



Diego Felipe Solorz... Juan David Loaiza ...

Daniel Aguilar Castro

Juan David Chacon ...

Diego Felipe Solorzano Aponte

**UNIVERSIDAD NACIONAL DE  
COLOMBIA  
SEDE BOGOTÁ  
FACULTAD DE INGENIERÍA  
DEPARTAMENTO DE SISTEMAS E INDUSTRIAL  
INGENIERÍA DE SOFTWARE 1  
BOGOTÁ D.C  
2024-2**

# **Levantamiento de Requerimientos**

## **Contexto del Negocio**

La comunidad universitaria enfrenta un problema significativo relacionado con el transporte. La movilidad en la ciudad representa un desafío diario para estudiantes, profesores y trabajadores de la universidad debido a la congestión vehicular y los altos costos de transporte, lo que afecta directamente su calidad de vida y sus actividades académicas.

Como solución, se propone una herramienta que ayude a mitigar esta problemática, ofreciendo a la comunidad universitaria un mecanismo que facilite compartir viajes dentro de la ciudad y sus alrededores. Esto permitiría ahorrar dinero, tiempo y garantizar un espacio más seguro y cómodo para desplazarse.

Actualmente, hay miembros de la comunidad que cuentan con vehículos propios y realizan el recorrido entre sus hogares y la universidad diariamente, pero muchas veces tienen asientos disponibles que no son aprovechados. Por otro lado, hay estudiantes, profesores y trabajadores que hacen trayectos similares utilizando el precario sistema de transporte público, lo cual resulta ser menos conveniente y más costoso. A estos últimos les sería de gran ayuda conocer a quienes disponen de vehículos para coordinar viajes compartidos, optimizando recursos y costos.

El objetivo principal es desarrollar una solución digital que permita a los miembros de la comunidad universitaria conectarse entre sí para organizar viajes compartidos de forma sencilla, pactando entre ellos los términos y costos del trayecto.

## **Descripción General del Software**

Se propone una aplicación web accesible desde dispositivos móviles y computadoras, diseñada con una interfaz sencilla y fácil de usar. Esta herramienta permitirá, para quienes buscan compartir viajes, registrar información clave como días, horarios, ubicaciones de salida y llegada.

Por otro lado, para quienes tienen vehículo propio, podrán publicar detalles de sus rutas y horarios habituales para que otras personas interesadas puedan contactarlos fácilmente a través de la app.

De esta forma, se busca fomentar una mayor interacción entre los usuarios y facilitar acuerdos para realizar viajes compartidos, promoviendo una alternativa más económica, eficiente y segura para la movilidad dentro de la comunidad universitaria. La aplicación facilita la conexión entre conductores y pasajeros, permitiendo gestionar pagos, planificar rutas y establecer contacto seguro entre usuarios. Sin embargo, la plataforma no es responsable por el cumplimiento de los acuerdos pactados entre usuarios.

Se integrarán funciones avanzadas para optimizar la experiencia de los usuarios, como un sistema de notificaciones en tiempo real, calificaciones entre usuarios y una pasarela de pago para facilitar las transacciones. Además, se implementarán medidas para mejorar la seguridad, como verificación de identidad y registro de historial de viajes, garantizando así una mayor confianza en el uso del servicio.

## **Beneficios esperados del proyecto**

El proyecto busca generar un impacto positivo tanto en los usuarios de la aplicación como en el equipo de desarrollo.

#### **Beneficios para la comunidad universitaria:**

- **Movilidad más económica y eficiente**, permitiendo compartir viajes y reducir costos de transporte.
- **Conexión más segura** entre usuarios con trayectos similares, facilitando la coordinación de viajes compartidos.
- **Reducción de la congestión vehicular**, al incentivar el uso compartido de vehículos y disminuir la cantidad de autos en circulación.
- **Mayor accesibilidad a opciones de transporte**, especialmente para estudiantes o trabajadores que no cuentan con un vehículo propio.

#### **Beneficios para el equipo de desarrollo:**

- **Adquisición de experiencia práctica** en tecnologías modernas como TypeScript, NestJS y React Native.
- **Trabajo en un entorno similar al profesional**, aplicando herramientas y metodologías ágiles en el desarrollo de software.
- **Desarrollo de habilidades de resolución de problemas**, enfrentando retos técnicos y adaptándose a nuevas tecnologías.
- **Experiencia en integración de servicios**, incluyendo autenticación, pasarelas de pago y notificaciones en tiempo real.

