ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"DISEÑO E IMPLEMENTACIÓN DE SISTEMA PARA MONITOREO DE CERCO VIRTUAL EN RECORRIDOS DE TAXIS"

INFORME DE PROYECTO INTEGRADOR

Previo a la obtención del Título de:

INGENIERO EN COMPUTACIÓN

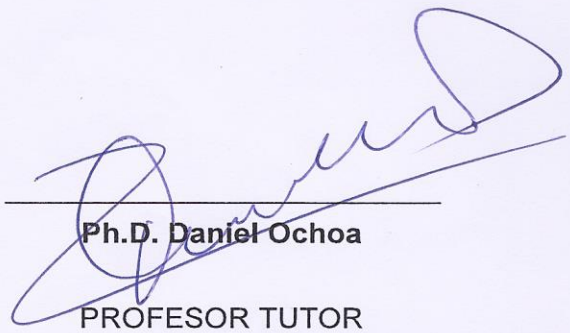
ANDRES SEBASTIAN CACERES VALDIVIEZO

DIEGO ROMARIO PALOMEQUE GOMEZ

GUAYAQUIL – ECUADOR

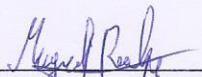
AÑO: 2017

TRIBUNAL DE EVALUACIÓN



Ph.D. Daniel Ochoa

PROFESOR TUTOR



Ph.D. Miguel Realpe

PROFESOR COLABORADOR

DECLARACIÓN EXPRESA

"La responsabilidad y la autoría del contenido de este Trabajo de Titulación, me(nos) corresponde exclusivamente; y doy(damos) mi(nuestro) consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Andrés Cáceres

Andrés Cáceres Valdiviezo

Diego Palomeque

Diego Palomeque Gómez

Resumen

El objetivo del proyecto consiste en desarrollar un sistema que integre los servicios de geotracking y geofencing para verificar que los taxis siguen su ruta establecida y que se emita una alarma a un sistema de monitoreo en caso de producirse un desvío mayor al establecido por un cerco virtual, y con esto poder reforzar la seguridad de los usuarios que utilizan los servicios de taxis.

El sistema consta de seis componentes que se conectan entre sí para cumplir con su función: dos aplicaciones móviles, un Web Server, un WebSocket Server, una base de datos y una aplicación web. Para la implementación de estos componentes se analizaron y se utilizaron varias tecnologías y complementos de desarrollo, los cuales facilitaron el desarrollo de los mismos. Las principales tecnologías que se utilizaron son: Nodejs, MongoDB, Socket.io, OSRM y Turfjs, los cuales aportaron para cumplir con la funcionalidad principal del sistema de los cuales tenemos el consumo de un API REST para el almacenamiento de los datos de las rutas, el cálculo de una ruta óptima entre dos puntos, comunicación en tiempo real entre los componentes y el cálculo del cerco virtual para la ruta.

Con el sistema desarrollado se puede realizar el seguimiento de un recorrido de taxi en tiempo real mediante un portal web de manera efectiva revisando el estado de la ruta. No se realizaron pruebas de campo debido a que no se cuenta con la infraestructura adecuada, pero se efectuaron pruebas de aceptación para verificar el funcionamiento del software.

Abstract

The objective of the project consists in developing a system that integrates the geo tracking and geofencing services in order to verify that the taxi drivers follow an established route, also to emit an alarm to a monitoring system when a major detour than the one previously established by the virtual fence happens, therefore with this we can improve the security of the clients that use the taxi services.

The system is made of six components that are connected between them in order to achieve their respective functions: 2 mobile applications, one web server, one websocket server, a database and a web application. In order to develop these components, some technologies and development complements were analyzed, these helped us the in development of the components, the main technologies that were used are:
NodeJS, MongoDB, Socket.IO, OSRM and Turf.js, which contributed with the main functionalities of the system. Among the main functionalities we can mention the consuming of the REST API for the storage of the routes' data, the calculation of the best route between two points, communication in real time between components and the calculation of the virtual fence for a route.

With the developed system we can make the tracking of a taxi's road in real time by a web portal checking the route's state.
No field tests were executed because we don't have the adequate infrastructure, but acceptance tests were run in order to verify the software's correct behavior.

ÍNDICE GENERAL

CAPÍTULO 1

1. INTRODUCCIÓN

1.1 Antecedentes

Entre los problemas de seguridad que preocupan a la ciudadanía en Ecuador se encuentra el secuestro exprés, los primeros casos de este delito en Ecuador se conocieron entre el 2000 y 2005, una de las modalidades identificadas por Agentes de Inteligencia de la Policía es en la que se utilizan taxis ya sea informales o legales [1]. De esta modalidad se identificaron dos variaciones: uno de los delincuentes se oculta en la cajuela con la complicidad del conductor de la unidad que aborda la víctima. En la otra los delincuentes siguen en autos la ruta especificada por el conductor de la unidad que aborda la víctima, y luego de recorrer ciertos tramos interceptan y abordan la unidad. En Manta y Guayaquil fueron las principales ciudades donde se radicó este delito en las cuales se observó un incremento de casos en el 2007 debido a la proliferación de taxis ilegales. Según datos del Ministerio del Interior señalan que en el 2013 se reportaron 494 casos de secuestro exprés en el país y solo sucedieron en las provincias de Carchi, Manabí y Guayas de las cuales 491 corresponden a la ciudad de Guayaquil [1][2].

En el 2017 se identificó una nueva modalidad de robo mediante secuestro exprés en la ciudad de Quito, en esta modalidad los antisociales captan a sus víctimas en sitios con mayor demanda de taxis, como zonas de diversión nocturna y centros comerciales. Durante el trayecto el conductor conversa con el pasajero para realizar un perfilamiento de una potencial víctima, mientras sus cómplices que se encuentran en un segundo vehículo esperan la señal del conductor para saber si el robo es apto o no, en el caso de serlo los antisociales introducen a la víctima en el segundo vehículo y la obligan a entregar sus tarjetas y claves, además las mantienen encerrada dentro de una casa por al menos 24 horas hasta que el sistema permita hacer una nueva transacción luego de haber retirado un monto límite permitido [3].

1.2 Definición del problema

Con la finalidad de reducir el índice de delincuencia y la necesidad de los usuarios de taxi a sentirse seguros en su trayecto, el Gobierno Nacional junto con la Agencia Nacional de Tránsito y el Sistema Integrado de Seguridad ECU-911 ejecutaron el proyecto "Transporte Seguro", la primera fase inició en el 2013 donde se instaló un kit de seguridad en unidades de transporte público y comercial (taxi convencional), 17.000 en buses y 38.000 en taxis, el kit para taxis consta de 1 grabador digital de video móvil, 1 gps, 2 cámaras de video, 1 UPS y 3 botones de pánico [4][5]. La segunda fase inició en noviembre del 2015, se instalaron 17.256 kits, 14.500 en taxis y 2.756 en transporte público, hasta la actualidad se han instalado un total de 72.256 kits [5][6]. Este mecanismo ha ayudado a reducir la delincuencia en los taxis, sin embargo, aún existe preocupación en la ciudadanía debido a que existen grupos delictivos especializados en secuestro exprés que falsifican los sellos municipales y adhesivos del proyecto "Transporte Seguro" y que además usan el kit de seguridad, pero cortan el sistema de cámaras y deshabilitan los botones de pánico [7][8].

1.3 Justificación

Actualmente existen empresas en Ecuador que cuentan con redes de transporte que conectan usuarios con vehículos a través de una aplicación móvil, como por ejemplo Easy Taxi, Uber y Cabify, las cuales llevan un historial de carreras realizadas y no mantiene en anonimato la información de los conductores [9], sin embargo, las políticas de seguridad manejadas por estas empresas no favorecen lo suficiente a los usuarios en ciertos casos entre los cuáles se incluye al delito de secuestro exprés, el cual en compañías como Uber no realiza el reembolso del monto robado, ni ofrece ayuda inmediata, dado que la comunicación puede realizarse en los días siguientes luego de que el cliente haya realizado la denuncia [10]. Además en Ecuador no existen servicios de transporte de taxis

que trabajen con una plataforma que integre tecnologías de geotracking y geofencing para reforzar la seguridad de los usuarios que usan este servicio.

1.4 Objetivos generales

Implementar un sistema que integre los servicios de geotracking y geofencing para verificar que los taxis sigan su ruta establecida y que se emita una alarma a un sistema de monitoreo en caso de producirse un desvío mayor al establecido por un cerco virtual (Geofence).

1.5 Objetivos específicos

- Implementar una aplicación móvil para que los pasajeros puedan pedir un taxi y mostrar el recorrido que siguen.
- Implementar una aplicación móvil para que los conductores puedan recoger a sus pasajeros y mostrar el recorrido que deben seguir.
- Implementar una aplicación web para monitorear que los taxis sigan las rutas establecidas.

CAPÍTULO 2

2. ANÁLISIS Y DISEÑO DE LA SOLUCIÓN

2.1 Descripción de la solución

Para fortalecer la seguridad a los usuarios que utilizan los servicios de taxis, se planteó desarrollar un sistema que utilice las tecnologías de geotracking y geofencing para que los usuarios puedan verificar que los taxis mantienen sus rutas establecidas y que se envíe una alerta a un sistema de monitoreo cuando ocurra una situación de riesgo. Se considera una situación de riesgo cuando en una ruta determinada ocurre uno de los siguientes casos:

A. El taxi abandona el cerco virtual establecido para esa ruta.
B. El usuario emite una alerta de auxilio mediante el botón de pánico de la aplicación.

Cuando llega alguna alerta al sistema de monitoreo, el usuario monitor puede comunicarse ya sea con el cliente o el conductor para verificar si alguno de los usuarios se encuentra en peligro, la comunicación se la puede realizar por medio de un chat que está integrado en las aplicaciones móviles y la aplicación web del monitor. Cabe señalar que el chat no solo se puede usar en situaciones de riesgos si no, en cualquier momento del recorrido de la ruta.

2.2 Alcance

2.2.1 Módulos y actores del sistema

Actores:

- **Cliente (Pasajero):** Es la persona que solicita el servicio de taxi.
- **Conductor:** Es la persona que trabaja para la compañía que ofrece el servicio de taxi y la encargada de recoger al pasajero para llevarlo a su destino.

- **Monitor:** Es la persona encargada de monitorear las alertas que se emiten en las rutas que siguen los taxis.

Módulos:

- Aplicación móvil para el pasajero
- Aplicación móvil para el Conductor
- Aplicación web de monitoreo de alertas de rutas

2.2.2 Requerimientos del sistema

Se realizó un levantamiento de requerimientos del sistema para poder realizar un diseño del mismo. Los requerimientos obtenidos fueron los siguientes:

- **Aplicación móvil para uso del cliente**

  Permitir al pasajero crear una cuenta para hacer uso del servicio.

  Permitir al pasajero ingresar a la aplicación mediante su cuenta.

  Mostrar su ubicación actual dentro del mapa.

  Permitir al pasajero establecer el lugar al que quiere llegar.

  Permitir al pasajero definir el recorrido que desea seguir para llegar a su destino.

  Mostrar al pasajero la duración aproximada del recorrido.

  Permitir al pasajero pedir un taxi para que lo recoja en su ubicación actual.

  Mostrar al pasajero los datos del conductor y el vehículo que lo va a recoger (nombre, placa del vehículo).

  Mostrar en tiempo real el recorrido que está siguiendo el taxista.

  Permitir al pasajero emitir una alerta de auxilio (botón de pánico).

  Permitir usar un chat para comunicarse con un usuario monitor que esté disponible.

Permitir al pasajero calificar la ruta como segura o peligrosa al final del recorrido.

Mostrar al pasajero que tan segura es una ruta en base a la calificación de otros usuarios.

- **Aplicación móvil para uso del conductor**

  Permitir al conductor crear una cuenta para hacer uso del servicio.

  Permitir al conductor ingresar a la aplicación mediante su cuenta.

  Mostrar su ubicación actual dentro del mapa.

  Mostrar al conductor datos del cliente (nombre, teléfono) que debe recoger.

  Mostrar en tiempo real su recorrido.

  Permitir al conductor solicitar al cliente un cambio de recorrido.

  Permitir usar un chat para comunicarse con un usuario monitor que esté disponible.

- **Aplicación web de monitoreo de alertas de rutas**

  Ver las rutas que están activas y en peligro.

  Visualizar cerca virtual de una ruta seleccionada.

  Ver una notificación dentro de la aplicación.

  Ver información de una ruta seleccionada.

  Enviar un mensaje a un usuario (cliente o conductor)

2.2.3 Historias de usuario

Para un mejor desarrollo de la aplicación se decidió el uso de esta herramienta, la cual nos permite guiar el diseño del software mediante una

descripción breve de la acción a realizar por el usuario y la razón de ejecutarla.

A continuación, mostramos las historias de usuario obtenidas de acuerdo a los requerimientos del sistema.

Historia de Usuario	USUARIO001
Nombre	Mostrar su ubicación actual dentro del mapa.
Actores	Cliente, Conductor
Descripción	El usuario quiere visualizar su ubicación dentro del mapa mientras transcurre el recorrido
Condiciones Previas	● El usuario pertenece a una ruta activa ● El usuario se encuentra en la vista principal.
Post-condición	Ninguna.
Escenario Principal	1. El usuario pertenece a una ruta activa. 2. El usuario observa su posición dentro del mapa.
Curso Alternativo	Ninguno.
Excepciones	Ninguno.

Tabla 2.1 Historia de usuario (Mostrar su ubicación dentro del mapa)

Historia de Usuario	USUARIO002
Nombre	Emitir alerta de auxilio
Actores	Conductor, Cliente

Descripción	El usuario emite una alerta de auxilio para dar a conocer qué se encuentra en problemas dentro del vehículo.
Condiciones Previas	● El usuario ha iniciado sesión
Post-condición	Se muestra un cuadro de confirmación indicando que la alerta ha sido enviada.
Escenario Principal	1. El usuario da click en el icono de *emitir alerta* 2. El sistema muestra un cuadro de confirmación, permitiendo al usuario aceptar o cancelar la acción. 3. El usuario acepta la acción y se emite la alerta.
Curso Alternativo	Ninguno.
Excepciones	3. 3.1. El usuario rechaza la acción 3.1.1. Se cierra el cuadro de confirmación. 3. 3.1. No se puede establecer comunicación con el servidor. 3.1.1. Se muestra un cuadro de diálogo mostrando e indicando el error de conexión.

Tabla 2.2 Historia de usuario (Emitir alerta de auxilio)

Historia de Usuario	USUARIO003

Nombre	Permitir al usuario ingresar a la aplicación mediante su cuenta.
Actores	Conductor, Cliente
Descripción	El usuario emite realiza la autenticación dentro de la aplicación móvil para hacer uso del sistema.
Condiciones Previas	● El usuario ha abierto la aplicación.
Post-condición	El Usuario es redirigido a la pantalla principal de la aplicación.
Escenario Principal	1. El usuario ingresa sus credenciales en el formulario. 2. El usuario presiona el botón 'Iniciar sesión' 3. El usuario ingresa a la pantalla principal.
Curso Alternativo	Ninguno.
Excepciones	2. 2.1. No se puede establecer comunicación con el servidor. 2.1.1. Se muestra un cuadro de diálogo mostrando e indicando el error de conexión.

