Trabajo de Fin de Máster

Aplicación Móvil para Optimizar el Transporte por Carretera

Mobile Application to Optimize Road Transport .

Jonay Zebensuí Herrera Santana

La Laguna, 9 de septiembre de 2019

Dª. **Pino Teresa Caballero Gil**, con DNI número 45.534.310-Z profesora Catedrática de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como directora

Dª. **Jezabel Molina Gil**, con DNI número 78.507.682-B profesora Ayudante Doctora adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como co-directora

C E R T I F I C A

Que la presente memoria titulada:

"*Aplicación Móvil para Optimizar el Transporte por Carretera.*"

ha sido realizada bajo su dirección por D. **Jonay Zebensuí Herrera Santana**, con DNI número 78.721.819-H.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 9 de septiembre de 2019

Licencia

Resumen

Este Trabajo Final de Máster tiene como objetivo identificar y controlar en tiempo real a conductores en vías públicas desde un servidor, llegando a sancionar por diferentes motivos según el tipo de conductor e infracción cometida. Por otro lado, también se pretende informar a los conductores si, dentro de un radio (en Km), existe algún conductor que no debería de estar en circulación por tener algún tipo de infracción vigente.

La información que se va acumulando en el servidor y que va creciendo en función del número de usuarios que hagan uso de la plataforma, puede llegar a ser muy interesante de cara a un análisis posterior para previsión de tráfico o puntos negros de concurrencia entre otros.

Palabras clave: DGT, Android Studio, GPS, BD, Node.js, GitHub, Sanción, Carnet

Abstract

This final master's project aims to identify and control in real time drivers on public roads from a server, reaching penalties for different reasons depending on the type of driver and infraction committed. On the other hand, it is also intended to inform drivers if, within a radius (in Km), there is a driver who should not be in circulation due to a serious offense in force.

The information that is accumulating on the server and that is growing depending on the number of users who make use of the platform, can become very interesting in the face of a subsequent analysis for traffic forecasting or black points of concurrence among others.

Keywords: *DGT, Android Studio, GPS, BD, Node.js, GitHub, Sanción, Carnet*

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Antecedentes

El grupo de Criptología de la Universidad de La Laguna CryptULL (vinculado al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna) se encuentra actualmente desarrollando diversos proyectos donde se aborda la Seguridad en Redes Inalámbricas, Diseño y Análisis de Protocolos Criptográficos, Aplicaciones Móviles Seguras, Criptoanálisis y Diseño de Cifrados en Flujo, y Métodos de Identificación Fuerte. Uno de ellos es el proyecto MOTAM (acrónimo que proviene de MOTUS que en latín significa MOVER) y que consiste en el desarrollo y la combinación de nuevos asistentes de conducción/aplicaciones con tecnologías de sensores/actuadores vehiculares para interactuar activamente con el entorno inteligente con el fin de incluir progresivamente conducción autónoma (piloto automático) en un número creciente de condiciones, pero que podrá ser utilizado desde el principio con infraestructuras viarias y vehículos no autónomos existentes.

El proyecto MOTAM combina información avanzada y Tecnologías de la Información y las Comunicaciones (TIC) para vehículos inteligentes y servicios de movilidad, incluyendo Sistemas Avanzados de Asistencia al Conductor o ADAS (Advanced Driver Assistance Systems), coches conectados y actuadores vehiculares para conducción autónoma, con sistemas básicos disponibles y componentes tales como navegación para automóviles y sensores. Por un lado, se aprovechará el alto poder de comunicación y cómputo de los teléfonos inteligentes con conectividad 4G para acelerar el despliegue a gran escala de aplicaciones vehiculares de asistencia al conductor. El entorno proveerá información veraz y confiable con la que apoyar las decisiones.

MOTAM demostrará la eficacia de la combinación de diferentes tecnologías para la interacción inteligente humano-vehículo-entorno que ayudará a aumentar la seguridad vial, y a reducir los atascos, la contaminación y el impacto del tráfico en el medioambiente. [14]

1.2. Estado actual del arte

En la actualidad existen aplicaciones móviles las cuales facilitan información al usuario acerca del estado de las carreteras, información sobre el estado del tráfico, obtención de mejores rutas hasta el destino deseado, acceso a las cámaras de tráfico en tiempo real, ubicación de radares, etc. El propósito de dichas aplicaciones es la de proveer al usuario de información fiable en tiempo real para una mejor toma de decisión de ruta lo mas óptima y segura para llegar al destino deseado en el menor tiempo posible y evitando posibles inconvenientes.

Actualmente existen herramientas que, a nivel empresarial, dotan de valor añadido a su actividad económica principal: dentro del sector de la logística y el transporte, conocer en cada momento dónde se encuentra cada activo de la empresa es una valiosa información, entendiendo como activo tanto a transportistas como a sus vehículos. De ésto se han percatado numerosas empresas y, actualmente, existen soluciones y herramientas para todo tipo de compañías de transporte. Ejemplos de ello pueden ser: Global M2M SIM [6], GPSWOX [9], FleetGO [4] o Transpoco [18].

Hasta ahora el enfoque de "asistencia al conductor" siempre ha venido marcado por el flujo de datos hacía el conductor pero nunca de él hacia el resto. Por el momento no existe ninguna aplicación para vehículos que controle en tiempo real (informando a la DGT) por donde se circula en cada momento, si se esta habilitado para circular, si se viola normas de tráfico (como exceder la velocidad por encima de lo permitido en la vía) y demás información que puede ser muy relevante para el resto de conductores. Por lo tanto, este trabajo fin de master presenta un nuevo enfoque para enriquecer y nutrir las herramientas que ya existen, de modo que cada vez sean más completas.

1.3. Objetivos y actividades a realizar

Este Trabajo Final de Máster se enmarca dentro de las líneas de investigación del proyecto MOTAM (1.1) cuyo propósito es el de diseñar y desarrollar un prototipo de aplicación móvil cliente-servidor cuyas funciones principales son la de reconocer al usuario del vehículo mediante huella dactilar y/o reconocimiento facial para posteriormente informar de su posición al servidor de la DGT **(Dirección General de Tráfico)**, monitorear su actividad, sancionar en caso que fuese necesario y mantener informados al resto de vehículos, delimitados por un radio de distancia, sobre conductores infractores. Todo esto en tiempo real.

Los objetivos propuestos para alcanzar en este Trabajo de Fin de Máster han sido los siguientes:

- Aprendizaje y manejo de la plataforma de desarrollo **Android Studio**.
- Aprendizaje y dominio de lenguajes y herramientas de desarrollo de aplicaciones WEB en el servidor: **Node-JS**, **NPM** y **Jade**.
- Estudio, conocimiento y dominio de servicios que ofrece la plataforma **Firebase** de Google, concretamente **Firebase Auth** y **Realtime Database**.
- Manejo y uso del control de version por excelencia **Github**.
- Aprendizaje y uso de API's de Google: ***Maps JavaScript API, Maps SDK for Android y Roads API***

Y las actividades relacionadas con estos objetivos son las que se detallan a continuación:

- Autenticación obligatoria mediante huella dactilar y/o reconocimiento facial del conductor cuando haga uso del vehículo.
- Una vez validado el usuario, enviar a la DGT la ubicación GPS en la que se encuentran cada 2 seg.
- La señal es recogida por el servidor y debe ser enviada a aquellos terminales móviles que se encuentren dentro de un cierto rango Ej: 10 Km. Este parámetro debe ser configurable.
- Si alguno de los conductores comete una infracción se indicará en el servidor de la DGT y conllevará un descuento de puntos y una sanción en caso de que se quede sin puntos. Aquí habrá que tener en cuenta los diferentes tipos de conductores y sus peculiaridades.
- Si tras hacer login, algún conductor no debería estar conduciendo porque se le ha retirado el carnet por consumir puntos, cambiar el tipo de señal que emite a una alerta y advertir a las autoridades.
- Diseñar y desarrollar el servidor de la DGT. En este se pueden visualizar todos los vehículos de forma anónima. Todos los puntos del mismo color sobre un mapa de google y si hay algún sancionado que esté circulando se podrá visualizar con otro icono y en este caso se pierde el anonimato.

Capítulo 2

Tecnologías usadas

2.1. Android Studio

Figura 2.1: Logotipo Android Studio

Android Studio es el IDE oficial para Android. En mayo de 2013 fue presentado en la conferencia de Google, y reemplazó a Eclipse como entorno de desarrollo integrado oficial para Android. La primera versión estable fue publicada en diciembre de 2014.

Se puede descargar para las plataformas Windows, MacOS y Linux. Su publicación para las plataformas Windows, MacOS y Linux ha sido gratuita mediante la Licencia Apache 2.0 y esta basado en el software de JetBrains IntelliJ IDEA. El propósito de su diseño esta pensado para el desarrollo de Android. [1]

2.2. Webstorm

Figura 2.2: Logotipo WebStorm

JetBrains WebStorm es otro entorno de desarrollo integrado (IDE) muy poderoso, pero en este caso para el lenguaje JavaScript tanto para el Front-End, como para el Back-End (con NodeJS).

Con WebStorm puedes desarrollar en JavaScript, HTML, CSS, etc, con el soporte necesario así como para marcos como Angular o React. Sistemas de control de versiones y varias herramientas (comentadas a continuación) pueden ser integradas en WebStorm. [19]

- Intelligent Editor con reconocimiento de código asistido mediante herramientas como el autocompletado de métodos, resaltado de sintaxis y coloreado y refactorizaciones (entre otros) se encuentra disponible para los siguientes lengujes: JavaScript, Node.js, TypeScript, ECMAScript 6, TypeScript, Dart y CoffeeScript, así como para HTML, CSS, Stylus, Less y Sass.
- Análisis de código *a*l vuelo, resaltado de errores y soluciones rápidas.
- Potente navegación en todo el proyecto y refactorizaciones avanzadas.
- Compatibilidad con marcos modernos: React, Angular, AngularJS, Vue.js, Express y más.
- Depurador incorporado para el código del lado del cliente y Node.js.
- Integración con herramientas de compilación (Grunt, Gulp), herramientas de calidad de código (JSHint, JSLint, ESLint, TSLint), corredores de prueba (Karma, Mocha, Jest, transportador) y VCS (Git, GitHub, Mercurial, SVN).

2.3. NodeJS, NPM y Jade

La relevancia que ha vuelto a adquirir Javascript en los últimos años ha sido gracias a los servidores con NodeJS, pasando de ser usado para códigos simples como validaciones de usuario en el Front-End a utilizarse para el desarrollo y puesta en funcionamiento de aplicaciones web mucho mas complejas.

Figura 2.3: Logotipos NodeJS, Jade y NPM

2.3.1. NodeJS

NodeJS es una librería y entorno escrito en Javascript y que es ejecutado en el propio servidor, se trata de un sistema basado en eventos y es **asíncrono**. Node ejecuta Javascript provisto del motor V8 implementado en Google Chrome el cual le permite proporcionar un entorno de ejecución en el lado del servidor con capacidades de compilar y ejecutar Javascript a velocidades muy altas. Esto se consigue porque dicho motor V8 no interpreta el Javascript sino que lo compila en código de máquina nativo. Node puede ser ejecutado en Windows, MacOS y/o Linux gracias a que es código abierto. Se ha comentado que NodeJS es asíncrono y es posible que se piense en la programación multi-hilo, es metodología consume demasiados recursos (un hilo por cliente), pero nada mas allá lejos de la realidad ya que Node esconde un poderoso sistema para procesar todas las operaciones intensivas de entrada/salida y se llama Bucle de Eventos capaz de gestionar todas estas operaciones de manera eficiente.

