Los JWT se pueden firmar de 2 formas, mediante una clave secreta utilizando el algoritmo HMAC o mediante criptografía asimétrica utilizando un par de claves pública privada con los algoritmos RSA o ECDSA.

En nuestro caso, se ha optado por el uso de HMAC combinado con SHA256 manteniendo el secreto en el servidor.

Un token JWT está formado por 3 partes diferenciables que son las siguientes:

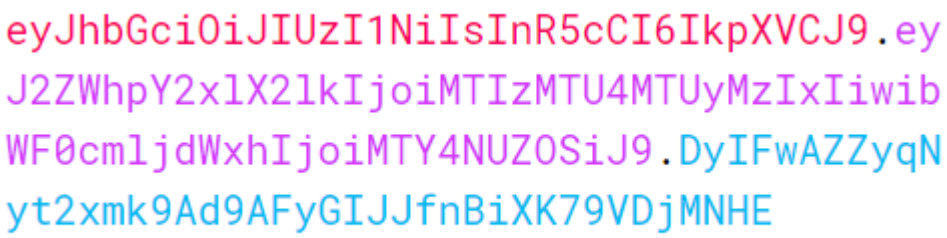

Figura 4-4: Estructura de un JSON Web Token codificado

El primer string (rojo) es el algoritmo de hashing utilizado
El segundo (morado) es el payload o mensaje a transmitir
El tercero (azul) es el hashing de la combinación de los 2 primeros y el secreto HMAC.

Los datos del JWT no están cifrados, simplemente están codificados en base64URL ya que únicamente interesa validar el acceso.

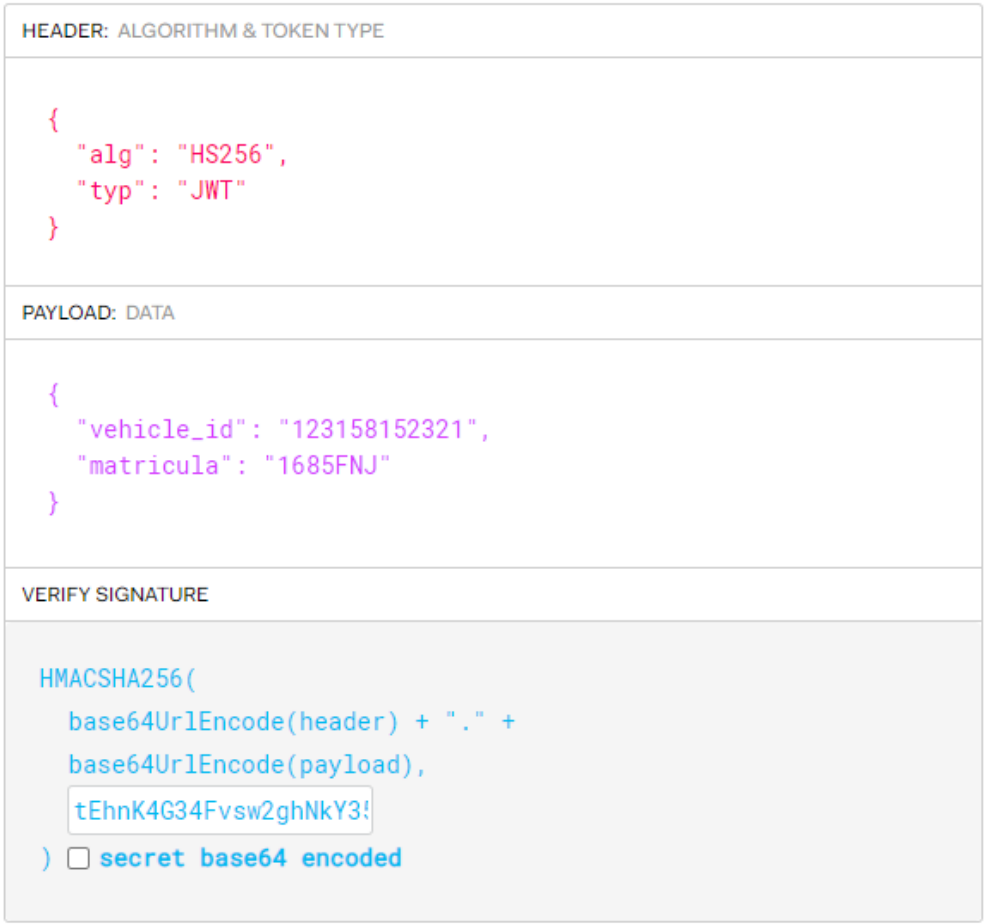

Figura 4-5: JWT decodificado para las peticiones de la App de los vehículos

¿Por qué son seguros los JWT?

Recordemos que con HMAC, se realiza un hashing del secreto concatenado con el mensaje a transmitir. Este mensaje por lo general, suele ser un objeto serializado que contiene entre otros datos el ID del usuario.

Cuando se genera un token JWT vinculado a un usuario, éste es utilizado en futuras peticiones para otorgar acceso a las rutas de la API. Si por cualquier razón se modifica un único bit del payload, al realizar la comprobación del HMAC en el servidor, éste sería totalmente diferente por lo que en este caso el usuario no estaría autorizado y se rechazaría el acceso. [21]

Este suele ser un típico caso en el que un atacante realiza suplantación de identidad modificando el atributo ID del payload para hacerse pasar por otro usuario.

De la misma forma, si un usuario intentase por ejemplo acceder a la lista de vehículos de otro usuario, aunque se disponga del token, como éste va directamente vinculado a su ID personal, cuando se realice la comprobación, los IDS serán diferentes por lo que el acceso será de igual forma denegado.

Todas las peticiones que se realizan a la API tienen esta comprobación, por lo que no es posible acceder sin autenticarse ni hacerse pasar por otro usuario.

4.7.2. Hashing de contraseñas

Otro aspecto vital a tener en cuenta en este sistema y prácticamente en cualquier sistema de información es mantener los datos personales de los usuarios de forma segura y evitar que puedan ser robados por un agente externo en caso de que se produzca un leak o fallo de seguridad.