Tabla 2.3 Historia de usuario (Ingresar a la aplicación)

Historia de Usuario	USUARIO004
Nombre	Permitir usar un chat para comunicarse con un usuario monitor que esté disponible.

Actores	Cliente, Conductor
Descripción	El usuario quiere enviar un mensaje a un usuario monitor conectado.
Condiciones Previas	● El usuario ha iniciado sesión dentro de la aplicación. ● Existe un usuario monitor conectado. ● El usuario se encuentra en una ruta activa.
Post-condición	Se actualiza el Diálogo de mensajes, mostrando el mensaje enviado.
Escenario Principal	1. El usuario selecciona la opción ‘Mensajes’. 2. El usuario selecciona al monitor con el que desea chatear. 3. El sistema muestra un cuadro de diálogo. 4. El usuario escribe el mensaje dentro del cuadro de diálogo. 5. El usuario da click en enviar.
Curso Alternativo	Ninguno.
Excepciones	5. 5.1. No se puede establecer comunicación con el servidor. 5.1.1. Se muestra un mensaje explicando el error.

Tabla 2.4 Historia de usuario (Usar chat para comunicarse con un usuario monitro)

Historia de Usuario	CLIENTE001

Nombre	Permitir al pasajero crear una cuenta para hacer uso del servicio.
Actores	Usuario Cliente
Descripción	El cliente quiere registrarse dentro de la aplicación para hacer uso de las funcionalidades principales de la misma.
Condiciones Previas	● El usuario ha abierto la aplicación.
Post-condición	Se le muestra al usuario la confirmación de creación de cuenta y es redirigido a la pantalla de 'Iniciar sesión'.
Escenario Principal	1. El usuario llena los campos de usuario, contraseña, nombre, cédula, número de celular. 2. El usuario hace submit del formulario 3. El servidor guarda la información del cliente.
Curso Alternativo	Ninguno.
Excepciones	2. 2.1. El usuario ha ingresado uno/varios campos en formato inválido. 2.1.1. Se le comunica al usuario del error y el/los campo/s inválidos. 3. 3.1. No se puede establecer comunicación con el servidor. 3.1.1. Se muestra un cuadro de

	diálogo mostrando e indicando el error de conexión.

Tabla 2.5 Historia de usuario (Permitir al pasajero crear una cuenta)

Historia de Usuario	CLIENTE002
Nombre	Permitir al pasajero establecer el lugar al que quiere llegar.
Actores	Usuario Cliente
Descripción	El cliente quiere iniciar un recorrido teniendo en cuenta el nombre del destino y su ubicación.
Condiciones Previas	● El usuario ha iniciado sesión dentro de la aplicación. ● El usuario no tiene una ruta activa.
Post-condición	Se muestra el botón de 'Solicitar taxi' en la parte inferior y se pinta la ruta que se generó para realizar el recorrido.
Escenario Principal	1. El usuario da click en la caja de texto 'Ingresar dirección'. 2. El usuario ingresa una dirección. 3. El usuario acepta la búsqueda de ruta para esa dirección. 4. El usuario selecciona una de las rutas disponibles mostradas debajo de la caja de texto.
Curso Alternativo	Ninguno.

Excepciones	3. 3.1. No existe dirección que coincida con la ingresada por el usuario 3.1.1. Se le comunica al usuario del error. 3. 3.1. No se puede establecer comunicación con el servidor. 3.1.1. Se muestra un cuadro de diálogo mostrando e indicando el error de conexión.

Tabla 2.6 Historia de usuario (Permitir al pasajero establecer lugar de destino)

Historia de Usuario	CLIENTE003
Nombre	Permitir al pasajero definir el recorrido que desea seguir para llegar a su destino.
Actores	Usuario Cliente
Descripción	El cliente quiere definir un recorrido personalizado dentro de la aplicación.
Condiciones Previas	● El usuario ha iniciado sesión dentro de la aplicación. ● El usuario no tiene una ruta activa.
Post-condición	Se muestra el botón de ‘Solicitar taxi’ en la parte inferior y se pinta la ruta que se generó para realizar el recorrido.

Escenario Principal	1. El usuario clickea el botón de icono ‘Recorrido personalizado’ disponible dentro de la aplicación. 2. El usuario selecciona de manera manual los puntos por los que debe cruzar su ruta. 3. El usuario selecciona la opción ‘LISTO’.
Curso Alternativo	Ninguno.
Excepciones	2. 2.1. No se puede establecer comunicación con el servidor. 2.1.1. Se cancela el modo de Ruta personalizada’.

Tabla 2.7 Historia de usuario (Permitir al pasajero definir el recorrido)

Historia de Usuario	CLIENTE004
Nombre	Permitir al pasajero pedir un taxi para que lo recoja en su ubicación actual.
Actores	Usuario Cliente
Descripción	El cliente quiere solicitar un taxi desde su aplicación para llegar a un destino.
Condiciones Previas	● El usuario ha iniciado sesión dentro de la aplicación. ● El usuario no tiene una ruta activa. ● El usuario ha creado una ruta.
Post-condición	Se muestra un cuadro de diálogo con la información

	del conductor que lo recogerá.
Escenario Principal	6. El usuario selecciona la opción ‘Solicitar taxi’. 7. Se muestra un diálogo de progreso que indica la solicitud de la ruta.
Curso Alternativo	Ninguno.
Excepciones	6. 6.1. No se puede establecer comunicación con el servidor. 6.1.1. Se muestra un mensaje explicando el error.

Tabla 2.8 Historia de usuario (Permitir al pasajero solicitar taxi)

Historia de Usuario	CLIENTE005
Nombre	Permitir al pasajero calificar la ruta como segura o peligrosa al final del recorrido.
Actores	Cliente
Descripción	El cliente quiere calificar una ruta de acuerdo a que tan segura le pareció para que el sistema pueda clasificar rutas similares.
Condiciones Previas	● El usuario ha iniciado sesión dentro de la aplicación. ● El usuario terminó su ruta. ● El sistema muestra un cuadro de diálogo con las calificaciones que se le puede dar a la ruta que siguió.

Post-condición	Se muestra la pantalla principal de la aplicación.
Escenario Principal	1. El usuario selecciona el número de estrellas qué le asigna a la ruta. 2. El usuario presiona aceptar. 3. El sistema envía la calificación asignada a la ruta.
Curso Alternativo	2. El usuario presiona cancelar 3. El cuadro de diálogo desaparece.
Excepciones	2. 2.1. No se puede establecer comunicación con el servidor. 2.1.1. Se muestra un mensaje explicando el error.

Tabla 2.9 Historia de usuario (Permitir al pasajero calificar la ruta)

Historia de Usuario	CONDUCTOR001
Nombre	Registrar Usuario Conductor
Actores	Usuario Conductor
Descripción	El conductor quiere registrarse en la aplicación.
Condiciones Previas	● El usuario debe abrir la aplicación móvil ● El usuario presiona la opción *registrar* dentro de la aplicación móvil
Post-condición	Al usuario se le muestra un diálogo de confirmación indicando que ya puede hacer uso de la aplicación

Escenario Principal	1. El usuario llena los campos de usuario, contraseña, nombre, placa del vehículo, descripción del vehículo, cédula, número de celular. 2. El usuario hace submit del formulario 3. El Servidor hace uso del endpoint de registro para guardar la información del usuario dentro del sistema.
Curso Alternativo	Ninguno.
Excepciones	2. 2.1. El usuario ha ingresado uno/varios campos en formato invalido. 2.1.1. Se le comunica al usuario del error y el/los campo/s inválidos. 3. 3.1. No se puede establecer comunicación con el servidor. 3.1.1. Se muestra un cuadro de diálogo mostrando e indicando el error de conexión.

Tabla 2.10 Historia de usuario (Registrar usuario conductor)

Historia de Usuario	CONDUCTOR002
Nombre	Visualizar información de la carrera a realizar
Actores	Conductor
Descripción	El conductor visualiza los datos de la carrera a realizar, indicando el nombre del pasajero y su

	número de teléfono.
Condiciones Previas	● Un usuario pasajero ha realizado el pedido de taxi, el servidor asigna a este conductor para que realice la carrera.
Post-condición	Ninguna.
Escenario Principal	1. El usuario establece su destino. 2. El usuario solicita el taxi. 3. Se muestra al usuario el diálogo con datos del conductor.
Curso Alternativo	Ninguno.
Excepciones	3. 3.1. No se encuentra un conductor disponible. 3.1.1. Se le notifica al usuario que no hay conductores disponibles.

Tabla 2.11 Historia de usuario (Visualizar información de carrera)

Historia de Usuario	MONITOR001
Nombre	Ver rutas activas y en peligro
Actores	Usuario Monitor
Descripción	El usuario monitor quiere ver las rutas activas clasificadas por los estados “Normal” y “Peligrando”
Condiciones Previas	● El usuario monitor debe abrir la aplicación web ● El usuario monitor se autentica de manera

	exitosa.
Post-condición	Al usuario se le muestra en un panel lateral derecho dividido en 2 pestañas: ‘Normal’ y ‘Peligrando’, las cuáles contienen una lista rutas activas.
Escenario Principal	1. El usuario mueve el cursor al panel lateral derecho 2. El usuario clickea en una de las 2 pestañas visibles. 3. El usuario visualiza las rutas activas dentro de la aplicación.
Curso Alternativo	Ninguno.
Excepciones	Ninguna.

Tabla 2.12 Historia de usuario (Ver rutas activas y en peligro)

Historia de Usuario	MONITOR002
Nombre	Visualizar cerca virtual de una ruta seleccionada.
Actores	Usuario Monitor
Descripción	El usuario monitor quiere ver una ruta dibujada dentro del mapa, mostrando un perímetro virtual alrededor de la ruta.
Condiciones Previas	● Existen rutas activas.
Post-condición	La ruta seleccionada se renderiza dentro del mapa con información básica del cliente y el

	conductor, y un perímetro virtual.
Escenario Principal	1. El usuario mueve el cursor al panel lateral derecho 2. El usuario clickea en el checkbox de una de las rutas mostradas dentro de los paneles. 3. Se renderiza la ruta junto con el cerco virtual en el mapa.
Curso Alternativo	Ninguno.
Excepciones	Ninguna.

Tabla 2.13 Historia de usuario (Ver cerca virtual de una ruta)

Historia de Usuario	MONITOR003
Nombre	Visualizar una notificación dentro de la aplicación.
Actores	Usuario Monitor
Descripción	El usuario monitor quiere visualizar una notificación enviada desde un recorrido activo, para conocer su causa, y el usuario que lo envió.
Condiciones Previas	• Existen notificaciones recibidas y no leídas por el usuario monitor.
Post-condición	Ninguna.
Escenario Principal	1. El usuario clickea en la barra de notificaciones. 2. El usuario visualiza la información de la

	notificación recibida, incluyendo: Usuario que la envió, números de contacto, causa.
Curso Alternativo	Ninguno.
Excepciones	Ninguna.

Tabla 2.14 Historia de usuario (Ver una notificación dentro de la aplicación)

Historia de Usuario	MONITOR004
Nombre	Ver información de una ruta seleccionada.
Actores	Usuario Monitor
Descripción	El usuario monitor quiere ver la información de una ruta activa dentro de la aplicación web.
Condiciones Previas	● Existen rutas activas
Post-condición	Ninguna.
Escenario Principal	1. El usuario mueve el cursor al panel lateral derecho. 2. El usuario selecciona el botón de icono que representa a “Ver información”. 3. Un diálogo aparece mostrando la información de la ruta incluyendo los campos: Nombres de los usuarios, Teléfonos de contacto de los usuarios.
Curso Alternativo	Ninguno.

Excepciones	Ninguna.

Tabla 2.15 Historia de usuario (Ver información de una ruta)

Historia de Usuario	MONITOR005
Nombre	Enviar un mensaje a un usuario (cliente o conductor)
Actores	Usuario Monitor
Descripción	El usuario monitor quiere enviar un mensaje a un usuario que pertenece a una ruta activa.
Condiciones Previas	● Existen rutas activas
Post-condición	Ninguna.
Escenario Principal	1. El usuario mueve el cursor al panel lateral derecho 2. El usuario selecciona dentro de una de las rutas activas el botón de icono que representa a un chat. 3. El usuario monitor selecciona al destinatario, al cual quiere enviar un mensaje. 4. El usuario escribe el mensaje. 5. El usuario pulsa en el botón de icono ¨enviar¨.
Curso Alternativo	Ninguno.
Excepciones	4. 4.1. No se encuentra el usuario destinatario seleccionado con conexión de internet y la aplicación activa.

	4.1.1. Se le notifica al usuario monitor que no se puede enviar el mensaje.

Tabla 2.16 Historia de usuario (Enviar un mensaje a un usuario)

2.3 Diseño del sistema

El sistema está compuesto por seis componentes que se conectan entre sí para cumplir con su función: dos aplicaciones móviles, un Web Server, un WebSocket Server, una base de datos y una aplicación web ver Figura 2.1.

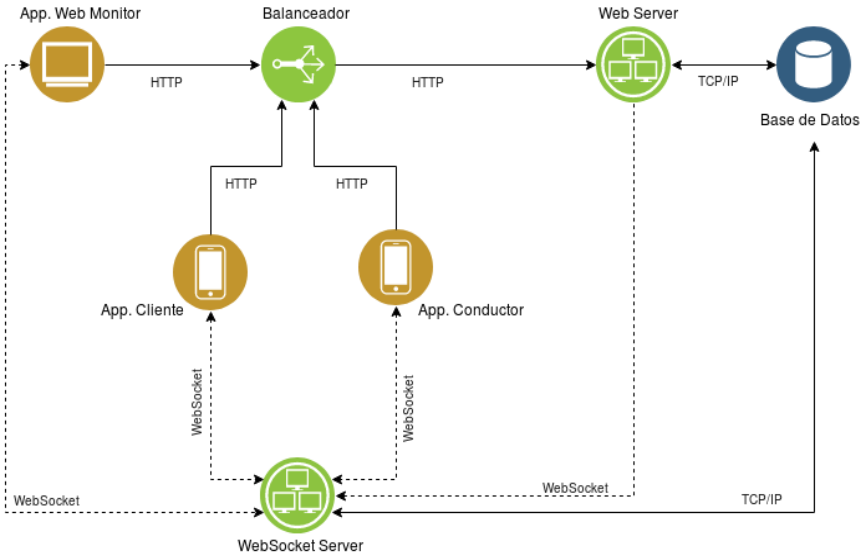


Figura 2.1 Diagrama de Arquitectura

La aplicación cliente se comunica con el Web Server para que este cree la ruta y le asigne un conductor para que lo lleve a su destino. La aplicación conductor se comunica con el Web Server para obtener la ruta que debe seguir para llevar al cliente a su destino, y rutas alternas en caso de que solicite un cambio de ruta, el cual tiene que ser confirmado por el cliente.

