

SED: Memoria final

# Geolocalización de vehículos

Abril de 2022



UNIVERSIDAD  
**COMPLUTENSE**  
MADRID

**Ernesto Aranda del Valle**  
Universidad Complutense de Madrid  
Madrid, España  
eraranda@ucm.es

## **Resumen**

Los vehículos se encuentran en nuestro día a día. Muchos de ellos, modernos, cuentan con tecnologías de asistencia al conductor, de comodidad para los pasajeros, pantallas multimedia, etc. Entre estas tecnologías está la de geolocalizar el vehículo cuando terminamos un viaje. Es de gran utilidad cuando se nos olvida donde exactamente lo dejamos o simplemente queremos prestarle el coche a alguien.

No obstante, gran parte del parque móvil en España lo componen vehículos que nos disponen de estas facilidades. Además, no todo el mundo está dispuesto a adquirir un vehículo nuevo debido al alto coste que supone y menos por el simple hecho de poder geolocalizarlo.

Es por ello que se ha ideado un sistema independiente el cual se explicará detalladamente a lo largo de este documento, el cual perfectamente puede situarse en un vehículo cualquiera o incluso llevar a un paseo que permita, no solo geolocalizar la última posición de un vehículo o persona, sino de calcular rutas completas y mostrar datos relativos de las mismas. Todo por un precio bajo lo cual permitiría dar esta funcionalidad a millones de vehículos en el mundo.

# Índice

<b>1. Introducción</b>	<b>4</b>
<b>2. Planificación</b>	<b>5</b>
2.1. Estimación de recursos hardware . . . . .	5
2.2. Estimación de recursos software . . . . .	5
2.3. Estimación temporal . . . . .	6
<b>3. Presupuesto</b>	<b>6</b>
3.1. Recursos hardware . . . . .	6
3.2. Recursos software . . . . .	6
3.3. Presupuesto final . . . . .	7
<b>4. Análisis</b>	<b>7</b>
4.1. Definición del problema . . . . .	7
4.2. Casuísticas y como resolverlas . . . . .	7
4.2.1. Costos . . . . .	7
4.2.2. Recepción de datos . . . . .	8
4.2.3. Procesamiento de datos . . . . .	8
4.2.4. Muestreo de datos . . . . .	8
<b>5. Diseño</b>	<b>9</b>
5.1. Diagrama de la arquitectura del sistema . . . . .	9
5.2. Diagrama de la interfaz de usuario . . . . .	10

<b>6. Implementación</b>	<b>11</b>
6.1. Servidor VPN . . . . .	11
6.2. Nodo de recepción GPS . . . . .	11
6.3. Nodo de procesamiento . . . . .	11
6.4. Servidor web . . . . .	11
6.5. Cliente . . . . .	12
<b>7. Conclusiones y vías futuras</b>	<b>12</b>
<b>8. Manual de usuario</b>	<b>13</b>
8.1. Instalación . . . . .	13
8.2. Pantalla inicial de la web . . . . .	14
8.3. Listado de rutas disponibles . . . . .	15
8.4. Ruta sin API . . . . .	16
8.5. Ruta con API . . . . .	17
<b>9. Bibliografía</b>	<b>18</b>

## 1. Introducción

En este proyecto se aborda el desarrollo de una aplicación para el rastreo de un vehículo a tiempo real con la posibilidad de almacenar rutas realizadas con anterioridad con los correspondientes datos registrados, meramente informativos, durante el progreso de la misma. Estos datos recogen valores de **duración** de ruta y **velocidad**: en parado, en movimiento, y total. También recoge valores de **distancia** recorrida.

El sistema lo conforman dos placas Raspberry, una para la obtención de los datos de localización del vehículo y otra para el cálculo de los datos mencionados previamente. Estarán constantemente tomando datos, por lo cual la última ruta generada mostrará a su vez la última localización obtenida.

- **Nodo receptor GPS:** Módulo de GPS para la obtención de la posición global y el envío de esta misma información. Se trata de una **Raspberry Pi Zero 2** con un hat GPS (SIM7070G). Lanza una VPN para la conexión con el módulo de procesamiento.
- **Nodo de procesamiento:** Recibe las posiciones del nodo GPS y las interpreta calculando datos como tiempo de parada, velocidad media, etc. La placa a usar es una **Raspberry Pi 3B+**, está en constante comunicación con el nodo GPS a través de la VPN previamente mencionada.
- **Servidor:** Nodo de almacenamiento y muestreo de la información. Este servidor es la misma **Raspberry Pi 3B+** que hace las veces de nodo de procesamiento. En él se almacena la web a mostrar, que está implementada en Node.js.
- **Cliente:** Interpretará la información recibida del servidor y la mostrará adecuadamente al usuario. Puede ser cualquier dispositivo con capacidad para lanzar un navegador web moderno (a poder ser Google Chrome) así como poder conectarse a la misma VPN.

