Trabajo de Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

# Aplicación Android y Servicio Web para la recolección en tiempo real de infracciones de velocidad, con representación en mapa mediante ELK y Kafka

Autor: David Madroñal Sánchez

Tutor: María Teresa Ariza Gómez

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

Sevilla, 2023

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

# Aplicación Android y Servicio Web para la recolección en tiempo real de infracciones de velocidad, con representación en mapa mediante ELK y Kafka


Autor:
David Madroñal Sánchez

Tutor:
María Teresa Ariza Gómez
Profesor titular

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



Sevilla, 2023

Trabajo de Fin de Grado: Aplicación Android y Servicio Web para la recolección en tiempo real de infracciones de velocidad, con representación en mapa mediante ELK y Kafka

Autor: David Madroñal Sánchez

Tutor: María Teresa Ariza Gómez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

*A mi familia*

*A mis amigos*

*A mis maestros*

# Agradecimientos

En primer lugar, agradecer a mi tutora, María Teresa Ariza Gómez, por brindarme la oportunidad de trabajar con ella, por su ayuda, consejos y tiempo.

En segundo lugar, agradecer a mis padres por confiar en mí y por todo el sacrificio que han hecho durante este tiempo. Agradecer a mis hermanos por aguantarme y apoyarme.

Agradecer a Carmen, por su infinita paciencia, ayuda y motivación. Además de ser el mayor apoyo de mi día a día.

Y por último, agradecer a los compañeros que me han acompañado durante esta etapa. Sobre todo, gracias a Fernando, Ferchu, Ocaña y Chazo por todas las risas y por todos los buenos momentos, sin vuestra amistad todo hubiera sido más duro.

*David Madroñal Sánchez*

*Sevilla, 2023*

# Resumen

Según la Dirección General de Tráfico (DGT), uno de cada cinco accidentes de tráfico con víctimas se puede relacionar directamente con el exceso de velocidad [1]. Por ello, la DGT, cada vez invierte más en campañas preventivas para evitar el exceso de velocidad, colocando más radares, fijos y móviles, y con Pegasus, el helicóptero de la DGT para controlar el tráfico desde el aire.

En este proyecto se ha desarrollo una aplicación móvil capaz de detectar y registrar automáticamente las infracciones de velocidad, gracias al GPS del dispositivo se obtiene la velocidad y la posición del vehículo en tiempo real y con un servicio web se obtiene la velocidad máxima de la carretera por la que se circula, en caso de ir a una velocidad mayor a ésta se registra la infracción de velocidad. Esto proporciona una herramienta efectiva para fomentar una conducción responsable y reducir el número de accidentes relacionados con la velocidad.

Además, la integración de tecnologías como Apache Kafka y ELK permite un procesamiento eficiente de los datos de las infracciones y su visualización en un mapa, lo cual facilita la identificación de las áreas de mayor incidencia y contribuye a la planificación de estrategias de seguridad vial más efectivas.

# Abstract

According to the Directorate General of Traffic (DGT), one out of every five traffic accidents with casualties can be directly related to speeding [1]. That is why the DGT is increasingly investing in preventive campaigns to avoid speeding, installing more fixed and mobile speed cameras, and using Pegasus, the DGT's helicopter for traffic control from the air.

In this project, a mobile application has been developed that is capable of automatically detecting and recording speed violations. Using the device's GPS, real-time vehicle speed and position are obtained, and a web service provides the maximum speed limit for the road being traveled. If the vehicle exceeds this speed limit, the speed violation is recorded. This provides an effective tool to promote responsible driving and reduce the number of speed-related accidents.

Furthermore, the integration of technologies like Apache Kafka and ELK allows for efficient processing of infringement data and its visualization on a map. This facilitates the identification of high-incidence areas and contributes to the planning of more effective road safety strategies.

# Índice

# ÍNDICE DE TABLAS

# ÍNDICE DE ILUSTRACIONES

# 1 INTRODUCCIÓN

*La tecnología es sólo una herramienta. En términos de motivar a los niños y lograr que trabajen juntos, el profesor es el recurso más importante.*

*- Bill Gates -*

## 1.1 Motivación

Este Proyecto se ha realizado con la finalidad de diseñar un sistema que ayude a la seguridad en carretera en tiempo real. Dicho sistema ayudará a detectar las infracciones de velocidad, gracias a la aplicación Android y al GPS del smartphone, y representar geográficamente los puntos donde se ha cometido dicha infracción, gracias a la integración de tecnologías que se comentará más adelante.

La representación geográfica facilita la identificación de las áreas de mayor incidencia y contribuye a la planificación de estrategias de seguridad vial más efectivas.

Una de las tecnologías a destacar es Apache Kafka, una plataforma distribuida para la transmisión de datos que permite publicar, almacenar y procesar flujos de eventos de forma inmediata y también suscribirse a ellos. Está diseñada para administrar los flujos de datos de varias fuentes, en este caso todos los usuarios que usen la aplicación Android, y enviarlos a distintas fuentes, en este caso la tecnología ELK, otra de las tecnologías a destacar.

La tecnología ELK, formada por ElasticSearch, Logstash y Kibana, es un conjunto de herramientas de código abierto utilizadas para la recopilación, indexación, búsqueda y visualización de datos en tiempo real.

## 1.2 Objetivos

Los objetivos de este proyecto son los siguientes:

- Desarrollar una aplicación Android capaz de detectar y registrar infracciones de velocidad en tiempo real, con una interfaz sencilla para todos los usuarios.
- Implementar un inicio de sesión con seguridad de token, lo cual implica diseñar y desarrollar un mecanismo de autenticación que utilice tokens seguros para contribuir a la seguridad y protección de los datos de los usuarios, así como a la prevención de accesos no autorizados a la aplicación.
- Integrar dos tecnologías clave para el procesamiento y visualización de los datos de infracciones en tiempo real: Kafka y ELK.
    - Kafka como tecnología de envío de datos en tiempo real.

- ELK que permite obtener y procesar los datos de Kafka y visualizarlos en un mapa, para una representación geográfica de las ubicaciones en las que se cometieron infracciones.

## 1.3 Antecedentes

El proyecto realizado en 2019 por Pablo Bermejo Pérez [2] y el realizado en 2020 por Sergio Mellado Contioso [3] para sus Trabajos de Fin de Grado han servido como inspiración e idea base para este proyecto.

El proyecto de Pablo Bermejo Pérez hacía uso de un dispositivo IoT dotado de un sensor GPS y de un acelerómetro con giroscopio, el cual funcionaba de forma autónoma y se incorporaba a un vehículo midiendo los parámetros de posición, velocidad, aceleración y dirección. También consultaba a un servicio web, creado por él, la velocidad máxima. Para ello tenía una base de datos en la que había importado datos de la DGT sobre velocidades máximas asociadas a tipos de vehículos. Al obtener la velocidad máxima comprobaba si el vehículo excedía dicha velocidad y en caso afirmativo avisaba al conductor con señales luminosas y acústicas. Además, se centraba en el desarrollo de una aplicación web, con la que el usuario podía suscribirse, mediante un número de matrícula, a la información que el context broker recopilara sobre su vehículo, así como consultar los datos recogidos y almacenados, tras la suscripción, en una base de datos.

Por otro lado, el proyecto de Sergio Mellado Contioso diseñaba una aplicación Android para la monitorización de la velocidad, posición, aceleración y la velocidad de rotación de los vehículos, obtenidos por los sensores del móvil. También, enviaba los datos recogidos por los sensores a un context broker y desarrollaba un servicio web para suscribirse al context broker, obtener la información y almacenarla en una base de datos.

En este proyecto se hace uso de un dispositivo móvil, basándose en la idea de Sergio Mellado Contioso, pero en este caso solo del sensor GPS. Además, se desarrolla una aplicación Android encargada de detectar infracciones de velocidad, basándose en la idea de Pablo Bermejo Pérez. En este caso, la aplicación Android consulta el servicio web “overpass-api”, el cual devuelve la velocidad máxima en tiempo real de la vía por la que se circula y en caso de cometerse una infracción se registra, tanto en una base de datos de tiempo real de Firebase, como en la tecnología ELK, la cual nos permite la representación geográfica de los puntos en los que se comete dicha infracción.

## 1.4 Descripción de la solución

A continuación, se presenta la solución desarrollada para poder alcanzar los objetivos establecidos en la memoria.

### 1.4.1 Funcionalidades

El proyecto presenta las siguientes funcionalidades:

- Autenticación y verificación de usuario mediante la aplicación Android, Firebase y API REST.
- Interfaz de usuario sencilla para el uso de cualquier usuario.
- Permitir guardar y modificar la matrícula vinculada a un determinado usuario.
- Permitir consultar las infracciones vinculadas a un determinado usuario.
- Detección en tiempo real de las infracciones.
- Procesamiento, almacenamiento y visualización de las infracciones en un mapa a tiempo real.
- Múltiples opciones de filtrados de datos en la representación geográfica.

### 1.4.2 Arquitectura del Sistema

La arquitectura del sistema, la cual se muestra en la ilustración 1, detalla el sistema desarrollado.

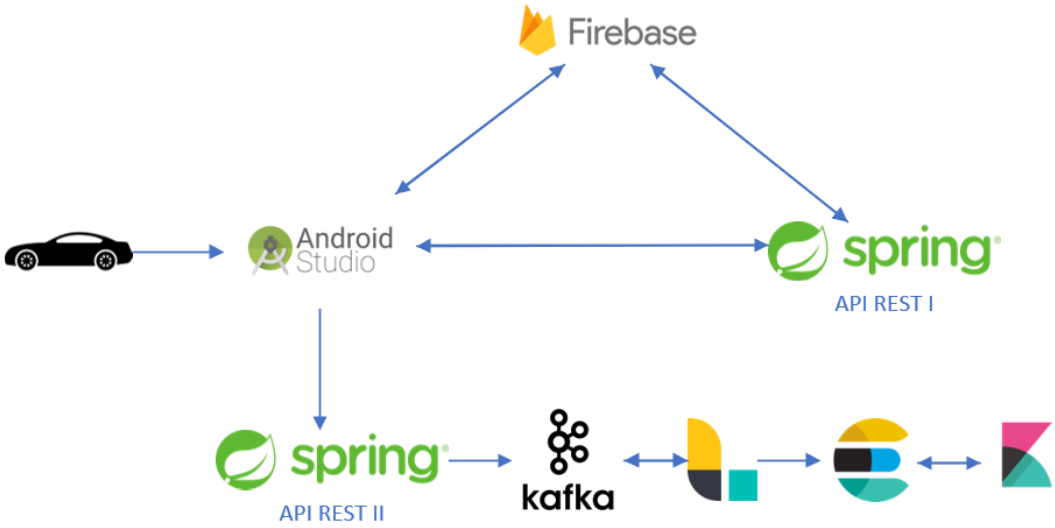



Ilustración 1: Arquitectura del sistema.

A continuación, se muestra la división de la arquitectura por bloques, ilustración 2 e ilustración 3, detallando los protocolos y puertos estándares usados para las conexiones entre las tecnologías.

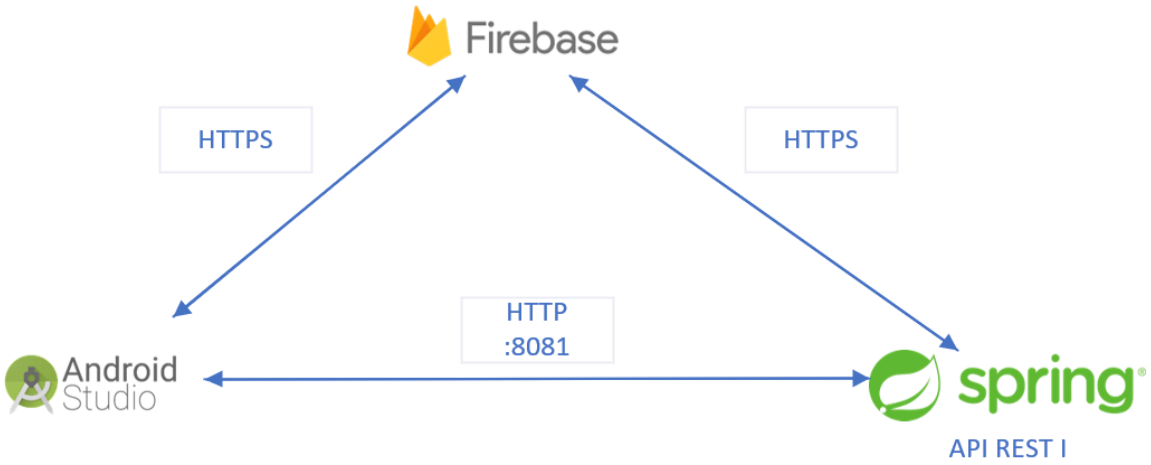



Ilustración 2: Arquitectura del inicio de sesión.

- Bloque I (ilustración 2): El primer bloque muestra la arquitectura del inicio de sesión, mediante autenticación con token, de la aplicación Android, detallando las conexiones necesarias para el inicio de sesión y la obtención de datos de la realtime database (base de datos en tiempo real) de Firebase.

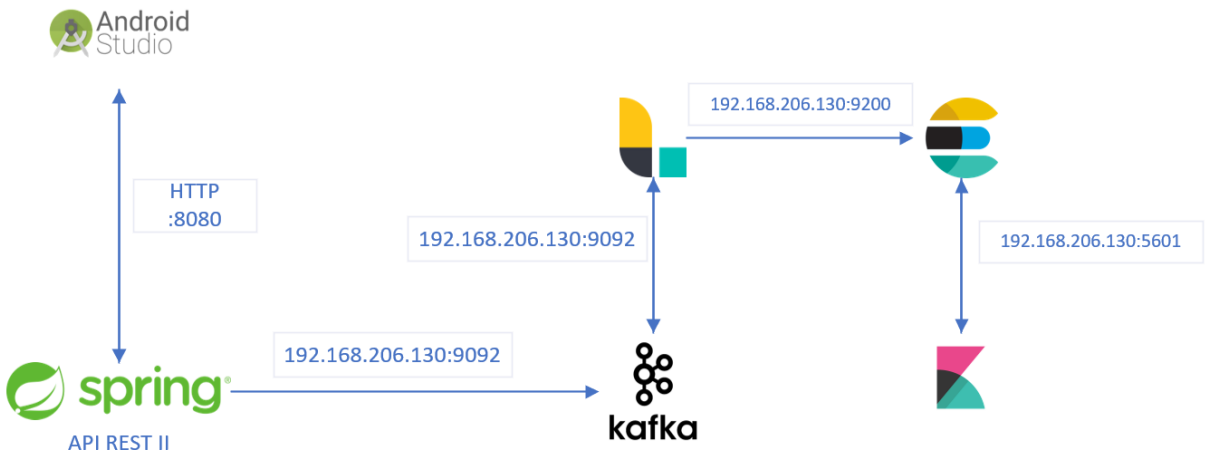



Ilustración 3: Arquitectura del registro de datos.

- Bloque II (ilustración 3): El segundo bloque muestra la arquitectura del registro de datos, mostrando las conexiones que se realizan desde que se detecta una infracción, desde la aplicación Android, hasta que llega a Kibana para su representación.

## 1.5 Estructura de la memoria

Para facilitar la lectura de esta memoria, en este apartado se resumen el contenido de cada capítulo:

1 **Introducción:** Presentación del problema en su contexto y solución adoptada.

2 **Recursos utilizados:** Recursos software y hardware empleados para la creación de la aplicación Android, los servicios web y las tecnologías utilizadas.

3 **Tecnologías utilizadas:** Resumen de las principales tecnologías empleadas para desarrollar el Proyecto.

4 **Aplicación desarrollada: Servicios Web:** Estructura de los dos servicios web desarrollados. Uno de ellos para el inicio de sesión y obtención de datos del usuario y el otro para el envío de los datos a Kafka.

5 **Aplicación desarrollada: Aplicación Android:** Estructura y análisis de la aplicación Android.

6 **Interfaz de aplicación Android:** Descripción de las funcionalidades que ofrece la aplicación Android de cara al usuario.

7 **Interfaz de Kibana:** Descripción de las funcionalidades que ofrece Kibana para la representación de las infracciones

8 **Conclusión y líneas futuras:** Ideas que podrían mejorar el proyecto en próximas versiones del mismo.

# 2 RECURSOS UTILIZADOS

Este capítulo se centra en comentar los recursos, tanto hardware como software, que se han utilizado para el desarrollo de este proyecto.