En este caso, el único dato sensible que se debe proteger son las contraseñas de los usuarios.

Si las contraseñas se guardasen en texto plano en la base de datos, cualquier atacante que consiguiese acceso a la base de datos, podría obtenerlas sin esfuerzo.

Otra tendencia suele ser almacenar el SHA256 o similar de las contraseñas de los usuarios. Pero esta práctica tampoco es muy segura ya que los usuarios suelen usar la misma contraseña en múltiples sitios web con patrones muy comunes como “casa123” o “nombre12” etc., patrones que pueden estar presentes en tablas arco iris.

Una buena solución y por la que se ha optado en este proyecto, es utilizar funciones hash completas como bcrypt [22].

Con bcrypt, se realiza el hashing correspondiente a la combinación del texto a proteger junto con un salt.

De esta forma, al no conocer el salt, el resultado final es un hashing muy complejo que probablemente no esté presente en ninguna tabla arcoíris.

Además, se puede incrementar la seguridad utilizando un salt diferente para cada usuario. De esta forma, usuarios con la misma contraseña en el mismo sistema, tendrán hashes totalmente distintos.

5 Integración, pruebas y resultados

5.1 Integración

Debido a que el sistema de FleetTracker está compuesto por 3 componentes principales, la integración de cada una de las partes se desarrolló primero en un entorno local para poder avanzar más rápido y facilitar la detección prematura de errores depurando en local, sin necesidad de externalizar la plataforma a un servidor público.

Primero se trabajó en las funcionalidades de la App de trackeo para los vehículos. Como se comentó previamente, la App emite la ubicación actual del dispositivo para poder localizarlo desde la Web de Tracking.

Al estar sobre un entorno local, no podemos realizar pruebas de seguimiento real en el exterior ya que no nos podemos comunicar con el servidor. Además, aunque se tuviese un servidor accesible desde el exterior, cada vez que se quisiera hacer alguna prueba habría que salir con el coche y probar el funcionamiento. Metodología muy ineficiente. Para ello, se desarrolló una opción con la que se puede poner la App en modo Debug y emular un itinerario real con una frecuencia de emisión de coordenadas determinada.

Una vez desarrolladas las funcionalidades básicas para poder emitir coordenadas, se desarrolló la parte del servidor para poder comunicarse con la app y renderizar el posicionamiento en el mapa web.

5.2 Pruebas

Para asegurar el correcto funcionamiento de la plataforma ante varios escenarios y comprobar el rendimiento del sistema, se hicieron diversas pruebas orientadas a la detección de fallos, experiencia de usuario y respuesta a la sobrecarga con el seguimiento de varios vehículos simultáneos.

5.2.1 Pruebas unitarias

Se realizaron pruebas unitarias independientes en cada módulo para comprobar que el procesamiento de información con las estructuras de datos y funciones creadas no fallaban y con ello, facilitar la detección de bugs más fácilmente.

Por la parte de la App, se realizaron pruebas unitarias para asegurarse de la correcta emisión de coordenadas tanto en entorno debug como real.

Especialmente, se probó la lectura de coordenadas del GPS en un trayecto real con el coche, además de las velocidades. Las coordenadas se emitían satisfactoriamente y se guardaban en un log para poder consultar posteriormente en un visualizador la cadencia de captura de coordenadas. Por otra parte, se comprobó que el velocímetro de la App marcaba la misma cifra que el velocímetro del coche, mostrando, por tanto, la velocidad real a la que se está circulando.

Una vez realizadas las pruebas de la App, se crearon itinerarios de prueba fijos en la WebApp para simular un recorrido a una velocidad constante de 40 kmh. (Todo esto sin tener ninguna comunicación con la App).

Se simularon también pérdidas de conexión momentáneas para simular que pasaba ante una falta de coordenadas en el buffer de lectura. El resultado fue satisfactorio ya que, ante esta situación, cuando se agotaban las coordenadas, el vehículo se detenía hasta que el buffer volvía llenarse hasta un cierto umbral a partir del cual, proseguía nuevamente la marcha.

Una vez que se comprobó el funcionamiento de estas pruebas unitarias, se continuó desarrollando la parte de comunicación entre ambas partes mediante los esquemas explicados previamente.

5.2.2 Pruebas responsive

Este apartado de pruebas es bastante importante de cara a ofrecer una solución funcional en cualquier dispositivo.

Desde el inicio del desarrollo, se diseñaron las interfaces teniendo en cuenta que debían funcionar en la mayoría de los dispositivos electrónicos que tuviesen acceso a internet (Ordenadores, teléfonos, tabletas etc.).

En primer lugar, cabe destacar la App, donde se diseñaron las interfaces mediante la herramienta Flexbox [23] compatible con React Native. Esto permite un diseño en el que se puede establecer el porcentaje de espacio que debe ocupar cada vista y conseguir cualquier tipo de Layout componiendo varios contenedores. Ya que internamente se toman como referencia porcentajes, el Layout se renderizará más grande o pequeño según las dimensiones del dispositivo.

Seguidamente, tenemos la WebApp donde se ha utilizado como Framework de diseño Bootstrap [24] con CSS3 [25] para ayudar a conseguir un Layout responsive en cualquier dispositivo. Hay que destacar la importancia de Bootstrap, ya que dispone de componentes y clases predefinidas con media queries que hacen posible que las vistas sean responsive sin introducir nada de código adicional. No obstante, para enlazar todos los componentes de la página, es necesario utilizar algunas clases CSS personalizadas.

5.2.3 Pruebas de rendimiento

Una vez finalizadas las pruebas anteriores y desarrollado las pruebas unitarias y de diseño, solamente queda probar la plataforma con varios dispositivos y comprobar el correcto funcionamiento del sistema.

En primer lugar, se realizó una prueba con varios emuladores Android (1 por vehículo) y se probó que el seguimiento funcionaba correctamente con 3 dispositivos simultáneos. La prueba resultó satisfactoria y se podía realizar el seguimiento de todos los vehículos por separado e individualmente sin detención alguna en la plataforma.