La comunicación entre placas Raspberry se realiza mediante MQTT (mosquitto). El nodo de procesamiento actúa como broker de *mosquitto* y el nodo de GPS publica en el broker *mosquitto* las posiciones que va recolectando del GPS. Como ya hemos mencionado, todo se hace a través de una VPN, lo cual proporciona una capa extra de privacidad y permite la conexión al servidor web por parte de un cliente que se conecte a esa misma VPN.

La información se muestra mediante una página web escrita en React.js haciendo uso de la API Maps de Google mediante la cual se pinta la hoja de ruta. Sobre este mapa se muestran de igual manera los datos obtenidos por el nodo de procesamiento. Existe

una lista de rutas disponibles seleccionable mediante la cual se puede seleccionar la ruta deseada. Al seleccionar una de las rutas muestran todos los datos disponibles asociados a la misma.

Con esta combinación de hardware y software se quiere dar solución al problema citado anteriormente. Una solución a mano de todo el mundo y con facilidad de uso puesto que las placas son de pequeño tamaño y se pueden colocar en casi cualquier lugar pudiendo aplicar esta solución a múltiples vehículos, e incluso para grabar rutas a pie o bici.

## 2. Planificación

### 2.1. Estimación de recursos hardware

- *Nodo receptor: Raspberry Pi Zero 2 + Hat GPS SIM7070G + USB Wifi con lector de SIM*
- *Nodo de procesamiento y servidor web: Raspberry Pi 3B+*
- *Cliente: MacBook Air 2020*
- *Servidor VPN: VPS S con 1 vCore, 512 MB Ram y 10 GB*

### 2.2. Estimación de recursos software

- *Sistema operativo de los nodos: Raspberry Pi OS*
- *Entorno de ejecución del servidor web: Node.js*
- *Lenguajes de marcado: HTML5*
- *Lenguajes de programación:*
  - Python
  - Javascript
- *Lenguajes de estilo: CSS3*

### 2.3. Estimación temporal

Debido a que el desarrollo del proyecto ha constado de 6 semanas a lo sumo, contando cada semana como 5 días laborales, en total por tanto, 30 días laborales. Han sido divididas de la siguiente manera:

- *Implementación de la recepción de señal GPS: 1 semana*
- *Implementación de obtención de datos mediante coordenadas GPS: 1 semana*
- *Implementación del servidor web: 2 semanas*
- *Implementación del sistema en tiempo real: 1 semana*
- *Puesta a punto y pruebas: 3 días*
- *Documentación: 2 días*

## 3. Presupuesto

### 3.1. Recursos hardware

Como el **cliente** usado es un portátil ya existente así como el **servidor VPN**, solo suponen gastos los **nodos** de recepción GPS y procesamiento, el **Hat GPS** y el **USB Wifi con lector de SIM** para la conexión a internet. El nodo de procesamiento supone **50€** mientras que el nodo GPS **20€**. El hat GPS cuesta **30€**. El USB Wifi unos **20€**.

### 3.2. Recursos software

El uso de bibliotecas no tiene ningún coste extra alguno así como la VPN tampoco supone ningún gasto más. No obstante, la *API Roads* de *Google* para suavizado de rutas supone un gasto estimado de **50€ / 5000** llamadas. En nuestro caso hemos usado un saldo que ha proporcionado *Google* inicialmente por cuestiones prácticas de **200€**. Esto supone que no ha habido gasto alguno aún en el proyecto en términos de software.

### 3.3. Presupuesto final

Ya que solo los recursos hardware han supuesto un gasto, el costo total final es la suma de todos los gastos de esta categoría, que en total sumaría un gasto de alrededor de **120€**. En este caso se han usado estas placas ya que han sido ofrecidas por la empresa **Ingenios telemáticos**, no obstante, pueden usarse placas distintas de precios reducidos con el mismo propósito pudiendo reducir bastante este presupuesto.

## 4. Análisis

### 4.1. Definición del problema

La idea con la que se creó este proyecto era la de crear un sistema de localización de vehículos para obtener información relevante para un usuario pero que fuese extensible a muchas funcionalidades tales como la apertura de garajes a distancia, notificaciones por hurto, etc. Para ello, se ha formado un sistema capaz obtener información útil como la **Última posición** del vehículo, **velocidad**, **tiempo de trayecto** o **distancia recorrida** que serán además de gran utilidad para los servicios previamente mencionados. Por ejemplo, para saber si nos acercamos a casa para abrir el garaje, sería útil conocer si nos estamos acercando y a qué velocidad para estimar cuando abrirla. Para obtener estos datos, es necesario obtener las coordenadas en tiempo real y extraer la información necesaria al momento. Con esto se puede saber la ruta que traza un vehículo junto a los datos previamente mencionados.

