****

**IES Maestre de Calatrava**

Paseo de la Universidad,1

13005-Ciudad Real

**F.P. INFORMÁTICA**

**CURSO 2018-19**

**Marzo - Diciembre**

**Convocatoria: Ordinaria**

**Técnico Superior en**

**Desarrollo de Aplicaciones Multiplataforma**

MEROTrack

**Nombre Alumno: Alberto Menchén Ruiz**

**DNI: 20616193M**

**Todos los apartados son obligatorios**

Se recomienda que el trabajo a realizar se desarrolle con arreglo al siguiente guion:

* Portada (**ANEXO** I)
* Índice de contenidos paginado.

**APARTADO I:** (Contenidos vinculados a los módulos de la especialidad de FOL)

1. **Definición de la Idea de Negocio** (Factor diferenciador y Utilidad del mismo)
2. **Análisis del Entorno de la Empresa.**
3. **DAFO y CAME**
4. **Viabilidad Económica y Técnica.**
5. **Trámites de puesta en marcha e impuestos a los que estás sujeto.**
6. **Política de prevención de Riesgos laborales en la Empresa**. (Riesgos expuestos-medidas preventivas adoptadas)

APARTADO II:

# TITULO

MEROTrack (Menchén-Romero-Transportes/Tracker)

# INTRODUCCIÓN (español)

MEROTrack es una aplicación web dinámica que permite, desde un panel de control en la web, mantener un seguimiento mediante geolocalización una flota de camiones y otros vehículos de transporte.

El tema de este proyecto fue elegido por la cercanía con el mundo del transporte, ya que toda mi familia se dedica en mayor o menor medida a esta actividad.

También fue elegido por la necesidad del transporte en el mundo moderno. Sin importar el producto, todo aquello que se produce ha de ser transportado. Las materias primas han de llevarse a las fábricas, y a su vez, los productos que estas fábricas generan han de ser distribuidos bien directamente al cliente, o a las tiendas donde se vayan a vender. Aunque no lo ‘veamos’, el transporte está detrás de todo aquello que consumimos y compramos.

Por tanto, una de las primeras necesidades que cubre MEROTrack es la gestión de viajes y repartos, haciendo especial hincapié en la optimización de rutas, a fin de ahorrar al máximo posible el gasto en combustible y más importante aún, tiempo, a la empresa (y al cliente).

La otra necesidad a cubrir es el control de la posición de vehículos, por dos principales factores:

* **El robo de vehículos, o su mercancía.** El cual es más común de lo que se piensa, al mantener geolocalizado el mismo, hay más posibilidades de recuperar el vehículo tras su robo.
* **Fraude.** Algunos conductores hacen ver como que trabajan o hacen más kilómetros de lo que realmente hacen, alargando los tiempos de descanso en enorme medida. Mantener un control en tiempo real, e histórico, de la posición de su vehículo ayuda en la evaluación del rendimiento de nuestros conductores.

# INTRO (english)

*MEROTrack is a dynamic webapp that allows, thanks to its web-based dashboard, keep track of a cargo vehicle fleet.*

*This topic has been chosen because of close to my family transportation is, due to the fact that almost all my entire family works in this field.*

*It has also been chosen because of how important the transport field is nowadays. No matter the product, everything once it’s produced must be carried around. From raw materials, which must be delivered to factories; to finished products, which must be delivered to the customer. Even if we don’t usually get to see the backstage, transportation is always behind every single thing we buy and consume.*

*Because of this, one of the first needs that MEROTrack must fulfill, is the management of trips and deliveries, with special interest in route and delivery optimization, achieving economical and time savings to the deliverer and the deliveree.*

*The second, and equally important neccessity to cover, is the management of the delivery vehicles themselves, for two main reasons:*

* ***Theft of the vehicle or its cargo.*** *More common than what it’s usually though, thanks to the integrated geotracker that sends its position in real-time, we can easily recover the stolen goods.*
* ***Fraud****. Some drivers pretend to work more time, or drive more miles than what they really do, by knowing where they are, we can effectively measure their actual performance.*

# RECURSOS MATERIALES Y PERSONALES PARA LA REALIZACIÓN DEL PROYECTO

A nivel personal, el recurso a aportar obviamente ha sido el tiempo, el cual estimo en unas 150 horas.

El módulo hardware encargado de la geolocalización de los vehículos, el rastreador GPS, está compuesto de una controladora programable Arduino modelo UNO v3, al cual se ha acoplado una *shield* marca DFRobotcon en el chip Sim908, el cual permite acceder a la red telefónica e internet mediante una tarjeta SIM, y obtener posiciones GPS mediante una antena integrada.

Debido a que el proyecto no se va a llevar a producción, y no hay una carga de trabajo importante, todo el software se ejecuta en un ordenador personal, y no se ha montado ningún servidor dedicado.

Se ha optado principalmente por software de código abierto para evitar costes y problemas de licencias. Algunos programas, librerías o frameworks utilizados son:

* Mapbox como proveedor externo de servicios de mapas
* BD PostgreSQL
* Backend en Spring BOOT
  + Spring Initializr para la inicialización del proyecto backend
  + Spring Data REST para exponer rutas del modelo al cliente
  + Spring Data JPA para el acceso a BD
  + Spring Security para la securización del API
  + Spring Web
  + Gradle para gestión de dependencias Java
  + Lombok
* Frontend en Angular 7
  + Angular4-Hal para interactuar con el backend
  + Nebular, componentes basados en bootstrap
  + Ngx-Admin como principal plantilla de diseño e interfaz
  + PrimeNG, algunos componentes
  + Leaflet como componente para visualización de mapas
    - Leaflet Routing Machine
    - Leaflet Control Geocoder

Los entornos y utlidades de desarrollo empleadas son, principalmente:

* Arduino IDE
* IntelliJ IDEA
* VSCode
* Angular CLI
* Docker
* Postman
* PgAdmin

# PRESUPUESTO ECONÓMICO

No se ha invertido una gran cantidad de recursos en la realización del proyecto.