## 2.1 Recursos Hardware

### 2.1.1 Ordenador de escritorio AMD Ryzen 5 3600

El recurso hardware que se ha utilizado para este proyecto es un ordenador de escritorio, utilizado para alojar Android Studio y la máquina virtual de Ubuntu, en la cual se lanzan los servicios web, Kafka y ELK. Las características del ordenador son las siguientes:


Ilustración 4: Procesador AMD Ryzen 5 3600.

- Procesador: AMD Ryzen 5 3600, 6 núcleos 3.6Hz.
- Memoria RAM: 16,0 GB.
- Tarjeta gráfica: NVIDIA GeForce RTX 2060.
- Almacenamiento: 1TB HDD + 500GB SDD.
- Sistema Operativo: Windows 10 Pro.

## 2.2 Recursos Software

### 2.3.1 Android Studio

La aplicación móvil ha sido desarrollada mediante el entorno de desarrollo integrado Android Studio, ya que es la herramienta oficial de desarrollo de Android. Además, se hace uso del emulador de Android, que éste integra, para probar la aplicación y poder realizar pruebas emulando rutas con el mapa que incorpora.


Ilustración 5: Logo Android Studio.

### 2.3.2 Navegador Web Google Chrome

El navegador utilizado para hacer uso de la interfaz web Kibana, la interfaz web Kafka Manager, la cual es la interfaz web de Kafka desarrollada por Yahoo, y la plataforma Firebase ha sido Google Chrome.

Ilustración 6: Logo Google Chrome.

### 2.3.3 Máquina Virtual Ubuntu

Para este proyecto se ha utilizado una máquina virtual, creada con VMware Workstation Pro, con el sistema operativo Ubuntu 20.04.4. En esta máquina virtual se lanzarán los servicios web desarrollados en Eclipse y las tecnologías Kafka y ELK. La configuración de esta máquina virtual se puede ver en la ilustración 7.

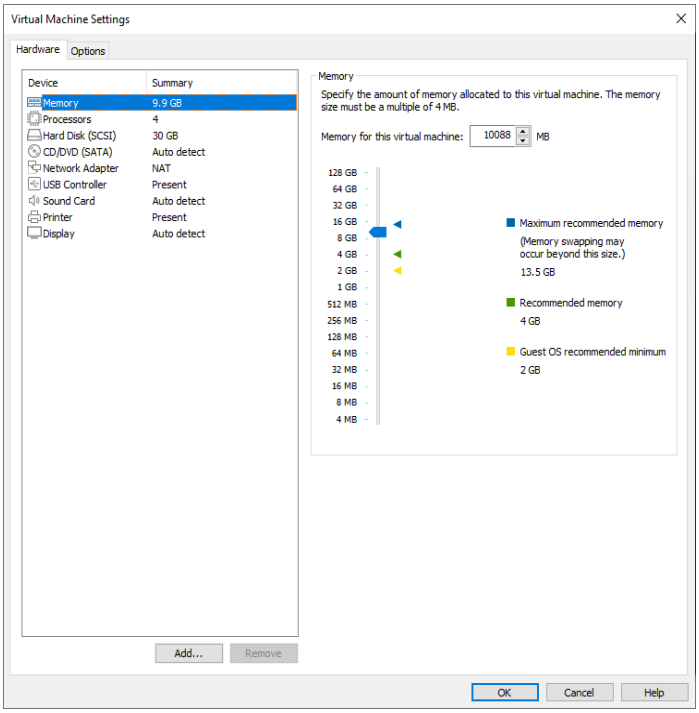



Ilustración 7: Configuración Máquina Virtual.

### 2.3.4 Eclipse

El entorno de desarrollo para implementar y desarrollar los dos servicios web API REST de este proyecto ha sido Eclipse, ya que éste proporciona un entorno confiable y eficiente para la implementación de dichos servicios.

Ilustración 8: Logo Eclipse.

# 3 TECNOLOGÍAS UTILIZADAS

Este capítulo resume las tecnologías utilizadas para el desarrollo de este proyecto.

## 3.1 Firebase

Firebase es una plataforma de desarrollo de aplicaciones móviles y web desarrollada por Google. Proporciona una variedad de servicios y herramientas para ayudar a los desarrolladores a crear aplicaciones de alta calidad de manera más rápida y eficiente. Dos de sus principales características que han sido usadas en este proyecto son:

- **Base de datos en tiempo real:** La base de datos Realtime Database de Firebase es una base de datos NoSQL en tiempo real, alojada en la nube que permite almacenar y sincronizar datos en tiempo real entre diferentes clientes o dispositivos. Destacar que la estructura de datos se basa en un árbol JSON, lo que significa que los datos se organizan en una estructura de tipo clave-valor.
- **Autenticación de usuarios:** proporciona un sistema de autenticación fácil de usar que permite a los desarrolladores gestionar el proceso de inicio de sesión y registro de usuarios de forma segura. Admite diferentes métodos de autenticación, como el usado en este proyecto, inicio de sesión mediante cuenta de Google.


Ilustración 9: Logo Firebase.

## 3.2 Docker-Compose

Para desplegar ELK y Kafka de forma fácil y sencilla se ha usado Docker-Compose, el cual es una herramienta que se utiliza para definir y administrar aplicaciones compuestas por varios contenedores de Docker. De forma fácil de entender, un contenedor Docker es una instancia aislada e independiente que contiene lo necesario para ejecutar una aplicación de manera portátil, el cual se crean a partir de una imagen de Docker.

Además, permite definir la configuración de una aplicación en un archivo YAML, en el cual se especifican los servicios, imágenes, volúmenes, variables de entorno y otras opciones relacionadas con los contenedores. Con este archivo de configuración, Docker-Compose, facilita la creación y ejecución de todos los contenedores necesarios para que la aplicación funcione correctamente.

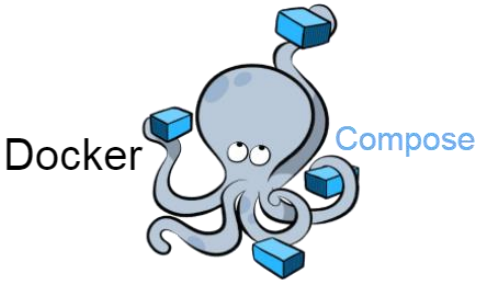



Ilustración 10: Logo Docker-Compose.

## 3.3 Apache Kafka

Para transmitir los datos desde el servicio web hasta ELK se ha usado Kafka, el cual es una plataforma de streaming distribuida de código abierto desarrollada por Apache Software Foundation. Se utiliza para el procesamiento y la transmisión de flujos de datos en tiempo real de manera escalable, confiable y de gran rendimiento.

Kafka está compuesto por varios elementos clave que trabajan juntos para habilitar la transmisión y el procesamiento de datos en tiempo real. Estos elementos son:

- **Productores:** Los productores son las entidades que generan y emiten los eventos o mensajes en los diferentes topics de Kafka. Además, envían los datos a los brokers de Kafka para que sean almacenados y procesados. En este caso será la API REST, a la cual se le enviaran datos a través de la aplicación Android.
- **Topics:** Los topics son categorías o temas a los que se envían los datos en Kafka. Los productores publican los mensajes en un topic específico, y los consumidores se suscriben y leen los mensajes de uno o varios topics.
- **Brokers:** Los brokers son los servidores de Kafka que almacenan y gestionan los datos. Los brokers reciben los mensajes de los productores, los almacenan de forma duradera y los hacen disponibles para los consumidores.
- **Consumidores:** Los consumidores son las entidades que se suscriben a uno o varios topics y reciben los mensajes publicados en ellos. Pueden procesar los datos en tiempo real o almacenarlos en otros sistemas para su posterior análisis o uso. En este caso será ELK.


Ilustración 11: Logo Kafka.

## 3.4 ElasticSearch, Logstash y Kibana

ELK es un conjunto de herramientas de código abierto utilizadas para la recopilación, indexación, búsqueda y visualización de datos en tiempo real, sus componentes son:

- **Logstash:** Es una herramienta de procesamiento de datos que permite la ingestión y transformación de datos de diferentes fuentes en tiempo real. Puede recopilar registros de diversas fuentes, en este caso de Apache Kafka, y normalizarlos para su posterior indexación en Elasticsearch.
- **Elasticsearch:** Es un motor de búsqueda y análisis distribuido diseñado para almacenar y recuperar grandes cantidades de datos en tiempo real. Proporciona una capacidad de búsqueda rápida y escalable, así como también permite realizar análisis y agregaciones complejas sobre los datos.
- **Kibana:** Es una interfaz web de usuario que permite visualizar y explorar los datos almacenados en Elasticsearch. Proporciona capacidades de búsqueda, filtrado, creación de gráficos y tableros interactivos para analizar y visualizar datos de manera intuitiva, en este caso será donde se visualizará el mapa con los puntos en los que se cometan las infracciones.

Ilustración 12: Logo ELK.

## 3.5 Overpass-api

La Overpass API es una interfaz de programación de aplicaciones (API) diseñada para acceder y consultar datos geográficos almacenados en el proyecto OpenStreetMap (OSM). OpenStreetMap es una colaboración de código abierto que crea y proporciona datos geoespaciales gratuitos y accesibles. La API Overpass permite realizar consultas específicas para recuperar datos geográficos detallados de OSM. Gracias a esta API, se obtiene la velocidad máxima y el nombre de la vía por la que se circula, proporcionándole la latitud y longitud de la posición en tiempo real.

# 4 APLICACIÓN DESARROLLADA: SERVICIOS WEB

En este punto se describen los dos servicios web desarrollados en este proyecto. Uno de ellos encargado de la comunicación con Firebase y el otro con Kafka.

En primer lugar, se muestra un diagrama de secuencia en el que se puede observar cómo se registra una infracción tanto en Firebase como en Kafka, ilustración 13.

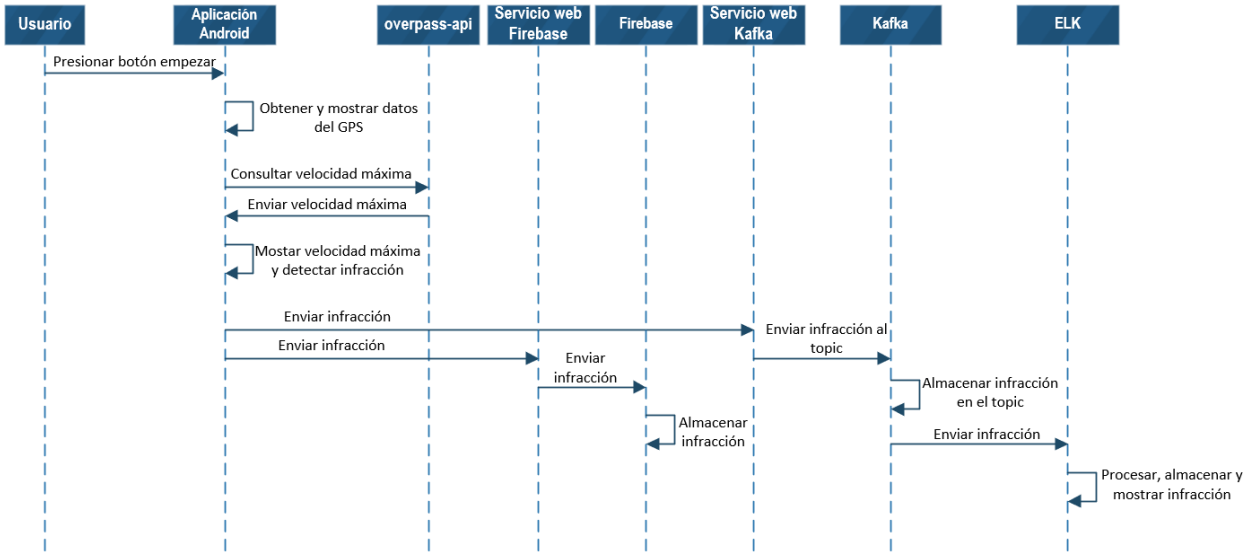



Ilustración 13: Detectar infracción – diagrama de secuencia.

## 4.1 Servicio web Firebase

Este servicio web se utiliza para realizar las siguientes operaciones relacionadas con Firebase:

- Mostrar matrícula.
- Modificar matrícula.
- Mostrar infracciones almacenadas.
- Almacenar infracción.

La configuración de este servicio web se detalla en el Anexo A.

A continuación, se mostrarán cuatro diagramas de secuencia de funcionalidades que implican Firebase:

- En el primer diagrama se muestra el inicio de sesión con cuenta de Google, ilustración 14.

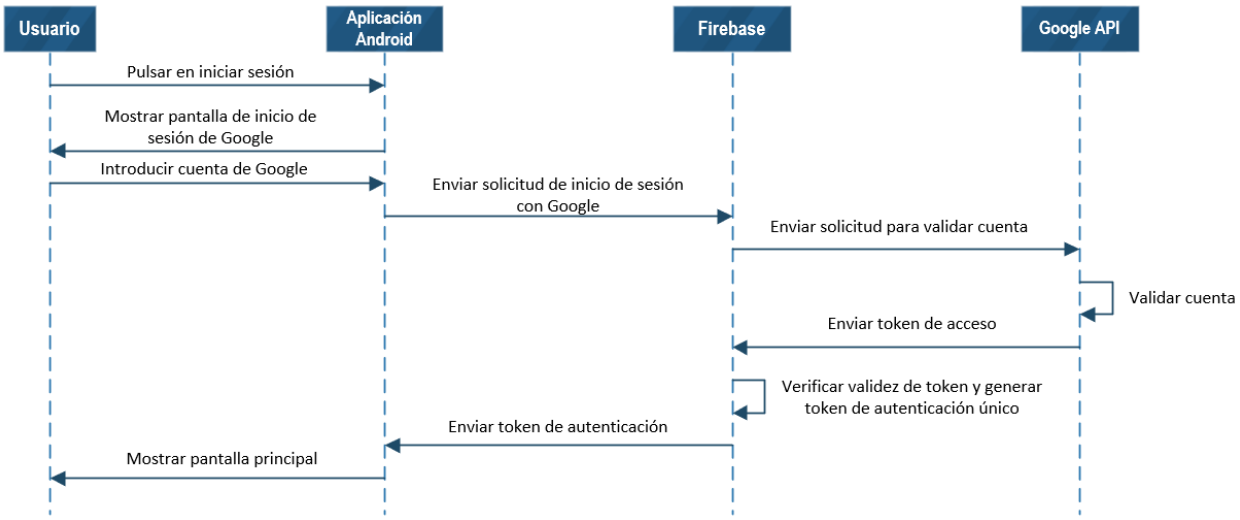



Ilustración 14: Inicio sesión – diagrama de secuencia.

- En el segundo diagrama se puede observar cómo se obtiene y se muestra la matrícula al usuario una vez que el usuario ha iniciado sesión y se encuentra en la pantalla principal, ilustración 15.

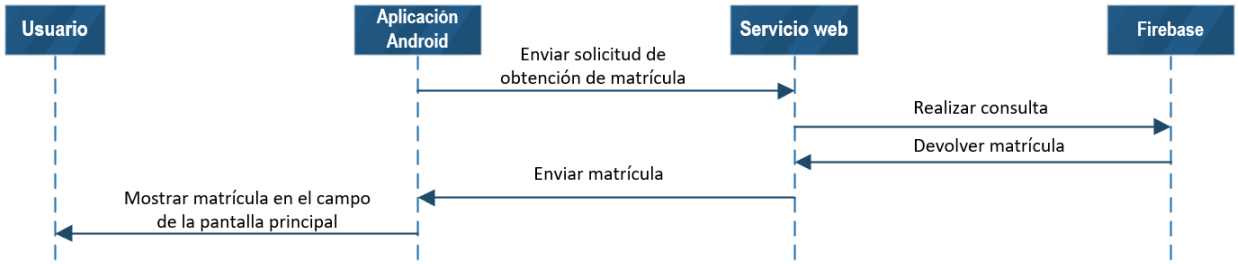



Ilustración 15: Mostrar matrícula – diagrama de secuencia.

- En el tercer diagrama se muestra cómo se guarda/modifica la matrícula del usuario desde la pantalla principal, ilustración 16.