El WebSocket Server se encarga de realizar la función de real time de envío de notificaciones y alertas a la aplicación cliente, conductor y monitor. Este componente recibe información GPS del conductor y cliente, y este se comunica a su vez con el Web Server para verificar si ocurre un abandono de cerco o un cambio de ruta no planificada.

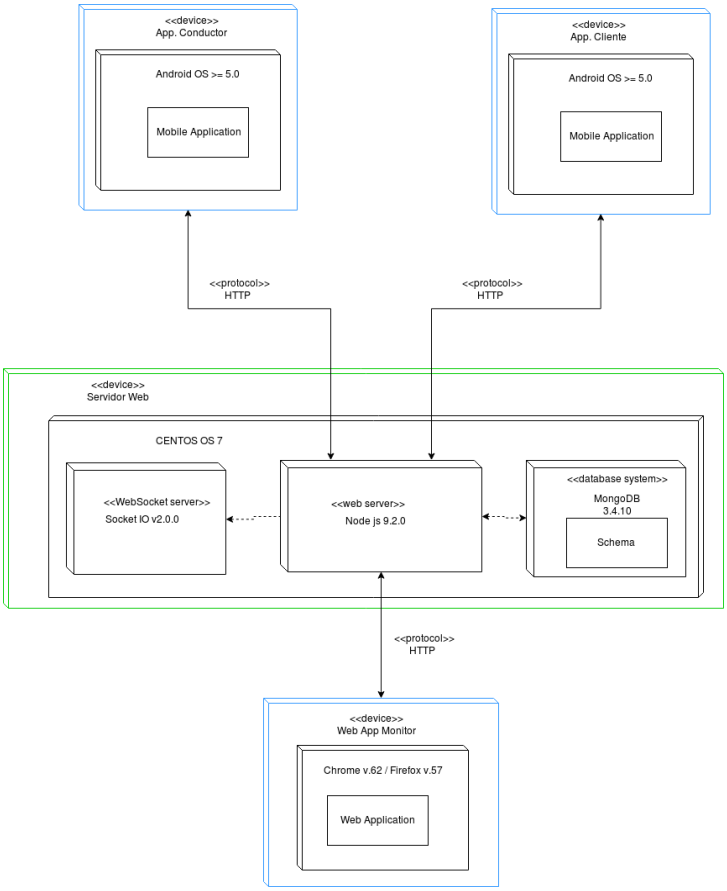


Figura 2.2 Diagrama de Despliegue

Las aplicaciones móviles del conductor y cliente deben ser compatibles con las plataformas de Android a partir de la versión 5.0 (Lollipop), más del 70% de dispositivos usan esta versión [11]. Para el almacenamiento de datos se utiliza la versión 3.4.10 de MongoDB, la cual es una base datos NoSQL orientada a documentos. Para el Web Server se utiliza la versión 9.2 de Nodejs y para el

WebSocket Server se utiliza la versión 2.0 de Socket.IO. En la siguiente sección se describen las características de estas tecnologías y otros complementos de desarrollo que se usaron.

2.4 Complementos de desarrollo

Para la implementación de este sistema se analizaron y se seleccionaron los siguientes complementos de desarrollo:

Nodejs

Nodejs es un entorno de ejecución para JavaScript de código abierto construido con el motor de alto rendimiento JavaScript V8 de Chrome, el cual está escrito en C++ [12]. Las principales características de este entorno son:

- Usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, lo cual lo hace liviano y eficiente.
- Usa un modelo de bucle de eventos como un entorno en vez de una librería, lo cual permite ingresar y salir de un bucle de eventos sin realizar una llamada de bloqueo que por lo general existe en otros sistemas.

Estas características hacen que Node esté diseñado para construir aplicaciones livianas, eficientes y de red escalables. En nuestro sistema usamos este entorno para construir el Web Server que es el encargado de manejar los múltiples requerimientos de las aplicaciones cliente, conductor y monitor.

MongoDB

Es una base de datos de código abierto que usa un modelo de datos orientado a documentos [13]. Es no relacional y nos ofrece de manera sencilla y con configuración mínima un sistema escalable y flexible con una sintaxis para realizar queries de manera sencilla y además son fáciles de aprender.

Entre sus principales características tenemos:

- Almacenamiento de datos en forma de documentos con estilo JSON.

- Las consultas ad hoc, la indexación y la agregación en tiempo real proporcionan formas poderosas de acceder y analizar sus datos.
- MongoDB es una base de datos distribuida en su núcleo, por lo que la alta disponibilidad, la escala horizontal y la distribución geográfica están integradas y son fáciles de usar.
- MongoDB es gratuito y de código abierto, publicado bajo la Licencia Pública General Affero de GNU.

Osmdroid y OsmBonusPack

Osmdroid es una librería para android que provee herramientas y vistas para interactuar con datos de OpenStreetMap [14]. Para renderizar el mapa con los datos de OSM se utiliza la vista de la clase MapView de osmdroid que es básicamente un remplazo de la clase MapView de Google. Adicionalmente existe un proyecto denominado OSMBonusPack que es una librería que complementa osmdroid con clases útiles para: Rutas y direcciones, puntos de interés, agrupamiento de marcadores, geocodificación y geocodificación inversa, entre otros.

Estas librerías la usamos en nuestro sistema para permitir la visualización e interacción del mapa en las aplicaciones móviles de cliente y conductor, así como para interactuar con los servidores OSRM y Nominatim para el cálculo de rutas y búsqueda de direcciones.

Open Source Routing Machine (OSRM)

OSRM es un motor de enrutamiento de alto rendimiento escrito en C++ diseñado para ejecutarse con datos de OpenStreetMap [15]. Los principales servicios que ofrece son:

- Nearest: Ajusta las coordenadas a la red de calles y devuelve las coincidencias más cercanas
- Route: encuentra la ruta más rápida entre coordenadas

- Table: calcula la duración de la ruta más rápida entre todos los pares de coordenadas proporcionadas.
- Match: ajusta las huellas de GPS ruidosas a la red de carreteras de la manera más plausible
- Trip: resuelve el problema del viajero ambulante utilizando una heurística de tipo greedy

Las principales ventajas de usar OSRM son:

- Open Source (2-clause BSD)
- Self Hosted: puede ser alojado en un servidor propio
- Utiliza una implementación de jerarquías de contracción y es capaz de calcular y generar la ruta más corta entre cualquier origen y destino en unos pocos milisegundos
- Portable
- Los datos de OSM pueden ser fácilmente exportado

Nominatim

Es un motor de búsqueda para datos de OpenStreetMap, permite realizar una búsqueda por nombre y dirección, y generar direcciones sintéticas de puntos OSM (geocodificación inversa) [16]. Nominatim referencia las características con nombre (o numeradas) con el conjunto de datos OSM y un subconjunto de otras características sin nombre (hoteles, iglesias, etc.). Los términos de búsqueda se procesan primero de izquierda a derecha y luego de derecha a izquierda en caso de que la primera falle. La búsqueda se la puede realizar con parámetros adicionales como: street, city, county, state, country, postal code y viewbox, este último permite obtener resultados de la búsqueda dentro de un área definida entre 2 puntos enviados como parámetro.

La llamada de este servidor en nuestro sistema se la realiza desde la aplicación del cliente para obtener una lista de direcciones que coincidan con la búsqueda realizada por el cliente para de esta manera permitirle elegir una de estas direcciones como destino.

Laflet

Leaflet es una librería JavaScript de código abierto para usar mapas interactivos con datos de OSM y que son visualmente amigables en dispositivos móviles [17]. Trabaja eficientemente en la mayoría de navegadores de escritorio como: Chrome, Firefox, Safari 5+, IE 7-11 y Opera 12+. Está diseñado para ofrecer simplicidad, usabilidad y buen rendimiento. Esta librería pesa acerca de 38 KB y cuenta con una gran cantidad de características necesarias para interactuar con el mapa, entre las principales características están:

- Manejo de Capas: Vector de capas (polilíneas, polígonos, círculos, rectángulos), marcadores, superposición de imágenes.
- Interacción con mapa: doble click zoom, zoom hacia un área, eventos de click y mouseover, arrastrar marcadores.
- Customizar elementos del mapa.
- Control del mapa: botón de zoom, intercambiador de capas.
- Manejo de estándar GeoJSON.

Esta librería la usamos en aplicación web monitor de nuestro sistema para permitir la visualización e interacción del mapa, así como la renderización de las rutas con su respectivo cerco virtual.

Turf

Turf es una librería JavaScript para realizar análisis espacial, incluye operaciones espaciales tradicionales, funciones de ayuda para crear datos en formato GeoJson, y herramientas para clasificar datos y crear estadísticas [18]. Las principales operaciones que ofrece esta librería son:

- Medición (área, distancia, punto medio)
- Mutación de coordenadas (flip, round, truncate)
- Transformación (rotación, traslación, escalado, buffer).

Las principales ventajas de usar esta librería son:

- Open source (MIT)
- Es simple, usa el formato de estándar abierto GeoJSON para representar datos geográficos, este formato es ampliamente usado en entornos web al permitir el intercambio de datos de manera rápida, ligera y sencilla
- Modular, es una colección de módulos pequeños y se usa solo lo que se necesita
- Se lo puede ejecutar en el servidor con Node.js

En este proyecto se lo usa principalmente para generar el cerco virtual (GeoFence) mediante el uso del método buffer que ofrece esta librería, el cual recibe como entrada los waypoints de la ruta y el radio con el que se defina el cerco.

Socket.IO

Socket.IO es una librería para aplicaciones web y móviles en tiempo real, permite la comunicación bidireccional en tiempo real entre clientes y servidores usando el protocolo WebSocket [19]. Esta librería provee la habilidad para:

- Realizar análisis en tiempo real
- Capacidad para realizar transmisión binaria
- Implementar mensajería instantánea
- Realizar aplicaciones para colaboración de documentos

Las principales ventajas de usar esta librería son:

- Open source (MIT)
- Confiabilidad, las conexiones son establecidas incluso en la presencia de proxies, balanceadores de carga, firewall personal y software antivirus
- Reconexión automática, a menos que se indique lo contrario, un cliente desconectado intentará volver a conectarse, hasta que el servidor vuelva a estar disponible.
- Detección de desconexión, permite que el servidor y el cliente sepan cuándo el otro ya no responde mediante un mecanismo heartbeat.
- Permite el uso de WebSocket como transporte

Esta librería se lo usa principalmente en este proyecto para la implementación del WebSocket Server, permitir la comunicación bidireccional del servidor con la aplicación cliente, conductor y monitor, y permitir que el sistema pueda realizar la función real time de envío de notificaciones y alertas.

CAPÍTULO 3

3. DESARROLLO DE LA SOLUCIÓN

3.1 Desarrollo de las aplicaciones móviles

Las aplicaciones móviles constan de 3 vistas con las que el usuario puede interactuar con el sistema, una vista para realizar autenticación, otra para creación de cuenta, y una vista principal donde el usuario puede realizar sus actividades principales.

Al iniciar la aplicación tanto cliente como conductor se realiza la conexión de los sockets clients con el WebSocket Server para abrir la comunicación bidireccional en tiempo real de las aplicaciones con el servidor. Esto se lo realiza usando la librería de **Socket.io** como se muestra en la Figura 3.1, además se envía información adicional del usuario al momento que se establezca la conexión para identificarlo del lado del servidor.

```
val socket = IO.socket(Env.SOCKET_SERVER_URL)
val id_user: String = User.instance._id
val role: String = User.instance.role
socket.on(Socket.EVENT_CONNECT) { it: Array<(out) Any!>!
    val userInfo = JsonObject()
    userInfo.addProperty( property: "_id", id_user)
    userInfo.addProperty( property: "role", role)
    socket.emit( event: "SENDINFO", userInfo)
```

Figura 3.1 Conexión del socket client con el WebSocket Server

3.1.1 Aplicación del cliente

En la vista principal de la aplicación del cliente, el usuario puede realizar las siguientes actividades:

- Ir a su ubicación actual dentro del mapa
- Buscar y elegir una dirección como destino

- Definir la ruta que desea seguir agregando marcadores por los puntos que desea que lo lleven
- Elegir destino agregando un marcador en algún lugar del mapa
- Seleccionar una ruta alterna por la que desea ir
- Solicitar un taxi a su ubicación actual
- Emitir una alerta de auxilio a los monitores de rutas
- Comunicarse con algún monitor que esté disponible por medio de un chat
- Calificar al final del recorrido el nivel de seguridad que considera que tuvo durante el trayecto.

Para obtener una lista de localizaciones del mapa en base a la dirección ingresada por el cliente se hace uso del servicio de geocodificación del servidor **Nominatim**, para esto usamos las clases de ayuda de la librería **OSMBonusPack**, ver Figura 3.2, con la cual se obtiene una lista de objetos con toda la información relacionada a las localizaciones que coincidieron con la busqueda.

```
override fun doInBackground(vararg params: Context?): List<Address> {
    val geoNominatim = GeocoderNominatim(Locale.getDefault(), System.getProperty("http.agent"))
    //uncomment for use own server
    geoNominatim.setService(Env.NOMINATIM_SERVER_URL)
    var addresses = listOf<Address>()
    try {
        addresses = geoNominatim.getFromLocationName(locationName,MAX_RESULTS,
                LOWER_LEFT_LATITUDE, LOWER_LEFT_LONGITUDE,
                UPPER_RIGHT_LATITUDE, UPPER_RIGHT_LONGITUDE)
    } catch (e: IOException) {
        return addresses
    }

    return addresses
}
```

Figura 3.2 Uso del servicio geocodificación de Nominatim

Como se observa en la Figura 3.2 con el método *getService* de la clase *GeocoderNominatim* que nos provee **OSMBonusPack** se especifica la url del host donde se encuentra nuestra propia instancia del servidor Nominatim para evitar usar el servidor original, el cual tiene limitaciones en

el número de búsquedas que se puede realizar al dia. Además, se especifica el número máximo de resultados que se desa de la búsqueda.

Con los resultados obtenidos se extrae la información relevante como nombre y posición de la dirección, la cual es mostrada al cliente por medio de un *RecyclerView*, ver Figura 3.3, además se usa la posición de la dirección que seleccione el cliente para realizar el cálculo de la ruta.

Figura 3.3 Resultados de la dirección buscada

El cliente también tiene la opción de definir los puntos por los cuales desea que lo lleve el conductor, para esto la aplicación le permite agregar marcadores entre los puntos de inicio y final del recorrido, cada vez que se agregue, cambie o se elimine uno de estos marcadores se vuelve a calcular la ruta, ver Figura 3.4.