El objetivo primordial de Node es *l*a de proporcionar un forma sencilla y fácil de desarrollar programas de red escalables sin necesidad de tener que preocuparse por los problemas de rendimiento ya que cada conexión que se realiza hace que se genere un evento dentro del proceso motor de Node en lugar de que se genere por cada conexión un hilo del sistema operativo (con su correspondiente asignación de memoria). NodeJS no permite bloqueos ni se bloquea por llamadas de E/S por lo tanto nunca se quedará *c*olgado, además, un servidor de Node es capaz de soportar miles de conexiones concurrentes. [15]

2.3.2. NPM

NPM (node package manager) es el gestor de paquetes javascript de **NodeJS**. Haremos uso de esta poderosa herramienta para instalar cualquier librería que necesitemos para nuestro proyecto en cuestión de segundos. Es muy similar al uso de *a*pt-get install de Linux, ya con una simple línea de comandos instalaremos todas las dependencias que podamos necesitar. [16]

2.3.3. JADE

Jade es lo que se conoce como un lenguaje de plantillas, un binario de NodeJS y una forma rápida de generar contenido. Haciendo uso únicamente del nombre de las etiquetas HTML, Jade implementa una estructura semántica, jerárquica y ordenada teniendo solo que preocuparnos en generar contenido y no en tener que aprender un nuevo lenguaje.

El código se basa en etiquetas indentadas y no en las etiquetas HTML comunes en las que tienes que abrir y cerrar con *«/ »"* dando como resultado final una plantilla definida correctamente y con orden. [10]

2.4. Firebase

Figura 2.4: Logotipo Firebase

Firebase fundada en 2011 por James Tamplin y Andrew Lee y comprada por Google en 2014, es una plataforma que permite de forma muy rápida, desarrollar aplicaciones para móviles de muy alta calidad. Disponible tanto para Android, iOS como para WEB, su propósito es la de simplificar la gestión de la aplicación, el crecimiento de los usuarios y por lógica, su productividad monetaria. Desde que fue adquirida por Google ha sido mejorada con la compra del equipo de Divshot. Ofrece diferentes servicios, pero solo comentaremos aquellos que han sido utilizados en este proyecto.

2.4.1. Firebase Authentication

Figura 2.5: Logotipo Firebase Authentication

Una de las necesidades fundamentales de la mayoría de las app es la de tener la capacidad de poder identificar usuarios para poder guardar sus datos en la nube de forma segura proporcionando la misma experiencia individualizada en todos los dispositivos del usuario.

Firebase Authentication ofrece servicios back-end, SDK muy sencillos de usar y bibliotecas de interfaz de usuario ya elaboradas. Además permite autenticar usuarios mediante correo electrónico/contraseña, números móviles y proveedores como Google, Facebook, Twitter, GitHub y Play Juegos.

Su integración con servicios de Firebase es total y gracias a que aprovecha los estándares OAuth 2.0 y OpenID Connect se puede incorporar de forma sencilla al back-end.

Cuando un usuario accede a la app, primero se obtienen las credenciales pertinentes (en función del método de login usado), se transmiten al SDK de Firebase Auth, son verificadas en el servicio de back-end y finalmente se responde al usuario de nuevo.

Cuando el acceso es realizado con éxito se puede acceder a la información básica del perfil del usuario y controlar el acceso del usuario a los datos almacenados en otros productos de la gama Firebase. [2]

2.4.2. Firebase Realtime Database

Figura 2.6: Logotipo Firebase Realtime Database

Encargada de almacenar y sincronizar datos con nuestra base de datos NoSQL en la nube permitiendo mantener sincronizados en tiempo real todos los clientes manteniendo disponible la información incluso cuando la app no dispone de conexión.

Firebase Realtime Database como ya hemos comentado, es una base de datos NoSQL alojada en la nube. Los datos son sincronizados en tiempo real mediante eventos con cada cliente que se encuentra validado en la plataforma y se almacenan en formato JSON. Cuando se compila la app multiplataforma haciendo uso de los SDK tanto de iOS, Android y Javascript, todos y cada uno de los clientes obtienen y comparten una instancia de la Realtime Database y reciben actualizaciones de manera automática con los datos que se van incorporando.

Realtime Database garantiza que el acceso a la base de datos desde el código del cliente es seguro y persistente de manera que cuando se interrumpe la conexión, los eventos se siguen activando en tiempo real proporcionando así una experiencia adaptable al usuario final. Una vez se recupera la conectividad, se sincronizan los cambios locales con los remotos que sucedieron mientras se estaba sin conexión combinando los posibles conflictos de manera automática.

Las reglas de seguridad de Firebase Realtime Database están basadas en un lenguaje flexible de reglas mediante expresiones las cuales definen como se debe estructurar los datos y cuando se puede leer o escribir. Firebase Auth permite al desarrollador definir quién tiene acceso a qué datos y como se acceden a ellos.

Al tratarse de una base de datos NoSQL tiene diferentes optimizaciones y funcionalidades en comparación con una base de datos relacional. La API de Realtime Database está diseñada para permitir solo operaciones que se puedan ejecutar rápidamente por lo que esto permite crear una experiencia de tiempo real excelente capaz de servir a millones de usuarios sin que la capacidad

de respuesta se vea afectada. El como deben acceder los usuarios a los datos determina como deben estar estructurados los mismos. [3]

2.5. Github

Figura 2.7: Logotipo GitHub

GitHub GitHub es una plataforma de desarrollo colaborativo que permite, mediante el uso del sistema de control de versiones Git, alojar proyectos de software. Esta escrito en Ruby on Rails y aunque inicialmente era conocido como Logical Awesome LLC, desde principios de 2010 opera bajo el nombre de GitHub Inc. Si haces uso de una cuenta gratuita todos los proyectos que alojes en dicha plataforma, serán públicos, pero si te das de alta con una cuenta de pago puedes hospedarlos en modo privado. [5]

2.6. API de Google Maps

Figura 2.8: Logotipo Google Maps API

Una **API** (Interfaz de Programación de Aplicaciones), se puede definir como una forma estructurada y ordenada de comunicar o interactuar módulos de

diferentes software. Las API permite a los desarrolladores diseñar programas personalizados para ciertos sistemas operativos simplificando en gran medida su desarrollo al no tener que escribir códigos desde cero ya que permiten usar funcionalidades definidas que a su vez contribuyen a la interacción con otro sistema o programa.

El servidor de aplicaciones de mapas **Google Maps**, ofrece un amplio abanico de API's para poder interactuar con los servicios que ofrece. En el desarrollo de esta proyecto se han hecho uso fundamentalmente de 3 tipos:

- **Maps Javascript API:** te permite personalizar mapas con tu propio contenido y las imágenes para mostrar en páginas web y dispositivos móviles. Maps JavaScript API presenta cuatro tipos básicos de mapas (hoja de ruta, satélite, híbrido y terreno) que puede modificar utilizando capas y estilos, controles y eventos, y varios servicios y bibliotecas. Esta API ha sido utilizada en la parte de desarrollo Servidor Web. [12]
- **Maps SDK for Android:** mediante esta API puedes agregar mapas basados en datos de Google Maps a su aplicación. La API maneja automáticamente el acceso a los servidores de Google Maps, la descarga de datos, la visualización del mapa y la respuesta a los gestos del mapa. También puede usar llamadas de API para agregar marcadores, polígonos y superposiciones a un mapa básico y para cambiar la vista del usuario de un área de mapa en particular. Estos objetos proporcionan información adicional para las ubicaciones de los mapas y permiten la interacción del usuario con el mapa. Se ha hecho uso de esta API en la parte de desarrollo Android. [13]
- **Routes API:** La API Roads identifica las carreteras por las que viajaba un vehículo y proporciona datos, como los límites de velocidad. Esta API ha sido utilizada en la parte de desarrollo Servidor Web. [8]

Capítulo 3

Desarrollo del sistema

Hasta ahora se ha descrito el estado del arte actual y se ha definido el Trabajo de Fin de Máster, especificando los objetivos, actividades a desarrollar y las tecnologías empleadas para su desarrollo. A continuación, antes de empezar a describir el desarrollo del mismo, se expondrán los inconvenientes surgidos durante el proyecto.

3.1. Precedente

El objetivo principal del proyecto esta enfocado a la creación de una aplicación móvil y un servidor web que simula a la DGT. Dicha aplicación móvil no esta orientada para que el usuario se la descargue e instale en su terminal con el objetivo de su uso y disfrute, sino para que sea implantada de manera **obligatoria** en su sistema de navegación Android del vehículo. Obviamente debido a la complejidad que supone para un proyecto final de master adquirir un sistema de navegación Android, se ha realizado el desarrollo e implementación como si fuese para un terminal móvil, pero el propósito real es la de integrarlo en un vehículo.

Partiendo de los objetivos planteados y las actividades a realizar, a continuación se describirá como ha sido el desarrollo de las diferentes partes que componen el sistema:

3.2. Base de Datos

Es la responsable de almacenar y validar los usuarios por una parte y almacenar las ubicaciones de los mismos que se envían de manera continua por la otra. Se divide en 2 secciones fundamentales:

- Autenticación de usuario (Firebase Authentication): Base de datos que contiene las credenciales de todos los usuarios cuya validación es mediante usuario (correo electrónico) y contraseña. La estructura de campos que

provee Firebase-Auth es como la que se describe en el cuadro 3.1. No se permite agregar ningún campo, por lo tanto, si se necesita alguno adicional se debe hacer uso de la Realtime Database. En este caso ha sido necesaria la incorporación de una estructura JSON llamada ***D*atos** teniendo como claves los **uid** de cada usuario y dentro de cada uno de ellos los siguientes campos descritos en el cuadro 3.2.

Propiedad	**Tipo**	**Descripción**
uid	string	El **uid** que se asignará al usuario recién creado. Debe ser una string que contenga entre 1 y 128 caracteres. Si no se proporciona, se generará automáticamente un uid aleatorio.
email	string	El correo electrónico principal del usuario. Debe ser un correo electrónico válido.
emailVerified	booleano	Indica si se verificó el correo electrónico principal del usuario. Si no se proporciona, el valor predeterminado es **false**.
phoneNumber	string	El número de teléfono principal del usuario. Debe ser un número de teléfono válido que cumpla con las especificaciones E.164.
password	string	La contraseña del usuario, sin encriptación. Debe tener al menos seis caracteres.
displayName	string	El nombre visible del usuario.
photoURL	string	La URL de la foto del usuario.
disabled	booleano	Si el usuario está habilitado o no. **true** indica que está inhabilitado; **false** indica que está habilitado. Si no se proporciona, el valor predeterminado es **false**.