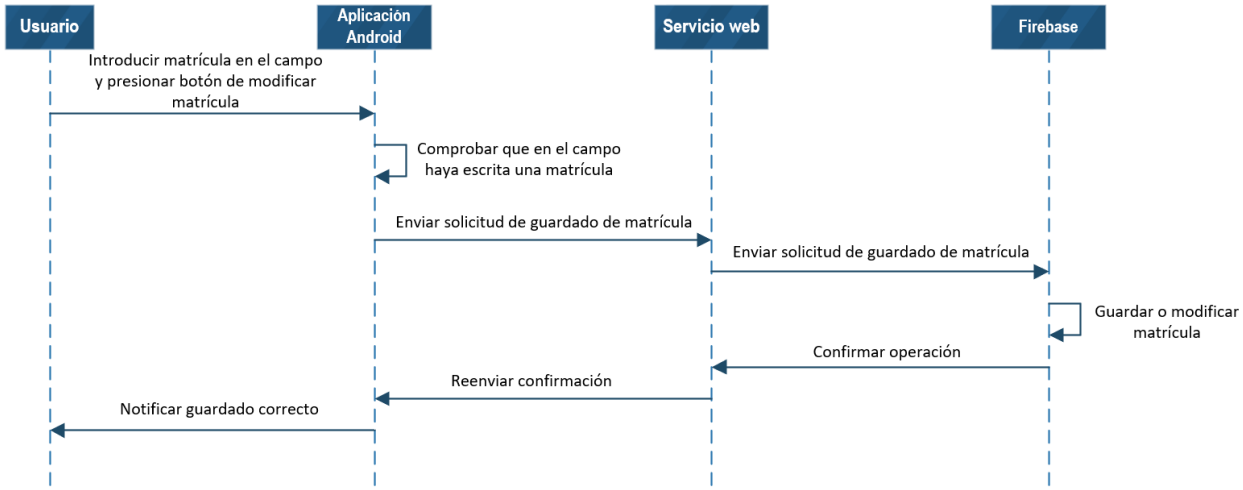



Ilustración 16: Modificar matrícula – diagrama de secuencia.

- En el cuarto diagrama se observa como se muestra el historial de infracciones una vez se presiona el botón de la pantalla principal, ilustración 17.

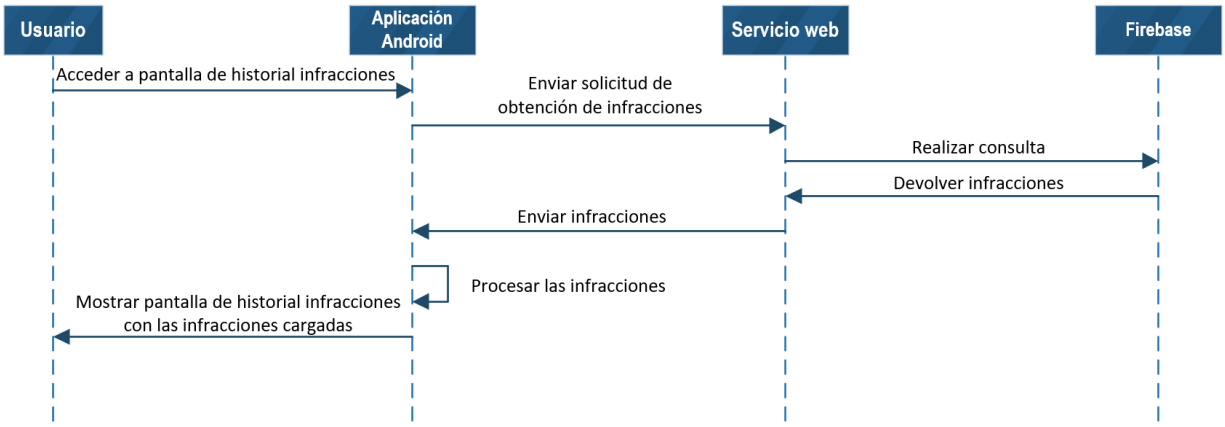



Ilustración 17: Mostrar historial de infracciones – diagrama de secuencia.

### 4.1.1 Estructura y funcionalidad

En este apartado se detalla la estructura y la funcionalidad del servicio web Firebase.

El proyecto está formado por los paquetes y las clases que se muestran en la ilustración 18.

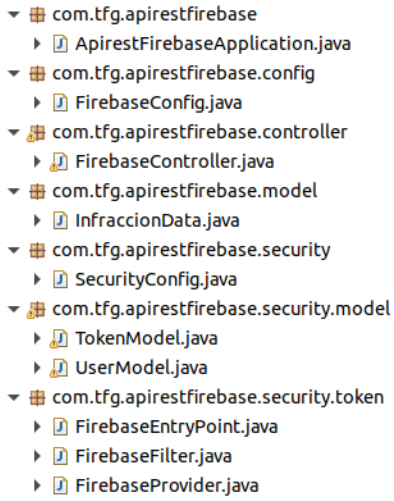



Ilustración 18: Paquetes y clases del servicio web Firebase.

- **ApirestFirebaseApplication (ilustración 19)**

  Es la clase principal de la aplicación. La anotación "@SpringBootApplication" indica que es una clase de aplicación Spring Boot y configura automáticamente la aplicación. El método "main" llama al método "run" de "SpringApplication" para iniciar la aplicación.

```java
package com.tfg.apirestfirebase;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class ApirestFirebaseApplication {

    public static void main(String[] args) {
        SpringApplication.run(ApirestFirebaseApplication.class, args);
    }

}
```

Ilustración 19: ApirestFirebaseApplication.

- **FirebaseConfig (ilustración 20)**

  Ésta es la clase de configuración para la inicialización de Firebase, como indica la anotación "@Configuration". El método "init" es anotado con "@PostConstruct", lo que significa que se ejecuta después de que la clase haya sido construida. Este método construye una instancia de "FirebaseOptions" utilizando las credenciales de Google y la URL de la realtime database de Firebase. Por último, se inicializa la aplicación de Firebase llamando al método "initializeApp".

```java
package com.tfg.apirestfirebase.config;

import java.io.IOException;

@Configuration
public class FirebaseConfig {

    private static final Logger logger = LoggerFactory.getLogger(FirebaseConfig.class);

    @Value("${firebase.config.path}")
    private String configPath;

    @PostConstruct
    public void init() throws IOException {
        ClassPathResource resource = new ClassPathResource(configPath);

        FirebaseOptions options = FirebaseOptions.builder()
                .setCredentials(GoogleCredentials.fromStream(resource.getInputStream()))
                .setDatabaseUrl("https://tfg-login-8ea38-default-rtdb.europe-west1.firebasedatabase.app")
                .build();

            FirebaseApp.initializeApp(options);
            logger.info("App name: {}", FirebaseApp.getInstance().getName());
    }
}
```

Ilustración 20: FirebaseConfig.

- **InfraccionData (ilustración 21)**

  Esta clase es un modelo que representa los datos de una infracción, los cuales son: fecha, longitud, latitud, velocidad y velocidad máxima. En la ilustración 21 no se muestra la clase completa, pero todos los atributos tienen el método getter y setter.

```java
package com.tfg.apirestfirebase.model;

public class InfraccionData {
     private String fecha;
        private String longitud;
        private String latitud;
        private String velocidad;
        private String velocidadMaxima;


        public InfraccionData(String fecha, String longitud, String latitud, String velocidad, String velocidadMaxima) {
            this.fecha = fecha;
            this.longitud = longitud;
            this.latitud = latitud;
            this.velocidad = velocidad;
            this.velocidadMaxima = velocidadMaxima;
        }

        public String getFecha() {
            return fecha;
        }

        public void setFecha(String fecha) {
            this.fecha = fecha;
        }
```

Ilustración 21: InfraccionData – Firebase.

- **SecurityConfig (ilustración 22)**

  Esta clase configura la seguridad de la aplicación, como indica su anotación "@Configuration" y "@EnableWebSecurity". Esta clase extiende de "WebSecurityConfigurerAdapter", que es una clase base proporcionada por Spring Security para facilitar la configuración de la seguridad en una aplicación web. El método "configure" es donde se configuran las reglas de seguridad para las solicitudes HTTP. Se configura un filtro de seguridad y proveedor de autenticación personalizados, "FirebaseFilter" y "FirebaseProvider" respectivamente.

```java
package com.tfg.apirestfirebase.security;

import org.springframework.beans.factory.annotation.Autowired;

@Configuration
public class SecurityConfig {

    @Autowired
    FirebaseEntryPoint entryPoint;

    @Autowired
    FirebaseProvider provider;

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf().disable().authorizeRequests().anyRequest().authenticated();
        http.exceptionHandling().authenticationEntryPoint(entryPoint);
        http.addFilterBefore(new FirebaseFilter(), BasicAuthenticationFilter.class);
        http.authenticationProvider(provider);
        return http.build();
    }

}
```

Ilustración 22: SecurityConfig.

- **TokenModel (ilustración 23)**

  Es una clase que implementa "AbstractAuthenticationToken" y representa un token de autenticación. Tiene una propiedad para almacenar el token.

```java
package com.tfg.apirestfirebase.security.model;

import javax.security.auth.Subject;

public class TokenModel extends AbstractAuthenticationToken {

    private String token;

    public TokenModel(String token) {
        super(null);
        this.token = token;
    }

    public String getToken() {
        return token;
    }

    @Override
    public Object getCredentials() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Object getPrincipal() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public boolean implies(Subject subject) {
        // TODO Auto-generated method stub
        return super.implies(subject);
    }

}
```

Ilustración 23: TokenModel.

- **UserModel (ilustración 24)**

  Esta clase implementa la interfaz "Authentication" y representa un usuario autenticado. Tiene una propiedad para almacenar el registro de usuario de Firebase e implementa los métodos de la interfaz "Authentication" para acceder a la información del usuario.

```java
package com.tfg.apirestfirebase.security.model;

import java.util.Collection;

public class UserModel implements Authentication{

    private UserRecord userRecord;

    public UserModel(UserRecord userRecord) {
        this.userRecord = userRecord;
    }

    @Override
    public String getName() {
        return userRecord.getDisplayName();
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Object getCredentials() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Object getDetails() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Object getPrincipal() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public boolean isAuthenticated() {
```

Ilustración 24: UserModel.

- **FirebaseEntryPoint (ilustración 25)**

  Esta clase implementa el método "commence()" de la interfaz "AuthenticationEntryPoint". Este método se invoca cuando una solicitud no está autenticada y debe ser manejada por el "AuthenticationEntryPoint". En este método, se responde con un error de autenticación.

```java
package com.tfg.apirestfirebase.security.token;

import java.io.IOException;

@Component
public class FirebaseEntryPoint implements AuthenticationEntryPoint {

    private static final Logger logger = LoggerFactory.getLogger(FirebaseEntryPoint.class);

    @Override
    public void commence(HttpServletRequest req, HttpServletResponse res, AuthenticationException e) throws IOException, ServletException {
        logger.error("fail in commence");

        res.sendError(HttpServletResponse.SC_UNAUTHORIZED, "unauthorized 401");
    }
}
```

Ilustración 25: FirebaseEntryPoint.

- **FirebaseFilter (ilustración 26)**

  Esta clase es un filtro de servlet que intercepta las solicitudes HTTP y extrae el token de autenticación de la cabecera de autorización. El método "doFilterInternal" se ejecuta para cada solicitud y extrae el token de la cabecera de autorización. Luego, crea una instancia de "TokenModel" y la establece en el contexto de seguridad utilizando "SecurityContextHolder".

```java
package com.tfg.apirestfirebase.security.token;

import java.io.IOException;

public class FirebaseFilter extends OncePerRequestFilter{

    private static final Logger logger = LoggerFactory.getLogger(FirebaseFilter.class);

    @Override
    protected void doFilterInternal(HttpServletRequest req, HttpServletResponse res, FilterChain chain)throws ServletException, IOException {
        try {
            String token = getToken(req);
            if (token != null) {
                SecurityContextHolder.getContext().setAuthentication(new TokenModel(token));
            }
        } catch (Exception e) {
            logger.error("Fail in do filter: ", e.getMessage());
        }

        chain.doFilter(req, res);
    }

    private String getToken(HttpServletRequest request) {
        String header = request.getHeader("Authorization");
        if (header != null && header.startsWith("Bearer ")) {
            return header.replace("Bearer ", "");
        }

        return null;
    }
}
```

Ilustración 26: FirebaseFilter.

- **FirebaseProvider (ilustración 27)**

  Esta clase es un proveedor de autenticación personalizado que utiliza el SDK de Firebase para verificar y autenticar el token de autenticación. Implementa la interfaz "AuthenticationProvider" de Spring Security. El método "supports" verifica si la clase de autenticación es compatible con este proveedor. Y el método "authenticate" se encarga de la autenticación del token utilizando el SDK de Firebase.

```java
package com.tfg.apirestfirebase.security.token;

import org.slf4j.Logger;

@Component
public class FirebaseProvider implements AuthenticationProvider{

    private static final Logger logger = LoggerFactory.getLogger(FirebaseProvider.class);

    @Override
    public Authentication authenticate(Authentication authentication) throws AuthenticationException {
        TokenModel token = (TokenModel) authentication;
        try {
            FirebaseToken firebaseToken = FirebaseAuth.getInstance().verifyIdToken(token.getToken(), true);
            String uid = firebaseToken.getUid();
            UserRecord userRecord = FirebaseAuth.getInstance().getUser(uid);
            return new UserModel(userRecord);
        } catch (FirebaseAuthException e) {
            logger.error("Fail: {}", getErrorCode(e.getAuthErrorCode()));
        }
        return null;
    }

    @Override
    public boolean supports(Class<?> authentication) {
        return TokenModel.class.isAssignableFrom(authentication);
    }

    private String getErrorCode(AuthErrorCode errorCode) {
        String error;
        switch (errorCode.toString()) {
            case "EXPIRED_ID_TOKE":
                error = "token expired";
                break;
            case "INVALID_ID_TOKE":
                error = "token invalid";
                break;
            case "REVOKED_ID_TOKE":
                error = "token revoked";
                break;
            default:
                error = "authentication fail";
                break;
```

Ilustración 27: FirebaseProvider.

- **FirebaseController**

  Esta clase es un controlador REST que maneja las solicitudes relacionadas con Firebase, como indica la anotación “@RestController”.