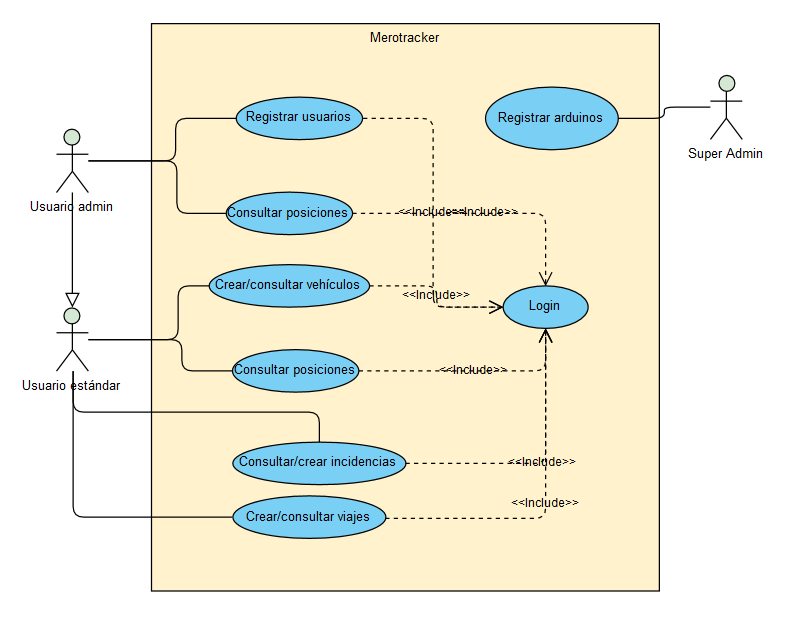
Debido a que no se ha llevado a producción, no ha sido necesario invertir en un servidor dedicado.

El resto de materiales estaban ya comprados antes del inicio del desarrollo. El portátil (~1000€ aprox), la arduino (~40€), y la shield SIM908 (~90€ en el momento de su compra, ahora 40€)

Los servicios de mapas son provistos por Mapbox; sin embargo, debido a que no se ha alcanzado el límite mensual de consultas, no ha supuesto ningún coste.

# FASES Y SECUENCIACIÓN DE LAS ACTIVIDADES DEL PROYECTO

## DIAGRAMA CASOS DE USO



Existen dos principales roles de uso, siendo el ‘Super admin’ el administrador total del sistema, que será el único que tenga acceso (directamente de la BD) a los identificadores privados y públicos de cada vehículo almacenado en la BD. Cabe destacar que, aunque se puedan crear vehículos en el panel web, estos no enviarán ninguna posición al ser necesario registrar previamente la Arduino que se encarga de la geolocalización.

El usuario ‘estándar’ es aquel usuario que puede acceder a los menús de creación/edición/eliminación de viajes, vehículos e incidencias; sin embargo, no puede acceder a la vista de posiciones ni a la de gestión de usuarios por motivos de seguridad.

El administrador, junto a las acciones a las que puede acceder el usuario estándar, sí que puede consultar la vista de posiciones (tanto la vista de flota, que muestra la última posición conocida de cada vehículo, como la vista histórica, que muestra las posiciones de un vehículo concreto en un rango de fechas). Y también puede acceder al panel de gestión de usuarios.

## DIAGRAMA DE CLASES UML

## Spoiler: No hay

### FRONTEND

El paradigma de ‘programación’ que sigue Angular, si bien usa clases ‘tradicionales’ en algunas partes, la inmensa mayoría del código se realiza mediante *Componentes*, los cuales están declarados por *Módulos,* y a los que se pueden inyectar *Servicios*.

Los componentes a su vez implementan Templates, Styles, y pueden implementar a su vez Directivas de entrada/salida.

Además de eso, un componente puede estar formado por varios archivos: su código, su template HTML, su hoja de estilos independiente…

Angular no sigue una metodología de programación que de pie a un Diagrama UML, debido a su complejidad, el hecho de que puede hacer uso de infinidad de librerías y componentes de terceros (en este proyecto se ha usado Nebular, PrimeNg y Leaflet) y el hecho de que muchos de los componentes son inyectados directamente con clases propias de Angular como el HTTPClient, por poner un ejemplo.

### BACKEND

Spring, pese a ser un framework de Java, tampoco hace un uso ‘tradicional’ de las clases. Gracias a las anotaciones (Java 1.5+), Spring genera en tiempo de compilación gran parte de la implementación y se encarga automáticamente de la inyección de dependencias y la configuración general de la aplicación. Esta forma de desarrollar tampoco daría pie a un diagrama UML al uso.

Aunque no haya una relación directa entre las múltiples partes de la aplicación Spring durante el desarrollo, el proyecto mantiene la siguiente estructura:

* Modelo: Los POJOS, los cuales contienen los datos de las columnas de las entidades de la BD y su mapeo JPA
* Repositorio: Permiten el acceso a datos, y exponen rutas REST de forma automática.
* Controllers: Para operaciones específicas, sobrescriben algunas rutas REST.
* Specifications: Contienen filtros de búsqueda para las consultas a la BD.
* Security: Configuraciones varias de seguridad y filtros de autorización/autenticación.

### ARDUINO

El programa que controla la Arduino, el rastreador, está escrito en C, no C++, el cual directamente no tiene soporte para clases.