### 4.2. Casuísticas y como resolverlas

#### 4.2.1. Costos

Toda esta información es de por sí algo que vemos en el día a día, pero que en un vehículo cualquiera no podríamos obtener mediante los sistemas empotrados que posee, al menos en aquellos de más antigüedad. Es por ello que se ha ideado un sistema sencillo y de bajo coste mediante *Raspberries* para que cualquier usuario pueda obtener esta funcionalidad para cualquier vehículo que desee, independientemente de lo reciente que sea. Como hemos visto previamente, el costo es de alrededor de **120€** que puede ser reducido si usamos placas de menor costo o incluso sensores GPS más asequibles, pudiendo reducir el valor incluso hasta los **60€** si usasemos un servidor web dedicado para hacer parte del procesamiento.

#### 4.2.2. Recepción de datos

La información básica que debemos obtener son las coordenadas GPS las cuales se recibirán mediante el **Hat GPS** y que el **nodo de recepción GPS** enviará mediante **MQTT mosquitto** al nodo de procesamiento.

#### 4.2.3. Procesamiento de datos

Además de las coordenadas GPS, necesitamos obtener las medidas derivadas de las mismas. No obstante, el nodo de recepción no es suficientemente potente para realizar todas las tareas. Por ello todos estos cálculos se realizan en un **nodo específico de procesamiento** que recibirá las coordenadas del nodo de recepción mediante **MQTT Mosquitto**.

#### 4.2.4. Muestreo de datos

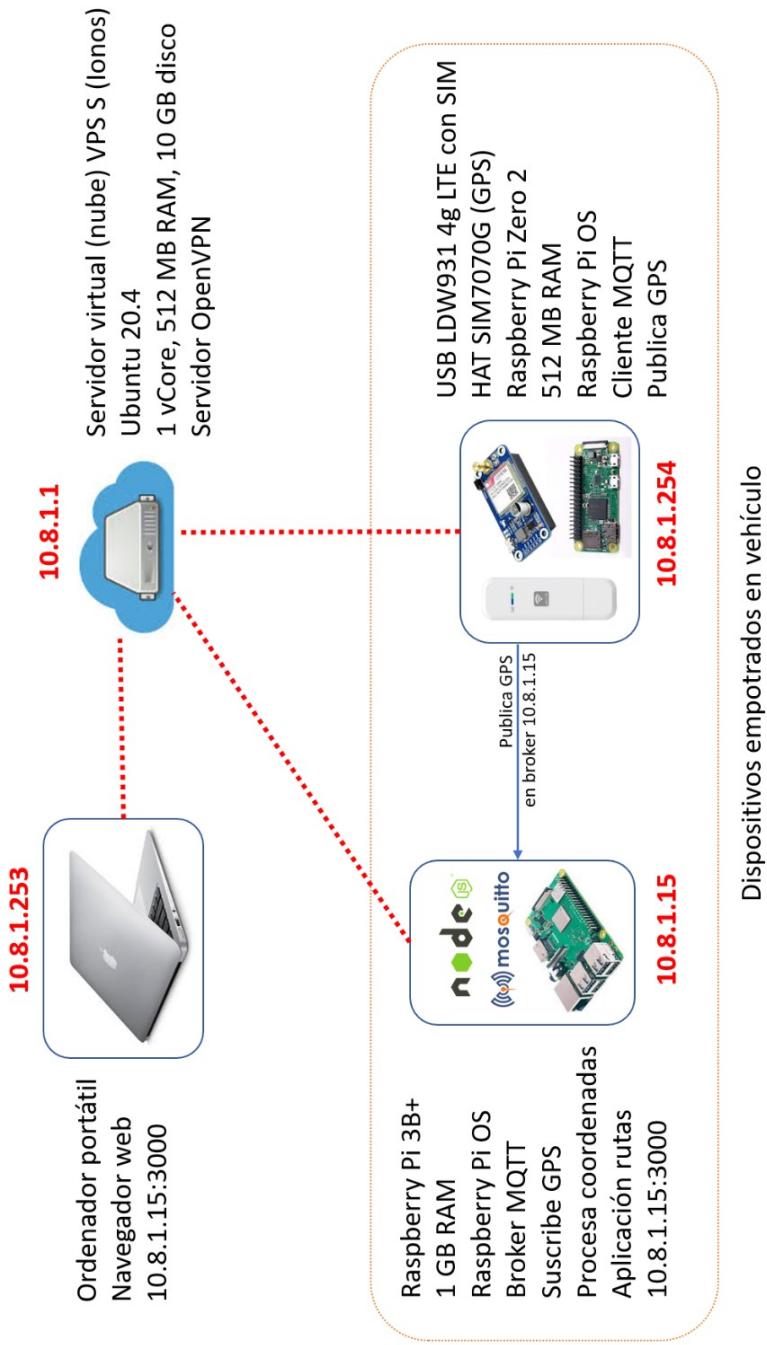
Obtenidos los datos necesarios, se definen los casos de uso en los que aplicar esta información. En este caso poseemos medidas numéricas (*velocidad, tiempo, distancia*) y coordenadas geográficas. Un usuario puede:

- Seleccionar la ruta actual
  - Visualizar la última posición del vehículo
  - Visualizar las medidas obtenidas para esa ruta
  - Obtener información actualizada de ruta
- Seleccionar una ruta pasada
  - Visualizar las medidas obtenidas para esa ruta
- Suavizar ruta cuando las coordenadas son irregulares
- Navegar por el mapa

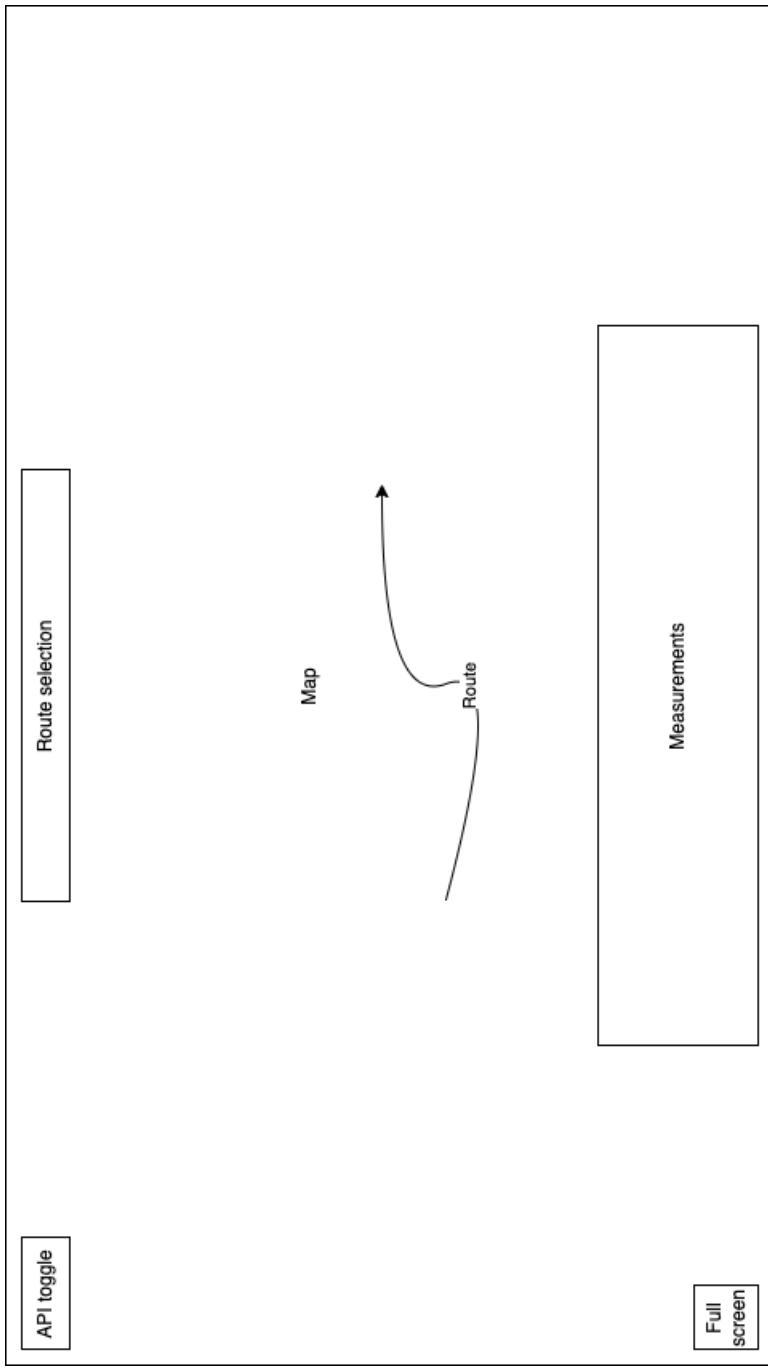
Para dar solución a todos estos puntos se creará una **página web interactiva** en la que se muestra un mapa y que permite seleccionar rutas mediante un **listado de rutas disponibles** incluyendo la actual o más reciente, siendo esta última la que muestra la última posición del vehículo. Cuando se selecciona una de estas rutas **se mostrarán en una tabla las medidas disponibles**. En caso de que el GPS tome coordenadas de manera errónea, se integra la **API Roads de Google** para suavizar la ruta y paliar estas irregularidades. El servidor se alojará en el mismo nodo de procesamiento.

## 5. Diseño

### 5.1. Diagrama de la arquitectura del sistema



## 5.2. Diagrama de la interfaz de usuario



## 6. Implementación

### 6.1. Servidor VPN

Es una máquina virtual que tiene instalado un servidor **OpenVPN** con la dirección **IP 10.8.1.1**. Para poder crear los certificados de servidor y clientes, se necesita obtener una firma de una entidad certificadora externa.

### 6.2. Nodo de recepción GPS

El **nodo GPS** usa un Hat SIM7070G con el que captura las coordenadas geográficas del GPS (latitud, longitud). Primero establece la conexión con el servidor MQTT y cada vez que recibe coordenadas válidas, las publica con el *topic gps*. Además, se suscribe al *topic gts* para recibir actualizaciones del *timestamp* desde el nodo de procesamiento ya que este nodo tiene restringido los puertos de salida para únicamente utilizar los necesarios para el establecimiento de la VPN y la comunicación con el nodo de procesamiento. Tiene instalado un cliente *OpenVPN* con la **IP 10.8.1.254**.