  - El método “getMatriculaByUserId” maneja una solicitud GET a "/usuarios/{userId}/matricula" y devuelve la matrícula asociada al usuario con el ID especificado, ilustración 28.
  - El método “getInfraccionesByUserId” maneja una solicitud GET a "/usuarios/{userId}/infracciones" y devuelve una lista de infracciones asociadas al usuario con el ID especificado, ilustración 29.
  - El método “setMatriculaByUserId” maneja una solicitud POST a "/usuarios/{userId}/matricula" y almacena la matrícula para el usuario con el ID especificado, ilustración 30.
  - El método “setInfraccionByUserId” maneja una solicitud POST a "/usuarios/{userId}/infracciones" y almacena una nueva infracción para el usuario con el ID especificado, ilustración 31.

```java
package com.tfg.apirestfirebase.controller;

import java.util.ArrayList;

@RestController
public class FirebaseController {

    private static final Logger logger = LoggerFactory.getLogger(FirebaseController.class);

    @GetMapping("/usuarios/{userId}/matricula")
    public ResponseEntity<String> getMatriculaByUserId(@PathVariable String userId) throws InterruptedException, ExecutionException {
        DatabaseReference ref = FirebaseDatabase
                .getInstance("https://tfg-login-8ea38-default-rtdb.europe-west1.firebasedatabase.app")
                .getReference("usuarios/" + userId + "/matricula");

        CompletableFuture<String> future = new CompletableFuture<>();

        ref.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                String matricula = dataSnapshot.getValue(String.class);
                future.complete(matricula);
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                logger.info("Fallo al encontrar matricula: {}", databaseError.getMessage());
                future.completeExceptionally(databaseError.toException());
            }
        });

        String matricula = future.get();

        if (matricula != null) {
            logger.info("La matricula es: {}", matricula);
            return ResponseEntity.ok(matricula);
        } else {
            logger.info("No hay matricula");
            return ResponseEntity.notFound().build();
        }
    }
```

Ilustración 28: getMatriculaByUserId.

```java
    @GetMapping("/usuarios/{userId}/infracciones")
    public List<InfraccionData> getInfraccionesByUserId(@PathVariable String userId) {
        DatabaseReference infraccionesRef = FirebaseDatabase
                .getInstance("https://tfg-login-8ea38-default-rtdb.europe-west1.firebasedatabase.app")
                .getReference("usuarios").child(userId).child("infracciones");

        CompletableFuture<List<InfraccionData>> future = new CompletableFuture<>();

        infraccionesRef.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                List<InfraccionData> infracciones = new ArrayList<>();
                for (DataSnapshot infraccionSnapshot : dataSnapshot.getChildren()) {
                    Map<String,String> infraccion = (Map<String,String>) infraccionSnapshot.getValue();
                    InfraccionData infracciondata = new InfraccionData(
                            infraccion.get("fecha"),
                            infraccion.get("longitud"),
                            infraccion.get("latitud"),
                            infraccion.get("velocidad"),
                            infraccion.get("velocidadMaxima"));
                    infracciones.add(infracciondata);
                }

                future.complete(infracciones);
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                future.completeExceptionally(databaseError.toException());
            }
        });

        List<InfraccionData> infracciones = new ArrayList<>();

        try {
            infracciones = future.get();
            logger.info("infracciones: {}", infracciones);
            return infracciones;
        } catch(Exception e) {
            logger.info("fallo con future: {}", e.getMessage());
            return infracciones;
        }
```

Ilustración 29: getInfraccionesByUserId.

```java
@PostMapping("/usuarios/{userId}/matricula")
public ResponseEntity<Void> setMatriculaByUserId(@PathVariable String userId, @RequestBody String matricula) throws InterruptedException, ExecutionException {
    DatabaseReference ref = FirebaseDatabase
            .getInstance("https://tfg-login-8ea38-default-rtdb.europe-west1.firebasedatabase.app")
            .getReference("usuarios");
    try {
        ref.child(userId).child("matricula").setValue(matricula, null);
        logger.info("La matricula se ha cambiado: {}", matricula);
        return ResponseEntity.ok().build();
    } catch (Exception e) {
        logger.info("Error al cambiar matricula: {}", e.getMessage());
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }
}
```

Ilustración 30: setMatriculaByUserId.

```java
@PostMapping("/usuarios/{userId}/infracciones")
public ResponseEntity<Void> setInfraccionByUserId(@PathVariable String userId, @RequestBody InfraccionData infraccionData) throws InterruptedException, ExecutionExcept
    DatabaseReference ref = FirebaseDatabase
            .getInstance("https://tfg-login-8ea38-default-rtdb.europe-west1.firebasedatabase.app")
            .getReference("usuarios");
    try {
        DatabaseReference referenceInfracciones = ref.child(userId).child("infracciones");

        String nuevaInfraccionKey = referenceInfracciones.push().getKey();
        Map<String, Object> infracionValues = new HashMap<>();
        infracionValues.put("fecha", infraccionData.getFecha());
        infracionValues.put("latitud", infraccionData.getLatitud());
        infracionValues.put("longitud", infraccionData.getLongitud());
        infracionValues.put("velocidad", infraccionData.getVelocidad());
        infracionValues.put("velocidadMaxima", infraccionData.getVelocidadMaxima());

        referenceInfracciones.child(nuevaInfraccionKey)
        .setValue(infracionValues, null);

        logger.info("La infraccion se ha guardado correctamente");
        return ResponseEntity.ok().build();
    } catch (Exception e) {
        logger.info("Error al guardar la infraccion: {}", e.getMessage());
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }
}
```

Ilustración 31: setInfraccionByUserId.

A continuación, se muestran dos tablas (tabla 1 y tabla 2) con las consultas y almacenamientos de datos mediante solicitud GET y otra tabla con las de solicitud POST:

| URI | Parámetros | Descripción |
|---|---|---|
| /usuarios/{parámetro}/matricula | “userId”: Id del usuario | Obtiene la matrícula asociada al Id del usuario |
| /usuarios/{parámetro}/infracciones | “userId”: Id del usuario | Obtiene las infracciones asociadas al Id del usuario. |

Tabla 1: Consultas de datos mediante GET – servicio web Firebase.

| URI | Parámetros | Descripción |
|---|---|---|
| /usuarios/{parámetro}/matricula | “userId”: Id del usuario | Almacena la matrícula del cuerpo de la solicitud asociándola al Id del usuario. |
| /usuarios/{parámetro}/infracciones | “userId”: Id del usuario | Almacena la infracción del cuerpo de la solicitud asociándola al Id del usuario. |

Tabla 2: Almacenamientos de datos mediante POST – servicio web Firebase.

## 4.2 Servicio Web Kafka

Este servicio web sirve para recibir la infracción de la aplicación Android y enviarla al topic configurado de Kafka. En este punto además se detalla la configuración necesaria de esta tecnología para el proyecto.

### 4.2.1 Estructura y funcionalidad

En este apartado se detalla la estructura y la funcionalidad del servicio web Kafka.

El proyecto está formado por los paquetes y clases que se muestran en la ilustración 32.

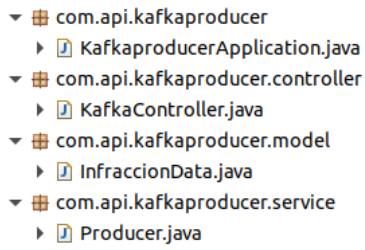



Ilustración 32: Paquetes y clases del servicio web Kafka.

- **KafkaproducerApplication (ilustración 33)**

  Es la clase principal de la aplicación. La anotación "@SpringBootApplication" indica que es una clase de aplicación Spring Boot y configura automáticamente la aplicación. El método "main" llama al método "run" de "SpringApplication" para iniciar la aplicación.

```
1  package com.api.kafkaproducer;
2
3  import org.springframework.boot.SpringApplication;
5
6  @SpringBootApplication
7  public class KafkaproducerApplication {
8
9      public static void main(String[] args) {
10         SpringApplication.run(KafkaproducerApplication.class, args);
11     }
12
13 }
```

Ilustración 33: KafkaproducerApplication.

- **InfraccionData (ilustración 34)**

  Esta clase es un modelo que representa los datos de una infracción, los cuales son: latitud, longitud, matrícula y velocidad.

```java
package com.api.kafkaproducer.model;

public class InfraccionData {
    private String lat;
    private String lon;
    private String mat;
    private String vel;

    public InfraccionData(String lat, String lon, String mat, String vel) {
        this.lat = lat;
        this.lon = lon;
        this.mat = mat;
        this.vel = vel;
    }

    public String getLat() {
        return lat;
    }
    public void setLat(String lat) {
        this.lat = lat;
    }
    public String getLon() {
        return lon;
    }
    public void setLon(String lon) {
        this.lon = lon;
    }
    public String getMat() {
        return mat;
    }
    public void setMat(String mat) {
        this.mat = mat;
    }
    public String getVel() {
        return vel;
    }
    public void setVel(String vel) {
        this.vel = vel;
    }
}
```

Ilustración 34: InfraccionData – Kafka.

- **Producer (infracción 35)**

  Esta clase es un servicio, "@Service" que se encarga de enviar el objeto "InfraccionData" a Kafka. El método "sendInfraccion" recibe un "topic" de Kafka y un objeto "InfraccionData" y utiliza un "KafkaTemplate" para enviar el objeto al topic especificado. Este servicio se utiliza en el controlador "KafkaController" para enviar la infracción recibida al topic de Kafka para su procesamiento posterior.

```java
package com.api.kafkaproducer.service;

import org.springframework.beans.factory.annotation.Autowired;

@Service
public class Producer {

    @Autowired
    private KafkaTemplate<String, InfraccionData> kafkaCoor;

    public void sendInfraccion(String kafkaTopic, InfraccionData infraccion) {
        System.out.println("Publishing to topic " + kafkaTopic);
        kafkaCoor.send(kafkaTopic, infraccion);
    }
}
```

Ilustración 35: Producer.

- **KafkaController (ilustración 36)**

  Esta clase es un controlador de API REST, "@RestController" que maneja las solicitudes relacionadas con el envío de infracciones a Kafka. El método "sendInfracciones" es una solicitud POST mapeada a la ruta "/kafkaproducer/infracciones". Recibe los parámetros: latitud, longitud, matrícula y velocidad. Luego, crea un objeto "InfraccionData" con los valores recibidos y lo envía a Kafka a través del productor para su procesamiento y publicación en un topic específico.

```java
package com.api.kafkaproducer.controller;

import org.springframework.beans.factory.annotation.Autowired;

@RestController
@RequestMapping("/kafkaproducer")
@CrossOrigin(origins = "*", methods= {RequestMethod.GET,RequestMethod.PUT,RequestMethod.POST})
public class KafkaController {

    @Autowired
    private Producer producer;

    //KafkaPost
    @PostMapping("/infracciones")
    public void sendInfracciones(@RequestParam("lat") String lat, @RequestParam("lon") String lon,
            @RequestParam("mat") String mat, @RequestParam("vel") String vel) {
        InfraccionData geo = new InfraccionData(lat, lon, mat, vel);
        producer.sendInfraccion("prueba",geo);
    }
}
```

Ilustración 36: KafkaController.

A continuación, se muestra una tabla (tabla 3) con la solicitud POST:

| URI | Parámetros | Descripción |
|---|---|---|
| / kafkaproducer/ infracciones | “lat”: latitud<br>“lon”: longitud<br>“mat”: matrícula<br>“vel”: velocidad | Almacena la infracción en el topic especificado de Kafka. |

Tabla 3: Almacenamiento de datos mediante POST – servicio web Kafka.

### 4.2.2 Configuración de Kafka

En este punto se detalla el fichero .yaml que se ha creado para el uso de esta tecnología, como crear un topic desde su interfaz web, Kafka-Manager, y la configuración necesaria para crear el productor desde el servicio web de Kafka.

#### 4.2.2.1 Fichero .yaml

El fichero .yaml, que se muestra en la ilustración 37, detalla la configuración de tres servicios necesarios para el uso de Kafka, con sus respectivas imágenes de Docker y configuraciones específicas. Este fichero se ha usado con la herramienta Docker-Compose, de la cual se ha hablado en el apartado 3.2, para crear y administrar los contenedores y servicios relacionados.

```yaml
version: "3"
services:
  zookeeper:
    image: zookeeper
    restart: always
    container_name: zookeeper
    hostname: zookeeper
    ports:
      - 2181:2181
    environment:
      ZOO_MY_ID: 1
  kafka:
    image: wurstmeister/kafka
    container_name: kafka
    ports:
    - 9092:9092
    environment:
      KAFKA_ADVERTISED_HOST_NAME: 192.168.206.130
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
  kafka_manager:
    image: hlebalbau/kafka-manager:stable
    container_name: kakfa-manager
    restart: always
    ports:
      - "9000:9000"
    environment:
      ZK_HOSTS: "zookeeper:2181"
      APPLICATION_SECRET: "random-secret"
    command: -Dpidfile.path=/dev/null
```

Ilustración 37: Fichero .yaml de Kafka.

A continuación, se detalla la utilidad de cada campo:

- versión: Indica la versión de la sintaxis del archivo .yaml que se está utilizando. En este caso, se utiliza la versión 3.
- services: Define los servicios que se ejecutarán en el entorno, en este caso: zookeeper, kafka y kafka_manager.
    - zookeeper: Configuración del servicio ZooKeeper.
        - image: zookeeper: Especifica la imagen de Docker utilizada para el contenedor de ZooKeeper.
        - restart: always: Indica que el contenedor se reiniciará automáticamente en caso de fallo.
        - container_name: zookeeper: Asigna un nombre al contenedor.
        - hostname: zookeeper: Configura el nombre de host del contenedor.
        - ports: 2181:2181: Mapea el puerto 2181 del host al puerto 2181 del contenedor.
        - environment: ZOO_MY_ID: 1: Establece una variable de entorno dentro del contenedor.
    - kafka: Configuración del servicio Kafka.
        - image: wurstmeister/kafka: Especifica la imagen de Docker utilizada para el contenedor de Kafka.

- container_name: kafka: Asigna un nombre al contenedor.
- ports: 9092:9092: Mapea el puerto 9092 del host al puerto 9092 del contenedor.
- environment: KAFKA_ADVERTISED_HOST_NAME: 192.168.206.130: Configura la dirección IP para Kafka.
- environment: KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181: Especifica la dirección y el puerto de ZooKeeper al que Kafka se conectará.

o kafka_manager: Configuración del servicio Kafka Manager.

- image: hlebalbau/kafka-manager:stable: Especifica la imagen de Docker utilizada para el contenedor de Kafka Manager.
- container_name: kafka-manager: Asigna un nombre al contenedor.
- restart: always: Indica que el contenedor se reiniciará automáticamente en caso de fallo.
- ports: 9000:9000: Mapea el puerto 9000 del host al puerto 9000 del contenedor.
- environment: ZK_HOSTS: "zookeeper:2181": Especifica la dirección y el puerto de ZooKeeper al que Kafka Manager se conectará.
- environment: APPLICATION_SECRET: "random-secret": Establece una clave de aplicación como variable de entorno.
- command: -Dpidfile.path=/dev/null: Configura un comando personalizado para ejecutar el contenedor, el cual asegura que el archivo de identificación no se almacene ni consuma espacio del sistema de archivos.

#### 4.2.2.2 Crear topic con Kafka-Manager

Existen dos formas para la creación de topics: mediante el uso de línea de comandos o con el uso de la interfaz gráfica, Kafka-manager. En este proyecto se ha optado por la segunda opción, ya que es más visual y fácil de usar. A continuación, se detallan los pasos:

1 Se inicia los tres servicios mencionados anteriormente en el fichero .yaml con el comando: sudo docker-compose -f <fichero_Kafka.yaml> up

2 En el navegador web, en este caso Google Chrome, se introduce la dirección IP de Kafka y el puerto configurado para Kafka-manager, en este caso: 192.168.206.130:9000.

3 En la pestaña superior de cluster se selecciona Add cluster.

4 Se introduce el nombre del cluster y los datos que se detallan en la ilustración 38. Luego se presiona “Save” y Go to cluster view.

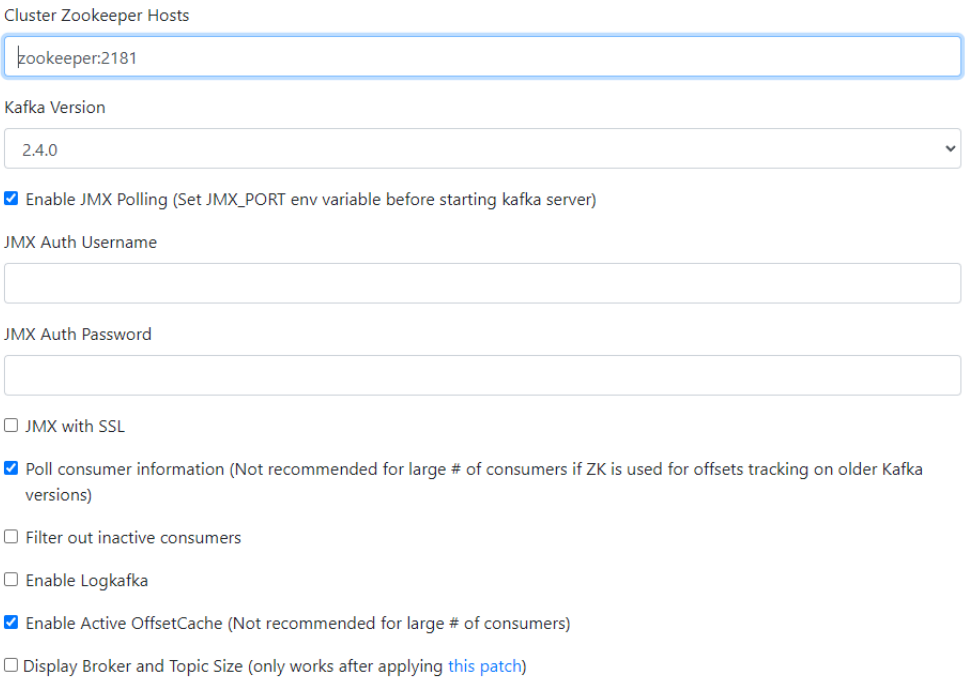



Ilustración 38: Detalles cluster Kafka.

5 En la pestaña de cluster, se selecciona “topic” y “create”. Se introduce el nombre del topic y se le da a “Create”.

Con esto, ya se habría creado el topic al cual el servicio web enviará información como productor y ELK obtendrá la información como suscriptor.

#### 4.2.2.3 Configuración del servicio web como productor de Kafka

A continuación, se detalla la configuración necesaria del servicio web para que actúe como productor Kafka:

1 Añadir las dependencias que se muestran en la ilustración 39 al proyecto del servicio web.

```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka-test</artifactId>
    <scope>test</scope>
</dependency>
```

Ilustración 39: Dependencias pom.xml Kafka.