Además, se abrieron múltiples instancias de la WebApp en el navegador y todas ellas podían acceder al seguimiento de forma simultánea sin un decremento de eficiencia en el seguimiento.

Por otra parte, se probó en paralelo en otros dispositivos móviles ubicados en una zona fuera de Madrid y se podía acceder también al seguimiento sin ningún problema.

Tras realizar estas pruebas, se demuestra que el sistema cumple con los requisitos de multi-acceso establecidos en el proyecto y la resistencia del sistema ante una carga de estrés.

Finalmente se realizó la prueba principal, que consiste en el seguimiento y control de velocidad a tiempo real de un vehículo desde casa.

Para ello, realicé una pequeña ruta por los alrededores de mi residencia y desde casa se grabó la pantalla de la web para verificar que el seguimiento funcionaba correctamente.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

El proyecto nació como una necesidad que tenían las empresas para poder gestionar sus vehículos, controlar adecuadamente su posicionamiento y reducir el consumo de combustible de éstos. Analizando la problemática y observando los costes y los problemas que tenían debido a estos motivos, es por lo que aproveché a desarrollar este trabajo.

Tras la realización de este TFG, se ha conseguido desarrollar un sistema de control de flotas con un seguimiento que tiene una precisión que, a día de hoy, no existe en el mercado y que ha cumplido todos los requisitos funcionales establecidos en el proyecto.

Durante el desarrollo de la plataforma, se han empleado tecnologías que no había empleado anteriormente como el uso de Sockets y la comunicación de información usando esta biblioteca entre varios dispositivos. Al ser un proyecto en el que están involucradas varias aplicaciones (WebApp, App móvil y servidor), cada una con una función específica, no ha sido una tarea tan sencilla y tuve que profundizar en la documentación y elegir la más adecuada para este tipo de proyecto en el que participan múltiples dispositivos de forma simultánea.

Además, considero que es un TFG bastante completo ya que, se ha desarrollado aplicando conocimientos aprendidos en algunas de las asignaturas principales cursadas a lo largo de la carrera.

Entre estas asignaturas se puede destacar Redes 1 y 2 para la comunicación entre los distintos componentes y la seguridad de la plataforma mediante hashing y firma digital para autenticar correctamente a los usuarios en los endpoints de la API REST, sistemas informáticos para la creación de la API REST y el diseño de la WebApp, estructuras de datos para la justificación del tipo de base de datos a utilizar y consultas sobre los modelos creados, cálculo para ayudar a desglosar y comprender las fórmulas para la orientación y cálculo de distancias, ingeniería del software como ayuda para estructurar y redactar este documento y análisis de algoritmos para usar las estructuras adecuadas para el almacenamiento y procesamiento de información.

6.2 Trabajo futuro

A pesar de ser una plataforma con funcionalidades únicas en el mercado, todavía quedan pasos para que el proyecto se convierta en una solución integral que puedan aprovechar al máximo las empresas.

En la tabla comparativa presente en la sección 2, se muestra que el TFG no dispone de un optimizador de rutas. Este optimizador es importante para que las empresas puedan introducir direcciones de recogida/entrega y en segundos se obtenga la ruta óptima para dichas direcciones. Ruta que, posteriormente sería asignada a algún vehículo de su flota para que pudiese despacharla al instante o al día siguiente.

Si combinamos las plataformas de Portefy Fleet con el trabajo desarrollado en este TFG se consigue como resultado una plataforma completa que contiene todas las funcionalidades necesarias para que los negocios puedan gestionar sus rutas, optimizarlas, despacharlas y tener el control sobre sus vehículos para saber en todo momento la ubicación y el comportamiento durante el curso de las jornadas laborales.

En cuanto a las funcionalidades propuestas para este TFG, se han cumplido con éxito todos los requisitos planteados. No obstante, se pueden incluir algunas mejoras para que la plataforma disponga de funcionalidades interesantes para los usuarios:

- **Mejora de las métricas de las grabaciones**: Actualmente, se muestra el tiempo del trayecto con las fechas de inicio y fin, kilómetros empleados y velocidad media. Se podrían incorporar más métricas como los tramos en los que se han superado las velocidades máximas, mapa de calor que muestre las zonas en las que el vehículo pasa más tiempo, y la posibilidad de generar informes conjuntos que contengan los datos de las grabaciones en un periodo de tiempo determinado.

Notificaciones: Se pueden incorporar notificaciones cuando los vehículos entren en algún punto predefinido en el mapa, avisar cuando se esté infringiendo los límites de velocidad de la vía en la que se circula actualmente, etc.

Referencias

[1] *Movolytics*
https://movolytics.es/

[2] *WebFleet*
https://www.webfleet.com/es_es/webfleet/

[3] *Portefy fleet*
https://www.portefy.com/fleet/

[4] *React Native*
https://reactnative.dev/ (consultado por última vez el)

[5] *Socket.IO Docs*
https://socket.io/docs/v4

[6] *NodeJS Docs*
https://nodejs.org/docs/latest-v10.x/api

[7] *Mongoose Docs*
https://mongoosejs.com/docs/guide.html

[8] *MongoDB*
https://www.mongodb.com/es

[9] *Google Maps APIs*
https://developers.google.com/maps?hl=es

[10] *OpenStreetMap*
http://project-osrm.org/docs/v5.24.0/api/#

[11] *Project OSRM (Open Source Routing Machine)*
http://project-osrm.org/docs/v5.24.0/api/#

[12] *Leaflet*
https://leafletjs.com/

[13] *Capas personalizadas para los mapas*
https://www.maptiler.com/

[14] *Leaflet AnimatedMarker*
https://github.com/openplans/Leaflet.AnimatedMarker

[15] *Leaflet RotatedMarker*
https://github.com/bbecquet/Leaflet.RotatedMarker

[16] *OSRM Route Service*
http://project-osrm.org/docs/v5.5.1/api/#route-service

[17] *Utilidades para las polylines*
https://gist.github.com/mhsattarian/85d829a9162e16ee79d7c7ec74ea6c37

[18] *Fórmula del semiverseno*
https://community.esri.com/t5/coordinate-reference-systems/distance-on-a-sphere-the-haversine-formula/ba-p/902128