## **Funcionalidades**

### **Registro de usuarios**

- **Registro y autenticación:** Usuarios pueden registrarse usando un correo electrónico y contraseña.
- **Perfiles de usuario:** Cada usuario tendrá un perfil con información básica: nombre, rol (estudiante, profesor, trabajador), foto opcional y datos de contacto.
  - En el caso del conductor, éste deberá añadir datos tales como su licencia de conducción, modelo y placas del vehículo.

### **Publicación de viajes**

- **Para conductores:**
  - Opción para registrar rutas con información como: origen, destino, horario aproximado, número de asientos disponibles y precio sugerido.
  - Posibilidad de editar o eliminar rutas publicadas.
- **Para pasajeros:**
  - Opción para buscar rutas disponibles según origen, destino y horario.
  - Solicitar unirse a un viaje compartido con un clic.

- Posibilidad de cancelar la solicitud de un viaje compartido antes de la confirmación del conductor.
- Opcionalmente, cancelar un viaje confirmado con condiciones establecidas, como un aviso previo.

### **Pasarela de pagos**

- Manejar desde la aplicación el monto a pagar y tener una confirmación del pago en ambos sentidos de la transacción priorizando el conductor.
- Manejar medios de pago electrónicos.

### **Comunicación entre usuarios**

- Sistema básico de mensajería dentro de la app para coordinar detalles del viaje.
- Sistema para pactar un precio.

### **Filtros de búsqueda**

- Filtrar rutas por:
  - Origen/destino (con opción de búsqueda aproximada).
  - Fecha y horario.

### **Mapa interactivo**

- Visualizar las rutas disponibles en un mapa, con marcadores para origen, destino y puntos de encuentro sugeridos.
- Seguimiento en tiempo real.

### **Sistema de reseñas y calificaciones**

- Los usuarios pueden calificar a conductores y pasajeros después de cada viaje, con comentarios opcionales.

### **Seguridad y confianza**

- Mostrar información básica del vehículo (modelo, placa) para los usuarios.
- Posibilidad de compartir el viaje en tiempo real con contactos extern

## **Notificaciones**

- Notificaciones en tiempo real respecto al estado del viaje.
- Notificaciones de mensajes.

## **Usabilidad y Personalización**

- Interfaz sencilla y fácil de usar, enfocada en la solicitud del viaje.
- Personalización básica del perfil (Foto y nombre).
- Modo oscuro.

## **Administración de viajes**

- Conductores y pasajeros pueden ver un historial de viajes (realizados o programados).
- Opción para cancelar un viaje con previo aviso.

## **Configuración y soporte**

- Configuración de perfil y preferencias del usuario.
- Acceso a un apartado de preguntas frecuentes y soporte técnico básico.



### PUNTO 3

Categoría	Funcionalidad	Prioridad (MoSCoW)	Complejidad
Registro de usuarios	Registro y autenticación Usuarios pueden registrarse usando un correo electrónico y contraseña.	Must Have	baja
Registro de usuarios	Perfiles de usuario Cada usuario tendrá un perfil con información básica: nombre, rol (estudiante, profesor, trabajador), foto opcional y datos de contacto. En el caso del conductor, éste deberá añadir datos tales como su licencia de conducción, modelo y placas del vehículo.	Must Have	baja
Publicación de viajes	Para conductores  Opción para registrar rutas con información como: origen, destino, horario aproximado, número de asientos disponibles y precio sugerido.  Posibilidad de editar o eliminar rutas publicadas.	Must Have	baja
Publicación de viajes	Para pasajeros  Opción para buscar rutas disponibles según origen, destino y horario.	Must Have	media

	Solicitar unirse a un viaje compartido con un clic.		
Pasarela de pagos	Manejar desde la aplicación el monto a pagar y tener una confirmación del pago en ambos sentidos de la transacción priorizando el conductor.	Must Have	media
Pasarela de pagos	Manejar medios de pago electrónicos.	Must Have	baja
Comunicación entre usuarios	Sistema básico de mensajería dentro de la app para coordinar detalles del viaje.	Should Have	media