### 6.3. Nodo de procesamiento

Está instalado el *broker MQTT mosquitto*. A través de este broker se suscribe como cliente al *topic gps* para recibir las coordenadas del **nodo GPS**. Conforme las recibe, las coordenadas se almacenan en un fichero .txt junto a su *timestamp*. Al mismo tiempo, publica cada dos minutos con el *topic gts* el *timestamp* del equipo, ya que este nodo sí tiene actualizada la hora. De esta manera se garantiza que los dos nodos están sincronizados con la misma hora. Tiene instalado un cliente *OpenVPN* con la **IP 10.8.1.15**.

### 6.4. Servidor web

En el mismo nodo de procesamiento se lanza el servidor web a través de un entorno **Node.js**. Este contiene una web escrita en **Javascript ES6 + React.js** que nos permite actualizar la información al instante, útil a la hora de mostrar información en tiempo real.

## 6.5. Cliente

Es una máquina cualquiera con posibilidad de instalar **OpenVPN** y tener acceso a navegadores web modernos, preferiblemente Chrome. Para establecer la conexión a la web debe conectarse al mismo *servidor VPN* mediante **OpenVPN**. Es necesario solicitar una IP para el mismo, en el caso de la máquina de pruebas será la **IP 10.8.1.13**

## 7. Conclusiones y vías futuras

Con este sistema podemos dar cabida a muchas funcionalidades aparte de la de geolocalización. Podemos obtener diversas medidas como velocidad, distancia, tiempo recorrido, etc. Todos estos datos pueden ser muy útiles a la hora de realizar análisis, más ahora cuando el **consumo de combustible** es algo crítico y hemos de encontrar la mejor manera de mover un vehículo, teniendo en cuenta la manera en la que una persona lo conduce así como por donde hacerlo con el objetivo de reducir los niveles contaminación así como gastos dinerarios.

Esta idea es extensible, como ya se ha dicho, al cálculo de consumos de combustible por trayecto, útil para conocer cual es la **mejor forma de conducir un vehículo** para distintos tipos de trayectos (cortos, medios o largos), tanto de combustión como híbridos o eléctricos. ¿Para qué sería útil esta medida? No todos los modelos de vehículos son iguales, algunos se conducen de manera distinta a otros. No muy lejos están los coches eléctricos que son de bajo consumo por ciudad y alto por autovía, cosa contraria a los motores de combustión. Para reducir estas debilidades, obtener la mejor manera de ir de un punto *A* a *B* sería algo muy útil para el conductor y saldríamos ganando todos, ya se que se **reduciría el consumo genérico de combustible** e igualmente la contaminación.

Por ejemplo, si queremos ir de *Barcelona* a *Madrid*, podríamos saber que velocidad media sería la más conveniente para alcanzar nuestro destino reduciendo el consumo de manera que el tiempo no se vea altamente afectado.

También pueden darse otros usos. El tener un sistema de geolocalización en tiempo real puede extender la **domotización** de nuestra casa. A priori suena algo extraño pero, ¿y si la puerta de un garaje se abriese automáticamente simplemente con detectar que se ha llegado a casa? El mundo de la domótica podría estar muy ligado a este proyecto si avanzase por esta vía. También puede darse uso como **alarma** en caso de que el vehículo en el que se instale se mueva de manera indeseada.

## 8. Manual de usuario

### 8.1. Instalación

Para poder hacer uso de esta web (con datos de ejemplo) en una máquina cualquiera nos harán falta tener instalados *Python 3.9* (o mayor) y *Node.js 12* (o mayor) y clonar el siguiente repositorio (en él se detalla igualmente toda la funcionalidad existente):

```
https://github.com/ernestoadv/VehicleTracker
```

Si se quieren mostrar datos propios, haría falta incluir en la carpeta *data* dentro de *measurements* archivos *.txt* con líneas de tres valores: **timestamp** (3 dígitos) del momento en el que se obtuvo la coordenada, **latitud** y **longitud** de la misma coordenada. Ejecutando el archivo *tracker.py* automáticamente generará la información necesaria en la carpeta correspondiente para el servicio web.