Tabla 3.1: Propiedades que admite Firebase Authentication

Propiedad	**Tipo**	**Descripción**
Puntos	int	Puntos de Carnet.
Radio	int	Rango expresado en Km que puede ver ese usuario.
Tipo	booleano	Tipo de conductor que pueden ser: Normal, Novel, Sancionado y de Transporte Público

Tabla 3.2: Campos adicionales en la Realtime Database

Figura 3.1: Estructura JSON de la tabla *a*ctualUsersLocations

- Almacenamiento en tiempo real de las ubicaciones de cada usuario Realtime Database (RT-DB): Inicialmente la única clave que existe en la RT-DB es la nombrada en el punto anterior, ***D*atos**. Una vez que los usuarios se conectan empiezan a enviar sus ubicaciones a 2 claves al mismo tiempo y se va construyendo un árbol de ubicaciones dinámicamente como se explica a continuación:

  - *a*ctualUsersLocations: Tabla donde se almacena la ubicación actual y que se va sobrescribiendo cada vez que los usuarios actualizan sus localizaciones. Sigue una estructura como la que se describe a continuación: *actualUsersLocations->uid->[latitud, longitud,timeStamp]*. Un ejemplo de esto lo tenemos en la figura 3.1
  - *l*ocationUsers: Tabla donde se almacena el histórico de ubicaciones de cada usuario con la siguiente estructura: *locationUsers->uid->timeStamp->[latitud, longitud, timeStringStamp]*. Para una mejor comprensión véase la figura 3.2.

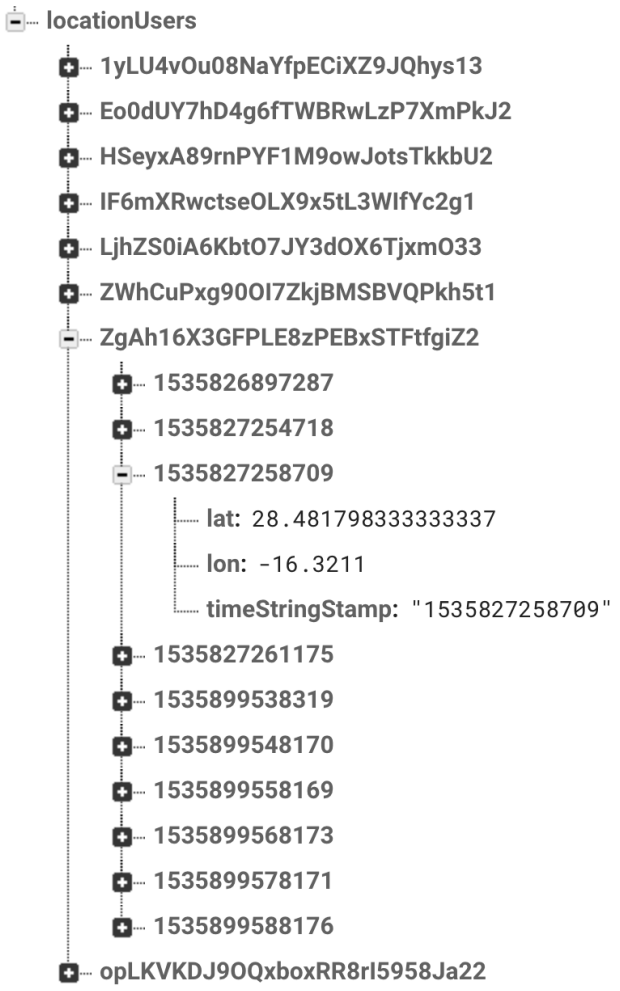

Figura 3.2: Estructura JSON de la tabla *l*ocationUsers

Tanto *locationUsers* como *actualUsersLocations* son creados por los usuarios desde la aplicación Android. Al mismo tiempo que se van incorporando localizaciones a *actualUsersLocations*, el servidor web crea dinámicamente otra nueva clave *myLocations* que tiene la estructura definida de la siguiente forma: *myLocations->uid(del usuario logueado)->uid(del usuario que se encuentre dentro de su rango)->[latitud, longitud,timeStamp]*. Véase la figura 3.3.

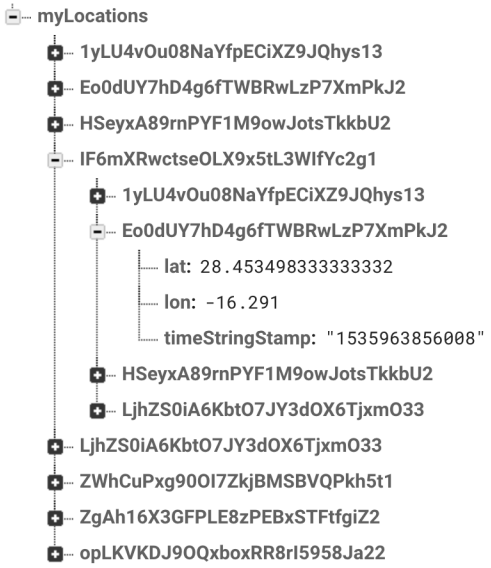

Figura 3.3: Estructura JSON de la tabla *m*yLocations

3.3. Aplicación Cliente

La aplicación del cliente se utilizará para proporcionar al usuario una interfaz donde visualizar el resto de vehículos ubicados en un mapa, datos del propio usuario como su nombre de usuario, radio de visualización, puntos, longitud y latitud y tipo de usuario. Además proporciona su ubicación en tiempo real a los servidores de la DGT. Desarrollada en Android Studio, "Motam_SDriver" parte del supuesto que el usuario debe ejecutarla e iniciar sesión para que el vehículo pueda ser arrancado, de lo contrario no debería permitir la ignición del mismo. Obviamente esta parte no se ha tenido en cuenta en el desarrollo ya que son necesarios elementos externos al mismo como por ejemplo los propios de la centralita y demás interconexionado del vehículo. Una vez ejecutada la aplicación nos encontramos con la pantalla de validación como se aprecia en la siguiente figura:

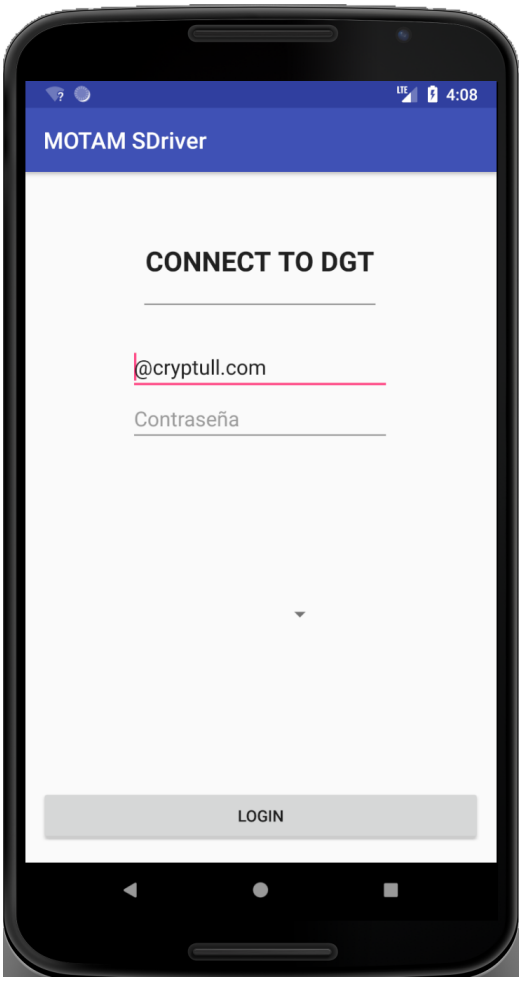

Figura 3.4: Captura de pantalla de inicio/validación

Para el prototipo del proyecto se ha añadido un menú desplegable que autorellenará por nosotros con los posibles usuarios desde "test001@cryptull.com"

hasta "test009@cryptull.com" tal y como se puede ver en la siguiente figura doble:

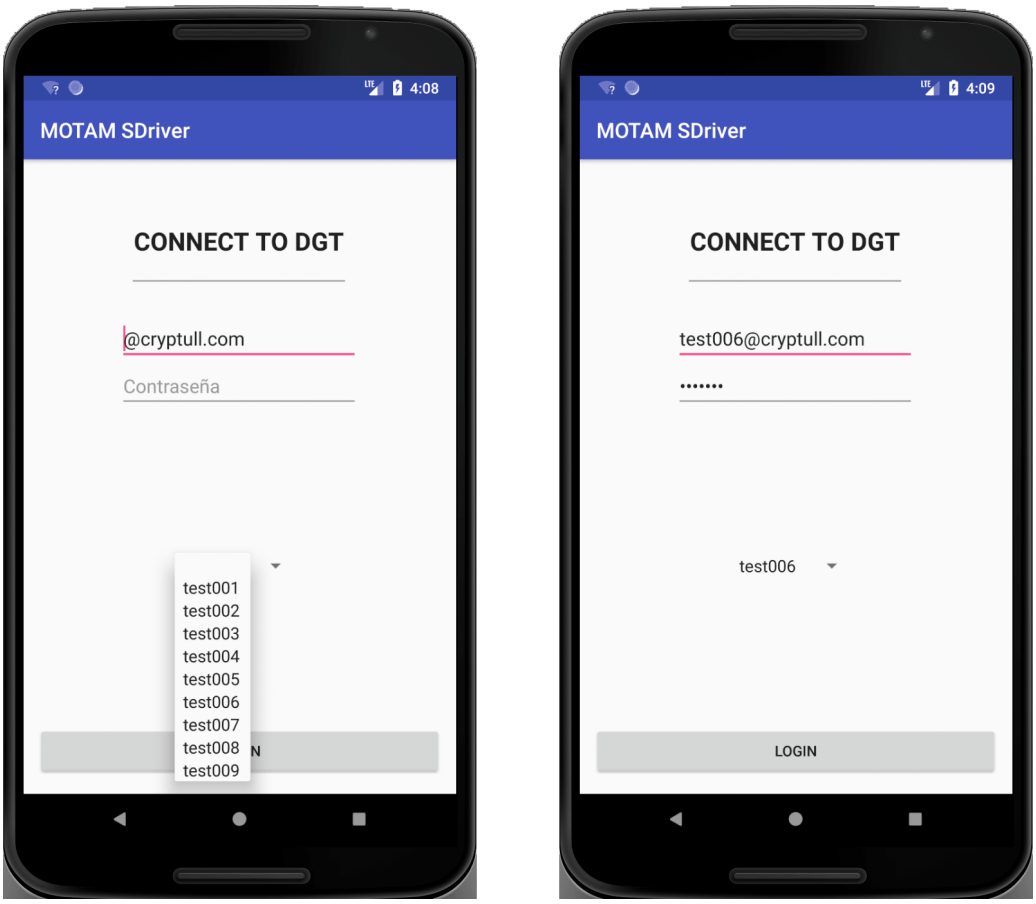

Figura 3.5: Capturas de inicio/validación y despliegue de usuarios de prueba

El 'layout' de la pantalla de inicio se define en el fichero 'activity_main.xml' y se compone de textos, un botón y una lista por lo que se omite el código del mismo al ser bastante sencillo, en cambio, si se mostrará el código java que gestiona esta parte de la aplicación y que se encuentra definido en 'LoginActivity.java' en la figura 3.6. Cuando se ejecuta la aplicación se definen los campos de usuario y contraseña, se crea un array con los usuarios 'test' y el código necesario para que, una vez seleccionado el usuario test, se rellenen los campos del login automáticamente. También se define la acción del botón 'Login' que consiste en validar los datos del formulario con ayuda de la función 'validateForm()' la cual comprueba que no están vacíos y que tienen el formato que deberían. Si los datos son correctos se procede a la validación con la 'FirebaseAuth' y en caso exitoso, nos enviará directamente al "main2Activity.java".