2 Añadir las propiedades que se muestran en la ilustración 40 al fichero application.properties. A continuación, se muestra una explicación de dichas propiedades:

a. spring.kafka.producer.bootstrap-servers: configura la dirección y puerto del servidor Kafka a los que el productor se conectará.

b. spring.kafka.producer.key-serializer: configura el serializador que se utilizará para la clave del mensaje que se enviará al topic de Kafka. En este caso un String.

c. spring.kafka.producer.value-serializer: configura el serializador que se utilizará para el valor

del mensaje que se enviará al topic de Kafka. En este caso un objeto Java que se serializa a formato JSON.

```
1 spring.kafka.producer.bootstrap-servers=192.168.206.130:9092
2 spring.kafka.producer.key-serializer: org.apache.kafka.common.serialization.StringSerializer
3 spring.kafka.producer.value-serializer: org.springframework.kafka.support.serializer.JsonSerializer
```

Ilustración 40: Application.properties servicio web Kafka.

# 5 APLICACIÓN DESARROLLADA: APLICACIÓN ANDROID

En este capítulo se describe la parte del proyecto de la Aplicación Android. La aplicación Android ha sido llamada como "ELKaf Velocity". La configuración necesaria de la aplicación Android relacionada con Firebase se detalla en el Anexo A.

A continuación, se muestra un diagrama con los casos de uso de la Aplicación Android, ilustración 41.

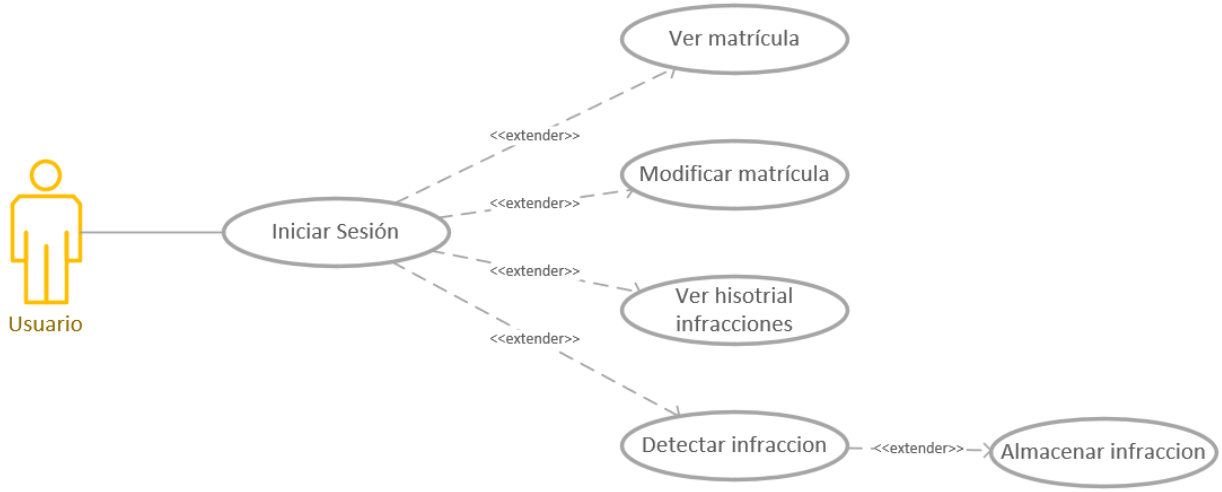



Ilustración 41: Diagrama de casos de uso – aplicación Android.

## 5.1 Estructura y funcionalidad

En este apartado se detalla la estructura de la aplicación y las clases de ésta, ilustración 42.

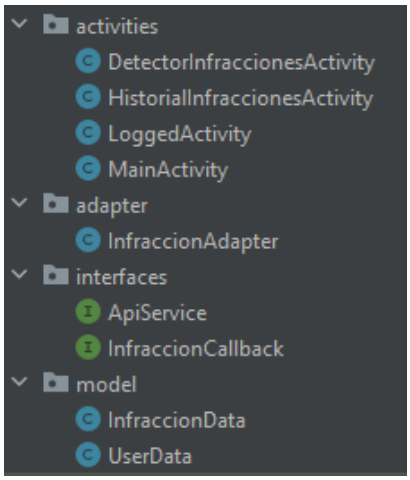



Ilustración 42: Paquetes y clases – aplicación Android.

A diferencia del capítulo anterior, en este apartado no se pondrán ilustraciones de las clases, ya que son clases más extensas, pero se detallarán los métodos y su utilidad de forma clara.

Se tienen dos clases que se usan como modelo para el usuario y para la infracción:

- **UserData**

  La clase UserData es una clase modelo que representa los datos del usuario registrado con los siguientes atributos:

  - **userId:** Id del usuario.
  - **token:** token asociado al usuario.
  - **matricula:** matrícula asociada al usuario.

  La clase también contiene los métodos getter y setter de dichos atributos y un constructor con todos los atributos. Además, esta clase implementa la interfaz "Serializable", lo que significa que los objetos de esta clase se pueden convertir en una secuencia de bytes para ser transferidos o almacenados y luego se pueden reconstruir a partir de esa secuencia de bytes.

- **InfraccionData**

  La clase InfraccionData es una clase modelo que representa los datos de la infracción cometido con los siguientes atributos:

  - **fecha:** fecha en la que se comete la infracción.
  - **longitud:** longitud en la que se comete la infracción.
  - **latitud:** latitud en la que se comete la infracción.
  - **velocidad:** velocidad a la que se comete la infracción.
  - **velocidadMáxima:** velocidad máxima de la vía en la que se comete la infracción.

  La clase también contiene los métodos getter y setter de dichos atributos y un constructor con todos los atributos.

En el paquete interfaces, existen dos clases:

- **ApiService**

  La clase ApiService es una interfaz que define los endpoints y los métodos para realizar llamadas a los servicios web de Firebase y de Kafka y al servicio web "overpass-api". A continuación, se detallan los métodos definidos en la interfaz:

- **Servicio web "overpass-api":**
  - **getVelMax(String url):** Realiza una solicitud GET a una URL específica del servicio web "overpass'api" y devuelve un objeto JsonObject en la respuesta, del cual se obtienen datos como la velocidad máxima y el nombre de la vía.
- **Servicio web Kafka:**
  - **sendKafka(String lat, String lon, String matricula, String velocidad):** Realiza una solicitud POST a la ruta "kafkaproducer/infracciones" con los parámetros de latitud (lat), longitud (lon), matrícula (matricula) y velocidad (velocidad).
- **Servicio web Firebase:**
  - **getMatriculaByUserId(String token, String userId):** Realiza una solicitud GET, a la ruta "/usuarios/{userId}/matricula" con el token de autorización (token) y el ID de usuario (userId). Devuelve la matrícula del usuario en la respuesta.
  - **setMatriculaByUserId(String token, String userId, String matricula):** Realiza una solicitud POST a la ruta "/usuarios/{userId}/matricula" con el token de autorización (token), el ID de usuario (userId) y la matrícula (matricula).
  - **getInfracciones(String token, String userId):** Realiza una solicitud GET a la ruta "/usuarios/{userId}/infracciones" con el token de autorización (token) y el ID de usuario (userId). Devuelve una lista de objetos "InfraccionData" que representan las infracciones del usuario en la respuesta.
  - **setInfraccion(String token, String userId, InfraccionData infraccionData):** Realiza una solicitud POST a la ruta "usuarios/{userId}/infracciones" con el token de autorización (token), el ID de usuario (userId) y un objeto InfraccionData (infraccionData).

La interfaz utiliza anotaciones de Retrofit para especificar el tipo de solicitud (GET o POST), las rutas de la API, los encabezados y los cuerpos de las solicitudes. Los tipos de respuesta esperados se especifican utilizando la clase Call de Retrofit junto con los tipos de datos correspondientes.

- **InfraccionCallback**

  La clase InfraccionCallback es una interfaz que define dos métodos de devolución de llamada (callback) para manejar los resultados de la obtención de infracciones y los errores en un contexto determinado. A continuación, se detallan los métodos definidos en la interfaz:

  - **onInfraccionesObtenidas(List<InfraccionData> infracciones):** Este método se llama cuando se han obtenido las infracciones de manera exitosa. Recibe como parámetro una lista de objetos InfraccionData que representan las infracciones obtenidas.
  - **onError(String mensajeError):** Este método se llama cuando se produce un error durante la obtención de las infracciones. Recibe como parámetro un mensaje de error que describe la causa del error.

  Esta interfaz se usa en la clase "HistorialInfraccionesActivity" que se verá a continuación.

En el paquete adapter, existe la clase:

- **InfraccionAdapter**

  La clase InfraccionAdapter es un adaptador personalizado utilizado en el contexto de un RecyclerView para mostrar una lista de infracciones en la pantalla "Historial Infracciones". A continuación, se detalla su funcionamiento:

  La clase extiende de RecyclerView.Adapter<RecyclerView.ViewHolder>, lo que indica que se

utilizarán elementos de vista "ViewHolder" dentro de un "RecyclerView".

El adaptador recibe una lista de objetos "InfraccionData" en su constructor, que representa las infracciones que se mostrarán en la lista.

- Los métodos "onCreateViewHolder", "onBindViewHolder" y "getItemCount" son métodos obligatorios de la clase RecyclerView.Adapter que deben implementarse.
- La clase InfraccionAdapter contiene dos clases internas estáticas: HeaderViewHolder y InfraccionViewHolder, que extienden de RecyclerView.ViewHolder. Estas clases se utilizan para mantener las referencias a los elementos de vista en las vistas de encabezado (título de las columnas) y de infracción (valores de las columnas) respectivamente. En el constructor de cada clase, se obtienen las referencias a los elementos de vista "TextView" mediante el método "findViewById()".
- **onCreateViewHolder(@NotNull ViewGroup parent, int viewType):** en este método se rellena el diseño de vista correspondiente según el tipo de vista "viewType". Si viewType es igual a HEADER_VIEW, se rellena el diseño del título de las columnas "item_encabezado", de lo contrario, se rellena el diseño de los valores de las columnas "item_infraccion". Luego, se crea y devuelve un objeto ViewHolder correspondiente al diseño rellenado.
- **onBindViewHolder(@NotNull RecyclerView.ViewHolder holder, int position):** en este método se asignan los datos de las infracciones a los elementos de vista del "ViewHolder". Si el "ViewHolder" es una instancia de "InfraccionViewHolder", se obtiene la infracción de la lista según la posición y se asignan los valores correspondientes a los elementos de vista "textFecha", "textPosicion", "textVelocidad" y "textVelocidadMaxima".
- **getItemCount()**: este método devuelve el número total de elementos en la lista de infracciones más uno, ya que se agrega una vista de encabezado adicional para el título de las columnas.
- **getItemViewType(int position):** este método determina el tipo de vista para una posición determinada. Si la posición es cero, se devuelve HEADER_VIEW (títulos de las columnas), de lo contrario, se devuelve ITEM_VIEW (valores de las columnas).

A continuación, detallamos las clases Activity:

- **MainActivity**

  La clase MainActivity es una actividad de Android que se utiliza como pantalla inicial de la aplicación. A continuación, se detallan los métodos y sus funciones en la clase:

  Esta clase contiene una instancia de "ActivityResultLauncher" llamada "resultLauncher". Esta instancia se utiliza para registrar y manejar el resultado de la actividad de inicio de sesión de Google. El resultado se recibe en el método "onActivityResult()" del "ActivityResultCallback", donde se obtiene la cuenta de Google firmada y se llama al método "**firebaseAuthWithGoogle()**" para realizar la autenticación en Firebase

  - **onCreate(Bundle savedInstanceState):** Este método se llama cuando se crea la actividad. Aquí se inicializan y configuran los componentes de la interfaz de usuario, se asocia al botón el inicio de sesión, usando el método "resultLauncher.launch()" y se crea una instancia del cliente de inicio de sesión de Google "mGoogleSignInClient" y de Firebase Authentication "mAuth".
  - **onStart():** Este método se llama cuando la actividad se vuelve visible para el usuario. Aquí se obtiene el usuario actualmente autenticado y se llama al método updateUI() para actualizar la interfaz de usuario en consecuencia.

- **firebaseAuthWithGoogle(String idToken):** Este método se utiliza para autenticar al usuario en Firebase utilizando las credenciales de Google. Recibe un token de ID de Google como parámetro y llama al método "signInWithCredential()" de "FirebaseAuth" para realizar la autenticación. Si la autenticación es exitosa, se llama al método "updateUI()" con el usuario autenticado.
- **updateUI(FirebaseUser user):** Este método se utiliza para actualizar la interfaz de usuario según el estado de autenticación del usuario. Si hay un usuario autenticado, se inicia la actividad "LoggedActivity" y se pasa a la pantalla LoggedActivity.

- **LoggedActivity**

  La clase LoggedActivity es una actividad de Android que representa la pantalla principal de la aplicación una vez que el usuario ha iniciado sesión. A continuación, se detallan los métodos:

  - **onCreate(Bundle savedInstanceState):** Este método se llama cuando se crea la actividad. Aquí se inicializan y configuran los componentes de la interfaz de usuario, se obtiene el usuario actualmente autenticado de Firebase, se muestra la matrícula del usuario y se configuran los listeners de los botones.
  - **showMatricula(String token, String userId):** Este método utiliza Retrofit para realizar una solicitud HTTP al servicio web Firebase y obtener la matrícula del usuario. La matrícula se muestra en el campo de entrada de texto "matriculaTextInput" y se almacena en UserData.
  - **saveMatricula():** Este método utiliza Retrofit para enviar una solicitud HTTP al servicio web Firebase y guardar la matrícula del usuario. La matrícula se obtiene del campo de entrada de texto "matriculaTextInput" y se almacena en UserData.
  - **signOut():** Este método se utiliza para cerrar la sesión del usuario. Llama al método signOut() de FirebaseAuth para cerrar la sesión actual, luego inicia la actividad "MainActivity" y nos devuelve a la pantalla inicial.
  - **irADeteccion(View view):** Este método se llama cuando se hace clic en el botón "iraDeteccion". Crea una nueva instancia de la actividad "DetectorInfraccionesActivity" y pasa los datos de usuario "userData" a través de un intent.
  - **irAInfracciones(View view):** Este método se llama cuando se hace clic en el botón "mostrarInfracciones". Crea una nueva instancia de la actividad "HistorialInfraccionesActivity" y pasa los datos de usuario "userData" a través de un intent.
  - **setUserData(String token, String userId, TextInputEditText matriculaTextInput):** Este método se utiliza para configurar los datos de usuario "userData". Crea una nueva instancia de UserData con el ID de usuario, el token y la matrícula proporcionados.
  - **showToast(String message):** Este método se utiliza para mostrar un mensaje emergente (Toast) en la pantalla con el mensaje proporcionado.

- **HistorialInfraccionesActivity**

  La clase HistorialInfraccionesActivity es una actividad de Android que muestra el historial de infracciones de un usuario. A continuación, se detallan los métodos de la clase:

  - **onCreate(Bundle savedInstanceState):** en este método se obtiene el objeto "UserData" enviado desde la actividad anterior a través del intent. Luego, se llama al método "getInfracciones" para obtener las infracciones del usuario utilizando el

token y el ID de usuario.

- **goBack():** este método se utiliza para regresar a la actividad anterior (LoggedActivity) cuando se hace clic en el botón "singOut".
- **getInfracciones(String token, String userId, InfraccionCallback callback):** este método se encarga de realizar una solicitud GET al servicio web Firebase para obtener las infracciones del usuario utilizando Retrofit. Se crea una instancia de "ApiService" y se realiza una llamada a "getInfracciones" pasando el token y el ID de usuario. La respuesta se maneja en los métodos onResponse y onFailure que hacen uso del método "onInfraccionesObtenidas" y "OnError" de la interfaz "InfraccionCallback", respectivamente.
- La implementación del método **onInfraccionesObtenidas** de la interfaz InfraccionCallback se llama cuando se obtienen las infracciones de manera exitosa. En este método, las infracciones se ordenan por fecha de manera descendente utilizando un comparador, y luego se crea y asigna el adaptador "InfraccionAdapter" al RecyclerView para mostrar las infracciones en forma de lista.
- La implementación del método **onError** de la interfaz InfraccionCallback se llama cuando se produce un error durante la obtención de las infracciones. En este caso, se crea una lista de infracciones de error y se asigna al RecyclerView un adaptador que muestra el mensaje de error.