[19] *Fórmulas de la aviación (Orientación)*
https://www.movable-type.co.uk/scripts/latlong.html

[20] *JSONWebToken*
https://jwt.io/

[21] *How JWT works*
https://flaviocopes.com/jwt/

[22] *Bcrypt para el hashing de contraseñas*
https://www.npmjs.com/package/bcrypt

[23] *Flexbox for React Native*
https://reactnative.dev/docs/flexbox

[24] *Bootstrap*
https://getbootstrap.com/docs/5.0/getting-started/introduction/

[25] *CSS Tutorial*
https://www.w3schools.com/css/

Glosario

API	Application Programming Interface
HTTP	HyperText Transfer Protocol
API REST	API Representational State Transfer
JSON	JavaScript Object Notation
IP	Internet Protocol
.apk	(Android Application package). Extensión de archive de las aplicaciones de Android
HTML	HyperText Markup Language
CSS/CSS3	Cascading Style Sheets
Tracking	Localización física de objetos o personas en un lugar determinado
WebApp	Aplicación Web
ID	Identificador
UNIX timestamp	Cantidad de ms que han pasado desde el 1 de enero de 1970 hasta una fecha específica
FPS	Fotogramas Por Segundo
Callout	Información de un marcador
Endpoint	Punto final de comunicación en un esquema API REST

Anexos

A Manual de instalación

A.1 Guía rápida de instalación

1. Descargar el .apk disponible en la siguiente URL: https://drive.google.com/file/d/149sAqEnkQskrybvRRwzXeKGQmicVeH7Y/view?usp=sharing
2. Acceder a http://ec2-34-253-152-137.eu-west-1.compute.amazonaws.com/

3. Loguearse con las siguientes credenciales:
   email: user@fleettracker.com
   password: test12

A.2 Instalación personalizada

Creación del entorno

Lo primero que se debe hacer es preparar un entorno de ejecución para lanzar el servidor Web donde reside la lógica del sistema.

Lo ideal es crear un servidor en la nube o de forma local, pero abriendo los puertos necesarios para que la aplicación móvil pueda emitir coordenadas correctamente.

Instalación de dependencias

Primero debemos instalar el entorno de ejecución NodeJS:

```
sudo apt install nodejs
```

Nota: Para un correcto funcionamiento, recomendamos el uso de Node v10.23.2 o superior

Una vez instalado, buscamos un directorio en el que clonar el proyecto disponible en GitHub. En mi caso lo instalo bajo /home

Clonamos el proyecto mediante:

```
git clone https://github.com/vssalcedo/TFG
```

Seguidamente, entramos en el directorio creado "/home/TFG" e instalamos las dependencias necesarias para el proyecto:

```
npm install
```

Una vez clonado el proyecto, debemos instalar la base de datos MongoDB. Esta instalación difiere según el tipo de sistema operativo que tengamos instalado, por lo que lo mejor es seguir las guías de la página oficial de MongoDB:

https://docs.mongodb.com/manual/tutorial/

Instalación del servidor Web

A continuación, debemos instalar el servidor Web para atender las peticiones y la comunicación entre los diferentes componentes. En mi caso he usado nginx pero se puede usar cualquier otro servidor web sin problema.

```
1.  apt-get install nginx
```

Seguidamente hay que configurar nginx para que actúe como un Reverse Proxy. Para ello modificamos el archivo de configuración normalmente ubicado en /etc/nginx/nginx.conf

Debemos añadir las siguientes líneas:

```
1.  server {
2.          listen       80;
3.          listen       [::]:80;
4.          server_name  ec2-34-253-152-137.eu-west-1.compute.amazonaws.com; # (IP del
    servidor)
5.
6.          location / {
7.               proxy_http_version 1.1;
8.               proxy_set_header   X-Forwarded-For $remote_addr;
9.               proxy_set_header Upgrade $http_upgrade;
10.              proxy_set_header Connection "upgrade";
11.              proxy_set_header   Host $http_host;
12.              proxy_pass         http://34.253.152.137:3000; # (Puerto donde se está
    ejecutando la aplicación de NodeJS)
13.         }
14.
15.     }
16.
17.
```

Una vez añadidas las líneas, se debe reiniciar nginx para actualizar los cambios mediante:

sudo service nginx restart

Finalmente, hay que configurar correctamente el firewall. Hay que asegurarse que en las reglas de entrada esté habilitado el tráfico http y crear una regla TCP personalizada para aceptar tráfico al puerto 3000 desde la IP del servidor.

HTTP	TCP	80	::/0
SSH	TCP	22	46.6.12.55/32
Regla TCP personalizada	TCP	3000	34.253.152.137/32

Figura A.1: Entradas necesarias del Firewall

Inicialización de MongoDB

Debemos crear la base de datos con la que se va a trabajar. El sistema está configurado por defecto para trabajar con una base de datos llamada tfg. Para crearla simplemente abrimos el intérprete de MongoDB ejecutando mongo en la terminal y ponemos el siguiente comando: use tfg.

Para comprobar que se ha registrado correctamente, podemos usar el comando show dbs.

Si se lista, la db ha sido configurada correctamente.

Por último, debemos añadir un usuario admin para poder gestionar la plataforma de flotas. Para ello, se debe hacer una petición POST al endpoint /auth/registerAdmin
Más información en el anexo REST API.

Puesta en marcha del sistema

Finalmente, ir a la carpeta del proyecto y acceder al directorio /server. Una vez allí, ejecutar

```
node app.js
```

Nótese que la ejecución del sistema se detendría una vez cerrada la sesión si nos conectamos por SSH o al interrumpir la ejecución. Si se quiere que el programa funcione en todo momento se debe daemonizar. Para ello se puede usar la herramienta pm2.

```
npm i -g pm2
```

```
pm2 start app.js
```

B REST API

De cara a poder realizar integraciones con otros dispositivos o poder escalar las funcionalidades actuales del proyecto, se aporta la documentación de la API REST con los endpoints actuales del sistema.

Endpoints para Login / Registro:

POST /auth/loginCourier

Realiza el login de un vehículo.