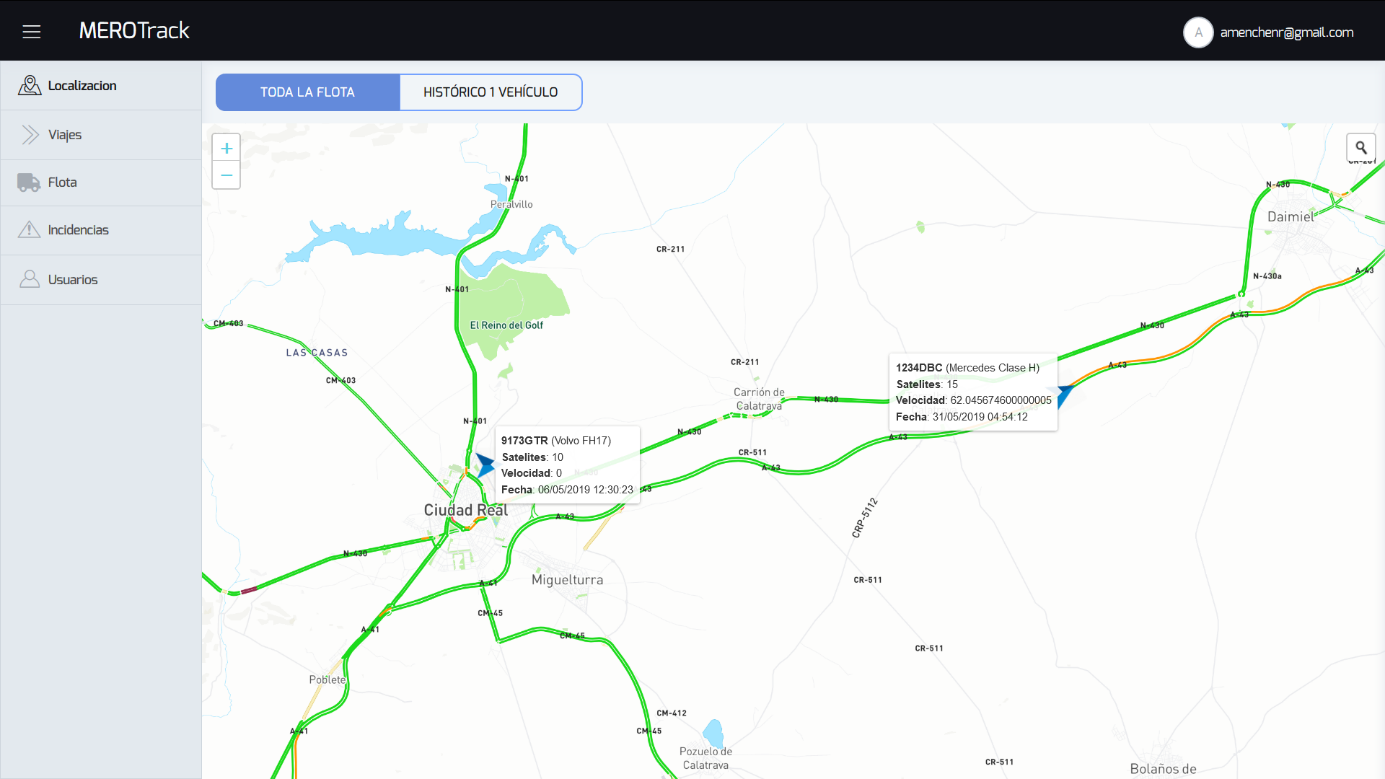
## DIAGRAMA E/R DE LA BASE DE DATOS

## INTERFACES GRÁFICAS

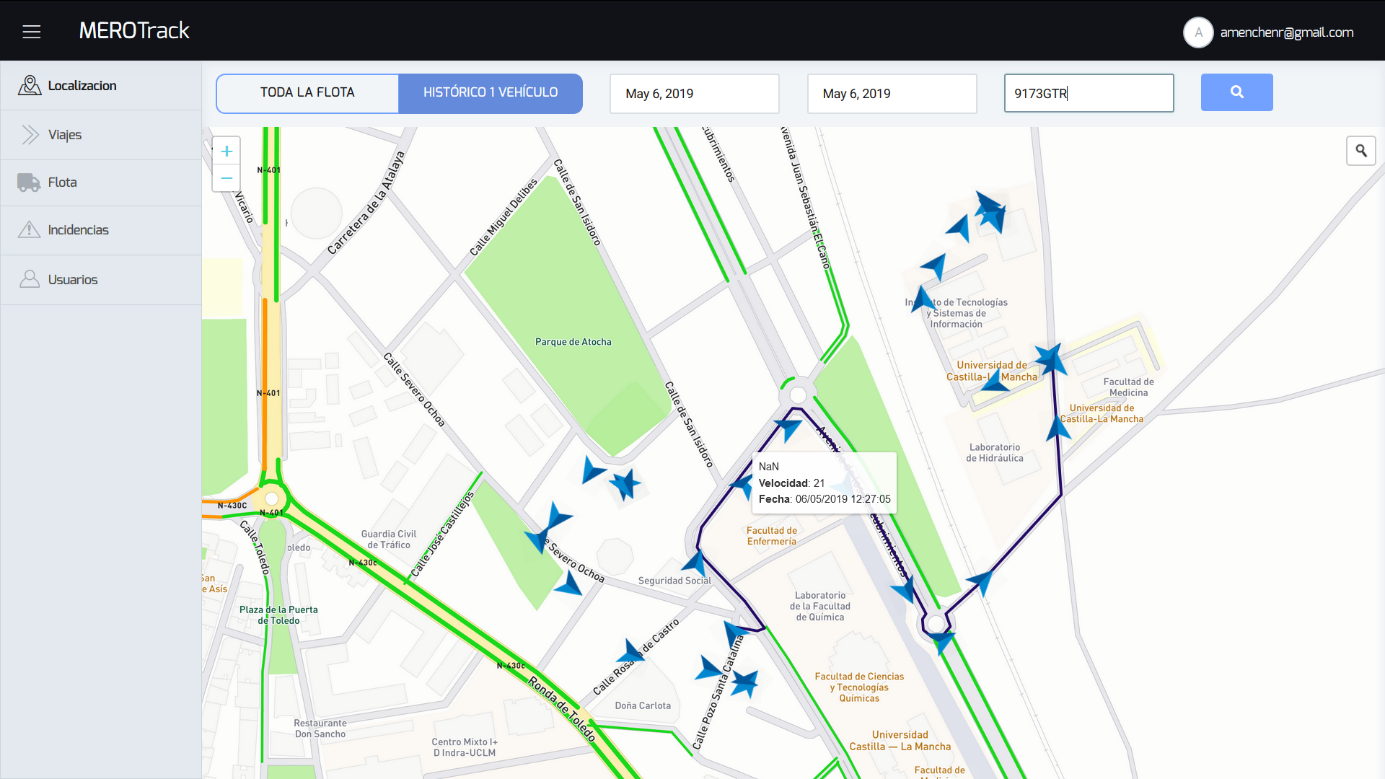
**LOG-IN:** Permite iniciar sesión en la aplicación