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mEmailField = (EditText) findViewById(R.id.email);
    mPasswordField = (EditText) findViewById(R.id.pass);

    Spinner spinner = (Spinner) findViewById(R.id.spinner);
    String[] users = {"","test001","test002","test003","test004","test005","test006","test007","test008","test009"};
    spinner.setAdapter(new ArrayAdapter<String>( context: this, android.R.layout.simple_spinner_item, users));

    spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {

        @Override
        public void onItemSelected(AdapterView<?> adapterView, View view, int pos, long id)
        {
            String user = (String) adapterView.getItemAtPosition(pos);
            mEmailField.setText(user + "@cryptull.com");
            mPasswordField.setText(user);
        }

        @Override
        public void onNothingSelected(AdapterView<?> parent)
        {    }
    });

    mAuth = FirebaseAuth.getInstance();
    mLoginBtn = (Button) findViewById(R.id.login);

    mAuthListener = (AuthStateListener) (firebaseAuth) → {

            if(mAuth.getCurrentUser()!=null){

                Comunicador.setObjeto(mAuth);
                Intent intent = new Intent( packageContext: LoginActivity.this, Main2Activity.class);
                startActivity(intent);
                finish();
                firsTime = true;
            }
            else {
                if (!firsTime) Toast.makeText( context: LoginActivity.this,  text: "Datos Incorrectos", Toast.LENGTH_SHORT).show();
            }
    };

    mLoginBtn.setOnClickListener((v) → {

            if (!validateForm()) {
                return;
            }

            LoginUser();
            firsTime = false;
    });
}
```

Figura 3.6: Método On Create()

Llegados a este punto el usuario esta validado y la app nos ha dirigido al siguiente layout ‘content_main2.xml’ cuya apariencia se puede ver en la figura 3.7.

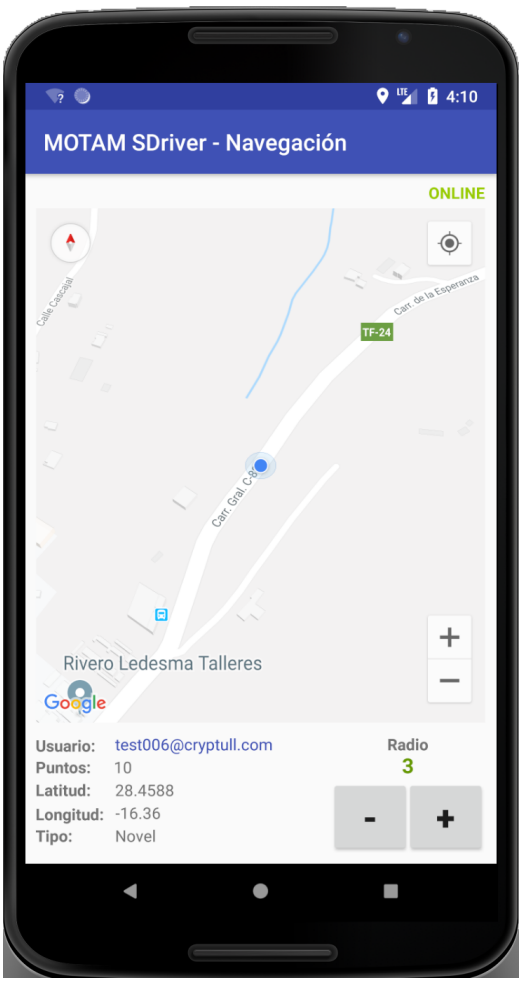

Figura 3.7: Captura de pantalla de navegación y datos de usuario

Los datos que se pueden apreciar en la captura son los siguientes:

- Mapa web: Centrado en la ubicación actual del vehículo con posibilidad de desplazarte por él, regresar a tu ubicación actual y realizar zoom in y zoom out.
- Datos de usuario: En la parte inferior izquierda podremos ver tanto el usuario, como los puntos de carnet, latitud, longitud y el tipo de conductor.
- Radio de visión: El usuario puede ver en su pantalla de navegación más vehículos que se encuentren dentro de dicho radio (en Km) pudiendo modificarlo solo si tiene 10 o mas puntos de carnet hasta un máximo de 50 Km.

En el punto anterior se ha comentado que la aplicación nos ha dirigido al layout 'content_main2.xml' cuyo contenido se gestiona desde 'Main2Activity.java' con ayuda de dos plantillas 'Comunicador.java' (Figura 3.8) encargada de transmitir el objeto del usuario 'mAuth' entre 'activities' y 'Ubicacion.java' (Figura

3.9), estructura con la que se envía a la base de datos las coordenadas obtenidas en cada momento.

```
class Comunicador {
    private static Object objeto = null;

    public static void setObjeto(Object newObjeto) { objeto = newObjeto; }

    public static Object getObjeto() {
        return objeto;
    }
}
```

Figura 3.8: Plantilla Comunicador.java

```
public class Ubicacion {

    public double lat;
    public double lon;
    private long timeStamp;

    public Ubicacion() {

    }

    public Ubicacion(double lat, double lon, long timeStamp) {
        this.lat = lat;
        this.lon = lon;
        this.timeStamp = timeStamp;
    }


    public void setLat(double lat) {
        this.lat = lat;
    }

    public void setLon(double lon) { this.lon = lon; }

    public void setTimestamp(long timestamp) {
        this.timeStamp = timestamp;
    }

    public String getTimeStringStamp() {

        return String.valueOf(timeStamp);
    }

    public double getLat() {
        return lat;
    }

    public double getLon() {
        return lon;
    }

}
```

Figura 3.9: Plantilla Ubicacion.java

En lo que respecta al 'Main2Activity.java', se comentará el código relacionado

con los agentes de escucha de la Firebase Realtime Database por ser lo mas relevante. Las variables que apuntan a las diferentes tablas son las siguientes:

- data: Datos de los campos adicionales del usuario autenticado.
- lUsers: Referencia a la clave “locationUsers”.
- aUsersl: Referencia a la clave “actualUsersLocastions”.
- myLocs: Referencia a la clave “myLocations”.

En las siguientes figuras veremos los agentes de escucha de ‘data’ (3.10), ‘myLocs’ (3.11) y también el código que detecta el cambio de ubicación GPS (3.12) :

```
156         ChildEventListener childEventListenerData = new ChildEventListener() {
157             @Override
158             public void onChildAdded(@NonNull DataSnapshot dataSnapshot, @Nullable String s) {
159
160
161                 switch (dataSnapshot.getKey()) {
162                     case "Puntos":  textPuntos.setText(dataSnapshot.getValue().toString());
163                         break;
164                     case "Radio":  textRadio.setText(dataSnapshot.getValue().toString());
165                         break;
166                     case "Tipo":  textTipo.setText(dataSnapshot.getValue().toString());
167                         break;
168                 }
169             }
170
171             @Override
172             public void onChildChanged(@NonNull DataSnapshot dataSnapshot, @Nullable String s) {
173
174                 switch (dataSnapshot.getKey()) {
175                     case "Puntos":  textPuntos.setText(dataSnapshot.getValue().toString());
176                         break;
177                     case "Radio":  textRadio.setText(dataSnapshot.getValue().toString());
178                         break;
179                 }
180             }
```

Figura 3.10: Agentes de escucha de ‘data’

- onChildAdded: Carga los datos del usuario a los campos del layout.
- onChildChanged: Actualiza los datos del usuario en el layout.

```
207         @Override
208         public void onChildAdded(DataSnapshot dataSnapshot, String previousChildName) {
209
210
211             Ubicacion data = dataSnapshot.getValue(Ubicacion.class);
212
213
214             if (mAuth.getUid() != dataSnapshot.getKey()){
215
216                 Marker mark = gMap.addMarker(new MarkerOptions()
217                         .position(new LatLng(data.getLat(),data.getLon()))
218                         .title(dataSnapshot.getKey())
219                         .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_YELLOW)));
220
221                 marcadores.put(dataSnapshot.getKey(),mark);
222             }
223
224         }
225
226
227         @Override
228         public void onChildChanged(DataSnapshot dataSnapshot, String previousChildName) {
229             if (mAuth.getUid() != dataSnapshot.getKey()){
230                 Ubicacion data = dataSnapshot.getValue(Ubicacion.class);
231                 marcadores.get(dataSnapshot.getKey()).setPosition(new LatLng(data.getLat(),data.getLon()));
232             }
233
234         }
235
236         @Override
237         public void onChildRemoved(DataSnapshot dataSnapshot) {
238
239             Ubicacion data = dataSnapshot.getValue(Ubicacion.class);
240
241             marcadores.get(dataSnapshot.getKey()).remove();
242             marcadores.remove(dataSnapshot.getKey());
243
244         }
```

Figura 3.11: Agentes de escucha de 'myLocs'

- onChildAdded: Añade al mapa con un icono amarillo, todos los usuarios que se encuentren dentro del radio permitido.
- onChildChanged: Actualiza la posición en el mapa del usuario en cuestión que acaba de enviar su nueva ubicación.
- onChildRemoved: Usuario eliminado, también se elimina del mapa.