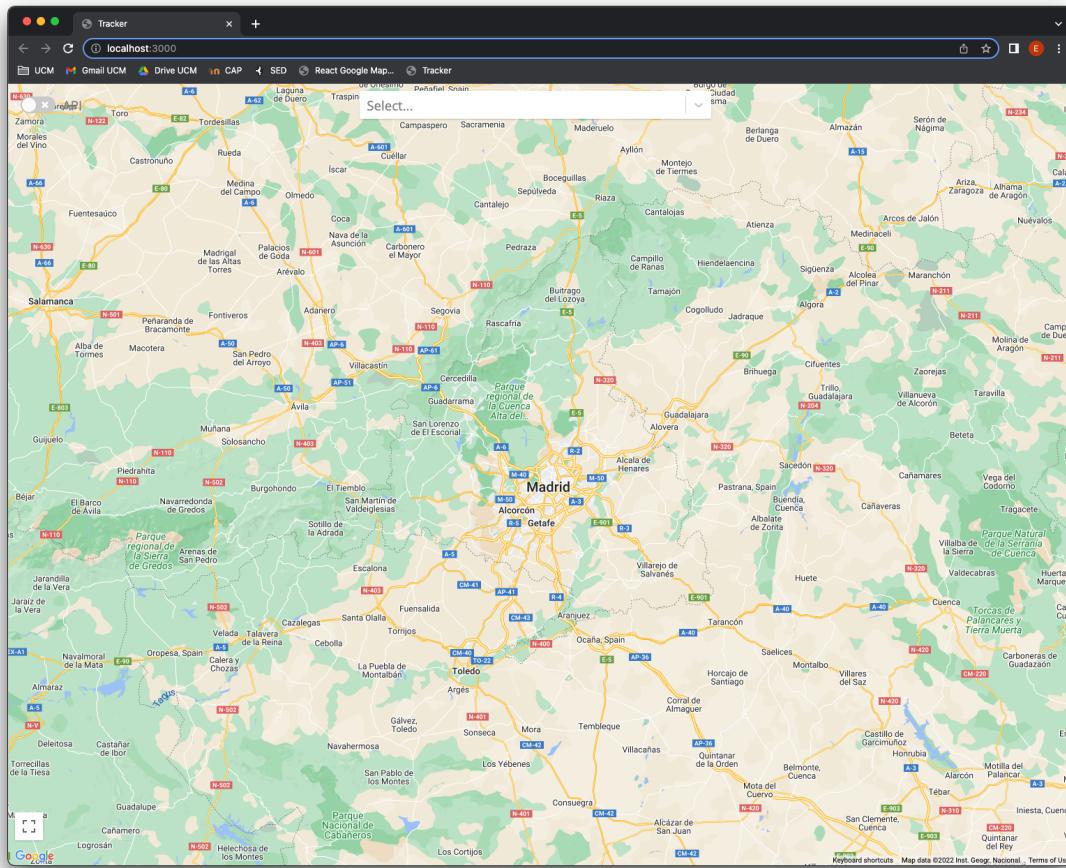
Para lanzar este servicio web necesitamos primero preparar el servidor *Node.js*. Es preferible abrir la web en un navegador *Chrome* moderno. Lo primero que haremos es instalar las dependencias que el proyecto necesita. Nos dirigiremos a la carpeta *tracker* y ejecutaremos el comando *npm install*. Una vez terminado, crearemos un archivo *.env* en ese mismo directorio que contendrá los siguientes valores:

```
GOOGLE_MAPS_API_KEY=your_api_key  
ROADS_API=https://roads.googleapis.com/v1/snapToRoads
```

Por último, lanzaremos el servidor usando el comando *npm start*. Este comando automáticamente lanzará un *localhost:3000* en un navegador *Chrome*. Si no existe o no se abre automáticamente, simplemente se escribe esa misma *URL* local y se abrirá un mapa como el que se muestra a continuación.