## 

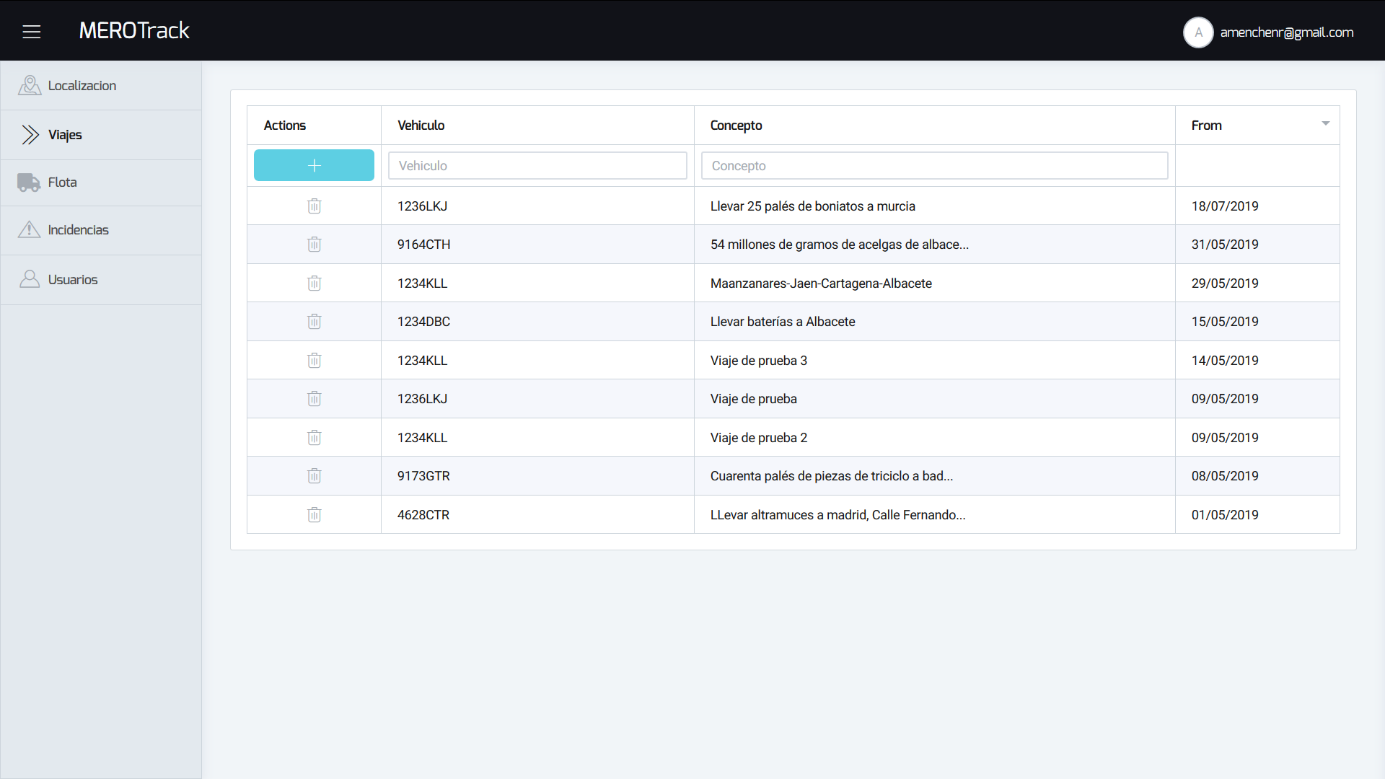
**PANEL DE POSICIONES:** En la modalidad ‘Toda la flota’ permite visualizar a tiempo real la última posición conocida de todos aquellos vehículos que tengan posiciones registradas. Indica también por colores el nivel de tráfico actual en las calles, así como la velocidad que llevaba el vehículo en ese momento.



En la modalidad ‘histórico 1 vehículo’, permite hacer una búsqueda por vehículo y rango de fechas, mostrando las posiciones del vehículo elegido durante el período seleccionado, mostrando una línea con la ruta más probable que haya tomado el vehículo.