```
290     //Cada vez que hay un cambio de ubicación GPS...
291     @Override
292     public void onLocationChanged(Location location) {
293
294         if (mAuth != null){
295             NumberFormat nf = NumberFormat.getInstance();
296             nf.setMaximumFractionDigits(4);
297
298
299             ubi.setLat(location.getLatitude());
300             ubi.setLon(location.getLongitude());
301             ubi.setTimestamp(System.currentTimeMillis());
302
303             LatLng currentPosition = new LatLng(ubi.getLat(), ubi.getLon());
304             gMap.animateCamera(CameraUpdateFactory.newLatLng(currentPosition));
305
306
307             textLat.setText(String.valueOf(nf.format(location.getLatitude())));
308             textLon.setText(String.valueOf(nf.format(location.getLongitude())));
309
310
311             //Se agrega la ubicacion y tiempo actual a la tabla "locationUsers" creando un histórico
312             locationUsers.child(mAuth.getUid()).child(ubi.getTimeStringStamp()).setValue(ubi);
313
314             //Se agrega la ubicación y tiempo actual a la tabla actualUserLocations y se va sobreescribiendo
315             aUsersl.child(mAuth.getUid()).setValue(ubi);
316         }
317     }
```

Figura 3.12: Detección de cambios de ubicación GPS

Cada vez que existe un cambio de localización se obtienen las nuevas coordenadas mediante 'LocationListener', se mueve el centro del mapa a la nueva coordenada, se envían a 'lUsers' para el histórico y cálculo de velocidad y también a 'aUsersl'.

3.4. Aplicación Servidor

La parte del servidor de la DGT ha sido desarrollada en nodeJS y se divide en dos funcionalidades principales Servidor DGT y Servidor WEB:

3.4.1. Servidor DGT (index.js)

El Servidor DGT será el responsable de recopilar toda la información recibida desde los dispositivos y enviará a cada uno de ellos aquella información que sea relevante respecto a los vehículos que los rodean dentro de su radio seleccionado. Este control de la actividad de los usuarios es realizada mediante la detección de eventos (On child_added, On child_changed y On child_removed) que proporciona Firebase Realtime Database de la tabla *actUserLocations* y se encarga de crear y mantener la tabla *myLocations*. Además de actuar en función de lo que ocurra en la BD, también existe una función periódica (IDLE) cuyo propósito es el de detectar usuarios inactivos por que han dejado de transmitir posición, entendiéndose que se han quedado sin cobertura, se han desconectado o simplemente han estacionado y han apagado el vehículo:

- On child_added: Cuando un usuario envía por primera vez su ubicación se comparan las distancias entre todos los usuarios activos y si es igual o menor que la permitida por el campo radio, se añade como subclave con la ubicación actual. Código en la figura 3.13.
- On child_changed: Cuando un usuario actualiza su ubicación sucede lo mismo que cuando inician sesión por primera vez y además también se añaden o eliminan subclaves de *myLocations* si las distancias entre vehículos empiezan o dejan de cumplir (respectivamente) con el radio especificado. Código en la figura 3.14.
- On child_removed: Cuando un usuario es eliminado por inactividad, esta función elimina todas las subclaves de *myLocations* en las que aparezca el usuario eliminado. Código en la figura 3.15.
- IDLE(x,y): Función repetitiva que controla los usuarios que dejan de transmitir. La "X" define el intervalo de tiempo en seg. entre ejecuciones mientras que la "Y" hace referencia al tiempo en segundos de inactividad máxima permitida. Código en la figura 3.16.

```
actUserLoc.on("child_added", function (snapshot) {

    console.log("Nuevo vehiculo!  --> " + snapshot.key);

    actUserLoc.once("value").then(function(snap) {

        let total = 0;
        let todos = new Array();

        todos = snap.val();
        total = snap.numChildren();

    /*
        Analizamos las distancias con el resto de vehiculos (si existen) y se van agregando a #myLocations
    */

        Object.keys(todos).forEach(function(key){

            console.log(time+ "snapshot: " + snapshot.key + "--------**--------- Key: " + key);

            if(snapshot.key != key ){
                let dist = parseInt(getKilometros(todos[key].lat,todos[key].lon,snapshot.val().lat,snapshot.val().lon));

                console.log("Distancia: " + dist);

                //Si el vehiculo A cumple con el radio lo agregamos
                if(dist <= usersData[snapshot.key].Radio){

                    let values = {lat:todos[key].lat,lon:todos[key].lon,timeStringStamp:todos[key].timeStringStamp}
                    myLocs.child(snapshot.key).child(key).set(values);

                }

                //Si el vehiculo B cumple con el radio se agrega tb
                if(dist <= usersData[key].Radio) {

                    myLocs.child(key).child(snapshot.key).set(snapshot.val());

                }
            }
        });
    });
});

```

Figura 3.13: Detección de evento “On child added” de *actUserLocations*

```
actUserLoc.on("child_changed", function(snapshot) {

        console.log("-->   actUserLoc on Changed!! ");

        let total = 0;
        let todos = new Array();

        actUserLoc.once("value", function(snap) {

            todos = snap.val();
            total = snap.numChildren();

            Object.keys(todos).forEach(function(key){

                console.log("snapshot: " + snapshot.key + " - Key: " + key);

                if(snapshot.key != key ){

                    let dist = getKilometros(todos[key].lat,todos[key].lon,snapshot.val().lat,snapshot.val().lon);

                    console.log("Distancia:  " + dist);

                    if(dist <= usersData[snapshot.key].Radio){

                        let values = {lat:todos[key].lat,lon:todos[key].lon,timeStringStamp:todos[key].timeStringStamp}
                        myLocs.child(snapshot.key).child(key).set(values);

                    }else{

                        myLocs.child(snapshot.key).child(key).set(null);

                        //console.log("actUserLoc => Eliminado: " + snapshot.key);
                    }


                    if(dist <= usersData[key].Radio){

                        myLocs.child(key).child(snapshot.key).set(snapshot.val());

                    }else{

                        myLocs.child(key).child(snapshot.key).set(null);

                    }
                }else {
                    //console.log("No se procesa porque es el mismo");
                }
            });
        });
});
```

Figura 3.14: Detección de evento “On child changed” de *actUserLocations*

```
actUserLoc.on("child_removed", function(snapshot) {

    myLocs.once("value", function(snap){

        let total = 0;
        let todos = new Array();

        todos = snap.val();
        total = snap.numChildren();


        console.log(" * TOTAL * :" + total);

        if (total > 0){

            Object.keys(todos).forEach(function(key){


                console.log(" * Usuario eliminado de actUserLoc * ");
                console.log(" ** " + myLocs.child(key) + " ** " + snapshot.key);

                myLocs.child(key).child(snapshot.key).set(null);

            });
        }
    });
});
```

Figura 3.15: Detección de evento “On child removed” de *actUserLocations*

```
function idle(intervalo,inactividad){

    setInterval(function(){

        actUserLoc.once("value").then(function(snap) {

            let total = 0;
            let todos = new Array();

            todos = snap.val();
            total = snap.numChildren();

        /*
            Analizamos las distancias con el resto de vehiculos (si existen) y se van agregando a myLocations
        */

            if (total > 0){

                Object.keys(todos).forEach(function(key){

                    var d = new Date();

                    let diferencia = (d.getTime() - todos[key].timeStringStamp );

                    console.log("Usuario: " + key + "  inactividad: " + diferencia );

                    if (diferencia > inactividad*1000){

                        console.log(key + " Inactivo!");
                        myLocs.child(key).set(null);
                        actUserLoc.child(key).set(null);

                    }else{

                        console.log(key + " Activo!");

                    }

                });
            }


        });

    },intervalo*1000,"JavaScript");
}
```

Figura 3.16: Función IDLE(x,y)

3.4.2. Servidor WEB (index.js, main.js y layout.jade)

Ésta será la herramienta principal de gestión y monitorización de todos los vehículos conectados a la plataforma en tiempo real. Esta diseñado y enfocado para el administrador de la DGT con el fin de poder obtener información real sobre la situación del tráfico. Se visualizará en un mapa la información sensible de cada vehículo, como por ej: UID, nombre de usuario, tipo de conductor, velocidad actual y modificar tanto el radio de visualización del vehículo como los puntos asociados al conductor.

El código fuente implementa la integración de los vehículos existentes en *actUserLocations* dentro del mapa que nos provee la API de Google Maps. Será posible la obtención de información en tiempo real de los vehículos en circulación (como velocidad, puntos, tipo de conductor y radio de visibilidad, etc.) pudiéndose gestionar los puntos y el radio de cada vehículo independientemente. Haciendo uso de los framework express.JS y socket.IO se ejecuta un servidor web (con motor Jade para el contenido y plantilla) el cual mostrará en tiempo real la ubicación de todos los vehículos sobre el mapa. En la figura 3.17 se muestran las declaraciones de lo comentado anteriormente y el llamamiento al método "listen" el cual ejecuta el servidor web. Con estas herramientas podemos tener control de la base de datos y de la comunicación entre "server-side" y "client-side" mediante sockets.

```
var express = require('express'),
    path = require('path'),
    jade = require('jade');

var app = new express();
var http = require('http').Server(app);
var io = require('socket.io')(http);

app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');
app.use(express.static(path.join(__dirname, 'public')));
app.get('/',function(req,res){
  res.render('layout', { title: 'Realtime DGT Server', subtitle: 'TFM CryptULL' });
});



var admin = require('firebase-admin');
var serviceAccount = require("./tfm-motamsdriver-303b6-firebase-adminsdk-rx7vn-3574b10a63.json");


admin.initializeApp({
    credential: admin.credential.cert(serviceAccount),
    databaseURL: "https://tfm-motamsdriver-303b6.firebaseio.com/"
});


.
.
.

http.listen(8000, function(){
    console.log('listening on *:8000');
});
```

Figura 3.17: Declaraciones de variables que gestionan BD y contenido WEB

En los siguientes puntos se describirá el código "server-side" y "client-side":

3.4.2.1. Server-side (index.js)

Se vuelve a hacer uso de los métodos de detección de eventos de Firebase pero en esta ocasión el objetivo es enviar información al "client-side" para representarla sobre el mapa y recibir posibles cambios de puntos y/o radio los cuales deben de ser actualizados en la BD. En la figura 3.18 se puede apreciar un resumen del contenido que gestiona la información entre servidor-cliente. A continuación se comentará brevemente la funcionalidad de cada método:

```
227 //Usuario de la DGT se conecta a la WEB
228 io.on('connection', function(socket){
229
230     console.log('Usuario conectado');
231
232     dbdatos.once("value", function(data) {
233         usersData = data.val();
234     });
235
236     socket.on('disconnect', function(){ ...
246
247     });
248
249     socket.on('update', function(uid,tipo,num){ ...
264
265     });
266
267     //Cuando hay cambios en datos de los usuarios
268
269     dbdatos.on("child_changed", function(snapshot) { ...
281
282     });
283
284     //Nuevo vehiculo que envia su localizacion
285
286     actUserLoc.on("child_added", function (snapshot) { ...
302
303     });
304
305     //Vehículo actualiza sus datos de localización
306
307     actUserLoc.on("child_changed", function(snapshot) { ...
397
398     });
399
400     //Vehículo se elimina por no enviar ubicación en X tiempo tambien se debe de borrar en el mapa
401
402     actUserLoc.on("child_removed", function(snapshot) { ...
416
417     });
418
```

Figura 3.18: Gestión de información en comunicación cliente-servidor

- socket.on 'disconnect' : Cuando un usuario se desconecta de la web es muy importante desvincular los agentes de escucha para evitar la duplicidad de código en tiempo real.
- socket.on 'update' : Desde el "client-side" se puede recibir acciones de actualización de la BD como el "radio" o los "puntos" de cualquier vehículo.
- dbdatos.on 'child_changed' : Si existiera algún cambio en la BD producida por el cliente o la parte Servidor DGT se enviaría al mapa para que fuese actualizada.
- actUserLoc.on 'child_added' : Todos los vehículos se envían al mapa cuando se carga por primera vez para que sean representados sobre él.

- actUserLoc.on 'child_changed' : Envía al "client-side" las actualizaciones de las posiciones de cada vehículo.
- actUserLoc.on 'child_removed' : Al eliminarse un vehículo también se debe de borrar del mapa por lo tanto se le envía el identificador para que se elimine el marcador correspondiente.

3.4.2.2. Client-side (main.js y layout.jade)

Con un diseño minimalista y simple a la par que intuitivo se divide en 5 partes, 3 de ellas describen la apariencia web y las otras 2 el código de la misma:

- Cabecera: Logo de la ULL y título del servidor web.
- Mapa web: Centrado inicialmente en unas coordenadas determinadas, según cargue la página solicita permiso al usuario para centrarla en su ubicación actual. Los vehículos vienen representados con un icono de color negro acompañado de la velocidad actual justo encima del mismo si tienen 1 o mas puntos y de color rojo si no tienen puntos. Véase un ejemplo con vehículos simulados en la siguientes figuras 3.19 y 3.26.

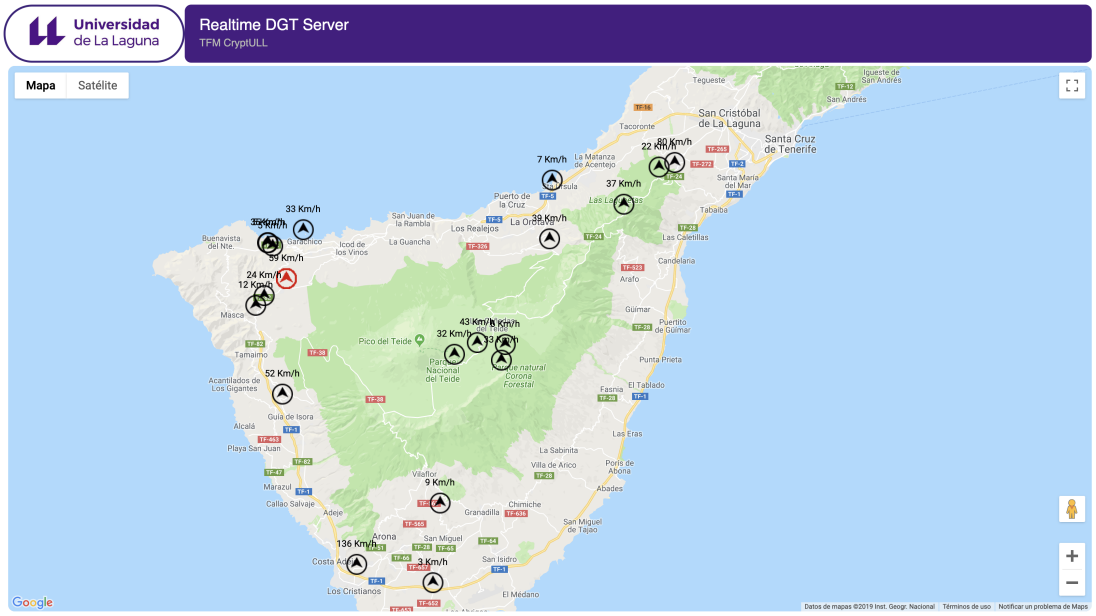

Figura 3.19: Ejemplo WEB con vehículos simulados

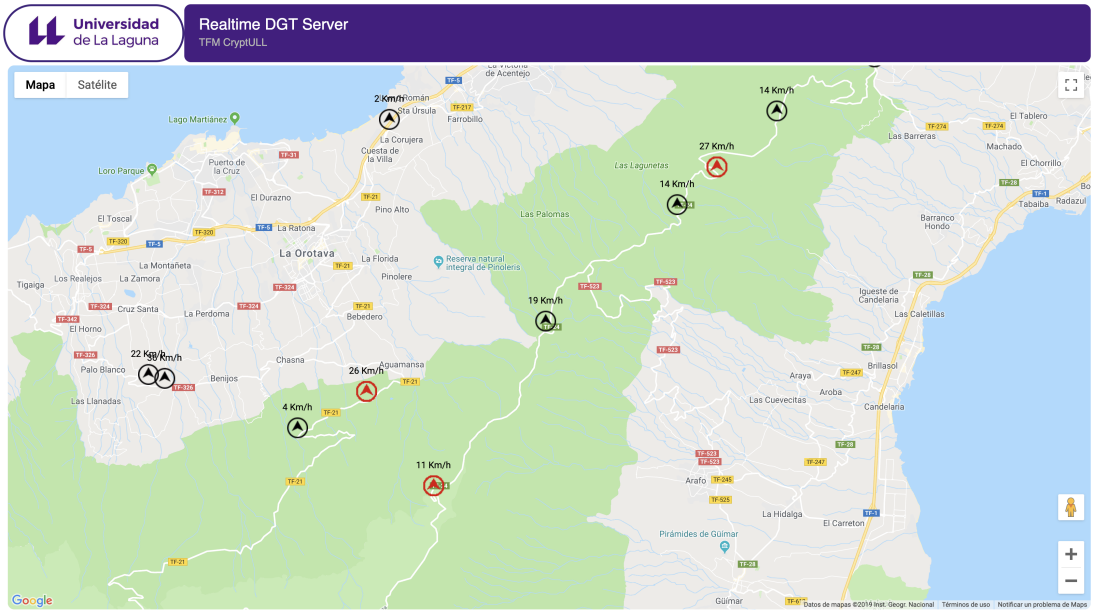

Figura 3.20: Ejemplo WEB con vehículos simulados ampliado

- Panel flotante: Al hacer clic sobre uno de los vehículos se despliega por la izquierda de la página un panel flotante el cual muestra la información del vehículo como su correo de login, uid, tipo, velocidad en el momento del clic, puntos y radios, pudiendo modificar estos 2 últimos. En las figuras 3.21 y 3.22 podemos ver un ejemplo.

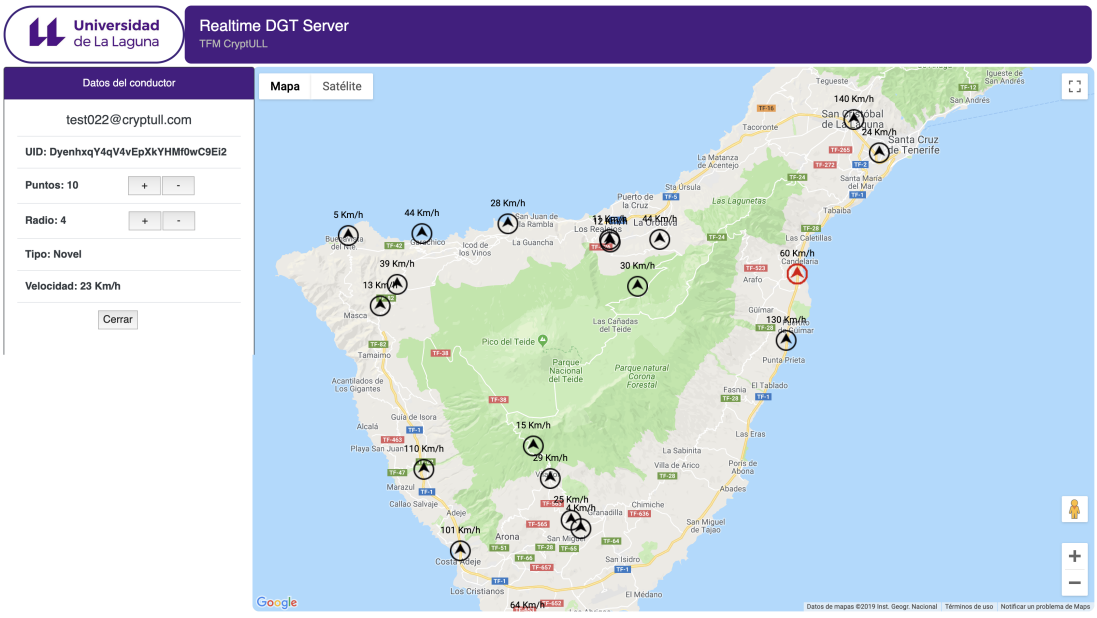

Figura 3.21: Ejemplo WEB con panel desplegado

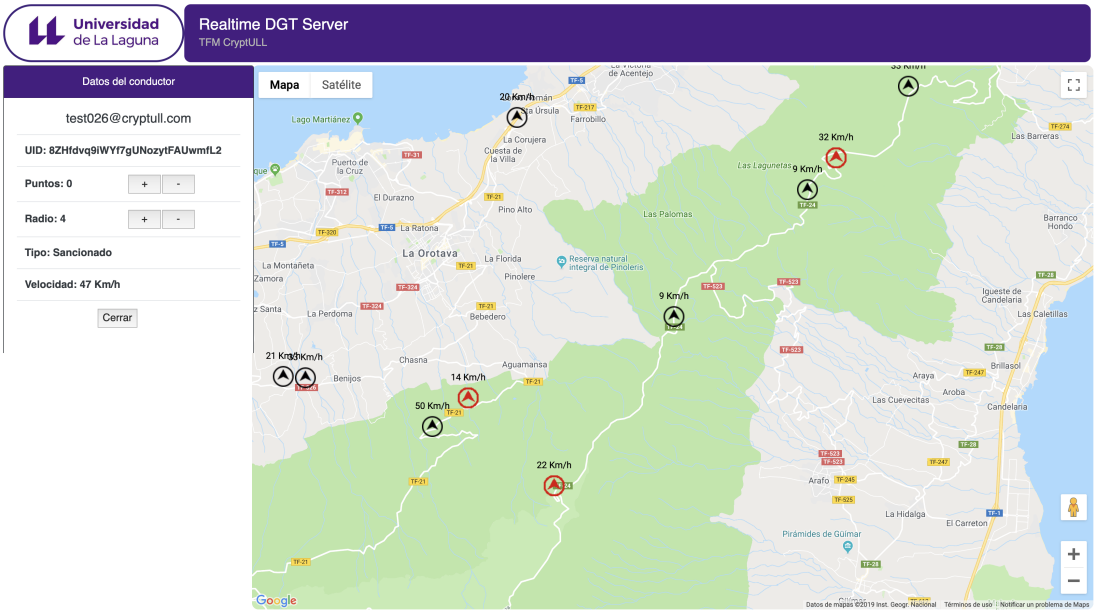

Figura 3.22: Ejemplo WEB con panel desplegado ampliado

- layout.jade: Plantilla que genera el código HTML con simples etiquetas según se puede apreciar en la figura 3.23.