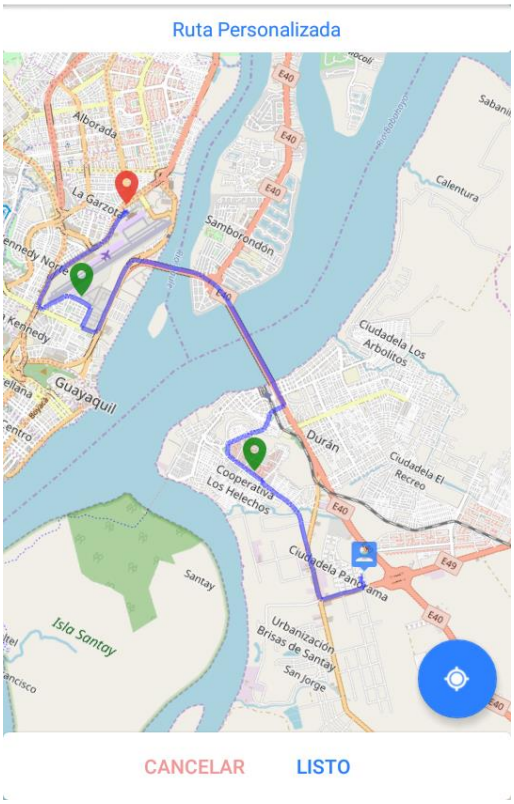


Figura 3.4 Ruta personalizada

Para el cálculo de la ruta se hace uso del servicio Route del servidor **OSRM**, para esto se realiza un HTTP request con la ayuda de la librería **Retrofit**, los parámetros que se especifican en el request son los puntos de inicio y fin de la ruta, los cuales se obtiene de la posición actual del cliente y la posición elegida como destino, además de otros parámetros adicionales como *alternatives* que se especifica para obtener rutas alternativas y las cuales son mostradas cuando el cliente desee tomar una ruta alterna a la obtenida, ver Figura 3.5. En caso de que el usuario defina la ruta con marcadores, se debe enviar como parámetros los puntos de cada uno de los marcadores para que se calcule una ruta que pase por cada uno de estos puntos. El resultado que se obtiene de esta petición es un string en formato JSON con información de las rutas calculadas, el cual es analizado para extraer dicha información.

```
override fun doInBackground(vararg params: Context?): Response<String>? {
    val serverCall = osrmAPI?.getOsrmRoutes(startLong, startLat, endLong, endLat)
    if (serverCall != null ) {
        return serverCall.execute()
    }
    return null
}

interface OsrmAPI {
    @GET( value: "route/v1/car/{fromLong},{fromLat};{toLong},{toLat}" +
            "?alternatives=true&overview=full&geometries=polyline&" +
            "steps=true&annotations=true")
    fun getOsrmRoutes(
            @Path( value: "fromLong") fromLong: String,
            @Path( value: "fromLat") fromLat: String,
            @Path( value: "toLong") toLong: String,
            @Path( value: "toLat") toLat: String): Call<String>
}
```

Figura 3.5 Request al OSRM Server

La información que se extrae se la guarda en la clase *Road* que nos provee **OSMBonusPack** para facilitar el manejo de los datos, ver Figura 3.6. Para renderizar la ruta en el mapa se utiliza un método de la clase *RoadManager* para construir la línea de puntos en base a los puntos de la ruta, ver Figura 3.7.

```
val route_geometry = jRoute.getString( name: "geometry")
road.mRouteHigh = PolylineEncoder.decode(route_geometry, precision: 10, hasAltitude: false)
road.mBoundingBox = BoundingBox.fromGeoPoints(road.mRouteHigh)
road.mLength = jRoute.getDouble( name: "distance") / 1000.0
road.mDuration = jRoute.getDouble( name: "duration")
//legs:
val jLegs = jRoute.getJSONArray( name: "legs")
for (l in 0 until jLegs.length()) {
    //leg:
    val jLeg = jLegs.getJSONObject(l)
    val leg = RoadLeg()
    road.mLegs.add(leg)
    leg.mLength = jLeg.getDouble( name: "distance")
    leg.mDuration = jLeg.getDouble( name: "duration")
    //steps:
    val jSteps = jLeg.getJSONArray( name: "steps")
    var lastNode: RoadNode? = null
    var lastRoadName = ""
    for (s in 0 until jSteps.length()) {
        val jStep = jSteps.getJSONObject(s)
        val jStepManeuver = jStep.getJSONObject( name: "maneuver")
        val jIntersections = jStep.getJSONArray( name: "intersections")
        for (j in 0 until jIntersections.length()){
            val node = RoadNode()
            node.mLength = jStep.getDouble( name: "distance") / 1000.0
            node.mDuration = jStep.getDouble( name: "duration")
            val intersection = jIntersections.getJSONObject(j)
            val location = intersection.getJSONArray( name: "location")
            val longitude = location[0] as Double
            val latitude = location[1] as Double
            node.mLocation = GeoPoint(latitude, longitude)
            road.mNodes.add(node)
        }//intersections
```

Figura 3.6 Extracción de información de las rutas

```
fun drawRoad(road: Road, userPos: GeoPoint, destinationPos: GeoPoint, score: Int?) {
    if (roadOverlays.isNotEmpty()) {
        roadOverlays.clear()
    }
    if (!road.mNodes.isEmpty()) {
        val roadOverlay = RoadManager.buildRoadOverlay(road)
        roadOverlay.color = ROAD_COLORS["chosen"]!!
        map?.overlays?.add(roadOverlay)
```

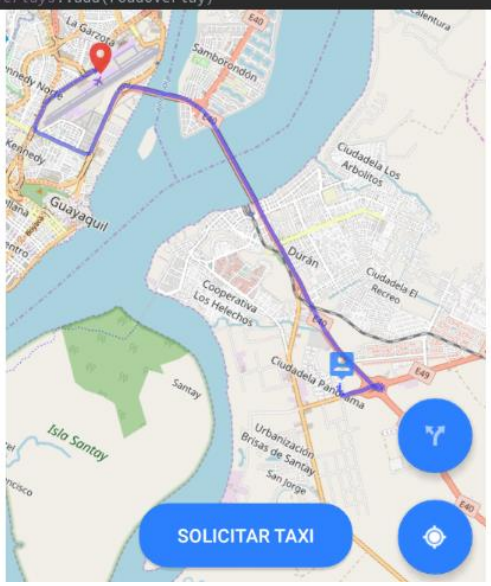


Figura 3.7 Renderizar ruta en el mapa

Cuando finaliza un recorrido, al usuario se le muestra un diálogo para que pueda calificarlo en base a la seguridad que sintió durante ese recorrido.

Como se mencionó anteriormente con la librería **Socket.io** se realiza la conexión de las aplicaciones con el WebSocket Server y de esta manera poder manejar los eventos de notificaciones, alertas, y transmisión de mensajes de texto en tiempo real, en la Tabla 3.1 se muestra una lista y una breve descripción de los eventos que son manejados en la aplicación cliente.

Evento	Descripción
ROUTE - POSITION DRIVER	Ocurre cada vez que cambia la posición del conductor, la cual es mostrada al cliente.
MONITOR - CONNECTED	Ocurre cuando se conecta un

	usuario monitor, se recibe su información y se agrega a la lista de usuarios disponibles en el chat.
MONITOR - DISCONNECTED	Ocurre cuando se desconecta un usuario monitor, el cual es eliminado de la lista de usuarios disponibles en el chat.
ROUTE - CHAT	Ocurre cada vez que se recibe un mensaje de algún usuario monitor.
DRIVER - CHOSEN	Ocurre cuando se encuentra un usuario disponible para realizar el recorrido solicitado por el cliente, se recibe la información del conductor y se la muestra al cliente.
ROUTE - START	Ocurre cuando la ruta se activa, es decir, cuando se inicia el recorrido.
ROUTE CHANGE - REQUEST	Ocurre cuando el conductor realiza una petición de cambio de ruta al cliente, se recibe información de la ruta y se la muestra al cliente.
ROUTE - FINISH	Ocurre cuando finaliza el recorrido.

Tabla 3.1 Socket events manejados en la aplicación del cliente

Cuando el cliente solicite un taxi, la aplicación realiza una petición al API REST para guardar la ruta solicitada y este se encargara de asignar a un conductor que esté disponible y que esté más cercano al cliente.

3.1.2 Aplicación del conductor

En la vista principal de la aplicación del conductor, el usuario puede realizar las siguientes actividades:

- Ir a su ubicación actual dentro del mapa
- Ver rutas alternas

- Solicitar al cliente un cambio de ruta
- Emitir una alerta de auxilio a los monitores de rutas
- Comunicarse con algún monitor que esté disponible por medio de un chat

Para el desarrollo de los requisitos funcionales de esta aplicación se reutilizaron algunos de los algoritmos y librerías que se usaron para la aplicación del cliente.

Cuando llega una solicitud de carrera de un cliente, al conductor se le muestra información del cliente que hizo la solicitud y además la ruta que debe seguir el conductor para llevar al cliente a su destino, la información de la ruta se la obtiene desde el manejador de socket events de la aplicación.

El conductor puede ver las rutas alternativas a la ruta solicitada si es que estas existen, y puede solicitar al cliente un cambio de ruta antes o luego de haber iniciado el recorrido. Usando la misma estrategia que se usó para la aplicación del cliente se obtienen las rutas mediante un HTTP request a el servicio Route del servidor **OSRM**, se extrae y se guarda la información de las rutas usando la clase *Road* que nos ofrece **OSMBonusPack** para finalmente construir los puntos de la ruta y renderizarlos en el mapa ver Figura 3.8.

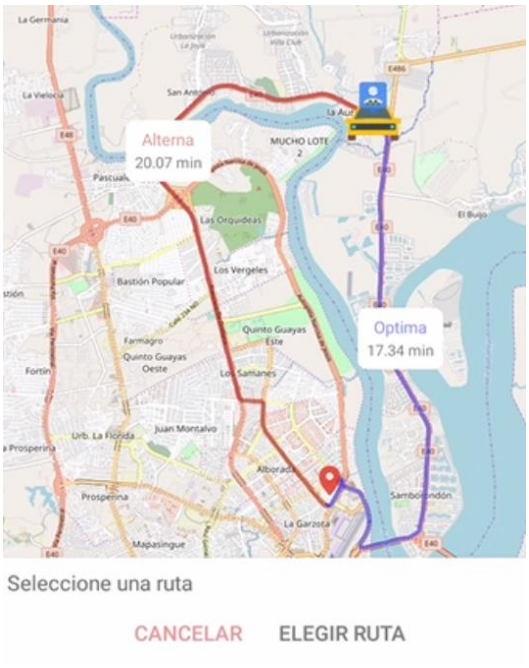


Figura 3.8 Ruta alterna

Cuando el conductor envía la solicitud de cambio de ruta, este debe esperar una notificación con un mensaje indicando si la ruta fue aceptada o negada por el cliente.

De la misma manera que en la aplicación del cliente se utilizó la librería **Socket.io** para manejar los eventos de notificaciones, alertas, y transmisión de mensajes de texto en tiempo real, en la Tabla 3.2 se muestra una lista y una breve descripción de los eventos que son manejados en esta aplicación.

Evento	Descripción
ROUTE - POSITION CLIENT	Ocurre cada vez que cambia la posición del cliente, la cual es mostrada al conductor.
MONITOR - CONNECTED	Ocurre cuando se conecta un usuario monitor, se recibe su información y se agrega a la lista de usuarios disponibles en el chat.

MONITOR - DISCONNECTED	Ocurre cuando se desconecta un usuario monitor, el cual es eliminado de la lista de usuarios disponibles en el chat.
ROUTE - CHAT	Ocurre cada vez que se recibe un mensaje de algún usuario monitor.
ROUTE REQUEST	Ocurre cuando un cliente solicita un taxi, se obtiene y se muestra información del cliente y la ruta solicitada.
ROUTE - START	Ocurre cuando la ruta se activa, es decir, cuando se inicia el recorrido.
ROUTE CHANGE - RESULT	Ocurre cuando el cliente acepta o niega el cambio de ruta solicitada por el conductor.
ROUTE - FINISH	Ocurre cuando finaliza el recorrido.

Tabla 3.2 Socket events manejados en la aplicación del conductor

3.2 Desarrollo de aplicación web

Para el desarrollo de la aplicación web realizamos la división de responsabilidades por módulos para obtener un mayor modularidad en el código, permitiendo un mejor entendimiento del código. Los módulos utilizados se muestran a continuación:

Auth.js:

Mediante este archivo exponemos 2 funciones disponibles al usuario llamadas login y logout las cuáles son utilizables por el usuario mientras se encuentra dentro de la aplicación, en la Figura 3.9 se las puede analizar.

Cuando el usuario presiona el botón para autenticarse llamamos a la función *login* la cual mediante la librería **Axios** envía las credenciales del usuario al servidor para su respectiva evaluación. Dependiendo de la respuesta del servidor

obtendremos un label que indica el estado de nuestra llamada; LOGIN representa qué recién se realizó la llamada, LOGIN_SUCCESS y LOGIN_FAIL indican la terminación de la solicitud HTTP, y a su vez la interfaz gráfica actualiza su estado dependiendo del label asignado.

```
export function login(username, password) {
  return {
    types: [LOGIN, LOGIN_SUCCESS, LOGIN_FAIL],
    payload: {
      request: {
        method: 'post',
        url: '/users/auth_monitor',
        data: {
          username,
          password,
        }
      },
    }
  };
}

export function logout() {
  return {
    types: [LOGOUT, LOGOUT_SUCCESS, LOGOUT_FAIL],
    payload: {
      request: {
        method: 'post',
        url: '/users/logout_monitor',
      },
    }
  };
}
```

Figura 3.9 Funciones expuestas por el módulo auth

A continuación, mostramos lo que ocurre en la interfaz gráfica luego de efectuar la llamada a la función login:

Al momento de iniciar el llamado al endpoint del servidor, dentro de nuestra aplicación tenemos el estado *loggingIn* qué es un booleano que representa el estado de autenticación del cliente, su asignación se puede ver en la Figura 3.10.

```
case LOGIN:
  return {
    ...state,
    loggingIn: true
  };
case LOGIN_SUCCESS:
  return {
    ...state,
    loggingIn: false,
    user: action.payload.data
  };
case LOGIN_FAIL:
  return {
    ...state,
    loggingIn: false,
    user: null,
    loginError: action.error
  };
```

Figura 3.10 Estado de nuestra aplicación durante el progreso de la autenticación.

Dentro de nuestro componente de react validamos el estado y dependiendo del mismo renderizamos nuestra vista. Cuando la variable *loggingIn* está en true nos indica que aún se está realizando el procesamiento de la autenticación del lado del servidor y estamos esperando su respuesta por ende mostramos un spinner y deshabilitamos las entradas.

Una vez que se termina la autenticación de manera exitosa ingresamos en la pantalla principal.

Map.js:

En este módulo adjuntamos el mapa proveniente de la librería **LeafletJS** al estado de nuestra aplicación. Esto se ejecuta una vez cuando iniciamos sesión, y es la instancia que se encarga del manejo de objetos renderizados dentro del mapa, en la Figura 3.11 podemos observar cómo realizamos la inicialización del mapa dentro de la aplicación.

```
init() {
  if (this.state.map) return;
  // this function creates the Leaflet map object and is called after the Map component mounts
  const map = L.map('map', config.params);
  L.control.zoom({ position: 'bottomleft' }).addTo(map);
  L.control.scale({ position: 'bottomleft' }).addTo(map);

  // a TileLayer is used as the "basemap"
  this.props.setMap(map);
}
```

Figura 3.11 Función que agrega el mapa al estado de la aplicación.