- **DetectorInfraccionesActivity**

  La clase "DetectorInfraccionesActivity" es una actividad de Android que se encarga de detectar infracciones de velocidad utilizando la ubicación del dispositivo y enviar la infracción a Firebase y Kafka, a través de los servicios web anteriormente comentados. A continuación, está la descripción de los métodos:

  - **onCreate(Bundle savedInstanceState):** Método que se llama cuando se crea la actividad. Aquí se inicializan los componentes de la interfaz de usuario, se configuran los listeners y se obtiene "UserData" de la actividad anterior.
  - **goBack():** Método que se llama cuando se hace clic en el botón "singOut". Su funcionalidad es regresar a la actividad anterior para regresar a la pantalla principal.
  - **checkLocation():** Método que verifica si la ubicación está habilitada en el dispositivo usando el método "isLocationEnabled()". Si no lo está, muestra un cuadro de diálogo de alerta llamando al método "showAlert()".
  - **showAlert():** Método que muestra un cuadro de diálogo para alertar al usuario de que la ubicación está desactivada y le proporciona la opción de ir a la configuración de ubicación para habilitarla.
  - **isLocationEnabled():** Método que verifica si la ubicación está habilitada en el dispositivo. Devuelve true si está habilitada, y false en caso contrario.
  - **showData(View view):** Método que se llama cuando se hace clic en el botón "showData". Dependiendo del estado actual, comienza la detección de infracción o la para.
  - **locationListenerBest:** Implementación de "LocationListener" que se utiliza para obtener los cambios de ubicación cada 5 segundos o 50metros. Cuando se produce un cambio de ubicación, actualiza las variables de latitud, longitud, velocidad y se obtiene la velocidad máxima con el método "getVelMax". Una vez los datos actualizados, muestra los datos en la interfaz de usuario y verifica si se ha cometido una infracción de velocidad, en el caso de haberse cometido una infracción se llama a los métodos "sendKafka" y "sendFirebase", además de mostrar una alerta y registrarlo en el campo de infracción. En el caso contrario se muestra en el campo de

infracción que no se ha cometido ninguna infracción.

- **getVelMax(double lat, double lon):** Método que realiza una solicitud GET, al servicio web "overpass-api" utilizando Retrofit para obtener la velocidad máxima permitida en la ubicación actual. Construye la URL de la solicitud con la latitud y longitud proporcionadas, procesa la respuesta, actualiza la interfaz de usuario con la velocidad máxima obtenida y la devuelve. Se crea una instancia de "ApiService" y se realiza una llamada a "getVelMax".
- **sendKafka(String latv, String lonv, String velv):** este método se encarga de realizar una solicitud POST al servicio web Kafka para registrar la infracción utilizando Retrofit. Se crea una instancia de "ApiService" y se realiza una llamada a "sendKafka" pasando la latitud, longitud, matrícula y velocidad.
- **sendFirebase(String latv, String lonv, String velv, String velMaxv):** este método se encarga de realizar una solicitud POST al servicio web Firebase para registrar la infracción utilizando Retrofit. Se crea una instancia de "ApiService" y se realiza una llamada a "sendFirebase" pasando el token, userId y infraccionData.
- **showToast(String message):** Método que muestra un mensaje de texto corto en forma de Toast en la pantalla.

# 6 INTERFAZ DE APLICACIÓN ANDROID

En este capítulo se analiza cada una de las pantallas de la aplicación Android y las funcionalidades que ofrecen al usuario. Las pantallas de la aplicación Android se pueden agrupar en las de inicio de sesión, las que obtienen y envían datos al servicio web relacionado con Firebase y la encargada de detectar las infracciones de velocidad y enviar los datos al servicio web relacionado con Kafka.

Para no repetir esta información por cada pantalla, se define que, salvo la pantalla inicial, todas las pantallas poseen un botón en la parte superior izquierda que te permite volver a la pantalla anterior. Además, en la pantalla principal al volver a la pantalla inicial se cierra la sesión.

A continuación, se muestran y describen cada una de las pantallas de la aplicación Android.

## 6.1 Pantalla inicial

La pantalla inicial de la aplicación, ilustración 43, permite:

- Ir a la pantalla de inicio de sesión de Google, ilustración 44, la cual es propia de Google, ésta permite iniciar sesión con una cuenta ya existente o crear una cuenta para iniciar sesión con ella.

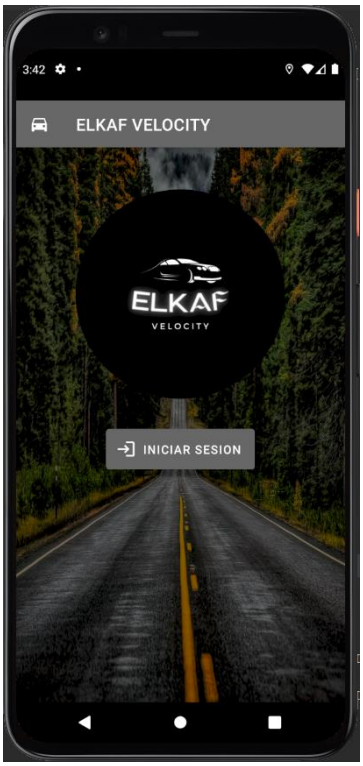



Ilustración 43: Pantalla inicial – App Android.

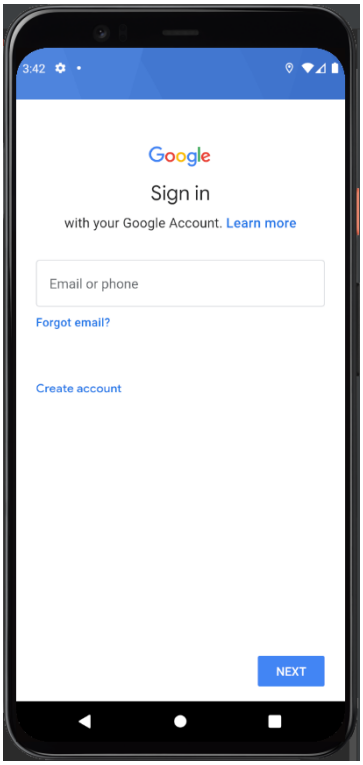



Ilustración 44: Pantalla inicio sesión Google – App Android.

## 6.2 Pantalla principal

La pantalla principal de la aplicación, ilustración 45, permite:

- Ir a la pantalla de detección de infracción.
- Ir a la pantalla del histórico de infracciones.
- Ver la matrícula que tiene asociada el usuario o en caso de nuevo usuario aparece el texto “Introduce Matrícula” en el campo “Matrícula”.
- Modificar la matrícula, para ello se debe modificar el texto que aparece en “Matrícula” y se presiona el botón “MODIFICAR MATRICULA”.

Las dos primeras opciones mostrarán dichas pantallas en el caso de que el usuario tenga una matrícula registrada, en caso contrario pedirá que se introduzca una matrícula.


Ilustración 45: Pantalla principal – App Android.

## 6.3 Pantalla historial de infracciones

La pantalla historial de infracciones, ilustración 46, permite:

- Visualizar todas las infracciones cometidas, ordenadas por fecha. Muestra la fecha, posición (latitud, longitud), velocidad a la que se cometió la infracción y la velocidad máxima.

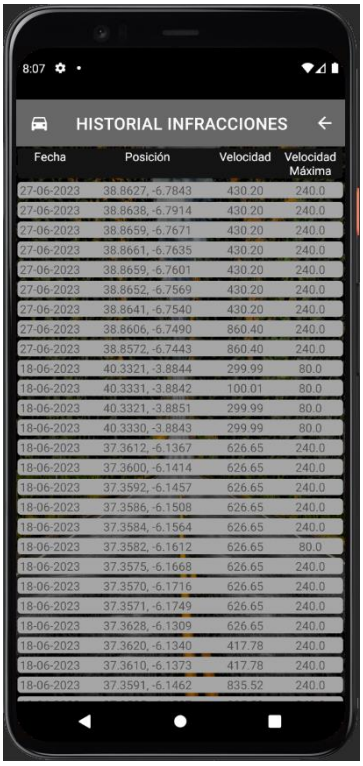



Ilustración 46: Pantalla historial de infracciones – App Android.

## 6.4 Pantalla detector infracción

La pantalla detectar infracción, ilustración 47, permite:

- Empezar/Parar la detección de infracción de velocidad.
- Ver los siguientes valores obtenidos por el GPS, que se actualizan cada 5 segundos o 50 metros recorridos:
    - Latitud en grados.
    - Longitud en grados.
    - Velocidad en forma de velocímetro, en km/h.
- Ver la velocidad máxima en km/h y el nombre de la vía obtenido del servicio web “overpass-api”, se actualiza al actualizarse los valores obtenidos por el GPS.
- Muestra en forma de notificación y, además, en un campo de texto si se ha cometido infracción o no con los últimos datos obtenidos por el GPS.
- Muestra el nombre de la vía por la que se circula.


Ilustración 47: Pantalla detectar infracción – App Android.

Aclarar sobre la ilustración 46, como se comentará en el capítulo 8, las pruebas se han realizado con el emulador de Android Studio por lo que se ha tenido que modificar la velocidad máxima, multiplicándola por dos, porque la velocidad más lenta del emulador ya superaba la velocidad máxima.

# 7 INTERFAZ DE KIBANA

En este capítulo se detallan dos de las pantallas de Kibana y las funcionalidades que ofrecen al usuario. Una de las pantallas de Kibana muestra los datos en forma de texto y otra de ellas muestra los datos en un mapa. Además, se detallará la configuración necesaria de Kibana para la creación de la pantalla del mapa.

Se comentan dos pantallas, vistas, que dispone Kibana para visualizar los datos. Las dos son en el apartado de la barra lateral izquierda "Analytics", una es la pantalla discover, que viene creada por defecto, y la otra Maps, la cual se indica cómo crearla en el último apartado de este capítulo.

También, ambas pantallas incorporan en la parte superior las siguientes opciones:

- Filtrado y consulta de datos en forma de lenguaje KQL[1] o en forma de seleccionable, ilustración 48, destacando en verde la forma de KQL y en rojo la forma de seleccionable.
- Filtrado y refresco de datos por tiempo. Permitiendo refrescar o mostrar los datos de los últimos segundos, minutos, horas, días, meses o años que se indiquen, ilustración 49 e ilustración 50.

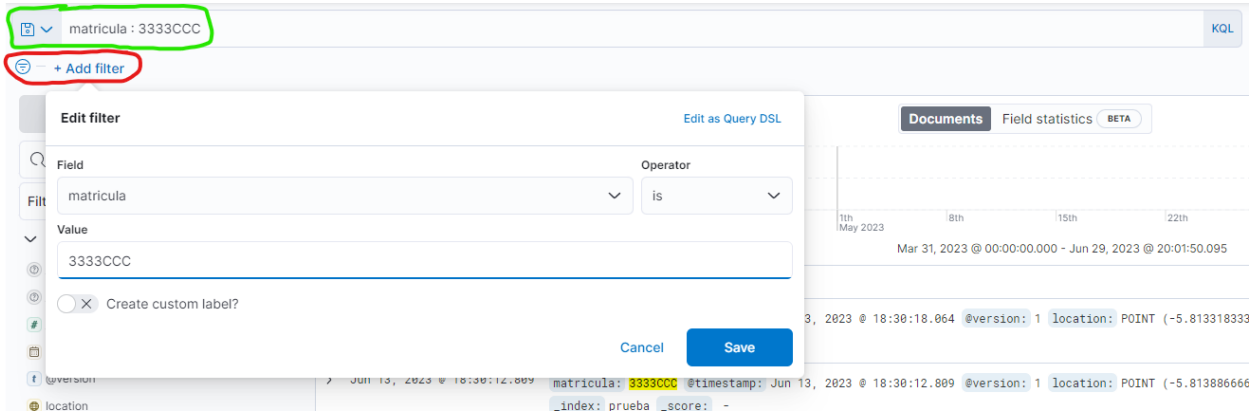



Ilustración 48: Filtrado de datos - Kibana.

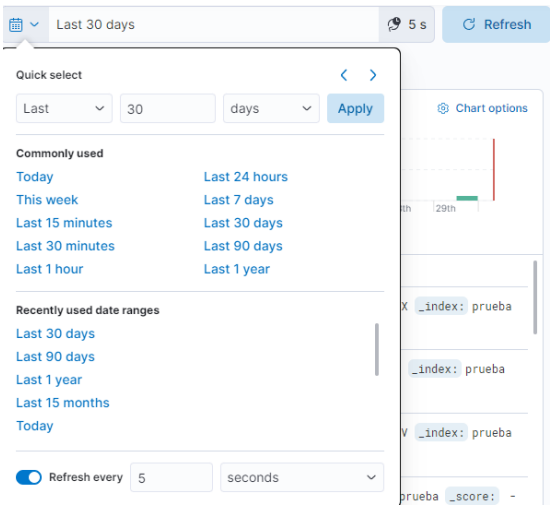



Ilustración 49: Filtrado por tiempo – Kibana

[1] Kibana Query Language es un lenguaje de consulta utilizado en Kibana que se utiliza para realizar consultas y filtrar datos para obtener información específica

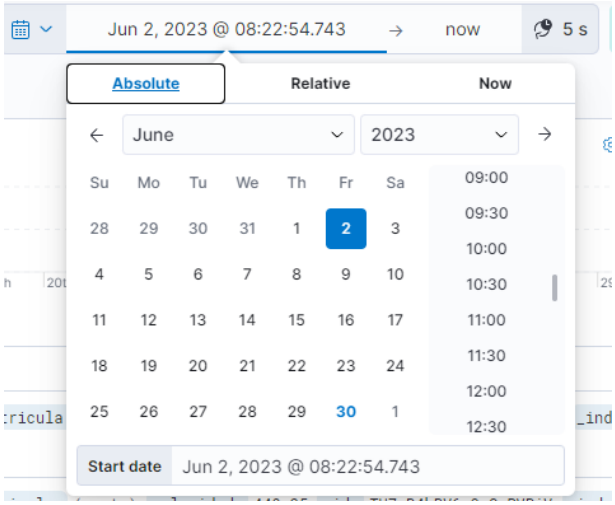



Ilustración 50: Otro filtrado por tiempo – Kibana.

## 7.1 Pantalla discover

La pantalla discover, ilustración 51, nos permite:

- Ver los valores de los campos del index, ordenados de más reciente a más antiguos.
- Ver la información de los valores de forma expandida en formato tabla o JSON, ilustración 52. Además, como también se puede ver en la ilustración 52, en la parte superior se permite ver cuantas infracciones se registraron un día en concreto poniendo el ratón sobre la barra verde.

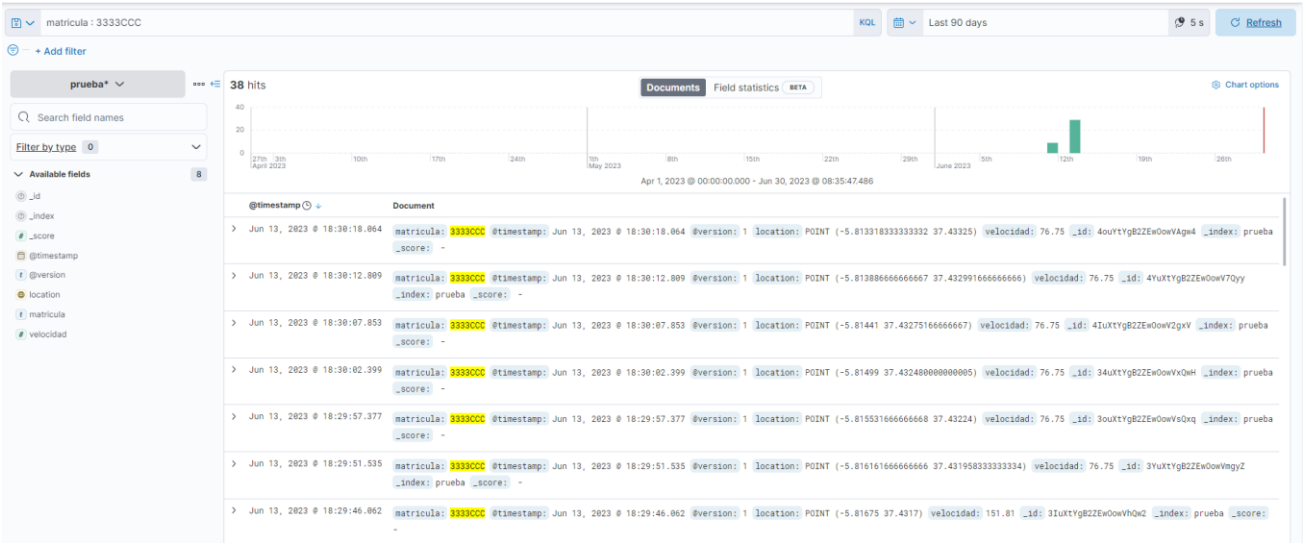



Ilustración 51: Pantalla discover – Kibana.