```
doctype html
html
    head
        title= title
        link(rel='stylesheet', href = '/stylesheets/bootstrap.min.css')
        link(rel='stylesheet', href ='/stylesheets/simple-sidebar.css')
        link(rel='stylesheet', href ='/stylesheets/style.css')
    body
        header
            .row
                #logo_ull
                    img(src= '/images/LOGO_ULL.jpg')
                #titulo
                    h1= title
                    h2= subtitle
        #division.col-12
        #wrapper
            #sidebar-wrapper
                .card.border-dark.mb-3
                    .card-header.titulo Datos del conductor
                    .card-body.text-dark
                        h5.card-title.center#usuario
                        table.table
                            tbody
                                tr
                                    th(scope="row")#uid
                                tr
                                    th(scope="row")#puntos
                                tr
                                    th(scope='row')#radio
                                tr
                                    th(scope='row')#tipo
                                tr
                                    th(scope="row")#velocidad
                                tr
                                    th(scope="row")#cerrar


            #map_canvas.container-fluid

        script(src='http://ajax.googleapis.com/ajax/libs/jquery/2.0.3/jquery.min.js')
        script(src='/socket.io/socket.io.js')

        script.
            var data = !{JSON.stringify(dats)};
        script.
            var socket = io();

        script(src='/javascripts/moment.js')
        script(src='https://maps.googleapis.com/maps/api/js?key=AIzaSyBA9qH3qXxZp2upzucQvEXJUiz5aeaeb6M&libraries=drawing,places')
        script(src='/javascripts/bootstrap.bundle.min.js')
        script(src='/javascripts/main.js')
```

Figura 3.23: Plantilla layout.jade

- main.js: Javascript del “client-side” cuyo código se irá comentando por

partes.

- $(document).ready(): Cuando la página es cargada en el navegador, el primer paso es llamar a la función "Initialize()" y luego añade las etiquetas de bootstrap para la cabecera. Véase código 3.24

```
72  $(document).ready(function() {
73
74      initialize();
75
76      $('#logo_ull').addClass('col-2');
77      $('#titulo').addClass('col-10');
78
79  });
```

Figura 3.24: Función ready()

- Initialize(): Crea el mapa de Google Maps partiendo de una coordenada predeterminada, posteriormente solicita la ubicación actual al navegador y en caso afirmativo, actualiza el centro del mapa. Código en la figura 3.25.

```
81  //Inicializar nuestro Google Map
82
83  function initialize() {
84
85      var center = new google.maps.LatLng(28.558599999999997,-16.2989);
86      var mapOptions = {
87          zoom: 12,
88          mapTypeId: google.maps.MapTypeId.ROADMAP,
89          center: center,
90      };
91      this.map = new google.maps.Map(document.getElementById('map_canvas'),
92          mapOptions);
93
94
95      navigator.geolocation.getCurrentPosition(success, error, options);
96
97  };
```

Figura 3.25: Función Initialize()

- agregarUbicaciones(): Mediante socket recibimos un listado json con las ubicaciones a añadir. Como se puede ver en la figura 3.26 se recorre la lista con un "for each". El primer 'if' comprueba si es una actualización de un vehículo existente y en caso afirmativo es eliminado del listado de marcadores. El siguiente 'if' determina si es un vehículo a eliminar (es eliminado como en el 1º 'if') y si se debe de agregar se ejecuta el 'else'. En el 'else' se decide que icono elegir

(negro o rojo) en función de los puntos, se calcula su velocidad actual (obteniendo media de las 3 ultimas velocidades), se crea el marcador con estos datos y finalmente se añade al marcador un 'Listener' para el clic el cual carga los datos en el panel y lo hace visible cuando se selecciona un vehículo. Por último, se agrega el marcador a la lista de marcadores y se mandan al mapa para su visualización.

```
function agregarUbicaciones(datos) {

    Object.keys(datos).forEach(function(key){

        if (MAPAPP.markers[key] != undefined){

            MAPAPP.markers[key].setMap(null);
            MAPAPP.markers[key] = null;

        }

        console.log("datos[key] : " + JSON.stringify(datos[key]));

        if (datos[key].Tipo == -1 ){

            MAPAPP.markers[key].setMap(null);
            MAPAPP.markers[key] = null;

        }else{

            let ico;

            if (datos[key].Puntos == 0){
                ico = '../icons/iconoKM_96x32-red.png';
            }else{
                ico = '../icons/iconoKM_96x32.png';
            }

            let velPro = datos[key].VelActual;

            if (velPro == undefined) velPro = "0";

            var marker = new google.maps.Marker({
                map: map,
                position: new google.maps.LatLng(datos[key].lat,datos[key].lon),
                shopname: key,
                details: moment(parseInt(datos[key].timeStringStamp)).format('MMMM Do YYYY, h:mm:ss a'),
                icon: ico,
                label: velPro + " Km/h"
            });


            marker.addListener('click', function () {

                if ((document.getElementsByClassName("toggled")).length == 0){

                    $("#wrapper").toggleClass("toggled");
                    $("#uid").text("UID: " + marker.shopname);
                    $("#usuario").text(datos[marker.shopname].email);
                    $("#puntos").text("Puntos: " + datos[marker.shopname].Puntos).append(btnActualizaPuntos);
                    $("#radio").text("Radio: " + datos[marker.shopname].Radio).append(btnActualizaRadio);
                    $("#tipo").text("Tipo: " + datos[marker.shopname].Tipo);
                    $("#velocidad").text("Velocidad: " + datos[marker.shopname].VelActual + " Km/h");
                    $("#cerrar").append(button);

                } else cerrarPanel();
            });

            MAPAPP.markers[key] = marker;
            MAPAPP.markers[key].setMap;

        }
    });
};
```

Figura 3.26: Función agregarUbicaciones()

3.5. Rutas y Simulación GPS

Con el fin de probar la funcionalidad del sistema propuesto, se han hecho diversas pruebas. Hay que tener en cuenta que para llevarlas a cabo sería necesario contar con un elevado número de dispositivos, donde cada uno se encuentre en cada vehículo y enviando información simultáneamente, siendo así una opción totalmente inviable. Por lo tanto se ha realizado una simulación de la **Aplicación Cliente** (3.3) de modo que se comporte o interactúe con la BD Firebase de igual manera y que a efectos del **Servidor DGT** no exista diferencia. Por otro lado, también hay que saber obtener y manejar un fichero de coordenadas que formen una ruta por carretera para poder saber que puntos GPS enviar a la base de datos. En los siguientes apartados se explica como se han llevado a cabo la solución de esta problemática.

3.5.1. Rutas GPS

Existen numerosos sitios web donde poder descargarse rutas en formato 'CSV' pero también se puede hacer uso de las herramientas de Google Maps para crear unas propias tal y como se ha realizado para este proyecto y posteriormente convertirlas de 'KML/KMZ' a 'CSV'. La ruta consiste en una vuelta a la isla de Tenerife partiendo de Santa Cruz hacia el sur de la isla y regresando por el norte (pasando por Las Cañadas en algún punto del tramo).
Durante algunas simulaciones de estas rutas, las velocidades calculadas no son próximas a la realidad. Esto es debido a que no coinciden los tiempos de envío y las distancias recorridas porque las rutas del Google Maps no están diseñadas para simularlas como se pretende en este proyecto, sino mas bien para seguirlas y llegar a un destino concreto. Por lo tanto, se hace uso de aplicaciones gratuitas de terceros para grabar con el móvil rutas realizadas con un vehículo real. Una vez obtenida, convertida a 'CSV' y simulada, las velocidades son totalmente iguales a las obtenidas realmente por lo tanto tenemos una simulación mucho mas fiable.

3.5.2. Simulación GPS

La idea básica de la simulación GPS es la de enviar cada cierto tiempo (establecidos en segundos) una ubicación (latitud y longitud) a la base de datos con una marca de tiempo asociada. En la siguiente figura 3.27 se muestra el código de la función que lee el fichero de entrada.

```
fs.createReadStream('./routes/Vuelta_Isla_TF.csv')
  .pipe(csv())
  .on('data', (row) => {

    tabla[num] = { "lat" : parseFloat(row["Y"]) , "lon" : parseFloat(row["X"]) };

    num++;

  })
  .on('end', () => {

    console.log('CSV file successfully processed');

    console.log("::::::::: Usuarios :::::::::: ");

    for (let i = 10 ; i <= 29 ; i++){

        let test = "test";

        if (i < 10) {
            test = test + "00" + i;
        }else if (i < 100){
            test = test + "0" + i;
        }else{
            test = test + i;
        }

        let email = test + "@" + "cryptull.com";

        admin.auth().getUserByEmail(email)
        .then(function(userRecord) {

            console.log("Iniciando usuario: " + userRecord.email + "    -> uid: " + userRecord.uid);

            setTimeout(function () {

                conductorSim(tabla,userRecord.uid,getRandomInt(1,Object.keys(tabla).length));

            }, (i-10)*1000);

        })
        .catch(function(error) {

            console.log("# Error # usuario no existe:", error);

        });
    }
  });
```

Figura 3.27: Simulación GPS

Una vez leído el fichero y almacenado en JSON se ejecuta el bucle que comienza a partir del usuario “test010@cryptull.com” hasta “test029@cryptull.com” en este caso. Cada usuario ejecuta un ‘setTimeout()’ pero con parámetros de intervalos distintos en función de su número “testXXX” con el objetivo de que no se ejecuten todas las simulaciones al mismo tiempo, si no que vayan ejecutándose de 1 en 1. Dentro de la función ‘setTimeout()’ que ejecuta cada usuario existe una función ‘setInterval()’ cuyo objetivo es la de repetir una tarea determinada cada X segundos infinitamente. La función que se ejecuta infinitamente es “conductorSim(gps,userID,i)” como se muestra en la figura 3.28 siguiente:

```
function conductorSim(gps,userID,i){


    setInterval(function(){

        var d = new Date();
        var tim = d.getTime();

        console.log(d + " :: " + userID + " is Sending...: " + gps[i].lat + "   " + gps[i].lon);

        let obj = { "lat": gps[i].lat, "lon": gps[i].lon, "timeStringStamp":  tim.toString()};

        lUsers.child(userID).child(d.getTime()).set(obj);
        actUserLoc.child(userID).set(obj);

        i++;

        if (i == Object.keys(gps).length){
            i = 0;
        }else console.log("----> i: " + i);

    },2000,"JavaScript");
}
```

Figura 3.28: Función conductorSim()

La función “conductorSim()” reciben 3 argumentos:

- gps: JSON con el contenido de la ruta entera.
- userID: Como su propio nombre indica se trata del UID del usuario.
- i: Índice de donde parte la ruta la simulación en cuestión. Se calcula con un random que va desde 1 hasta el tamaño del JSON (gps). De esta manera cuando se simulan varios vehículos no empezarán todos desde el mismo sitio para evitar así mucha concurrencia de iconos en la misma zona.

3.6. Poblar BD para simulación GPS

Inicialmente para poder realizar pruebas se crearon 10 usuarios que van desde test001@cryptull.com hasta test010@cryptull.com, los cuales han sido usados desde los móviles virtuales de Android Studio. Con el objetivo de poner a prueba la aplicación es necesario realizar una simulación mas numerosa, para ello se debe crear una BD de usuarios mucho mayor. Este objetivo se ha logrado mediante la función "poblar(tam)" que crea 'tam' usuarios (en este caso hasta test100@cryptull.com) y también se añaden los campos extra en la BD (clave Datos) de la Firebase Realtime con la función "poblarDatos(tam)". El código de ambas funciones sería como se muestra en las figuras 3.29 y 3.30 siguientes:

```
function poblar(tam){

    let test = "test"
    let mail = "";
    let cont = 0;

    let i = 101; //Desde que usuario quieres empezar

    let bucle = setInterval(function(){

        if (cont == tam-1) clearInterval(bucle);

        if (i < 10) {
            test = test + "00" + i;
        }else if (i < 100){
            test = test + "0" + i;
        }else{
            test = test + i;
        }

        mail = test + "@" + "cryptull.com";

        admin.auth().createUser({
          email: mail,
          emailVerified: true,
          password: test,
          disabled: false
        })
          .then(function(userRecord) {
            // See the UserRecord reference doc for the contents of userRecord.
            console.log("Successfully created new user:", userRecord.uid);
          })
          .catch(function(error) {
            console.log("Error creating new user:", error);
          });


        console.log("Creado => Usuario:    %s        Contraseña: %s", mail,test);


        test = "test";
        mail = "";

        i++;
        cont++;