Body:

- matricula (String): Matrícula del vehículo a utilizar

Respuesta:

Si el login es correcto, se devolverá un objeto con la siguiente información:

```
{
   token, // Token JWT para realizar próximas peticiones
   Matricula,  // Matrícula del vehículo
   vehicle_id // ID del vehículo
}

```

Si el login es incorrecto, se devolverá un objeto como el siguiente:

```
{error: 'invalid_matricula'}
```

POST /auth/loginAdmin

Realiza el login de un usuario en la WebApp.

Body:

- email (String):Email del usuario
- password: Password del usuario

Respuesta:

Si el login es correcto, se devolverá un objeto con la siguiente información:

```
{
   token, // Token JWT para realizar próximas peticiones
   admin_id // ID del usuario
}

```

Si el login es incorrecto, se devolverá un objeto como el siguiente:

```
{error: 'invalid_credentials'}
```

POST /auth/registerAdmin

Realiza el registro de un nuevo usuario en la WebApp.

Body:

- email (String):Email del usuario
- password: Password del usuario

Respuesta:

Si el registro se realiza correctamente, se devolverá un objeto con la siguiente información:

```
{status: 'ok'}
```

Endpoints para gestión/listado de vehículos:

GET /vehicles/

Lista la información de todos los vehículos

Respuesta:

JSON con la información de los vehículos

GET /vehicles/get/{vehicleID}

Obtiene los detalles específicos de un vehículo.

Parámetros:

vehicleID: ID del vehículo

Respuesta:

JSON con la información completa del vehículo.

PATCH /vehicles/updateEstado/{vehicleID}/{estado}

Actualiza el estado de un vehículo (Online u Offline)

Parámetros:

vehicleID: ID del vehículo

estado (String): Estado del vehículo. Los valores pueden ser ‘online’, ‘offline’

Respuesta:

JSON con la información del vehículo actualizada

GET /vehicles/getCount

Obtiene el número de vehículos registrados

Respuesta:

JSON con el número de vehículos registrados

GET /vehicles/getOnlineCount

Obtiene el número de vehículos cuyo estado es ‘online’

Respuesta:

JSON con el número de vehículos cuyo estado es ‘online’

POST /vehicles/add

Añade un nuevo vehículo a la plataforma

Body:

- matricula (String): Matrícula del vehículo
- modelo(String): Modelo del vehículo
- marca(String): Marca del vehículo

Respuesta:

Si el registro es satisfactorio, se devuelve un JSON con la información del vehículo

POST /vehicles/delete/{vehicleID}

Elimina un vehículo de la plataforma

Parámetros:

- vehicleID: ID del vehículo

Respuesta:

Si el registro es satisfactorio, se devuelve un JSON con la lista de vehículos actualizada

POST /vehicles/delete

Elimina todos los vehículos de la plataforma

Parámetros:

- vehicleID: ID del vehículo

Respuesta:

Si la eliminación es satisfactoria, se devuelve un JSON con la lista de vehículos actualizada. (Array vacío)

Endpoints para la gestión de las grabaciones de los recorridos

POST /recordings/pushTmpCoords/{vehicleID}

Actualiza el array de coordenadas temporales de un vehículo.

Parámetros:

- vehicleID: ID del vehículo

Respuesta:

Si se actualizan las coordenadas correctamente,, se devuelve un JSON como el siguiente:

```
{ code ‘ok’ }
```

POST /recordings/removeCoords/{vehicleID}

Elimina las coordenadas temporales de un vehículo

Parámetros:

- vehicleID: ID del vehículo

Respuesta:

Si se eliminan las coordenadas correctamente,, se devuelve un JSON como el siguiente:

```
{ code ‘ok’ }
```

POST /recordings/save/{vehicleID}

Transforma el array de coordenadas temporales actual en una Polyline para poder visualizarla en el historial.

Parámetros:

- vehicleID: ID del vehículo

Respuesta:

Si se eliminan las coordenadas correctamente, se devuelve un JSON como el siguiente:

```
{ code ‘ok’ }
```

Si no hay una cantidad suficiente de coordenadas para generar el Polyline, se devuelve un JSON como el siguiente:

```
{code: 'not_enough_data'}
```

POST /recordings/enable/{vehicleID}

Habilita el flag para poder realizar grabaciones para un vehículo

Parámetros:

- vehicleID: ID del vehículo

Respuesta:

Si se habilita correctamente, se devuelve un JSON como el siguiente:

```
{ code ‘ok’ }
```

POST /recordings/disable/{vehicleID}

Deshabilita el flag para poder realizar grabaciones para un vehículo

Parámetros:

- vehicleID: ID del vehículo

Respuesta:

Si se deshabilita correctamente, se devuelve un JSON como el siguiente:

```
{ code ‘ok’ }
```

GET /recordings/isRecordingEnabled/{vehicleID}

Devuelve un boolean que indica si las grabaciones están o no habilitadas

Parámetros:

- vehicleID: ID del vehículo

Respuesta:

JSON con el booleano del estado de las grabaciones

GET /recordings/get/{vehicleID}

Lista las grabaciones de un vehículo específico

Parámetros:

- vehicleID: ID del vehículo

Respuesta:

JSON con la información de las grabaciones

POST /recordings/clear/{vehicleID}

Elimina todas las grabaciones de un vehículo específico

Parámetros:

- vehicleID: ID del vehículo

Respuesta:

Si se eliminan correctamente las grabaciones, se devuelve un JSON como el siguiente:

```
{ code ‘ok’ }
```