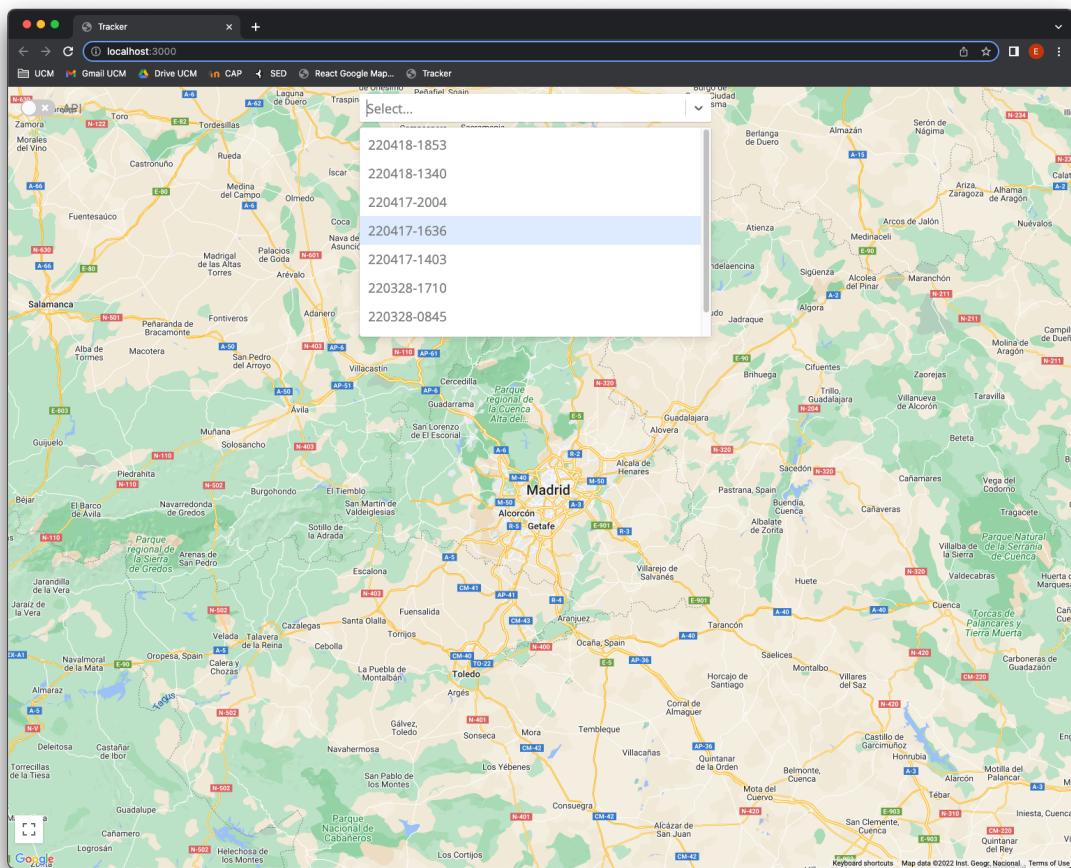
## 8.2. Pantalla inicial de la web

Esta pantalla aparece cuando cargamos la página antes de realizar ninguna acción. En ella encontramos tres botones: **API**, **Pantalla completa** y un *Seleccionable vertical*. El primero habilita la *API de Google* para suavizar las rutas en caso de que las coordenadas sean irregulares, el segundo *amplia el mapa* a todo el viewport del monitor actual y el último *muestra la lista de rutas* almacenadas disponibles para mostrar.



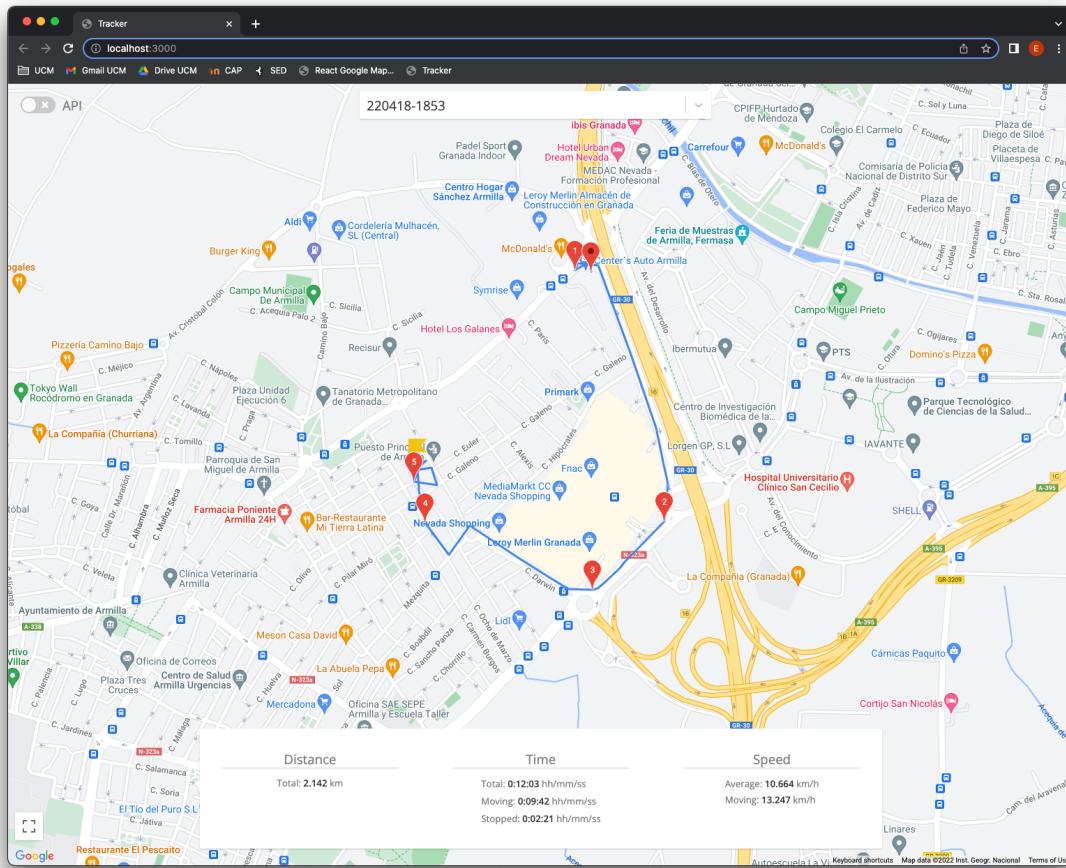
### 8.3. Listado de rutas disponibles

Haciendo click en el seleccionable se mostrarán todas las rutas disponibles. Cuando hay una ruta actual en progreso, aparecerá con el nombre *Current route* en la primera posición de la lista.