Routes.js:

Dentro de este archivo exponemos al monitor las funciones necesarias para realizar un manejo básico de las rutas dentro del mapa. Al cargar la vista principal hacemos una llamada **HTTP** mediante la librería **Axios** al servidor pidiendo la información de las rutas activas e información básica de los usuarios que la conforman, la función que realiza esto se puede observar en la Figura 3.12.

```
export function getActiveRoutes() {
  return {
    types: [GET_ROUTES, GET_ROUTES_SUCCESS, GET_ROUTES_FAIL],
    payload: {
      request: {
        method: 'get',
        url: '/routes/active',
      },
    }
  };
}
```

Figura 3.12 Función utilizada para traer las rutas activas.

Además de esto, tenemos funciones necesarias para el manejo de las rutas dentro del mapa, las cuáles nos permiten dibujarlas y removerlas, por ejemplo para renderizar una ruta, debemos enviarle el router qué es una variable creada desde **Leaflet** para indicar ciertas opciones de configuración de la ruta que será dibujada, le enviamos la ruta que consta de todos los puntos que serán dibujados, está función se puede visualizar en la Figura 3.13.

```
export const addRenderedRoute = (
  router,
  routeId,
  route,
  map,
  iconDriver,
  iconClient
) => (dispatch, getState) => addRouteToMap(
  router,
  route,
  map,
  iconDriver,
  iconClient
).then((values) => {
  dispatch({
    type: ADD_RENDERED_ROUTE,
    payload: {
      line: values.line,
      popup: values.popup,
      polygon: values.polygon,
      markerDriver: values.markerDriver,
      markerClient: values.markerClient,
      router,
      routeId,
    }
  });
});

export const deleteRenderedRoute = (map, routeId) => ({
  type: DELETE_RENDERED_ROUTE,
  payload: {
    map,
    routeId,
  }
});
```

Figura 3.13 Funciones usadas para el manejo de las rutas dentro de la interfaz gráfica.

Una vez que ejecutamos nuestra función desde nuestro componente de **ReactJS**, junto con la función *buffer* que nos provee la librería **Turf**, se crea el polígono a partir de los puntos de la ruta como se puede ver en la Figura 3.14.

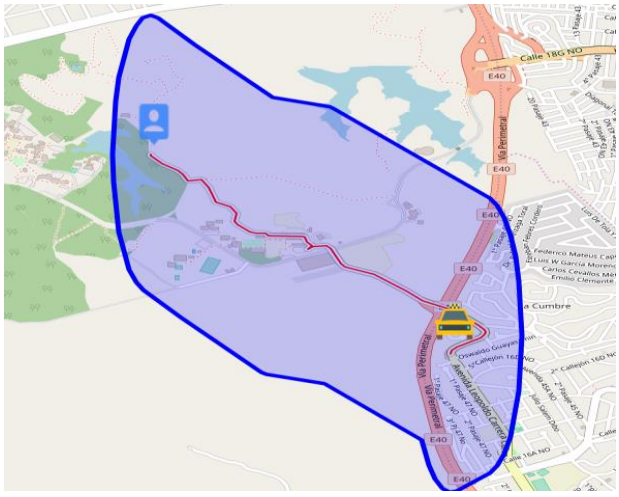


Figura 3.14 Aplicación web del monitor mostrando ruta activa.

Aparte del uso de los módulos, luego de autenticarse iniciamos el registro del socket, esto ocurre en el archivo **Routelist.js,** lo cual se observa en la Figura 3.15.

```
export default class RouteList extends React.Component {
  constructor(props) {
    const socket = io('http://localhost:9000', { reconnect: true });
    super(props);
    this.state = {
      socket,
      notificationSystem: null,
      chat: new Map(),
      selectedChat: { route_id: null, role: null },
      messageDialog: {
        open: false,
        title: 'SELECCIONE EL USUARIO'
      },
      ChatboxDialog: {
        open: false,
```

Figura 3.15 Estado inicial del componente RouteList.

Después de inicializar el componente y ya teniendo el socket activo suscribimos al socket a varios canales por los cuáles el Websocket Server le envía mensajes de manera asíncrona y dentro de nuestra aplicación capturamos ese mensaje y realizamos una acción en base a ello.

La manera en que suscribimos el socket al canal fue hecha con indicaciones del API de la librería **socket-io-client.** Cuando realizamos la conexión enviamos un mensaje al canal *SENDINFO* el cual nos ayuda a registrar al usuario monitor dentro del servidor para ser usado por el servidor más adelante, esto se puede observar en la Figura 3.16.

```
socket.on('connect', () => {
  socket.emit('SENDINFO', { ...this.props.user, role: 'monitor' });
});
```

Figura 3.16 Enviando información del monitor apenas realizamos la conexión al WebSocket Server.

Dado que dentro de nuestra interfaz web tenemos la capacidad de ver/esconder alguna ruta en tiempo real, usamos un checkbox dibujado al lado izquierdo de cada resumen de la ruta para activar/desactivar el geotracking en dicha ruta. Primero filtramos la ruta que ha sido seleccionada, dependiendo del valor del atributo *route_id* asignado a dicho checkbox. Luego revisamos si ha sido activado el checkbox o desactivado. Dependiendo de ello si está activado, entonces removemos esa ruta del mapa y detenemos el geotracking a dicha ruta, caso contrario (recién se selecciona el checkbox) pintamos la ruta en el mapa usando la función *addRouteToMap,* luego por medio de la función *joinRoom* iniciamos el geotracking a la ruta seleccionada, y luego de esto ya podemos ver el estado de la ruta en tiempo real, este procedimiento se puede visualizar en la Figura 3.17.

```
handleCheck(event, isInputChecked) {
  const routes = this.props.routes.filter(route => route._id == event.target.getAttribute('route_id'));
  if (isInputChecked) {
    const routeRendered = this.props.routesRendered[event.target.getAttribute('route_id')];
    this.routeListCheckbox[event.target.getAttribute('route_id')] = true;
    if (routeRendered) {
      return;
    }
    if (routes.length === 1) {
      this.addRouteToMap(routes[0]);
      this.joinRoom(routes[0]);
      this.state.notificationSystem.addNotification({
        title: 'Mostrando ruta',
        level: 'success',
        position: 'br',
        autoDismiss: 1
      });
    }
  } else {
    this.routeListCheckbox[event.target.getAttribute('route_id')] = false;
    this.removeRouteFromMap(routes[0]);
    this.leaveRoom(routes[0]);
  }
}
```

Figura 3.17 Enviando información del monitor apenas realizamos la conexión al WebSocket Server.

Desarrollo del chat:

Esta parte de la aplicación web se encarga del manejo de mensajería con usuarios en rutas activas dentro de la aplicación, para ello tenemos ciertas

variables vinculadas al chat dentro de nuestro sistema las cuáles se muestran en la Figura 3.18.

En el estado de la aplicación creamos inicialmente una variable llamada *chat* tipo diccionario el cual es usado para acceder rápidamente a los chats usando el id de la ruta y el rol del usuario con el que se quiere chatear como *llave*, y una lista de mensajes es el *valor*.

La variable *selectedChat* representa el id de la ruta y rol que son seleccionados por el monitor, gracias a ella en cualquier parte de nuestra aplicación sabremos si el cuadro de diálogo del chat está o no abierto, en base a ello podemos reaccionar de manera diferente cuando llega un mensaje.

La variable *ChatboxDialog* representa el cuadro de diálogo que aparece cuando seleccionamos el chat con un usuario, básicamente lo usamos para abrir o cerrarlo por medio de la variable *ChatboxDialog.open*.

Figura 3.18 Estado del chat dentro de la aplicación

Para realizar el envío de un mensaje primero obtenemos el valor tenemos dentro de nuestra caja de texto usando la variable *evt*. Luego de adjuntar el identificador del usuario remitente que se encuentra en la variable *from,* adjuntamos el identificador del usuario destinatario y el id de la ruta a la que pertenece, esta información se encuentra dentro de la variable *selectedChat* previamente mencionada y finalmente usamos nuestro websocket para realizar el envío del payload por el canal ‘ROUTE-CHAT’ el cual es reenviado por el servidor de websockets al respectivo cliente destinatario.

Una vez que enviamos este mensaje, ahora si actualizamos los mensajes existentes y corremos la función *forceUpdate* para renderizar el mensaje recién enviado dentro de la aplicación, este proceso de envío de mensaje dentro de la aplicación web se puede observar en la Figura 3.19.

```
sendMessage(evt) {
  let message = {
    from: this.props.user._id,
    position: 'left',
    type: 'text',
    text: evt.target.value,
    date: Date.now(),
  };
  try {
    this.state.socket.emit('ROUTE - CHAT', { ...this.state.selectedChat, message });

    message = { ...message, position: 'right' };
    // datasource feed
    const routeId = this.state.selectedChat.route_id;
    const role = this.state.selectedChat.role;
    const dupleRouteChat = this.state.chat.get([routeId, role].toString());
    if (!dupleRouteChat) {
      this.state.chat.set([routeId, role].toString(), []);
    }
    this.state.chat.get([routeId, role].toString()).push(message);
    this.forceUpdate();
  } catch (e) {
    console.error('Something wrong happened, ', e);
  }
}
```

Figura 3.19 Envío de mensaje del monitor hacia un usuario (Cliente, Monitor)

3.3 Desarrollo de servidor

3.3.1 API REST

Este servicio en general consta de un conjunto de funciones y procedimientos que nos ayudan a establecer una comunicación HTTP con las aplicaciones móviles usadas por el cliente y el conductor, y por la aplicación web usada por el monitor.

Este servicio web ha sido dividido en 2 partes principales: Manejo de usuarios y rutas. Hay endpoints realizados para ser consumidos por los usuarios para su respectiva autenticación, filtrado, creación y actualización de los mismos.

En la Tabla 3.3 se puede observar los principales endpoints que el API soporta.

Función	Método	Ruta
Crear usuario	POST	v1/users
Actualizar usuario	PATCH	v1/users/:id
Eliminar usuario	PATCH	v1/users/:id
Obtener usuario	GET	v1/users/:id
Crear ruta	POST	v1/routes

Tabla 3.3 Endpoints del API

Cuando el cliente solicita un taxi, el servicio web se encargará de crear en la base de datos la ruta solicitada y selecciona entre los conductores disponibles al que esté más cercano al cliente. Para seleccionar el conductor más cercano se utiliza el método *distance* que nos provee la librería **Turf**, este método mide las distancia que existe entre dos puntos, de esta manera se mide la distancia que existe entre la posición de cada uno de los conductores y el cliente, para finalmente elegir al conductor cuya distancia sea la menor, en la Figura 3.20 se muestra el código donde se hace la comparación usando este método.

```
const tmpRoute = await Route.findOne({
  driver: driver._id,
  status: { $in: ['active', 'pending'] },
}).exec().catch((e) => {
  console.error('Something bad happened, ', e);
});
if (!tmpRoute) {
  const user = await User.get(driver._id);
  const position = point([user.location.coordinates[1], user.location.coordinates[0]]);
  if (distance(start, position) < maxDistance) {
    driverChosen = driver;
    maxDistance = distance(start, position, { units: 'kilometers' });
  }
}
```

Figura 3.20 Uso de Turf para elegir al conductor más cercano.

En el momento que un conductor es seleccionado, se notifica al conductor la solicitud de carrera, y al cliente el conductor que ha sido seleccionado, esto se lo realiza emitiendo un *socket event* a través del servidor WebSocket ver Figura 3.21.

```
req.app.io.to(driverChosen.socketId).emit('ROUTE REQUEST', { user, route });
if (client) {
  req.app.io.to(client[0].socketId).emit('DRIVER - CHOSEN', driver);
} else console.info("Client disconnected")
```

Figura 3.21 Envío de notificación al cliente y conductor.

Para el caso en que el conductor solicite un cambio de ruta y esta sea aceptada por el cliente, el servidor web debe crear la nueva ruta y enviar una notificación a los monitores para que se refleje esta nueva ruta en el mapa con su respectivo cerco virtual, de igual manera se debe notificar al conductor que el cliente aceptó su solicitud para que pueda seguir el recorrido por la nueva ruta. Esto se lo realiza emitiendo un *socket event* a través del servidor WebSocket, ver Figura 3.22.

```
if (err) return err
req.app.monitors.forEach((monitor) => {
  req.app.io.to(monitor.socketId).emit('ROUTE CHANGE - RESULT', status="ok", route[0]);
});
const driverID = route[0].driver._id;
const filterDrivers = req.app.drivers.filter(driver => driver._id == driverID);
if (filterDrivers) {
  req.app.io.to(filterDrivers[0].socketId).emit('ROUTE CHANGE - RESULT', status="ok", route[0]);
}
```

Figura 3.22 Envío de notificación a los monitores y el conductor.

3.3.2 Servidor WebSocket

Debido a que se necesitan notificaciones en tiempo real para diferentes eventos que ocurren mientras la aplicación está ejecutándose, se le asigna

a cada usuario conectado un socket el cual se comunica con el servidor de Socket.IO y emite eventos al servidor, el cuál puede emitir mensajes a los diferentes sockets conectados.

```
const server = http.createServer(app);
const io = require('socket.io').listen(server);

server.listen(9000);
console.info('Socket Server listening in port 9000');
```

Figura 3.23 Inicialización del servidor de websockets.

El servidor reacciona a los eventos que son emitidos por los sockets, los cuáles pueden visualizarse en la Tabla 3.4.

Evento	Descripción
SENDINFO	Nos ayuda a agrupar a los sockets por rol.
JOIN ROUTE	Nos indica qué un socket desea ingresar a una ruta.
LEAVE ROUTE	Nos indica cuando un socket sale de una ruta.
FENCE DANGER	Nos indica qué ha ocurrido una salida del cerco virtual y los usuarios monitor deben ser notificados.
PANIC BUTTON	Nos indica que un usuario solicita ayuda mediante el botón de pánico de la aplicación.
CHAT - GET MONITORS	Nos permite obtener los monitores qué se encuentran conectados, se envía su información a los usuarios clientes.

ROUTE - CHAT	Redirige un mensaje de un monitor a algún usuario dentro de una ruta activa.
CHAT - SEND FROM USER	Redirige un mensaje enviado desde el usuario (Conductor o cliente) hacia los usuarios monitores.
POSITION	Nos ayuda a obtener la posición actual del cliente o conductor.
ROUTE CHANGE - REQUEST	Nos indica que un conductor solicitó un cambio de ruta.
ROUTE CHANGE - RESULT	Nos indica que el cliente acepto o rechazo el cambio de ruta.

Tabla 3.4 Eventos del lado del servidor

Cuando un socket se conecta llega un evento al servidor para guardar la información del socket en la lista de sockets activos los cuales se encuentran divididos por el rol del usuario al cual le pertenece el socket, ver Figura 3.24.

```
socket.on('SENDINFO', (data) => {
  let _data = {};
  const userInfo = {};

  // _data JSON containing the info.
  _data = (typeof (data) === 'string') ? JSON.parse(data) : data;

  userInfo._id = _data._id;
  userInfo.role = _data.role;
  userInfo.socketId = socket.id;
  socketClients.set(socket.id, userInfo);
  switch (userInfo.role) {
    case 'client':
      clients.push(userInfo);
      break;
    case 'driver':
      drivers.push(userInfo);
      break;
    case 'monitor':
      monitors.push(userInfo);
      notifyMonitorConnected(_data);
      break;
    default:
      break;
  }
});
```

Figura 3.24 Almacenamiento de información de los sockets activos.