C Maquetación

C.1 Maquetas App móvil

Figura 0-1: Maqueta de la pantalla de inicio de sesión en la App móvil

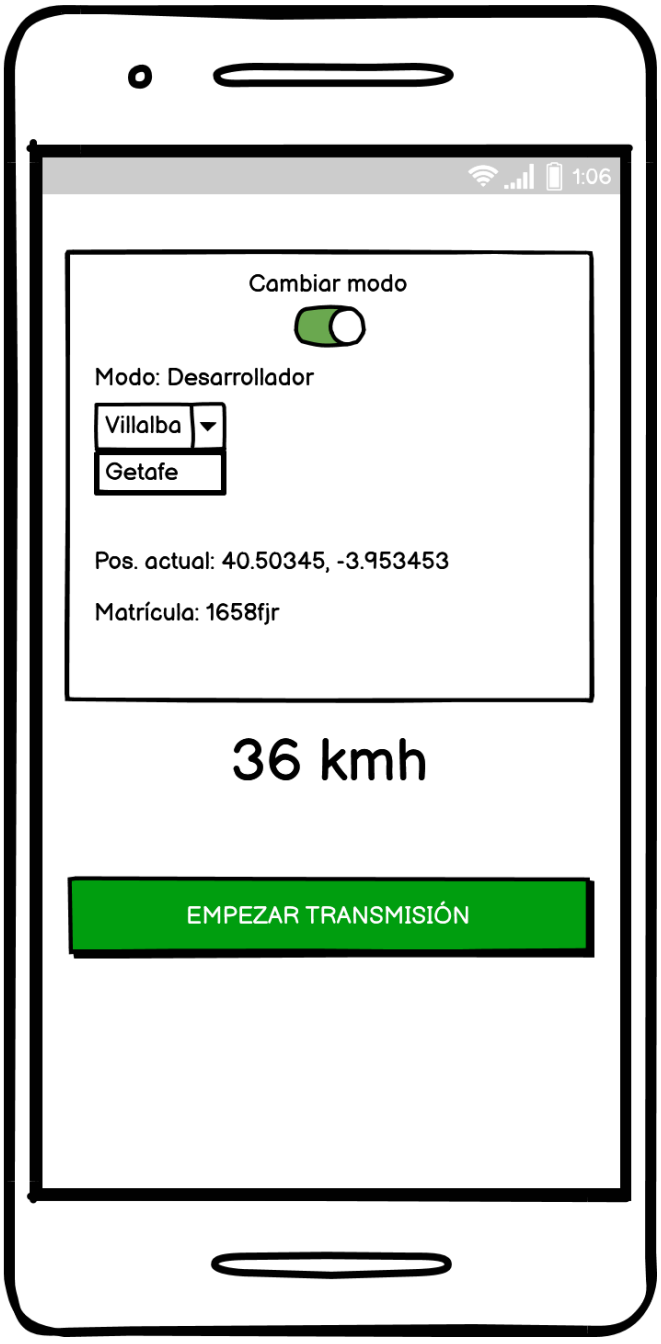

Figura 0-2: Maqueta de la vista home en modo Debug

Figura 0-3: Maqueta de la vista home en modo Live

A.2 Maquetas WebApp

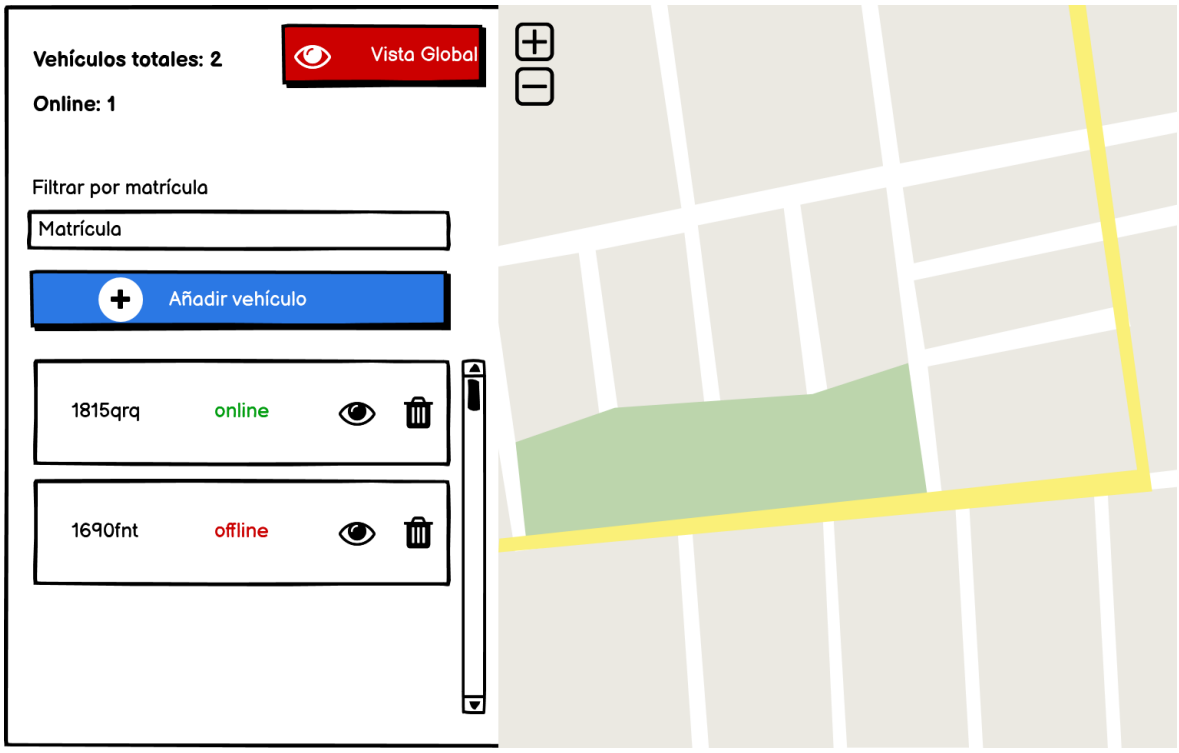

Figura 0-4: Maqueta de la vista home de la web

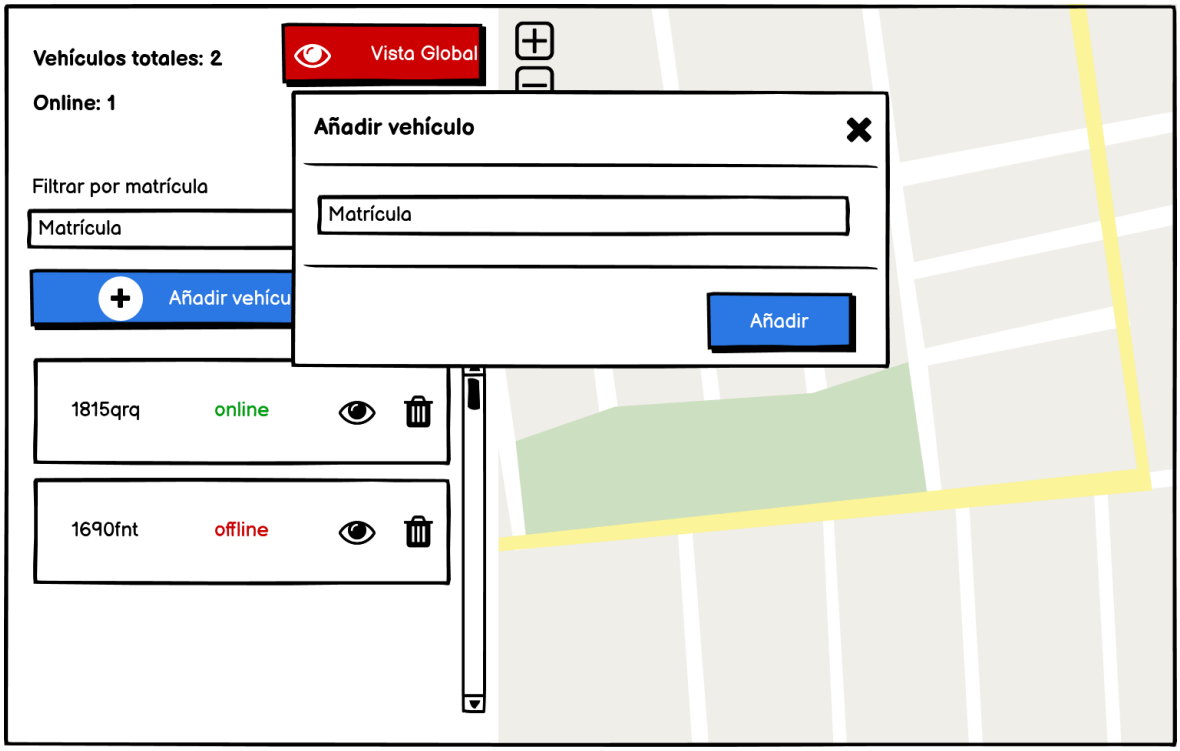

Figura 0-5: Añadir nuevo vehículo al sistema

Figura 0-6: Historial de recorridos de un vehículo específico

Figura 0-7: Grabación en curso de un vehículo específico

D Estructuras de directorios

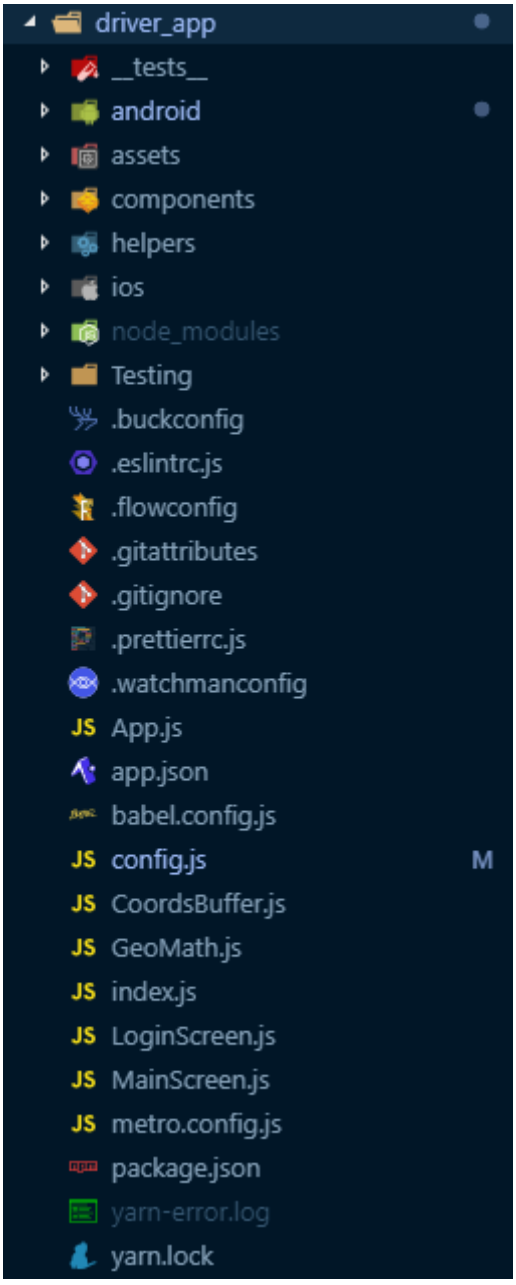

Figura 0-8: Estructura de directorios de la App móvil

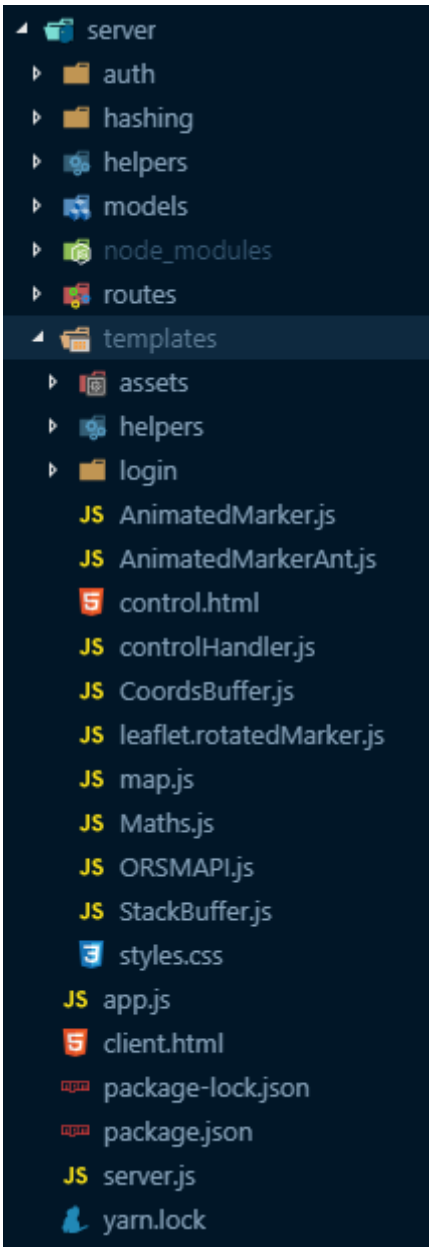

Figura 0-9: Estructura de directorios del servidor en NodeJS

E Problemas con la Geolocalización en Background

La aplicación móvil encargada de transmitir las coordenadas para poder realizar el seguimiento, funciona exclusivamente cuando la App está en primer plano. Si cerramos la aplicación, el dispositivo dejará de transmitir coordenadas, por lo que el vehículo permanecerá parado hasta que se vuelva a situar en primer plano la aplicación.