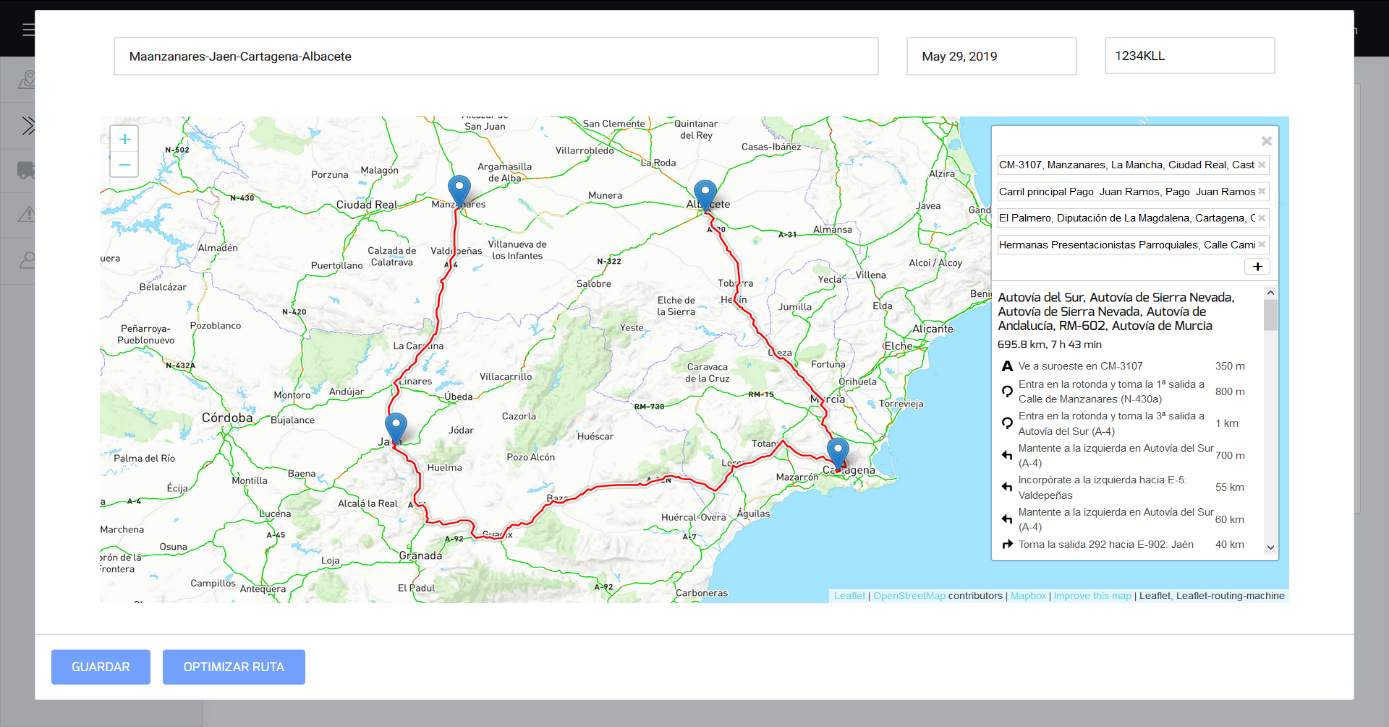


**PANEL DE VIAJES:** Muestra un listado con los viajes realizados o a realizar. La tabla se puede ordenar o filtrar por los campos de vehículo o de concepto. Se pueden borrar viajes. Al hacer click en una de las filas o en el botón ‘+’ (nuevo), se abre el panel de edición de viaje.