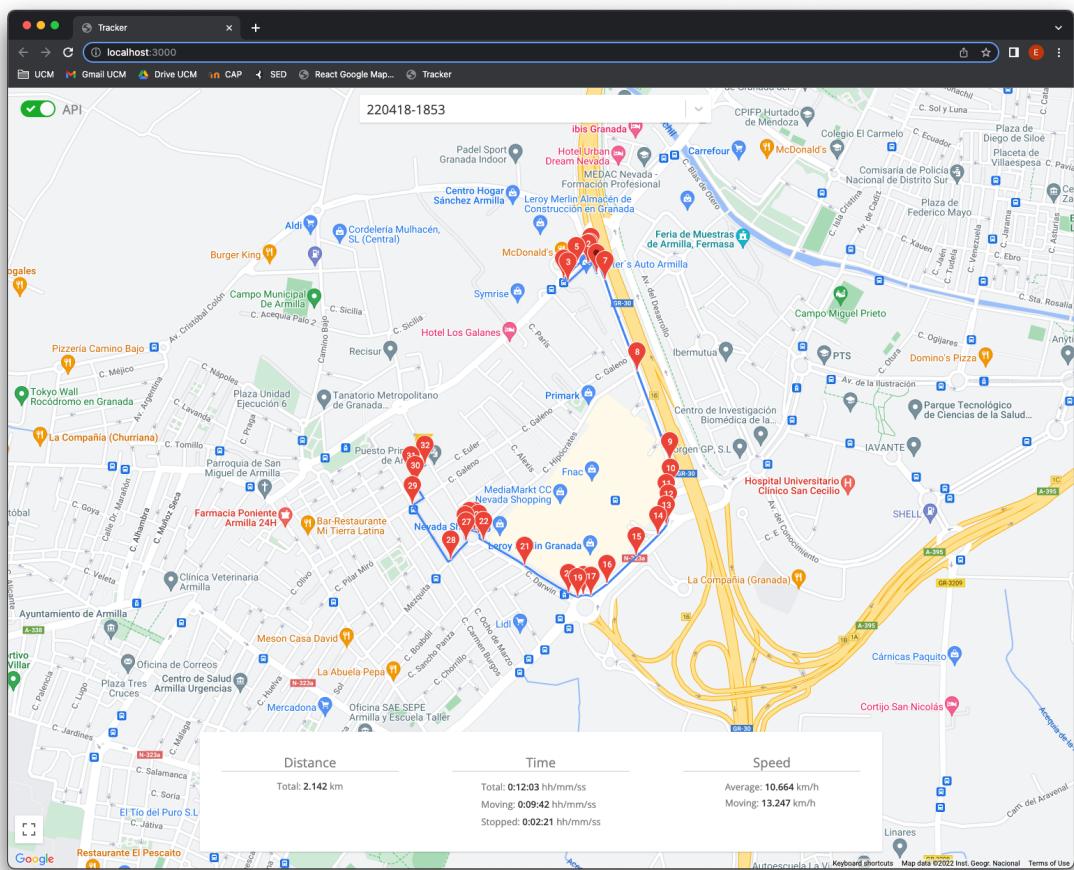
## 8.4. Ruta sin API

Al seleccionar una ruta cualquiera, se pintará sobre el mapa siguiendo una serie de *marcadores* numerados en orden de aparición. El último será una bandera mientras que el primero un marcador sin número. Por defecto, y como ocurre en este caso, no se aplica la *API* de suavizado de rutas de Google. Podemos ver también abajo como la plantilla de datos se rellena para esa ruta con valores de **distancia**, **tiempo** y **velocidad**.



## 8.5. Ruta con API

En este caso se ha usado la *API* para suavizar la ruta. Se puede distinguir de la anterior ruta como la línea sigue la carretera tomando las curvas de las calles y no cortando esquinas como hacia el previo ejemplo.



## 9. Bibliografía

- Designthemes. (2022, 8 marzo). Instalando Broker MQTT en Raspberry PI. Tienda y Tutoriales Arduino. <https://www.prometec.net/instalando-mosquitto-en-raspberry/>
- Google. (2022). Google Maps Platform Documentation — Maps JavaScript API —. Google Developers. <https://developers.google.com/maps/documentation/javascript?hl=es>
- Vehicle Tracker. (2022). GitHub. <https://github.com/ernestoadv/VehicleTracker>
- Waveshare. (2020). SIM7070G Cat-M/NB-IoT/GPRS HAT - Waveshare Wiki. SIM7070G Cat-M/NB-IoT/GPRS HAT. [https://www.waveshare.com/wiki/SIM7070G\\_Cat-M/NB-IoT/GPRS\\_HAT](https://www.waveshare.com/wiki/SIM7070G_Cat-M/NB-IoT/GPRS_HAT)