Hay otras bibliotecas que permiten el trackeo de coordenadas mientras la App está en segundo plano, pero debido a que este servicio puede llegar a consumir mucha batería, las capas personalizadas de Android desarrolladas por los distintos fabricantes de smartphones, tienden a bloquear el servicio a pesar de que se concedan permisos expresamente en la aplicación.

Para solventar este problema, se pueden configurar unos ajustes que varían según el fabricante. Existe una página con instrucciones para cada marca de dispositivo, se puede encontrar en la página: https://dontkillmyapp.com/

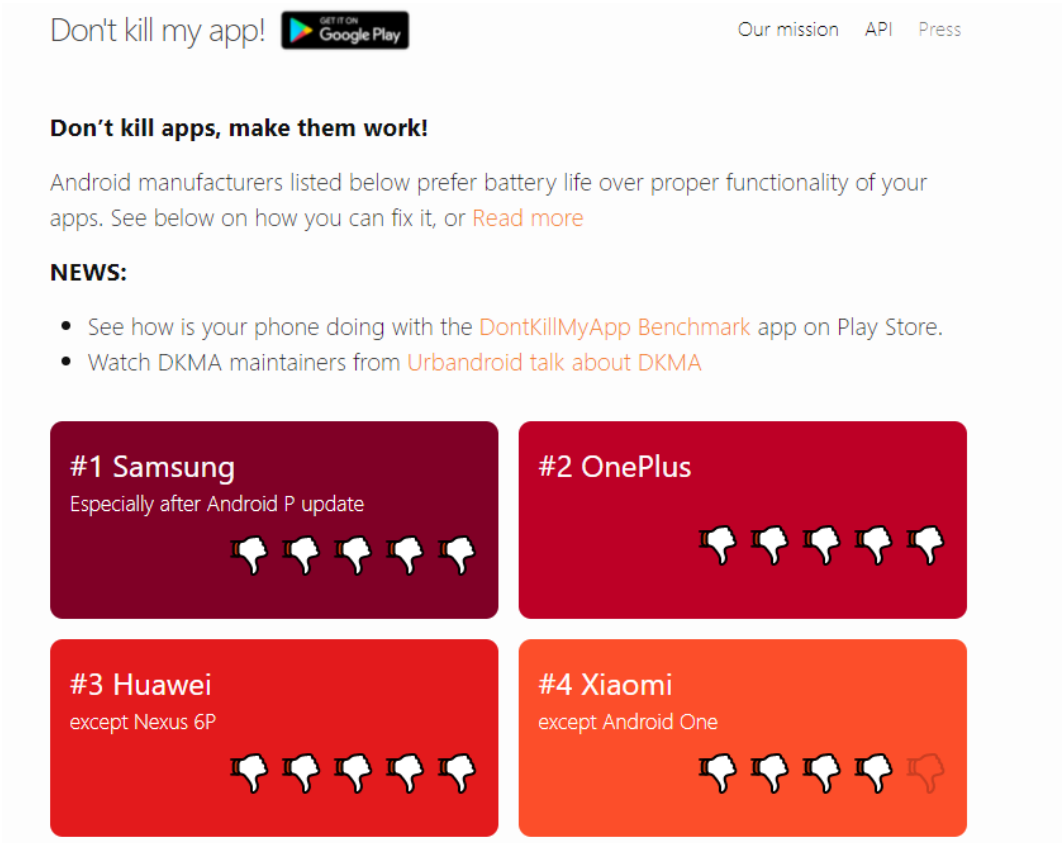

Figura 0-10: Diferentes marcas de dispositivos con problemas de Background

F Fórmulas de Haversine y Bearing en Javascript

```
1.  const toRad = x => x * Math.PI / 180;
2.  const toDegrees = x => x * 180 / Math.PI;
3.
4.  function bearing(startLat, startLng, destLat, destLng) {
5.      startLat = toRad(startLat);
6.      startLng = toRad(startLng);
7.      destLat = toRad(destLat);
8.      destLng = toRad(destLng);
9.      y = Math.sin(destLng - startLng) * Math.cos(destLat);
10.     x = Math.cos(startLat) * Math.sin(destLat) -
11.         Math.sin(startLat) * Math.cos(destLat) * Math.cos(destLng - startLng);
12.     brng = Math.atan2(y, x);
13.     brng = toDegrees(brng);
14.     return (brng + 360) % 360;
15. }
16.
17. function haversineDistance(start, end) {
18.     let lon1 = start[0];
19.     let lat1 = start[1]
20.     let lon2 = end[0];
21.     let lat2 = end[1];
22.     let R = 6371;
23.     let x1 = lat2 - lat1;
24.     let dLat = toRad(x1);
25.     let x2 = lon2 - lon1;
26.     let dLon = toRad(x2)
27.     let a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
28.       Math.cos(toRad(lat1)) * Math.cos(toRad(lat2)) *
29.       Math.sin(dLon / 2) * Math.sin(dLon / 2);
30.       let c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
31.       let d = R * c;
32.     return d * 1000;
33. }
34.
```