Cada vez que un usuario cliente o conductor emite su posición al servidor WebSocket, este debe reenviar la posición a los usuarios monitor y al usuario cliente o conductor que estén suscritos al *room* de la ruta, ver Figura 3.25, de esta manera cada uno de los usuarios pueden ver las posiciones de los demás en tiempo real.

```
if (_data.role === 'driver') {
  io.to(_data.route_id).emit('ROUTE - POSITION DRIVER', { position: _data.position, routeId: _data.route_id });
} else if (_data.role === 'client') {
  io.to(_data.route_id).emit('ROUTE - POSITION CLIENT', { position: _data.position, routeId: _data.route_id });
} else {
  console.error('PLEASE ADD THE ROLE TO THE DATA PAYLOAD.');
}
```

Figura 3.25 Reenvío de posición de los usuarios

Además de realizar el envío de la posición también se debe verificar el estado de la ruta, esto quiere decir verificar si la ruta puede ser inicializada, finalizada, o se encuentra en peligro. Una ruta puede inicializar cuando el conductor está a menos de dos metros de distancia del cliente, esta distancia se la mide usando el método *distance* de la librería **Turf,** cuando esto sucede se emite la notificación correspondiente a los usuarios cliente, conductor y monitor para indicar que la ruta está activa. Una ruta puede finalizar cuando la distancia tanto del conductor y del cliente está a menos de 30 metros de su destino, de la misma manera la distancia se mide con usando el método *distance*, cuando esto sucede se emite la notificación a los usuarios cliente, conductor y monitor para indicar que la ruta finalizó. También se verifica si la posición que envía el usuario se encuentra dentro o fuera del cerco virtual de la ruta, para esto usamos el método *isPointInPolygon* de la librería **Turf**, en caso de que el usuario se encuentre fuera del polígono se emite una notificación de abandono de cerco a los monitores, ver Figura 3.26. Todos estos parámetros de distancias pueden ser configurados en el servidor. Las distancias que son utilizadas para iniciar y finalizar el recorrido, y para definir el cerco pueden ser configuradas en el servidor.

```
const linestring1 = lineString(coordinates);
const _point = clientPos;
const buffered = buffer(linestring1, 1, { units: 'kilometers' });
const isInBuffer = isPointInPolygon(_point, buffered.geometry);
if (!isInBuffer) {
  monitors.forEach((monitor) => {
    io.to(monitor.socketId).emit('ROUTE - DANGER', { routeId: route._id, outofBuffer: true });
  });
}
```

Figura 3.26 Verificar si usuario está dentro

Además, dentro del servidor realizamos el cálculo de la posible calificación que se le puede asignar a una ruta representada por la variable coordinates, para realizar este proceso obtenemos las rutas que han sido calificadas anteriormente y las guardamos en la variable routes.

Ahora iteramos las rutas y verificamos si alguna tiene una forma parecida a la guardada en la variable coordinates verificando si el 80% de los puntos dentro las rutas iteradas se encuentran cercanos a un máximo de 20 metros de los puntos dentro de la variable *coordinates*, si este es el caso guardamos su calificación dentro de la variable *scores*. Al terminar el proceso promediamos las calificaciones y retornamos una posible calificación de esta ruta. Si la ruta no tiene la forma de ninguna de las guardadas enviamos -1 que representa incertidumbre en el estado de la ruta. Este proceso se puede observar en la Figura 3.27.

```
// coordinates are sent from android client
exports.getRouteScore = async (coordinates) => {
  let totalScore = 0;
  const routes = await Route.listRatedRoutes();
  const scores = [];
  routes.forEach((route) => {
    let counterEqualPoints = 0;
    const linestring1 = lineString(route.points.coordinates);
    const buffered = buffer(linestring1, 0.02, { units: 'kilometers' });
    coordinates.forEach((pointEvalRoute) => {
      const _point = point(pointEvalRoute);
      const isInBuffer = isPointInPolygon(_point, buffered.geometry);
      if (isInBuffer) {
        counterEqualPoints += 1;
      }

      if (counterEqualPoints >= route.points.coordinates.length * 0.8) {
        scores.push(route.safeScore);
      }
    });
  });
  if (scores.length === 0) { totalScore = -1; } else {
    totalScore = scores.reduce(add, 0) / scores.length;
  }
  return Math.ceil(totalScore); // returning the upward integer
};
```

Figura 3.27 Obtener posible calificación de ruta.

CAPÍTULO 4

4. PRUEBAS Y RESULTADOS

En este capítulo se describen las pruebas funcionales que se realizaron en el sistema y se presentan los resultados de las mismas.

Pruebas de la aplicación móvil cliente

Escenario	Buscar un sitio destino
Caso de prueba	Buscar un sitio en el mapa dentro de la ciudad de Guayaquil
Datos de entrada	“Mi comisariato”
Resultado Esperado	Una lista de direcciones de Guayaquil que coincidan con el dato de entrada
Resultado Obtenido	
Estado	Exitoso

Tabla 4.1 Buscar un sitio destino

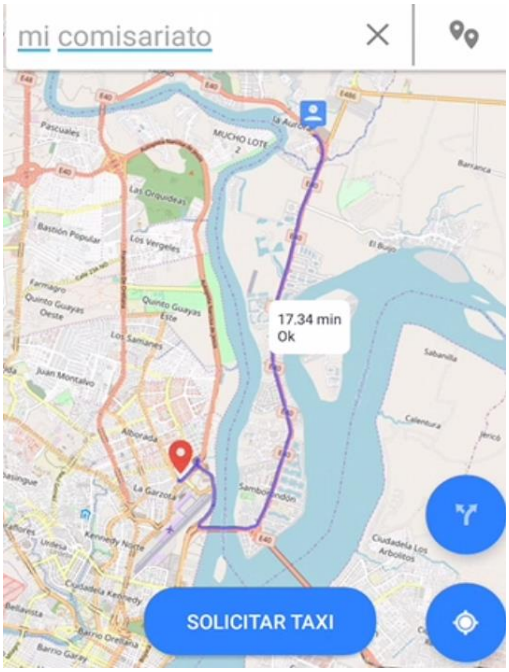
Escenario	Mostrar ruta
Caso de prueba	Mostrar en el mapa la ruta óptima calculada desde la ubicación actual a la ubicación destino
Datos de entrada	Ubicación actual: [-2.057856, -79.876340] “Vía Samborondón” Destino: [-2.1445, -79.8854] “Mi comisariato Av. Américas”
Resultado Esperado	Ver ruta óptima calculada entre ubicación actual y destino
Resultado Obtenido	
Estado	Exitoso

Tabla 4.2 Mostrar ruta

Escenario	Mostrar rutas alternativas
Caso de prueba	Mostrar en el mapa las rutas alternativas disponibles dada la ubicación actual y destino
Datos de entrada	Ubicación actual: [-2.057856, -79.876340] “Vía Samborondón” Destino: [-2.1910, -79.8927] “Mi comisariato Av. Américas”

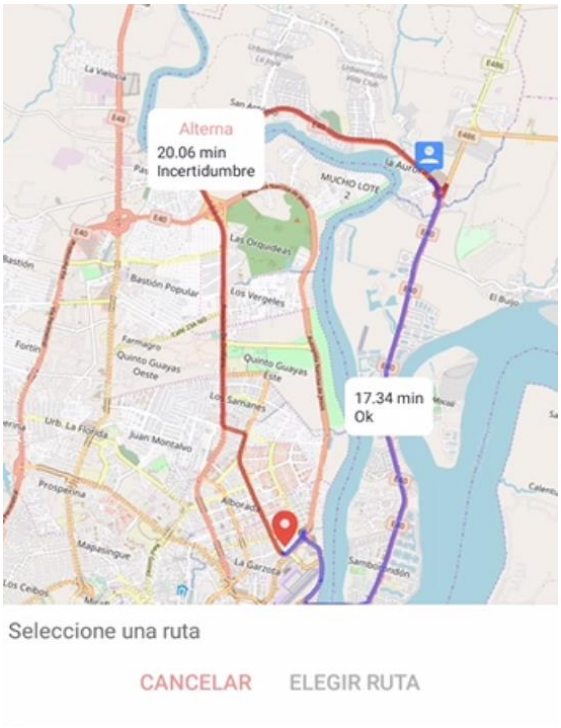
Resultado Esperado	Ver rutas disponibles entre ubicación actual y destino
Resultado Obtenido	
Estado	Exitoso

Tabla 4.3 Mostrar rutas alternativas

Escenario	Solicitar taxi
Caso de prueba	Solicitar un taxi disponible a la ubicación actual
Datos de entrada	Ubicación actual: [-2.057856, -79.876340] “Vía Samborondón”
Resultado Esperado	Se muestre un diálogo con información del conductor y el vehículo seleccionado para la ruta

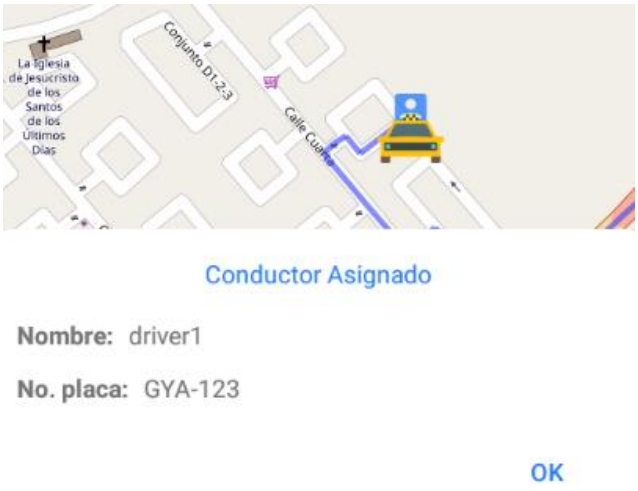
Resultado Obtenido	
Estado	Exitoso

Tabla 4.4 Solicitar taxi

Pruebas de la aplicación móvil conductor

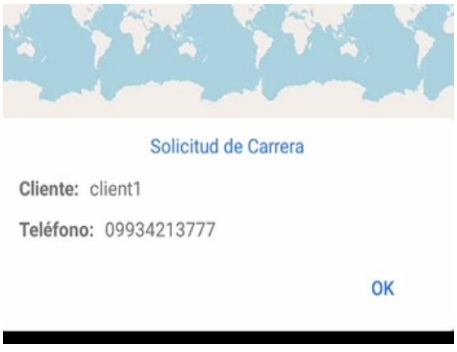
Escenario	Solicitud de carrera
Caso de prueba	Recibir una solicitud de carrera
Datos de entrada	Ubicación del cliente: [-2.057856, -79.876340] “Vía Samborondón” Destino: [-2.1910, -79.8927] “Mi comisariato Av. Américas”
Resultado Esperado	Se muestre ruta solicitada y un diálogo con información del cliente que solicitó la carrera
Resultado Obtenido	
Estado	Exitoso

Tabla 4.5 Solicitud de carrera

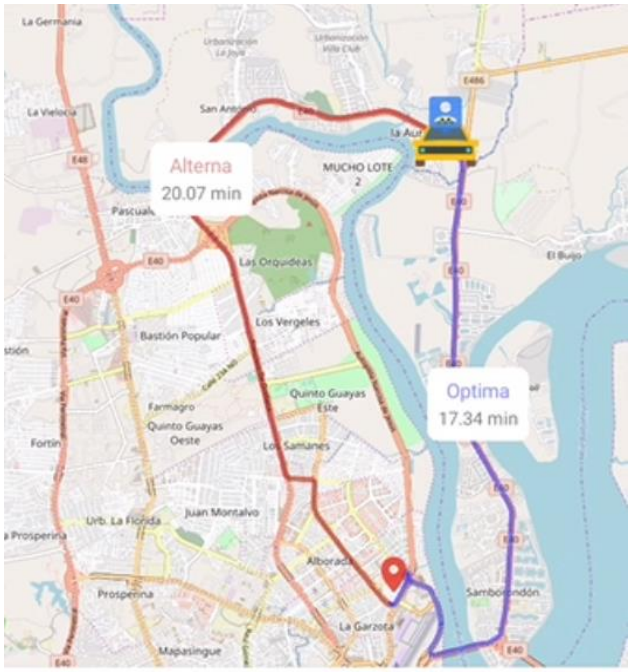
Escenario	Mostrar rutas alternativas
Caso de prueba	Mostrar en el mapa las rutas alternativas disponibles dada la ubicación actual y destino
Datos de entrada	Ubicación actual: [-2.057856, -79.876340] “Vía Samborondón” Destino: [-2.1910, -79.8927] “Mi comisariato Av. Américas”
Resultado Esperado	Ver rutas disponibles entre ubicación actual y destino
Resultado Obtenido	
Estado	Exitoso

Tabla 4.6 Mostrar rutas alternativas

Escenario	Solicitar cambio de ruta
Caso de prueba	El cliente acepta el cambio de ruta solicitada

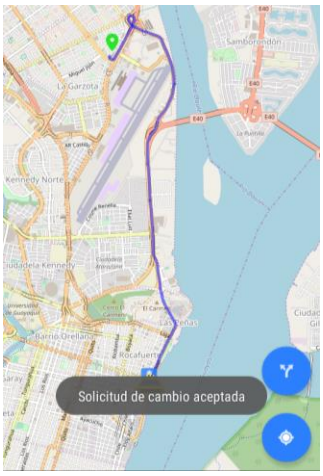
Datos de entrada	Nueva ruta solicitada
Resultado Esperado	Se muestra la notificación de la nueva ruta creada, además se muestra la ruta dentro de la lista de rutas en estado activo.
Resultado Obtenido	
Estado	Exitoso

Tabla 4.7 Solicitar cambio de ruta

Pruebas de la aplicación web monitor

Escenario	Se genera una ruta activa
Caso de prueba	Una ruta generada realiza la transición de su estado PENDIENTE -> ACTIVA.
Datos de entrada	Nueva ruta solicitada
Resultado Esperado	Se muestra la notificación de la nueva ruta creada, además se muestra la ruta dentro de la lista de rutas en estado activo.