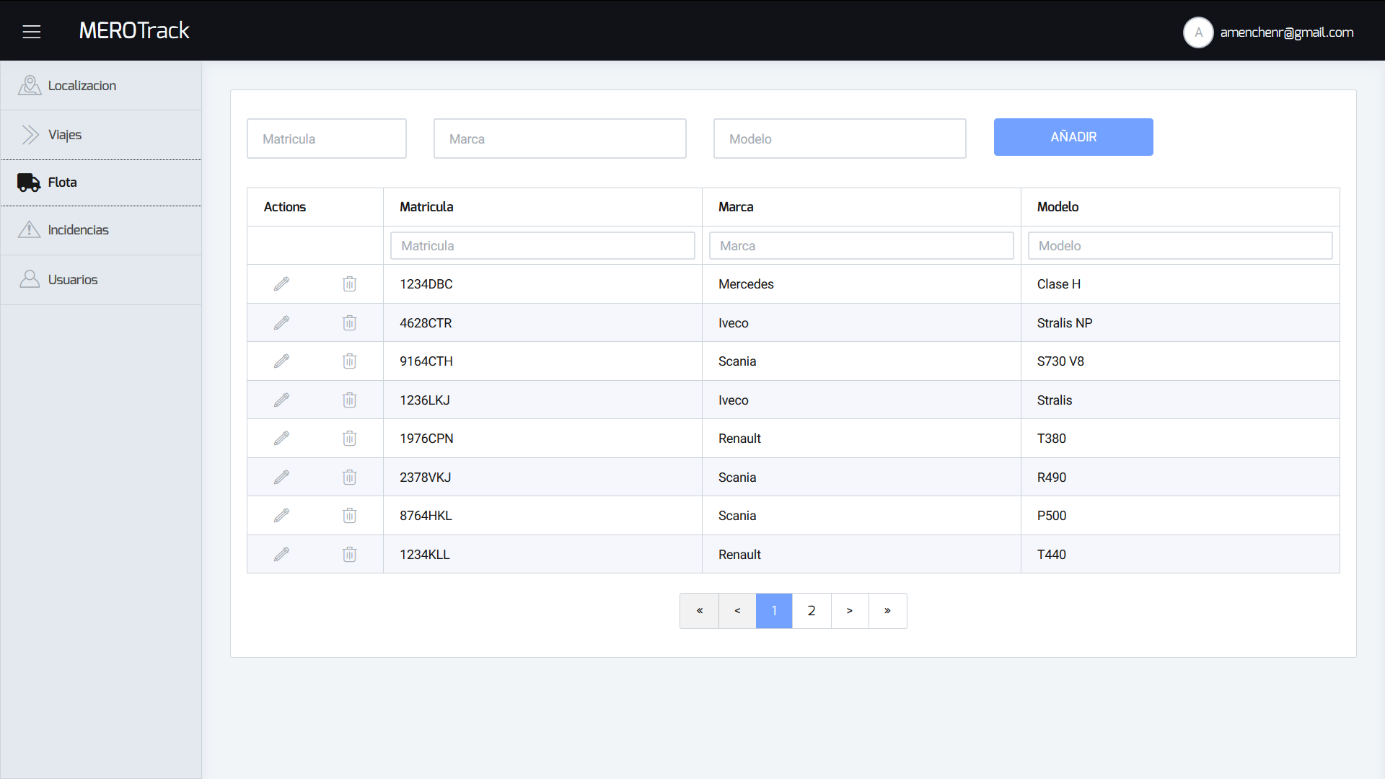


El panel de edición nos permite establecer la descripción del viaje, la fecha en la que se va a realizar, y el vehículo que lo va a realizar. La elección de vehículo es mediante un campo de búsqueda autocompletado.

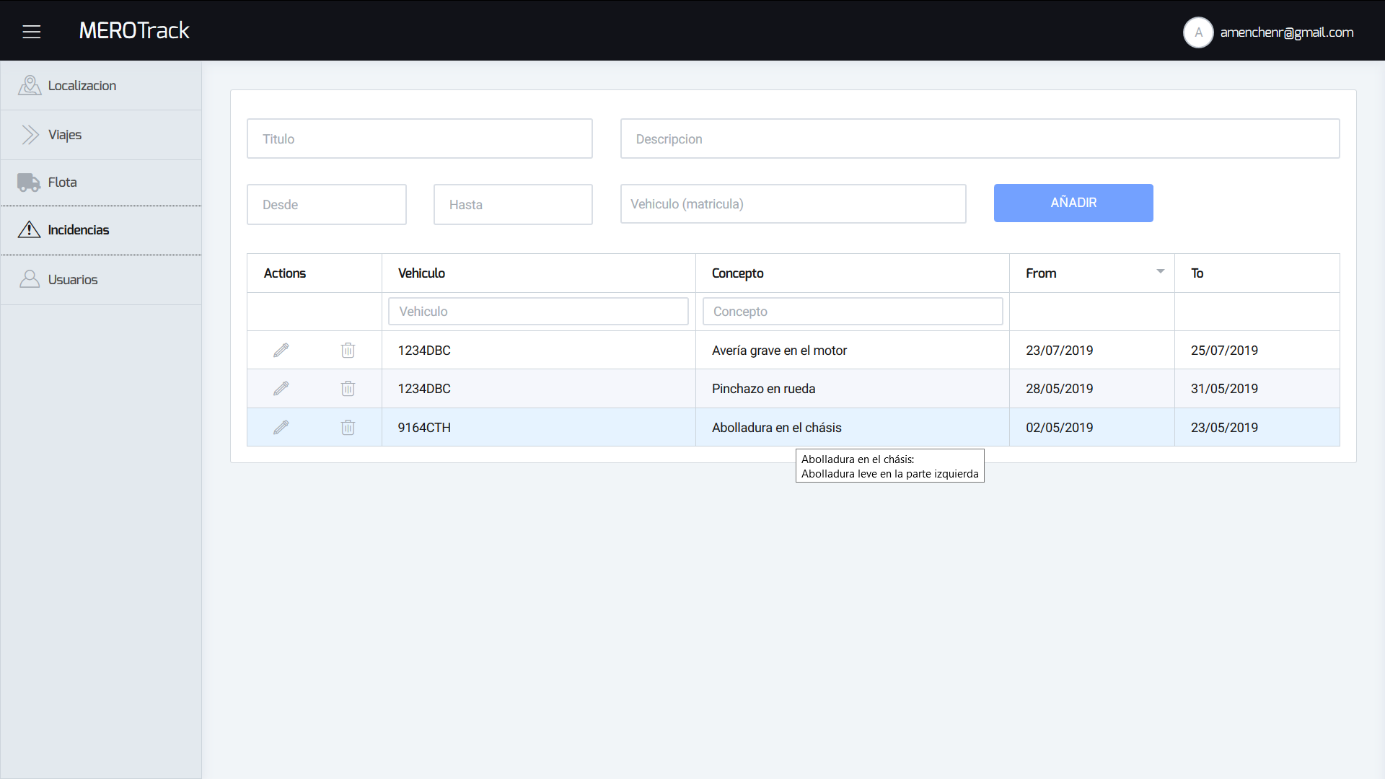
Podemos arrastrar los marcadores, hacer click en el mapa o buscar la dirección en el panel derecho del mapa. Una vez establecidos los puntos, nos genera la ruta más eficiente junto con el tiempo, kilometraje e instrucciones. El botón ‘optimizar ruta’ nos recalcula la ruta de la forma más óptima posible. Tras ello, podemos guardar el viaje.