    },100,"JavaScript");
}
```

Figura 3.29: Función poblar()

```
function poblarDatos(tam){

    for (let i = 10 ; i <= 100 ; i++){

        let test = "test";

        if (i < 10) {
            test = test + "00" + i;
        }else if (i < 100){
            test = test + "0" + i;
        }else{
            test = test + i;
        }


        let email = test + "@" + "cryptull.com";

        admin.auth().getUserByEmail(email)
        .then(function(userRecord) {

            console.log("Actualizando datos de: " + userRecord.email + "    -> uid: " + userRecord.uid);

            let puntos = getRandomInt(0,16);
            let radio = getRandomInt(1,11);
            let tipoConductor = { "1": "Transporte Publico" , "2": "Novel", "3": "Normal"};

            let conductor;

            if (puntos == 0){
                conductor = "Sancionado";
            }else{

                conductor = tipoConductor[getRandomInt(1,4)];

            }

            let data = {"Puntos": puntos, "Radio": radio, "Tipo": conductor};

            dbdatos.child(userRecord.uid).set(data);

            console.log("### > " + JSON.stringify(data));
            console.log("# # # # # # # # # # # # # # #");

        })
        .catch(function(error) {

            console.log("# Error # usuario no existe", error);

        });

    }
}
```

Figura 3.30: Función poblarDatos()

Capítulo 4

Conclusiones y trabajos futuros

En los últimos años se ha producido un enorme crecimiento de las tecnologías móviles y muchos de esos avances han sido aplicados en la industria automovilística. El objetivo es dotar a los vehículos con cierta inteligencia y herramientas de control con el fin de hacerlos más seguros y, por ende, contribuir a que las carreteras sean más seguras.

No obstante, parte de la securización de las carreteras también pasa por el control y monitorización de los conductores que hacen uso de ellas. Esto es lo que se pretende con la aplicación desarrollada en este Trabajo de Fin de Máster: poder monitorizar en cada momento a los conductores que hacen uso de las vías y prevenir, o reducir, las situaciones de riesgo en carretera.

Esta herramienta facilitaría la supervisión de los conductores por parte de la DGT y, a su vez, proporcionaría asistencia en tiempo real para todos los conductores y facilitaría la toma de decisiones ante imprevistos como atascos o accidentes.

Por otra parte, la aplicación sienta las bases para nuevas iteraciones de desarrollos futuros, como puede ser la ampliación de las funcionalidades de la misma o la optimización de lo que ya existe. Otros trabajos futuros también podrían abarcar el resto de procesos que se incluyen en el pipeline de la Ingeniería del Software:

- Automatización del proceso de compilación del código.
- Inclusión del código en herramientas de integración continua (CI), entrega continua (CD) y despliegue continuo.
- Inclusión del código de la parte servidor en un contenedor, de manera que la aplicación sea lo más portable y escalable posible.

Capítulo 5

Conclusions and future works

In recent years there has been a tremendous growth in mobile technologies and many of these advances have been applied in the automobile industry. The objective is to provide vehicles with some intelligence and control tools in order to make them safer and, therefore, contribute to making roads safer.

However, part of the road safety also goes through the control and monitoring of the drivers who use them. This is what is intended with the application developed in this Master's Thesis: to be able to monitor drivers who use the roads at all times and prevent, or reduce, risk situations on the road.

This tool would facilitate the supervision of drivers by the DGT and, in turn, provide real-time assistance for all drivers and facilitate decision-making in case of unforeseen events such as traffic jams or accidents.

On the other hand, the application lays the foundations for new iterations of future developments, such as the extension of its functionalities or the optimization of what already exists. Other future work could also cover the rest of the processes included in the Software Engineering pipeline:

- Automation of the code compilation process.
- Inclusion of the code in tools of continuous integration (CI), continuous delivery (CD) and continuous deployment.
- Inclusion of the server part code in a container, so that the application is as portable and scalable as possible.

Capítulo 6

Presupuesto

En este capítulo se establece un presupuesto que indica el coste de implementar este Trabajo de Fin de Máster si se encargase por la DGT.

6.1. Introducción y coste por hora

A continuación se muestra una tabla con las actividades realizadas en este TFM. Se indica la duración en horas de implementación para cada una y el precio por hora calculado.

El precio por hora que se considerará en este presupuesto es de 35€/hora.

6.2. Funcionalidades requeridas (Android Studio)

Actividad	**Duración**	**Precio**
Validación usuario/contraseña	10 horas	350 €
Representación de vehículos en mapa (Sancionados y NO Sancionados)	80 horas	2800 €
Envío de ubicación GPS a la DGT	12 horas	420 €
Modificación de radio en KM	5 horas	175 €
Subtotal	**107 horas**	**3745 €**

Tabla 6.1: Tabla de actividades, duración y precios

6.3. Funcionalidades requeridas (Servidor nodeJS)

Actividad	**Duración**	**Precio**
Análisis y cálculo continuo de distancias entre vehículos para su posterior envío a cada usuario	20 horas	700 €
Cálculo continuo de la velocidad instantánea para posibles sanciones	10 horas	350 €
Control de usuarios sancionados	5 horas	175 €
Servidor WEB de la DGT	30 horas	1050 €
Subtotal	**65 horas**	**2275 €**

Tabla 6.2: Tabla de actividades, duración y precios

6.4. Funcionalidades extra

Actividad	**Duración**	**Precio**
Diseño y desarrollo de un Simulador GPS real con rutas reales	25 horas	875 €
Poblar Base de Datos para el Simulador GPS	1 horas	35 €
Subtotal	**26 horas**	**910 €**

Tabla 6.3: Tabla de actividades, duración y precios

6.5. Coste y duración total

Actividad	**Duración**	**Precio**
Funcionalidades requeridas (Android Studio)	107 horas	3745 €
Funcionalidades requeridas (Servidor nodeJS)	65 horas	2275 €
Funcionalidades extra	26 horas	910 €
Total	**198**	**6930 €**

Tabla 6.4: Precio y duración total

Capítulo 7

Inconvenientes

7.1. Inconvenientes

Durante el desarrollo del proyecto han existido dos inconvenientes fundamentales que han impedido llevar a cabo objetivos planteados inicialmente. Por un lado la validación mediante huella dactilar o Face ID (7.1.1) y por otro, la obtención del límite de velocidad de la vía por la que se circula (7.1.2).

7.1.1. Validación mediante Huella o Face ID

En el punto 1.3 se especifica como forma de acceder a la aplicación el método de autenticación del usuario con el sistema mediante huella dactilar y/o reconocimiento facial. Según la documentación para desarrolladores [1] de Android, concretamente, en el apartado Sistema Android Keystore (dentro de Best Practise / Security), describe con exactitud el mecanismo que se lleva acabo en Android para la gestión de claves criptográficas el cual cito textualmente a continuación:

"*...El sistema* ***Android Keystore*** *te permite almacenar claves criptográficas en un contenedor para que resulte más difícil extraerlo del dispositivo. Una vez que las claves se encuentran en el almacén de claves, se pueden usar para realizar operaciones criptográficas con el material de claves restante no exportable...*"

"*...****Funciones de seguridad:*** *El sistema Android Keystore protege el material de claves contra usos no autorizados. En primer lugar, Android Keystore reduce el uso no autorizado de material de claves fuera del dispositivo Android evitando la extracción de este material de los procesos de la aplicación y del dispositivo Android en su totalidad...*

"*...****Prevención de extracción:*** *El material de claves de las claves de Android Keystore está protegido contra la extracción mediante dos medidas de seguridad:*

- "*El material de claves nunca ingresa al proceso de la aplicación. Cuando una aplicación realiza operaciones criptográficas usando una clave de Android Keystore, el texto sin formato, el texto cifrado y los mensajes en segundo plano que se firmarán o verificarán se envían a un proceso del sistema que se ocupa de las operaciones criptográficas. Si el proceso de la app está comprometido, el atacante puede usar las claves de la app.* ***Sin embargo, no podrá extraer el material de claves (por ejemplo, para usarlo fuera del dispositivo Android).***"

Si partimos de como gestiona Android las claves criptográficas y la propia naturaleza del proyecto, se torna inviable poder almacenar dichas claves en una base de datos externa de Firebase (como es en este caso). Debido a este problema se ha optado por un inicio de sesión mediante **usuario\contraseña** como método de validación.

7.1.2. API ROAD (speedlimit)

Con la última actualización de política de uso de las API de Google, se hace obligatorio la incorporación de una clave API que se consigue mediante una suscripción a Google Maps Platform. Para la realización de este trabajo se ha tenido que crear dicha suscripción en modalidad 'Free', donde sería suficiente introducir una tarjeta de débito para que te concedan 300$ de crédito. Con este 'Free Plan' puedes acceder a la mayoría de las API de Google y realizar muchas acciones con ella, pero concretamente para la API ROAD Speedlimit (API que proporciona el límite de velocidad dadas unas coordenadas), según comentan en la documentación oficial es obligatorio un 'Google Maps APIs Premium Plan' el cual es totalmente de pago y te cobran por su uso. Por este motivo no se ha podido realizar las acciones de sanción pertinentes sobre los usuarios. No obstante, sí que se calcula la velocidad de cada vehículo y con perspectivas de líneas futuras contando con un plan premium, sería bastante sencillo a la par que viable añadir el código necesario para comparar velocidad y sancionar en caso necesario. [17]

Bibliografía

[1] Android Developer. https://developer.android.com/.

[2] Firebase Authentication. https://firebase.google.com/docs/auth/?hl=es-419.

[3] Firebase Realtime Database. https://firebase.google.com/docs/database?hl=es-419.

[4] FleetGO. https://fleetgo.com.

[5] Github. https://github.com.

[6] Global M2M SIM. https://www.globalm2msim.com/gps-trackers.

[7] Google Maps Platform - Maps. https://cloud.google.com/maps-platform/maps/?hl=es-419&sign=0.

[8] Google Maps Platform - Routes. https://cloud.google.com/maps-platform/routes/?hl=es-419.

[9] GPSWOX. https://www.gpswox.com.

[10] Jade. https://medium.com/@gorilonmax/jade-una-herramienta-eficaz-para-creaci%C3%B3n-de-contenido-y-plantillas-1bcc17d863ed.

[11] JavaScript Timing Events. https://www.w3schools.com/js/js_timing.asp.

[12] Maps Javascript API. https://developers.google.com/maps/documentation/javascript/tutorial.

[13] Maps SDK Android. https://developers.google.com/maps/documentation/android-sdk/intro.

[14] MOTAM. https://cryptull.webs.ull.es/MOTAM.

[15] nodeJS. https://nodejs.org/en/.

[16] NPM. https://www.npmjs.com/.

[17] Speed Limits. https://developers.google.com/maps/documentation/roads/speed-limits.

[18] Transpoco. https://www.transpoco.com.

[19] Webstorm. https://www.jetbrains.com/webstorm.