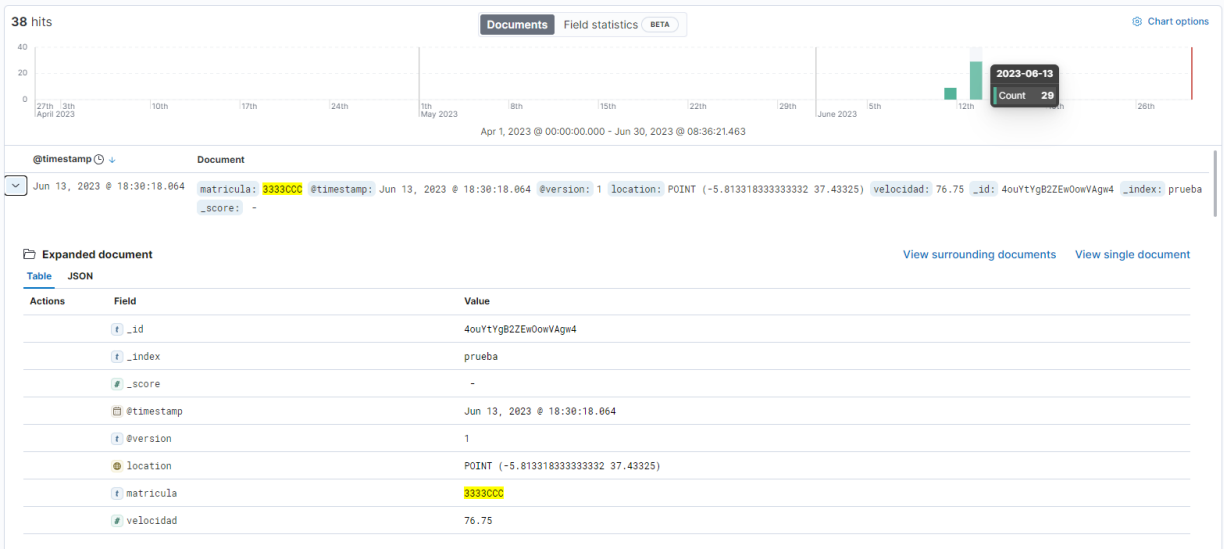



Ilustración 52: Pantalla discover información expandida – Kibana.

## 7.2 Pantalla maps

La pantalla maps, ilustración 53, permite:

- Interactuar con el mapa, en el cual se registran las infracciones gracias al tipo de dato “geo_point”. Si el mapa tiene poco zoom, como el de la ilustración 53, al seleccionar un punto se mostrarán cuantas infracciones hay cerca del punto seleccionado. Al hacer más zoom, ilustración 54, se puede apreciar mejor los puntos donde se han cometido dichas infracciones.
- Al tener seleccionado un punto se puede hacer click sobre la matrícula para aplicar automáticamente el filtrado de esa matrícula sin tener que usar los campos de la parte superior de filtrado.

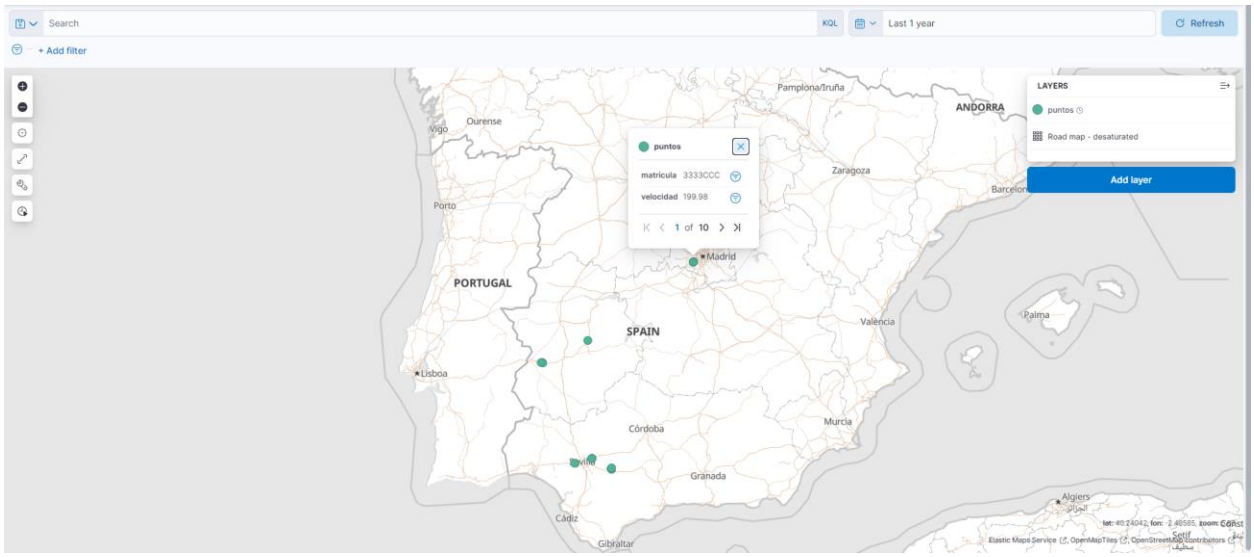



Ilustración 53: Pantalla maps con poco zoom – Kibana.

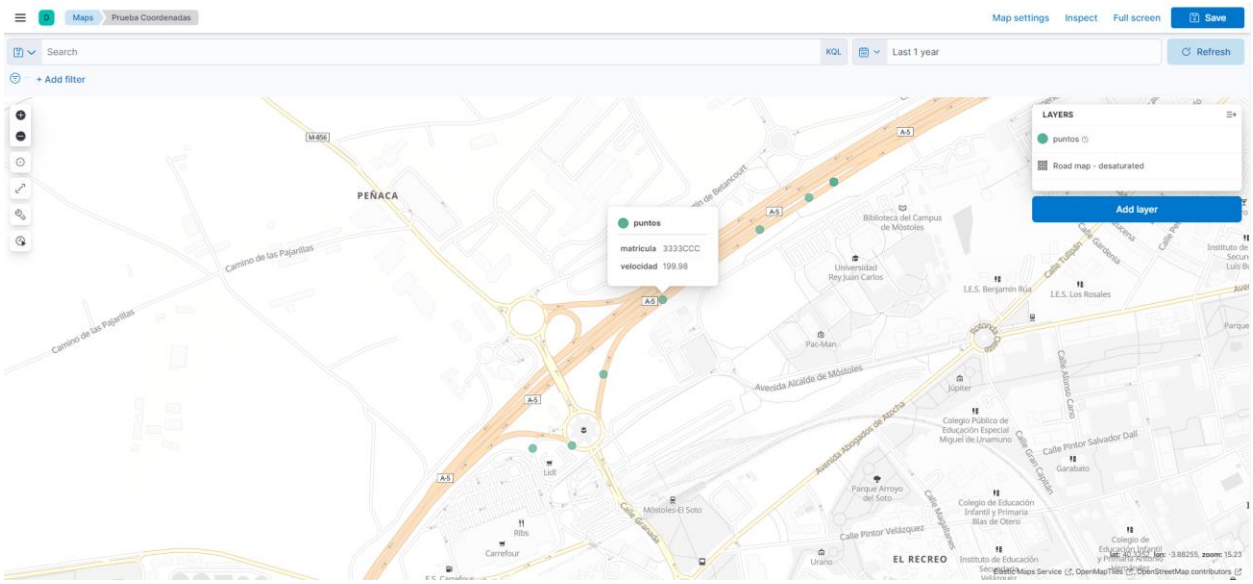
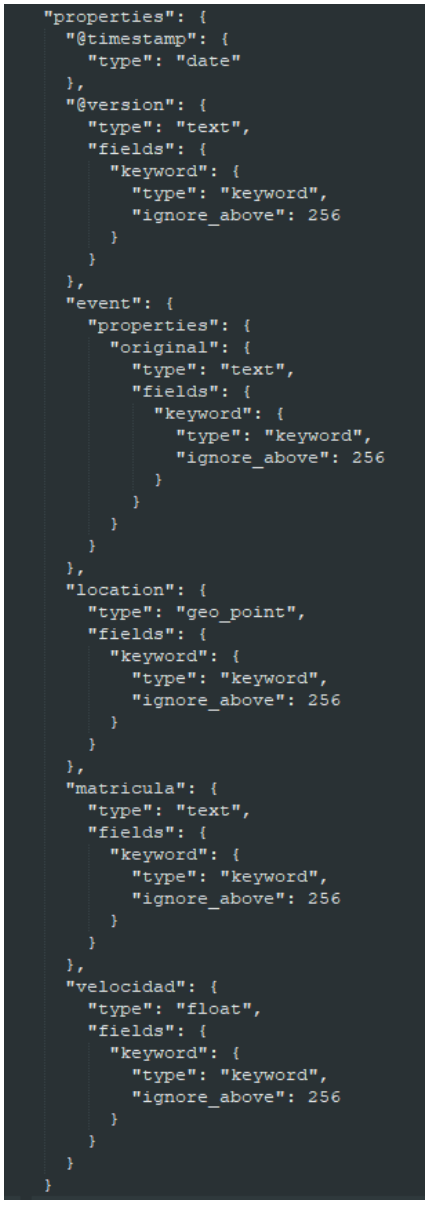



Ilustración 54: Pantalla maps con más zoom – Kibana.

## 7.3 Configuración Kibana

En este punto se detalla la configuración necesaria de Kibana para la correcta representación de los datos, así como la creación de la pantalla Maps para la representación geográfica de las infracciones.

A continuación, se muestra el mapping del index, ilustración 55, para especificar los campos del index y de qué tipo son estos.



```
"properties": {
  "@timestamp": {
    "type": "date"
  },
  "@version": {
    "type": "text",
    "fields": {
      "keyword": {
        "type": "keyword",
        "ignore_above": 256
      }
    }
  },
  "event": {
    "properties": {
      "original": {
        "type": "text",
        "fields": {
          "keyword": {
            "type": "keyword",
            "ignore_above": 256
          }
        }
      }
    }
  },
  "location": {
    "type": "geo_point",
    "fields": {
      "keyword": {
        "type": "keyword",
        "ignore_above": 256
      }
    }
  },
  "matricula": {
    "type": "text",
    "fields": {
      "keyword": {
        "type": "keyword",
        "ignore_above": 256
      }
    }
  },
  "velocidad": {
    "type": "float",
    "fields": {
      "keyword": {
        "type": "keyword",
        "ignore_above": 256
      }
    }
  }
}
```


Ilustración 55: Mapping del index de Kibana.

De este mapping se destaca los siguientes campos, que ya se contemplaban en el filtro del fichero de configuración de Logstash, en el Anexo B:

- **Location** de tipo geo_point, con el cual se puede representar el punto en el mapa
- **Matrícula** de tipo texto, el cual representa la matrícula del vehículo.
- **Velocidad** de tipo float, el cual representa la velocidad a la que se cometió la infracción.

Para añadir este mapping al index se tienen que seguir los siguientes sencillos pasos [4]:

1 En la barra lateral izquierda seleccionar Dev Tools dentro de la categoría Management para abrir la consola.

2 Introducir PUT /nombre_index/_mapping y pegar el mapping de la ilustración 55.

Ya solo queda la creación del Maps para poder representar las infracciones de forma geográfica. Para la creación de este Maps se tienen que seguir los siguientes pasos:

1 En la barra lateral izquierda seleccionar Maps dentro de la categoría Analytics.

2 Presionar el botón create map.

3 Presionar Add layer.

4 Seleccionar Documents como tipo de layer para la representación de puntos.

5 Seleccionar el Data view que se quiera representar, en este caso el del index mencionado anteriormente y en la pestaña Geospatial field seleccionar el campo location, también mencionado anteriormente.

6 Presionamos Add layer.

7 Se introduce un nombre y se añade los campos matricula y velocidad, ilustración 56, y se presiona save&close.

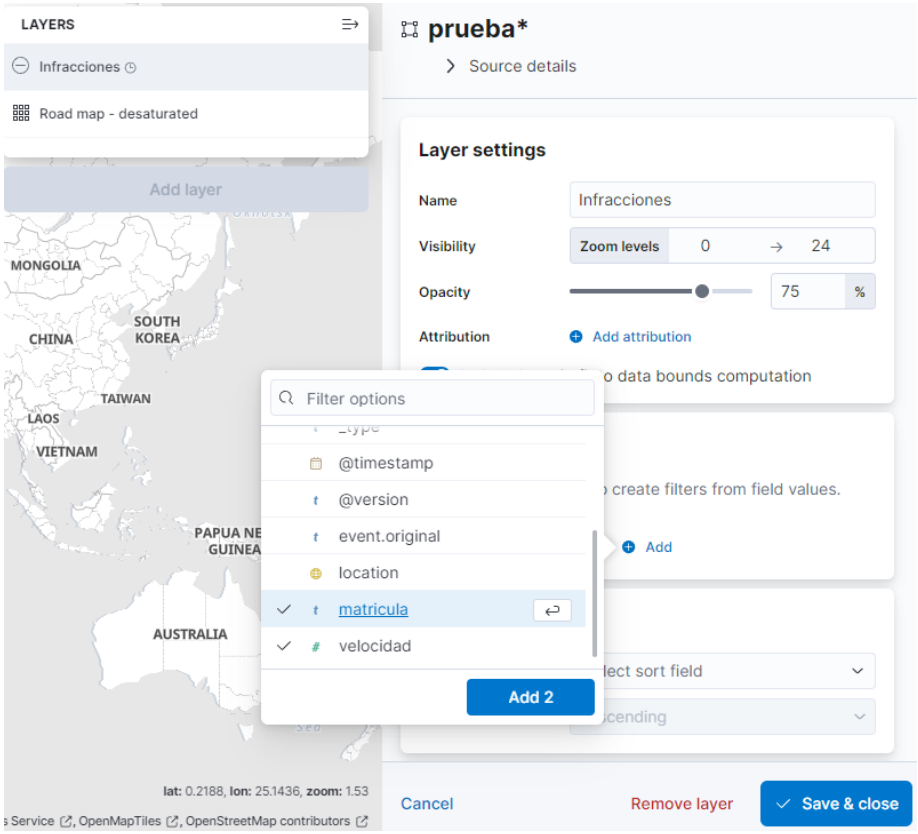



Ilustración 56: Creación de Maps Kibana.

# 8 CONCLUSIÓN Y LÍNEAS FUTURAS

Como conclusión, en este proyecto, se ha desarrollado una aplicación Android que permite a los usuarios iniciar sesión utilizando su cuenta de Google mediante la integración de Firebase. Además, la aplicación utiliza un servicio web y Firebase con su realtime database para almacenar la matrícula del vehículo asociada al usuario e infracciones de velocidad en dicha base de datos. Estas infracciones se pueden visualizar en una pantalla dentro de la aplicación.

La aplicación utiliza el GPS para obtener la velocidad y la posición del usuario, y también accede a una API externa para obtener la velocidad máxima permitida en esa ubicación. Si la velocidad del usuario supera la velocidad máxima permitida, se envía la ubicación, matrícula y velocidad a un servicio web que a su vez lo envía a Kafka. La información es consumida por ELK (Elasticsearch, Logstash y Kibana), y finalmente se representa en un mapa en Kibana mostrando la ubicación del usuario junto con la velocidad y la matrícula correspondiente.

Este proyecto ha demostrado la viabilidad de la integración de múltiples tecnologías y servicios para crear una aplicación Android con funcionalidades avanzadas, como la detección de infracciones de velocidad y su representación en un mapa interactivo. El uso de Firebase ha facilitado el almacenamiento y la recuperación de datos en tiempo real, mientras que Spring Boot ha permitido crear servicios web eficientes y escalables. La combinación de Kafka y ELK ha posibilitado la transmisión y el procesamiento de datos en tiempo real, proporcionando una visualización geográfica clara y concisa de las infracciones de velocidad.