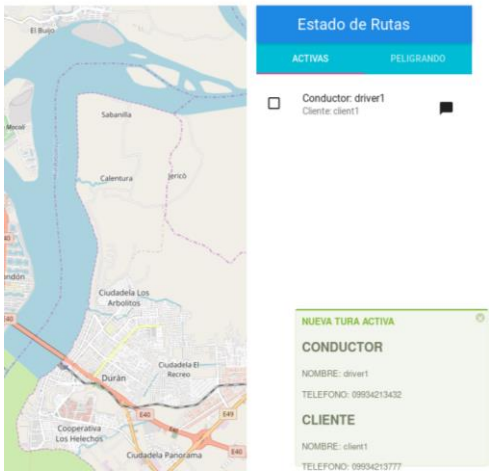
Resultado Obtenido	
Estado	Exitoso

Tabla 4.8 Ruta activa generada

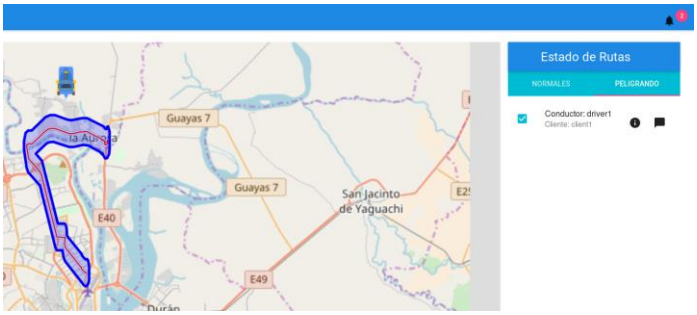
Escenario	Mostrar ruta en tiempo real
Caso de prueba	El monitor selecciona una ruta disponible en la lista de rutas.
Datos de entrada	
Resultado Esperado	La ruta se renderiza mostrando la ubicación del cliente y el monitor en tiempo real.
Resultado Obtenido	
Estado	Exitoso

Tabla 4.9 Mostrar ruta

Escenario	Cliente abandona cerco virtual
Caso de prueba	El cliente abandona el perímetro.

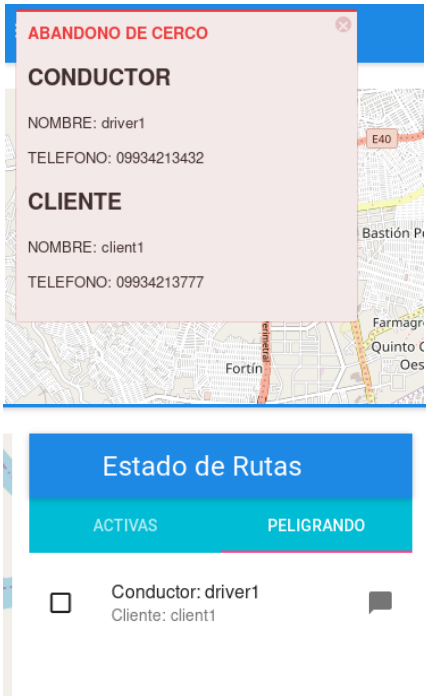
Datos de entrada	
Resultado Esperado	La aplicación monitor cambia el estado de la ruta de ACTIVA -> PELIGRANDO, se muestra la notificación de abandono de ruta y se actualizan las listas.
Resultado Obtenido	
Estado	Exitoso

Tabla 4.10 Abandono de cerco

Escenario	Mostrar notificación de alerta de auxilio
Caso de prueba	El cliente envía una alerta de auxilio.
Datos de entrada	
Resultado Esperado	La aplicación monitor cambia el estado de la ruta de ACTIVA -> PELIGRANDO, se muestra la notificación de alerta de auxilio y se actualizan las listas.
Resultado Obtenido	

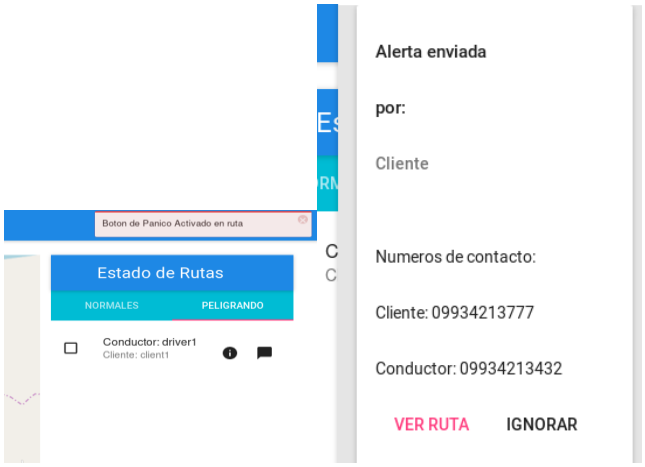
	
Estado	Exitoso

Tabla 4.11 Notificación de alerta de auxilio

Pruebas del servidor

Escenario	Crear usuario
Caso de prueba	Crear un usuario desde la aplicación cliente
Datos de entrada	{ "name": "diego", "cedula": "0974561232", "mobile": "0974521678", "username": "dieropal", "password": "1234567", "role": "client" }
Resultado Esperado	El servidor responda con status 200
Resultado Obtenido	Status: 200
Estado	Exitoso

Tabla 4.12 Crear usuario cliente

Escenario	Crear usuario
Caso de prueba	Crear un usuario desde la aplicación conductor

Datos de entrada	{ "name": "Romario", "cedula": "0914231232", "mobile": "0874351678", "username": "rom", "password": "2134523", "role": "driver", "vehicle_plate": "GYB-324", "vehicle_description": "Auto chevrolet aveo family" }
Resultado Esperado	El servidor responda con status 200
Resultado Obtenido	Status: 200
Estado	Exitoso

Tabla 4.13 Crear usuario conductor

Escenario	Crear ruta
Caso de prueba	Crear nueva ruta solicitada por el cliente
Datos de entrada	{ "start": { "type": "Point", "coordinates": [-79.8799873, -2.1936348] }, "end": { "type": "Point", "coordinates": [-79.8925899, -2.1546264] }, "points":[[-79.88005, -2.193617], [-79.879875, -2.193023], [-79.880575, -2.192786], [-79.880952, -2.194035], [-79.889973, -2.191965], [-79.888074, -2.185886], [-79.889025, -2.182875], [-79.889361, -2.181904], [-79.891787, -2.175528], [-79.89169, -2.173943], [-79.891868, -2.167607], [-79.892047, -2.165517], [-79.894577, -2.155516], [-79.894716, -2.154909], [-79.894317, -2.153173], [-79.892623, -2.154327], [-79.89259, -2.154626]], "client": "5a72682979d0ac57572e3551", "route_index": 0 }
Resultado Esperado	El servidor responda con status 200
Resultado Obtenido	Status: 200
Estado	Exitoso

Tabla 4.14 Crear una ruta

Escenario	Actualizar ruta
Caso de prueba	Actualizar ruta solicitada por el cliente

Datos de entrada	{ "routeId": "5a726fd3179de25f05be669f", "points":[[-79.88005, -2.193617], [-79.879875, -2.193023], [-79.880575, -2.192786], [-79.880952, -2.194035], [-79.889973, -2.191965], [-79.888074, -2.185886], [-79.879873, -2.169909], [-79.880563, -2.17055], [-79.89097, -2.171088], [-79.891774, -2.169277], [-79.891868, -2.167607], [-79.892047, -2.165517], [-79.894577, -2.155516], [-79.894716, -2.154909], [-79.894317, -2.153173], [-79.892623, -2.154327], [-79.89259, -2.154626]], "route_index": 1 }
Resultado Esperado	El servidor responda con status 200
Resultado Obtenido	Status: 200
Estado	Exitoso

Tabla 4.15 Actualizar ruta

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

Gracias a la integración con SocketIO se minimiza el polling al mínimo para permitir a las aplicaciones reaccionar a los eventos que ocurren en el lado del servidor.

Se implementó un sistema web y móvil que muestra el uso del geofencing para mejorar la seguridad del cliente durante su trayecto en un taxi.

Se puede realizar el seguimiento de un recorrido de taxi en tiempo real mediante un portal web de manera efectiva revisando el estado de la ruta.

No se realizaron pruebas de campo debido a que no se cuenta con la infraestructura adecuada, pero se efectuaron pruebas de aceptación para verificar el funcionamiento del software.

El sistema no siempre va a calcular la ruta óptima real debido a que no se están considerando datos de tráfico.

Recomendaciones

Usar librerías de código abierto para tener un mejor entendimiento de las librerías qué se utilizan y mejorar la confianza en las mismas.

Usar librerías muy conocidas por la comunidad como **TurfJS** para la generación del cerco virtual, esta librería es ampliamente aceptada y al ser de código abierto, la comunidad la ha ido manteniendo durante largo tiempo haciendo que la generación de nuestro cerco sea bastante buena cubriendo toda la ruta en el 100% de las rutas monitoreadas.

Considerar datos de tráfico en tiempo real para realizar el cálculo de la ruta, y de esta manera poder obtener la ruta óptima real en cualquier hora del día.

BIBLIOGRAFÍA

[1] El Comercio. “La Ruta Del Secuestro Exprés Hasta Ecuador.” [online]. Disponible en: www.elcomercio.com/actualidad/seguridad/ruta-del-secuestro-expres-hasta.html

[2] El Universo. “Cifras del secuestro exprés se ‘esconden’ entre otros delitos.” [online]. Disponible en: https://www.eluniverso.com/noticias/2014/01/19/nota/2049456/cifras-secuestro-expres-se-esconden-otros-delitos

[3] La Hora. (2017, Octubre). “Detectan Nuevo Método Para Secuestro 'Exprés' en Ecuador.” [online]. Disponible en: https://lahora.com.ec/losrios/noticia/1102103818/detectan-nuevo-metodo-para-secuestro-expres-en-ecuador

[4] ANT. “Transporte Seguro.” *[online].* Disponible en: www.ant.gob.ec/index.php/transporte-seguro.

[5] ANT. (2015, Octubre). “ANT Iniciará la segunda fase de instalación de kits de seguridad a nivel nacional - Agencia Nacional De Tránsito Del Ecuador.” *[online]. Disponible en:* http://www.ant.gob.ec/index.php/noticias/1313-ant-iniciara-la-segunda-fase-de-instalacion-de-kits-de-seguridad-a-nivel-nacional#.Whzi99-YUUt

[6] ANT. (2017). “Inversión del Gobierno Nacional para el Proyecto Transporte Seguro.” [online]. Disponible en: http://www.ant.gob.ec/phocadownload/transporte/seguro/2017/inversion_transporte_seguro.pdf

[7] *El Comercio.* (2015, Marzo). “Bandas Usan Logos Clonados En Secuestros Exprés.” *[online]. Disponible en:* www.elcomercio.com/actualidad/bandas-logos-taxi-clonacion-secuestros.html

[8] Ecuavisa. (2014, Octubre). “Asaltos en taxi seguro” [online]. Disponible en: www.ecuavisa.com/articulo/noticias/nacional/86584-asaltos-taxi-seguro-nueva-modalidad-secuestro-expres-guayaquil

[9] Syed Irfan Ajma, 2017. “Ridesharing vs. Taxi – Watch this Exciting Duel of the Century Unfold” [online]. Disponible en: https://www.ridester.com/ridesharing-vs-taxi/

[10] Guilherme Tagiaroli. (2017, Febrero 25). Titulo (edición) [online]. Disponible en: https://motherboard.vice.com/en_us/article/4x5kbm/brazils-love-affair-with-uber-has-been-ruined-by-kidnapping-robbery-and-murder

[11] Android Developers. (2017). “Dashboards” [online]. Disponible en: https://developer.android.com/about/dashboards/index.html

[12] Node.js. (2017). Disponible en: https://nodejs.org/es/about/

[13] Mongodb. (2017). Disponible en: https://www.mongodb.com/

[14] Osmdroid - OpenStreetMap Wiki. (2017). “Osmdroid” [online]. Disponible en: https://wiki.openstreetmap.org/wiki/Osmdroid

[15] Open Source Routing Machine - OpenStreetMap Wiki. (2017). “Open Source Routing Machine” [online]. Disponible en: https://wiki.openstreetmap.org/wiki/Open_Source_Routing_Machine

[16] Nominatim - OpenStreetMap Wiki. (2017). “Nominatim” [online]. Disponible en: https://wiki.openstreetmap.org/wiki/Nominatim

[17] Leaflet. (2017). [online]. Disponible en: http://leafletjs.com/

[18] Turfjs Github. (2017). [online]. Disponible en: https://github.com/Turfjs/turf

[19] Socketio Github. (2017). [online. Disponible en: https://github.com/socketio/socket.io