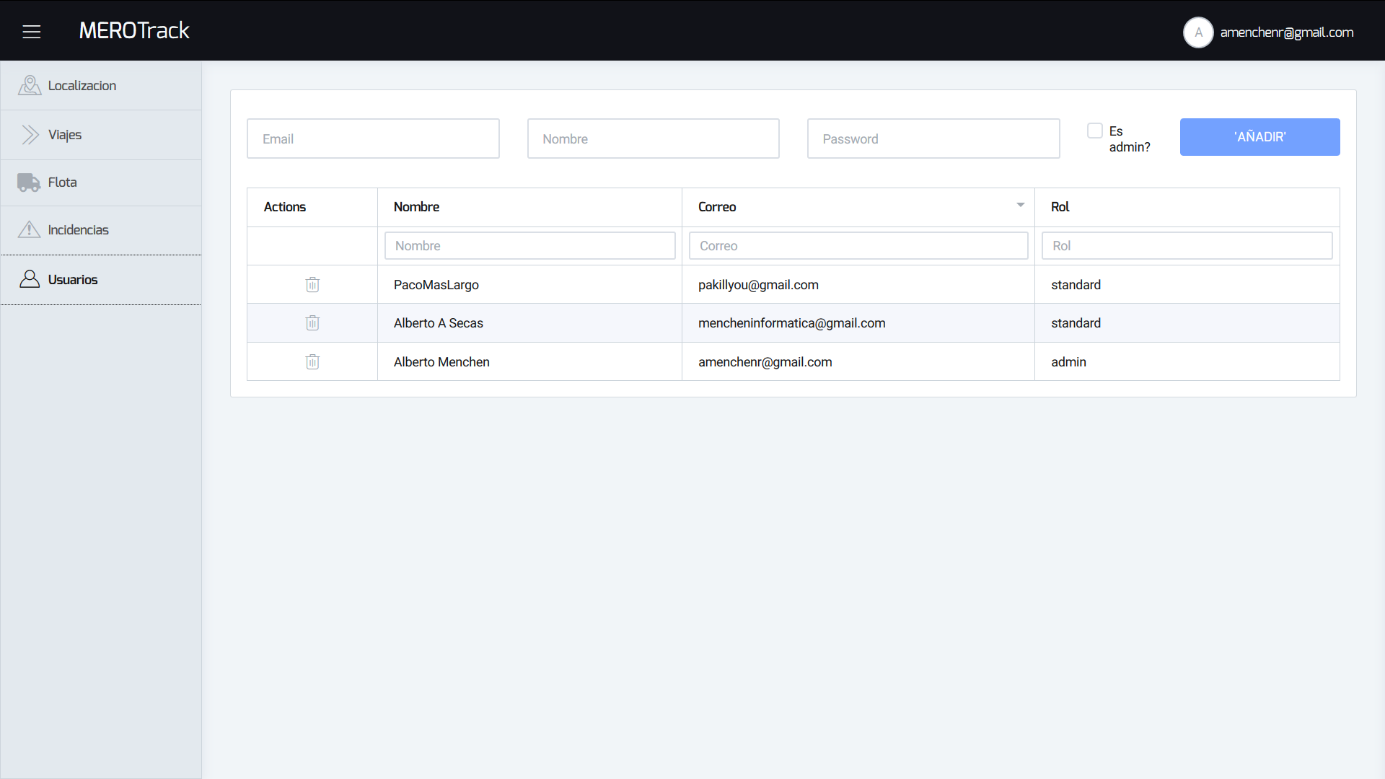
**FLOTA:** Permite visualizar, insertar, eliminar o editar vehículos. Cabe destacar que no se tiene acceso a los componentes públicos y privados del identificador usado por la Arduino para la geolocalización, algo a lo que solo tiene acceso el administrador de la BD.



**INCIDENCIAS:** Igual que la ventana anterior, permite editar, insertar eliminar y consultar las diversas incidencias (averías, retrasos, etc…) que puedan sucederle a un vehículo.



**Usuarios:** Este menú solo es accesible a usuarios con privilegio de administrador





## DESARROLLO DEL CÓDIGO E INTERACCIÓN CON LA BASE DE DATOS

### BACKEND

Lo primero a realizar será el backend, ya que es el pilar central de la aplicación, puesto que es lo que se va a comunicar con las Arduino y lo que va a proveer de datos a la aplicación frontal angular.

Es importante destacar que gracias a la nueva versión de Hibernate ORM utilizada por Spring, no es necesario crear la base de datos ‘a mano’ mediante SQL; lo primero a realizar pues, será crear los objetos del modelo de la aplicación en java, y anotarlos mediante notación JPA:



Nótese (y nunca mejor dicho), como las notaciones se encargan del mapeo. Notaciones como @Entity, @Table, @Column o @OneToMany, nos permite escribir directamente los nombres de columnas, tablas, o incluso las relaciones y posibles constraints que posteriormente se generarán de forma automática en la base de datos a la que conectemos nuestro backend.

Otras notaciones como @Data, @NoArgsConstructor o @EqualsAndHashCode no son notaciones JPA sino notaciones Lombok; las cual permiten entre otras cosas, generar constructores, getters, setters, toString, hashCode…. Sin tener que escribirlos a mano o que los genere el IDE, ahorrando en tiempo y ganando legibilidad en el código.

El siguiente paso será crear los repositorios. Spring data REST permite exponer rutas del servidor automáticamente, que serán las que nos permitan acceder a los datos del servidor. Tan solo tenemos que declarar un interfaz repositorio y anotarlo con @RepositoryRestResource.



Esto debería ser suficiente para exponer las rutas necesarias a los datos (en este caso, a la tabla de viajes, ‘Trip’), ya que se generarán automáticamente los controladores y servicios que se encargarán de procesar nuestras peticiones al servidor. Sin embargo, para operaciones específicas que requieran de una implementación algo especial, necesitaremos crear los controladores ‘manualmente’.

Esto ultimo lo he hecho por ejemplo, a la hora de crear vehículos (ya que hay que generar aleatoriamente un id público y un id privado, que se componen de un string aleatorio cada uno), a la hora de obtener las posiciones de un vehículo, ya que la Arduino envía los datos de una forma muy específica al servidor, y hay que parsearlo adecuadamente (aparte de comprobar las credenciales de seguridad), el login, o búsquedas específicas.

Aquí un ejemplo de un controller personalizado, en este caso de creación de vehículos:



Tras ello, queda muy poco para finalizar el backend. En el código se pueden ver ejemplos de ‘Specifications’, las cuales permiten definir criterios de búsqueda específicos a la hora de realizar consultas.