Aunque este proyecto ha logrado cumplir con los objetivos planteados, existen algunas áreas que podrían mejorarse en futuros desarrollos. Algunas de estas mejoras son:

- Implementación de un sistema de clasificación y priorización de infracciones: Sería útil asignar una prioridad a cada infracción de velocidad registrada, para que las autoridades puedan tomar medidas rápidas y eficientes en función de la gravedad de cada infracción.
- Mejora en la precisión de la detección de velocidad máxima: Si bien se ha consultado una API REST para obtener la velocidad máxima de la vía por la que circula el usuario, puede haber situaciones en las que la precisión no sea óptima. Sería beneficioso investigar y utilizar métodos adicionales para mejorar la precisión de la detección de velocidad máxima.

Por último, una mejora relevante para el desarrollo futuro de esta aplicación sería el despliegue en la nube de todos los componentes involucrados. Actualmente, las pruebas se han realizado utilizando el emulador de Android Studio, lo cual puede limitar la precisión y la representación realista de situaciones en el mundo real. De hecho, se ha tenido que modificar la velocidad máxima, multiplicando ésta por dos, aunque el emulador de Android Studio ofrece 5 velocidades, la velocidad más lenta que éste ofrece supera la velocidad máxima.

Al migrar el sistema a la nube, se podrían realizar pruebas reales utilizando dispositivos móviles físicos y obtener datos más precisos.

En resumen, este proyecto ha sentado las bases para una aplicación Android completa y funcional que integra diversas tecnologías y servicios para detectar y representar infracciones de velocidad en tiempo real. Las mejoras mencionadas anteriormente podrían proporcionar una mayor usabilidad, funcionalidad y precisión, y podrían ser consideradas en futuros desarrollos de la aplicación.

# Anexo A: Configuración de Firebase

Ya se ha detallado que es Firebase y las dos características principales que se usan en el proyecto en el punto 3.1 de esta memoria. En este anexo se especifica como configurar Firebase desde su interfaz web y lo que hay que añadir a la aplicación Android y al servicio web para el correcto funcionamiento entre ellos.

## A.1 Autenticación con Google

Lo primero es incorporar Firebase a nuestro proyecto Android y seleccionar cuenta de Google como inicio de sesión o método de autenticación, para ello existe una guía oficial [5] de la cual se han realizado los siguientes pasos, se destacarán algunos pasos importantes dentro de ellos:

1. Tener creado un proyecto en Android Studio.
2. Ir a la consola de firebase [6] e iniciar sesión con una cuenta de Google.
3. Una vez iniciada la sesión, se selecciona “agregar proyecto” y se continua con los pasos de creación del proyecto.
4. Se agrega Firebase a la aplicación Android seleccionando el icono de Android, como se muestra en la ilustración 59, y se siguen los pasos, de los cuales destacamos:
    a. Agregar el google-service.json, el cual tenemos que descargar, a la carpeta app del proyecto Android.
    b. Agregar el repositorio y dependencia detalladas en la ilustración 57 al build.gradle a nivel de proyecto y el plugins y dependencias detalladas en la ilustración 58 al build.gradle a nivel de app.

```
buildscript {
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:4.2.0"
        classpath 'com.google.gms:google-services:4.3.15'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}
```

Ilustración 57: Repositorios y dependencias build.gradle(Project)

```
plugins {
    id 'com.android.application'
    id 'com.google.gms.google-services'
}
dependencies {

    implementation platform('com.google.firebase:firebase-bom:32.0.0')
    implementation 'com.google.firebase:firebase-analytics'
```

Ilustración 58: Plugins y dependencias buil.gradle(App)

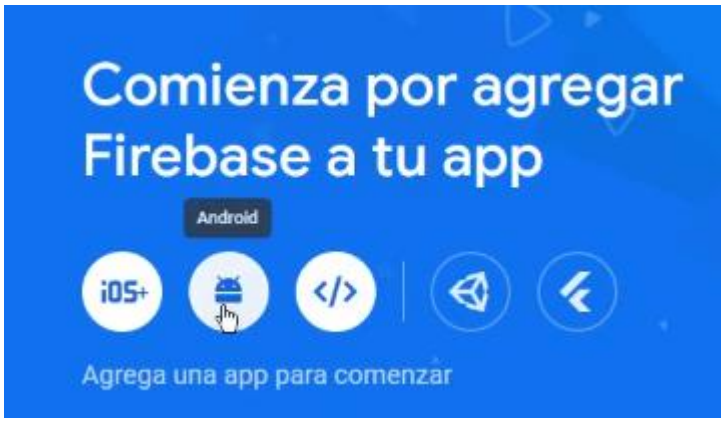



Ilustración 59: Agregar Firebase a aplicación Android

5 Importar las dependencias detalladas en la ilustración 60 al build.gradle a nivel de app, para el BoM de Firebase a fin de declarar la dependencia de la biblioteca de Android para Firebase Authenticacion, además, el SDK de Servicios de Google Play a la app.

```
implementation platform('com.google.firebase:firebase-bom:31.5.0')
implementation 'com.google.firebase:firebase-auth'
implementation 'com.google.android.gms:play-services-auth:20.5.0'
```

Ilustración 60: Mas dependencias build.gradle(App)

6 Habilitar el método de Acceso con Google, para ello se selecciona la pestaña “Authentication” en la barra lateral de la izquierda de la pantalla console de Firebase y se selecciona Google.

## A.2 SDK de Firebase Admin para servicio web

El siguiente paso es agregar el SDK de Firebase Admin al servicio web, para ello tenemos una guía oficial [7] de la cual se han realizado los siguientes pasos, se destacarán algunos pasos importantes dentro de ellos:

1 Tener creado un proyecto de spring boot.

2 Importar la dependencia detallada en la ilustración 61 en el pom.xml.

```
<dependency>
  <groupId>com.google.firebase</groupId>
  <artifactId>firebase-admin</artifactId>
  <version>9.1.1</version>
</dependency>
```

Ilustración 61: Dependencia pom.xml Firebase

3 Inicializar el SDK, para ello, en la pantalla principal de la consola, ir a la pestaña configuración del proyecto, luego cuentas y servicio, seleccionamos Java y se selecciona "generar nueva clave privada". Esta clave privada se guarda en la ruta "src/main/resources" con el nombre "serviceAccountKey.json".

4 En el fichero "application.properties" se crea la variable que se muestra en la ilustración 62.

```
1 firebase.config.path=serviceAccountKey.json
```

Ilustración 62: Path de la configuración de Firebase

5 Incluir el código necesario al servicio web para que cuando se reciba cualquier petición se autentique con "Bearer token". Dicho código se detalla en el apartado 5 de la memoria.

# Anexo B: Configuración ELK

Ya se ha hablado de lo que es ELK, de sus características en el capítulo 3 de esta memoria y de la configuración de Kibana y creación de la pantalla Maps en el capítulo 7 de esta memoria. En este punto veremos el fichero .yaml que se ha creado para el uso de esta tecnología y cómo configurar Logstash para suscribirse a Kafka, transformar los datos obtenidos y almacenarlos en ElasticSearch.

## B.1 Fichero .yaml

El fichero que podemos ver en la ilustración 63 detalla la configuración de ElasticSearch y Kibana, dos de las tres tecnologías que componen ELK, con sus respectivas imágenes de Docker y configuraciones específicas [8].

```yaml
version: '3.7'
services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.1.0
    container_name: elasticsearch
    restart: always
    environment:
      - xpack.security.enabled=false
      - discovery.type=single-node
    ulimits:
      memlock:
        soft: -1
        hard: -1
      nofile:
        soft: 65536
        hard: 65536
    cap_add:
      - IPC_LOCK
    volumes:
      - elasticsearch-data:/usr/share/elasticsearch/data
    ports:
      - 9200:9200

  kibana:
    container_name: kibana
    image: docker.elastic.co/kibana/kibana:8.1.0
    restart: always
    environment:
      - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
    ports:
      - 5601:5601
    depends_on:
      - elasticsearch
volumes:
  elasticsearch-data:
```

Ilustración 63: Fichero .yaml de ElasticSearch y Kibana.

A continuación, se detalla la utilidad de cada campo:

- version: '3.7': Indica la versión de la sintaxis de Docker Compose que se está utilizando en el archivo.
- services: Define los servicios que se ejecutarán en contenedores, en este caso elasticsearch y kibana.
    - elasticsearch: Es el nombre del servicio de Elasticsearch.

- image: Especifica la imagen de Docker utilizada para el servicio, en este caso, la imagen de Elasticsearch versión 8.1.0.
- container_name: Define el nombre del contenedor que se creará para el servicio.
- restart: always: Indica que el contenedor se reiniciará automáticamente en caso de que se detenga.
- environment: Permite definir variables de entorno dentro del contenedor de Elasticsearch. En este caso, se establecen dos variables: xpack.security.enabled=false para deshabilitar la seguridad de X-Pack y discovery.type=single-node para configurar Elasticsearch en modo de un solo nodo.
- ulimits: Establece los límites de recursos para el contenedor. En este caso, se configura memlock (bloqueo de memoria) y nofile (cantidad máxima de archivos abiertos) con valores específicos.
- cap_add: Permite agregar capacidades adicionales al contenedor. En este caso, se agrega la capacidad IPC_LOCK, que se utiliza para permitir el bloqueo de memoria en Elasticsearch.
- volumes: Mapea el volumen llamado "elasticsearch-data" al directorio dentro del contenedor donde Elasticsearch almacena sus datos.
- ports: Mapea el puerto 9200 del host al puerto 9200 del contenedor.

o kibana: Es el nombre del servicio de Kibana.

- container_name: Define el nombre del contenedor que se creará para el servicio.
- image: Especifica la imagen de Docker utilizada para el servicio, en este caso, la imagen de Kibana versión 8.1.0.
- restart: always: Indica que el contenedor se reiniciará automáticamente en caso de que se detenga.
- environment: Permite definir variables de entorno dentro del contenedor de Kibana. En este caso, se establece ELASTICSEARCH_HOSTS como http://elasticsearch:9200, lo que indica que Kibana se conectará a Elasticsearch en el host "elasticsearch" en el puerto 9200.
- ports: Mapea el puerto 5601 del host al puerto 5601 del contenedor.
- depends_on: Define una dependencia entre servicios. En este caso, se indica que Kibana depende de que el servicio de Elasticsearch esté en funcionamiento antes de iniciar.

- volumes: Define un volumen llamado "elasticsearch-data" que se utilizará para persistir los datos de Elasticsearch fuera del contenedor.

## B.2 Configuración Logstash

En este apartado se detalla el archivo de configuración utilizado por Logstash, ilustración 64, en el cual se especifica la entrada, el filtro y la salida de datos. Dicho archivo se debe ubicar en la ruta “etc/logstash/conf.d”.

```
input {
    kafka {
      bootstrap_servers => "192.168.206.130:9092"
      topics => ["prueba"]
      codec => "json"
   }
}

filter {
    json{
      source => "message"
    }
    mutate {
        convert => {"lat" => "float"}
        convert => {"lon" => "float"}
        convert => {"vel" => "float"}

        add_field => ["location", "%{lat},%{lon}"]
        add_field => ["matricula", "%{mat}"]
        add_field => ["velocidad", "%{vel}"]
        remove_field => ["event","lat", "lon","mat","vel"]
    }
}

output {
   elasticsearch {
      hosts => ["192.168.206.130:9200"]
      index => "prueba"
      workers => 1
     }
}
```

Ilustración 64: Fichero configuración Logstash.

A continuación, se expone la estructura y utilidad de cada campo:

- Input: kafka: Configura Logstash para leer datos de un clúster de Kafka.
    - bootstrap_servers: Especifica la dirección y el puerto de los servidores de Kafka desde los que se leerán los datos.
    - topics: Define los temas de Kafka de los que se leerán los mensajes. En este caso, se utiliza el tema "prueba".
    - codec: Establece el códec de entrada para decodificar los mensajes. Aquí se utiliza el códec JSON para procesar mensajes en formato JSON.
- filter:
    - json: Analiza el campo "message" y convierte su contenido de JSON a una estructura de datos.
    - mutate: Realiza varias transformaciones en los campos de los eventos.
        - convert: Convierte los campos "lat", "lon" y "vel" en tipo de datos float.
        - add_field: Agrega nuevos campos al evento. En este caso, se crean los campos "location", "matricula" y "velocidad" a partir de otros campos existentes en el evento.
        - remove_field: Elimina los campos "event", "lat", "lon", "mat" y "vel" del evento.

- Output: elasticsearch: Configura Logstash para enviar los datos procesados a Elasticsearch.
    - hosts: Especifica la dirección y el puerto del servidor de Elasticsearch al que se enviarán los datos.
    - index: Define el nombre del índice en Elasticsearch donde se almacenarán los datos. En este caso, se utiliza el índice "prueba".
    - workers: Establece el número de trabajadores (hilos) que Logstash utilizará para enviar los datos a Elasticsearch. Aquí se especifica un único trabajador.

Se destaca de este fichero el filter, debido a que es el que se encarga de convertir los tipos de datos obtenidos de Kafka a el tipo de dato que se necesita en Elasticsearch para su correcta representación en Kibana. La forma en la que se define el campo “location” es muy importante, ya que como se detalla en el siguiente apartado, se define ese campo como tipo “geo_point”, el cual necesita estar formado por dos float.

# Anexo C: Puesta en marcha y Ejecución del proyecto

Para la puesta en marcha del proyecto son necesarios los siguientes requisitos.

## C.1 Instalaciones necesarias

Se necesita instalar las siguientes herramientas:

- Máquina virtual. Dentro de la máquina virtual instalar:
  - Docker-Compose.
  - Eclipse.
- AndroidStudio.

## C.2 Importar proyectos

Se tiene que importar los proyectos de los servicios web del capítulo 4 en Eclipse de la máquina virtual y el proyecto Android del capítulo 5 a AndroidStudio.

## C.3 Configurar las tecnologías

Se necesita configurar las siguientes tecnologías:

- Kafka: Como se indica en el capítulo 4.2.2.
- ELK: Como se indica en el Anexo B y capítulo 7.3.
- Firebase: Como se indica en el Anexo A.

## C.4 Desplegar las tecnologías

Se necesita desplegar Kafka, Logstash y Elasticsearch junto a Kibana, en la máquina virtual, para ello:

- Arrancar la máquina virtual
- Lanzar los siguientes comandos para el despliegue de las tecnologías comentadas:
  - sudo docker-compose -f <fichero_kafka.yaml> up.
  - sudo systemctl start logstash.service.
  - sudo docker-compose -f <fichero_ELK.yaml> up.

## C.5 Arrancar servicios web y aplicación Android

Se necesita arrancar los servicios web, iniciando los proyectos de Eclipse y arrancar la aplicación Android en AndroidStudio.

## C.6 Abrir interfaz de Kibana

Por último, se abre en el navegador web, Google Chrome, la interfaz de Kibana, para ello se accede a la URL configurada en el fichero .yaml, en este caso: 192.168.206.130:5601

# REFERENCIAS

[1] Dirección General de Tráfico, «Conducir con exceso de velocidad,» [En línea]. Available: https://www.dgt.es/muevete-con-seguridad/evita-conductas-de-riesgo/conducir-con-exceso-de-velocidad/#:~:text=Uno%20de%20cada%20cinco%20accidentes,con%20el%20exceso%20de%20velocidad.

[2] P. B. Pérez, Aplicación y Servicio web para la gestión de información de sensores usando Fiware y Spring, Sevilla: Universidad de Sevilla, 2019.

[3] S. M. Contioso, Aplicación Android y Servicio Web Spring para la monitorización de datos obtenidos en un vehículo haciendo uso de la plataforma FIWARE, Sevilla: Universidad de Sevilla, 2020.

[4] elastic, «Indices put mapping,» [En línea]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/8.1/indices-put-mapping.html.

[5] Google, «Autentica con Google en Android,» [En línea]. Available: https://firebase.google.com/docs/auth/android/google-signin?hl=es.

[6] Google, «Consola Firebase,» [En línea]. Available: https://console.firebase.google.com/.

[7] Google, «Agrega el SDK de Firebase Admin a tu servidor,» [En línea]. Available: https://firebase.google.com/docs/admin/setup?hl=es-419#java.

[8] elastic, «Getting started with the Elastic Stack and Docker-Compose,» [En línea]. Available: https://www.elastic.co/es/blog/getting-started-with-the-elastic-stack-and-docker-compose.