ANEXOS

ANEXO A1

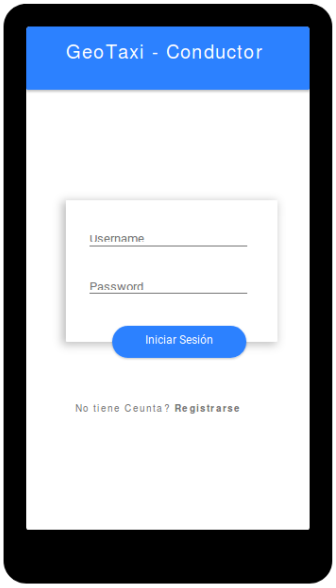


Figura A1.1 Prototipo de inicio de sesión

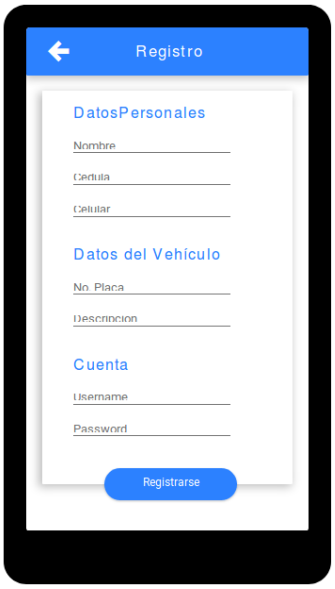


Figura A1.2 Prototipo de registro conductor

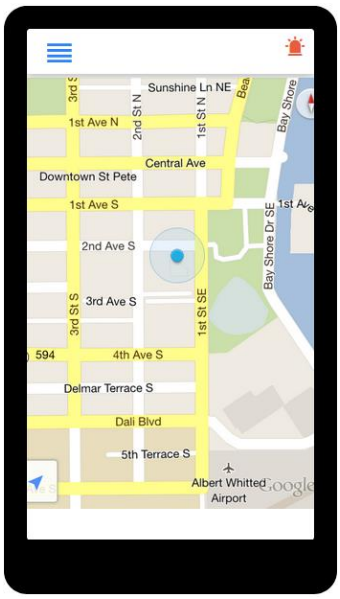


Figura A1.3 Prototipo pantalla principal conductor

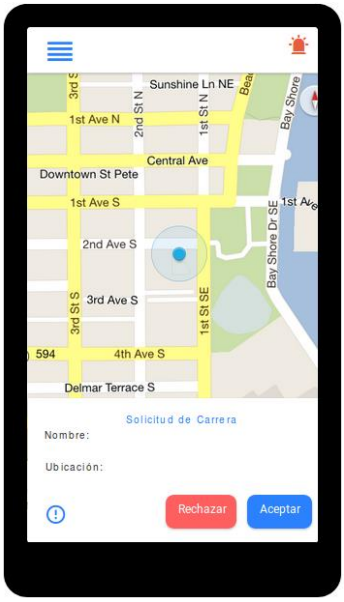


Figura A1.4 Prototipo solicitud de carrera conductor

Figura A1.5 Prototipo Selección de ruta alterna conductor

Figura A1.6 Prototipo Confirmación de la nueva ruta

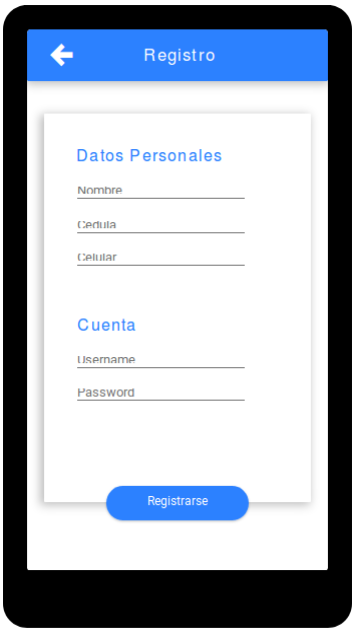


Figura A1.7 Prototipo de Registro cliente

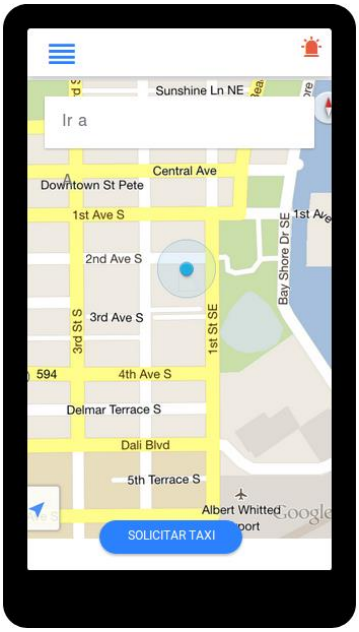


Figura A1.8 Prototipo de vista principal del cliente

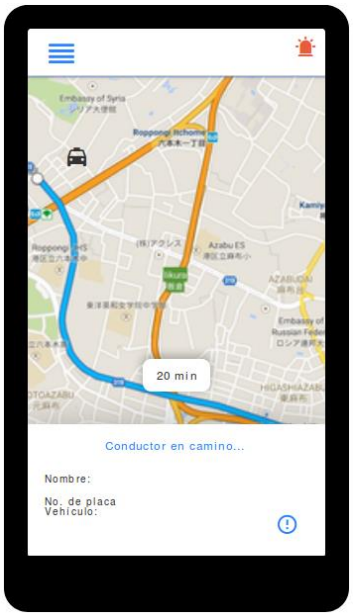


Figura A.9 Prototipo de notificación con información del conductor

Figura A.10 Prototipo de diálogo de solicitud de cambio de ruta

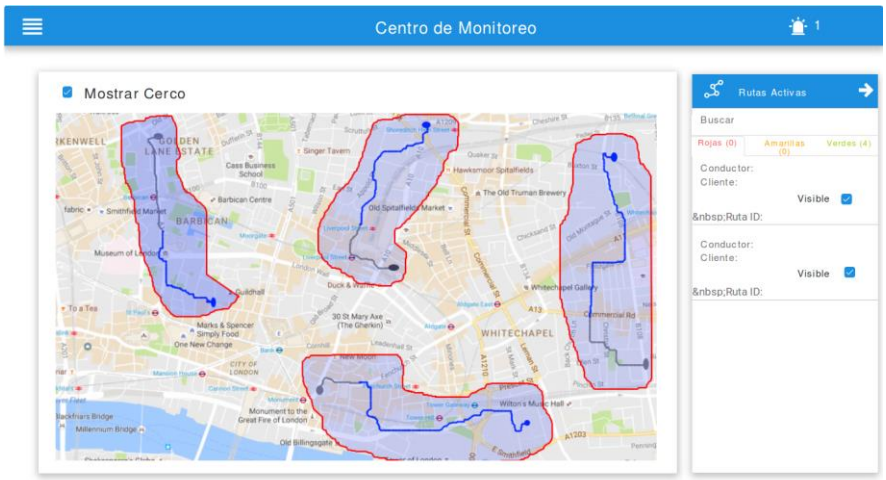


Figura A.11 Prototipo de lista de rutas activas

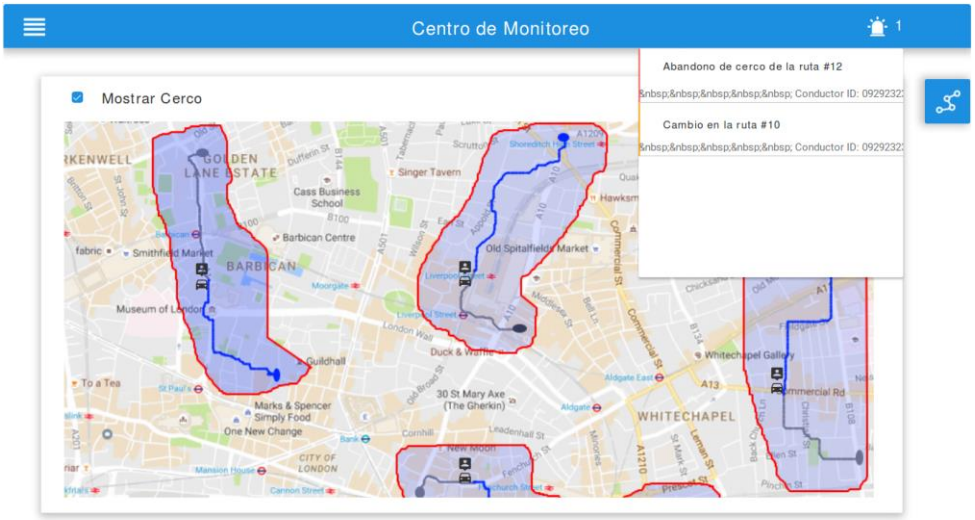
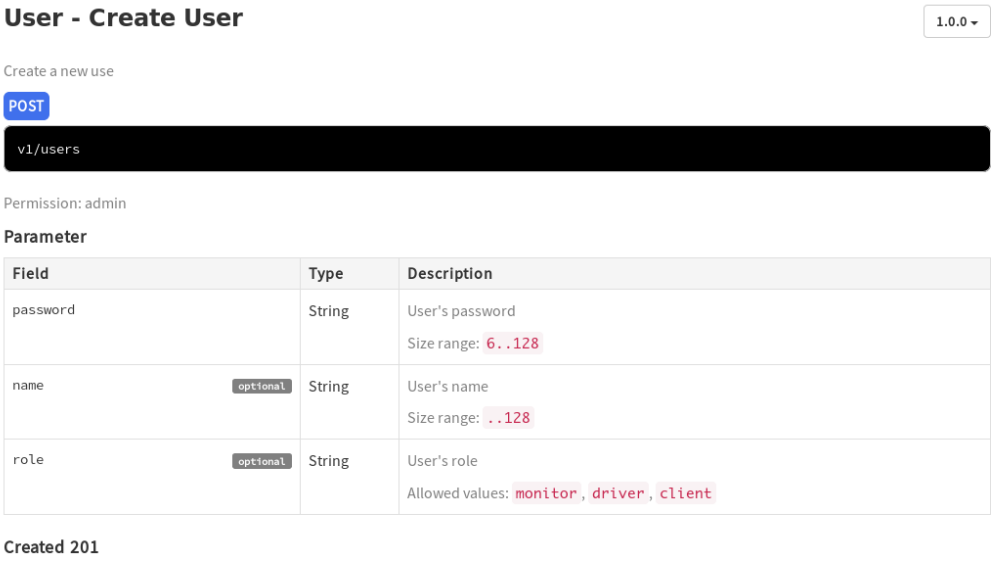


Figura A.12 Prototipo de barra de notificaciones de alertas

ANEXO A2



User - Create User 1.0.0

Create a new use

POST

v1/users

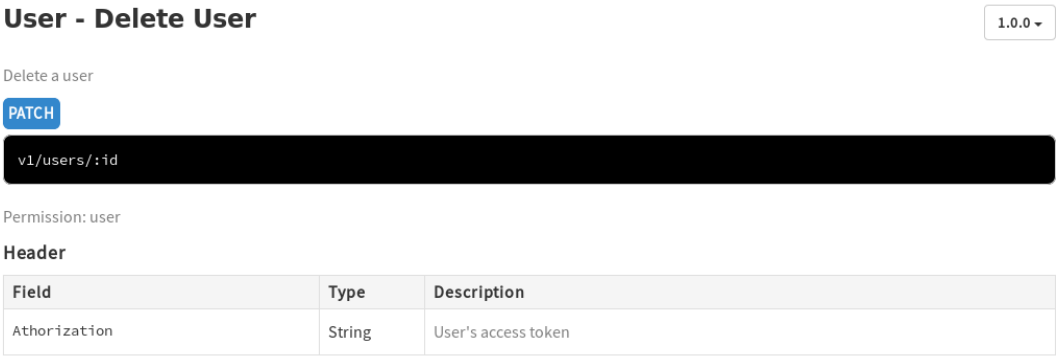
Permission: admin

Parameter

Field	Type	Description
password	String	User's password Size range: 6..128
name optional	String	User's name Size range: ..128
role optional	String	User's role Allowed values: monitor, driver, client

Created 201

Figura A2.1 Descripción de endpoint usado para la creación de un nuevo usuario.



User - Delete User 1.0.0

Delete a user

PATCH

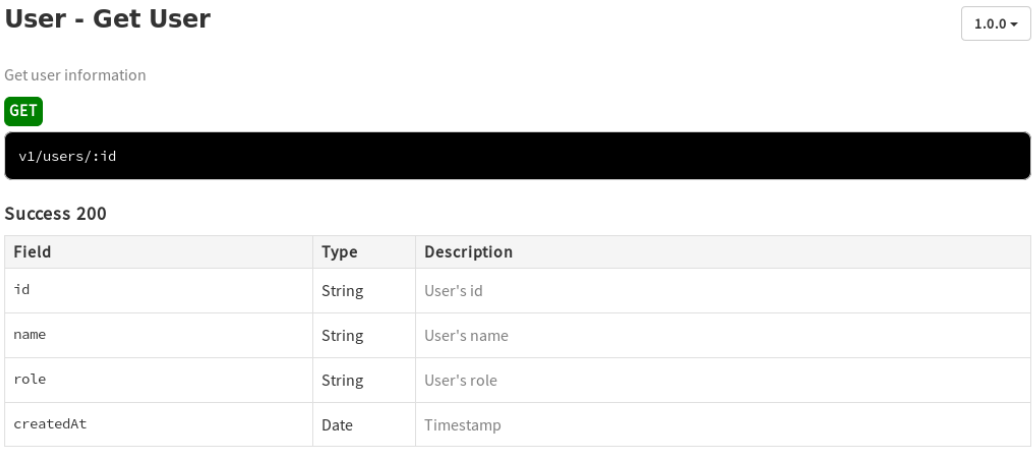
v1/users/:id

Permission: user

Header

Field	Type	Description
Athorization	String	User's access token

Figura A2.2 Descripción de endpoint usado para la eliminación de un nuevo usuario.



User - Get User

1.0.0

Get user information

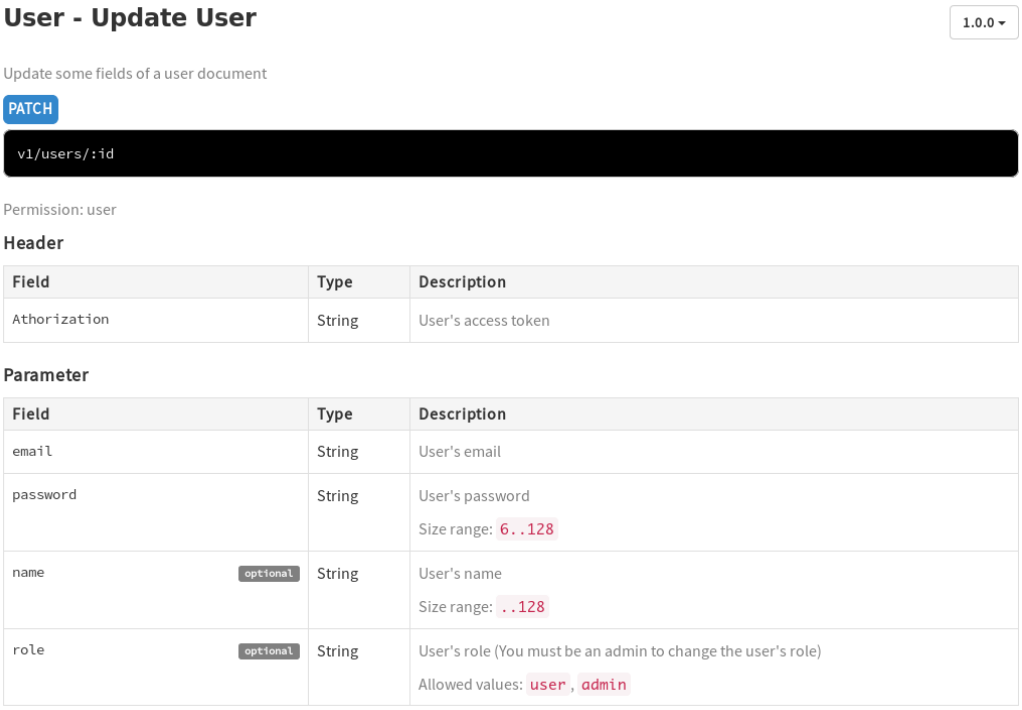
GET

```
v1/users/:id
```

Success 200

Field	Type	Description
id	String	User's id
name	String	User's name
role	String	User's role
createdAt	Date	Timestamp

Figura A2.3 Descripción de endpoint usado para obtener un usuario por medio de su ID



User - Update User

1.0.0

Update some fields of a user document

PATCH

```
v1/users/:id
```

Permission: user

Header

Field	Type	Description
Athorization	String	User's access token

Parameter

Field	Type	Description
email	String	User's email
password	String	User's password Size range: `6..128`
name optional	String	User's name Size range: `..128`
role optional	String	User's role (You must be an admin to change the user's role) Allowed values: `user`, `admin`

Figura A2.4 Descripción de endpoint usado para actualizar un usuario

Routes - Create route

1.0.0

Create a new route

POST

```
v1/routes
```

Parameter

Field	Type	Description
status optional	String	Route's status Allowed values: `inactive`, `active`, `cancelled`, `finished`
driver optional	User	Id of driver in route
client optional	User	Id of client in route(Optional)
points optional	Multipoint	Start of route
start optional	Point	Starting Point of the route.
end optional	Point	Ending Point of the route.

Created 201

Figura A2.5 Descripción de endpoint usado para crear una nueva ruta.

Routes - List Routes

1.0.0

Get a list of routes

GET

```
v1/users
```

Parameter

Field	Type	Description
page optional	Number	List page Default value: `1` Size range: `1-`
perPage optional	Number	routes per page Default value: `1` Size range: `1-100`
dateTime optional	Date	TimeStamp saving route's start
points optional	Multipoint	Start of route
start optional	Point	Starting Point of the route.
end optional	Point	Ending Point of the route.
status optional	String	Route's status Allowed values: `inactive`, `active`, `cancelled`, `finished`

Figura A2.6 Descripción de endpoint usado para crear una nueva ruta.