También hay que implementar los filtros de autenticación y autorización para securizar el backend. Aunque para facilitar el desarrollo fue de lo que último hice, para evitar problemas.

### ARDUINO

Lo siguiente a realizar fue el código que se encarga de la Arduino. En la página de DFRobot hay multitud de ejemplos de cómo usar la shield SIM908, que es la que realmente se encarga de realizar las tareas de conexión a red y a GPS.

El trabajo que realiza la Arduino es construir y enviar los llamados comandos AT a la shield. Por tanto, hay que programar la Arduino de tal forma que envíe mediante su conexión serial integrada dichos comandos al chip SIM908.

Algunos de estos comandos AT son:

* **AT+CPIN=”1234”:** Establece que el PIN de la SIM es 1234
* **AT+CGPSPWR=1:** Activa el sensor GPS
* **AT+HTTPPARA="URL","servidor.com/pos?lat=12&lon=25”:** 
  + Establece que la URL a acceder es servidor.com con los parámetros lat=12 y lon=25
* **AT+HTTPACTION=0:** Accede a la anterior URL mediante el método GET

La secuencia que realiza el programa de la Arduino es:

1. Configuración inicial de entrada y salida
2. Configuración inicial de red telefónica
3. Activar el sensor GPS
4. Espera alrededor de dos minutos (para dar tiempo al sensor a calibrar su posición)
5. Entra en bucle, cada 30 segundos envía una señal al servidor con su posición.

Por motivos de hardware (la memoria de la Arduino apenas deja 2 Mb libres para variables), no se ha podido implementar ninguna validación programática de control de errores en caso de problemas de conexión; si hay un error el módulo se reinicia totalmente.

### FRONTEND

Una vez montado el servidor, y con datos de prueba suficientes provistos por el módulo rastreador, lo siguiente que implementé fue el frontend de la aplicación, la página web en Angular.

Para lograr un diseño más agradable, me decidí por una biblioteca de componentes llamada Nebular, el cual provee componentes básicos (layout, inputs, datepickers…), con diseño basado en Bootstrap.

Tras instalar nebular (y su template ngx-admin) mediante el gestor de paquetes NPM, había que implementar objetos del modelo, los *Services* (para acceder a los datos del backend), y las distintas páginas de la web.

Lo siguiente es un ejemplo de objeto básico, servicio de datos, y servicio de tabla (para conectar los datos del backend, de forma paginada, con las distintas tablas de la web):



Cabe mencionar que cada una de las *rutas* de la web es un componente independiente, contenido en su propio módulo. Esto es, cada página está compuesto de un *Template* (el HTML) y un *Componente* (el código javascript que controla las acciones, eventos, formularios, y otras acciones de ese html), los módulos son una forma de hacer saber a Angular qué componentes cargar en qué momento (para evitar problemas de rendimiento y consumo excesivo de red)

A continuación se muestra un ejemplo de *Template*, nótese el uso de *directivas* de [entrada] y (salida), aunque también pueden ser [(bidireccionales)]; estas directivas son la forma de indicarle a Angular, directamente en el HTML, dónde encontrar el código que se encarga de procesar eventos o de contener datos.



Así mismo, algunas de estas directivas harían referencia al siguiente evento ‘onDelete()’, y a la propiedad ‘source’ establecida en el constructor.



## PRUEBAS DE SOFTWARE

### ARDUINO

Las pruebas realizadas han sido la realización de varios viajes (dentro de Ciudad Real, y Manzanares-Ciudad Real). Debido a las limitaciones de memoria de la Arduino y lo antiguo que es el hardware (la shield SIM908 es del 2013) no se han podido realizar comprobaciones muy exhaustivas; sin embargo, los resultados son lo suficientemente satisfactorios para la antigüedad y las limitaciones del hardware.

### BACKEND

No se han realizado pruebas ‘personalizadas’ con JUnit. Sin embargo, algunas pruebas de integración de Spring Boot (conexión a base de datos, inyección de Beans y dependencias) se realizan de forma automatizada al lanzar la aplicación, o se pueden ejecutar mediante Gradle, con el comando ‘gradle check’

### FRONTEND

Angular permite la automatización de test previo a la compilación (o mediante el comando ‘ng test’) mediante archivos spec.ts, sin embargo, debido a la importancia de la interacción del usuario, las pruebas realizadas no han sido unitarias, si no ‘manuales’, intentando buscar los casos extremos en los que la aplicación pueda romper.

## SEGURIDAD

Debido a la creciente preocupación en cuanto a seguridad digital, considero importante añadir este punto mencionando los sistemas de seguridad implementados en el proyecto. Toda la aplicación está protegida contra inyección SQL y XSS (Cross-Site scripting).

El acceso está controlado y regulado mediante el estándar JWT (Json WebToken), tanto a nivel de interfaz, protegiéndose el propio angular de accesos indebidos a rutas de la web mediante AuthGuard; como en el propio servidor, controlando que las peticiones al mismo provengan de un usuario autenticado. Las contraseñas se cifran mediante cifrado Blowfish/BCrypt.

Las Arduino siguen un protocolo propio de autenticación para prevenir fraude (envío de posiciones falsas, a fin de despistar) y comprobación en caso de error en la red y corrupción de datos:

Todos los vehículos en la base de datos tienen un doble Id: dos cadenas generadas aleatoriamente con letras y números de 16 caracteres de duración, uno público y otro privado. La Arduino, en la petición al servidor, incluye como parámetro su componente público, y cifra todos los datos (latitud, longitud, fecha y su id público) ADEMÁS de su componente privado (el cual solo conoce el servidor y la propia Arduino), mediante cifrado MD5, impidiendo de esta manera el fraude, al ser necesario conocer el componente privado del Id.