Comunicación entre usuarios	Sistema para pactar un precio.	Must Have	2	media
Filtros por búsqueda	Filtrar rutas por <ul style="list-style-type: none"> <li>• Origen/Destino</li> <li>• Fecha y horario</li> </ul>	Should Have	1	baja
Mapa interactivo	Visualizar las rutas disponibles en un mapa, con marcadores para origen, destino y puntos de encuentro sugeridos.	Should Have	5	Alta
Mapa interactivo	Seguimiento en tiempo real	Could Have	3	Alta
Sistema de reseñas y calificaciones	Los usuarios pueden calificar a conductores y pasajeros después de cada viaje, con comentarios opcionales.	Could Have	2	Media



Seguridad y confianza	Mostrar información básica del vehículo (modelo, placa) para los usuarios.	Should Have	1	Baja
Seguridad y confianza	Posibilidad de compartir el viaje en tiempo real con contactos externos.	Could Have	3	Alta
Notificaciones	Notificaciones en tiempo real respecto al estado del viaje.	Could Have	2	Media
Notificaciones	Notificaciones de mensajes.	Should Have	2	Media
Usabilidad y personalización	Interfaz sencilla y fácil de usar, enfocada en la solicitud del viaje.	Should Have	5	Alta
Usabilidad y personalización	Personalización básica del perfil (Foto y nombre).	Should Have	2	Media
Usabilidad y personalización	Modo oscuro	Could Have	2	Media
Administración de viajes	Conductores y pasajeros pueden ver un historial de viajes (realizados o programados).	Should Have	2	Media
Administración de viajes	Opción para cancelar un viaje con previo aviso.	Must have	2	Media
Configuración y soporte	Configuración de perfil y preferencias del usuario.	Could have	2	Media
Configuración y soporte	Acceso a un apartado de preguntas frecuentes y soporte técnico básico.	Should Have	2	Media

### **Distribución de Trabajo**

- **Semana 1 (Días 1 - 7)**
  - Configuración del entorno de desarrollo y base de datos.
  - Desarrollo del módulo de registro de usuarios:
    - Registro y autenticación.
  - Publicación de viajes
    - Para conductores
    - Para pasajeros
- **Semana 2 (Días 8 - 14)**

- Diseño responsivo de la interfaz para móvil y computadora.
  - Pasarela de pagos.
    - Transacción en ambos sentidos
    - Manejar medios de pago electrónicos.
  - Comunicación entre usuarios
  - Sistema básico de mensajería
- **Semana 3 (Días 15 - 21)**
    - Comunicación entre usuarios
    - Sistema para pactar un precio.
    - Filtros por búsqueda
    - Filtrar por origen/destino, fecha y horario.
    - Mapa interactivo
    - Visualización de rutas disponibles.
- **Semana 4 (Días 22 - 30)**
    - Mapa interactivo
    - Seguimiento en tiempo real.
    - Sistema de reseñas y calificaciones.
      - Calificación de conductores y calificaciones
    - Seguridad y confianza
    - Mostrar información básica del vehículo.
    - Viaje compartido en tiempo real.
- **Semana 5 (Días 31 - 39)**
    - Notificaciones
    - Estado del viaje.
    - Mensajes.
    - Usabilidad y personalización.
    - Interfaz
- **Semana 6 (Días 40 - 47)**
    - Usabilidad y personalización.
    - Personalización del perfil
    - Modo oscuro
    - Administración de viajes
    - Historial de viajes
    - Opción de cancelar viaje
- **Semana 7 (Días 48 - 55)**
    - Configuración y soporte
    - Perfil y preferencias de usuario.
    - Apartados de preguntas
    - Integración de todos los módulos.
    - Pruebas de usuario para asegurar que el sistema sea intuitivo y funcional.





## PUNTO 4: Análisis gestión de software

### Tiempo

A continuación, para medir el tiempo vamos a utilizar una tabla la cual nos va a dar un tiempo estimado de desarrollo de cada funcionalidad ya mencionada utilizando un sistema el cual consiste en dar un tiempo estimado que denominaremos “n”, este tiempo n se refiere a cuánto supone el desarrollador (a) terminar una funcionalidad, a este tiempo se le va a agregar un tiempo que denominaremos “m”, donde m significa el tiempo en errores inesperados, este tiempo “m” va a ser calculado con la siguiente fórmula:  $m = n/2$ . Aunque sencilla esta fórmula nos permite asegurarnos que las funcionalidades estén listas en el tiempo previsto, teniendo en cuenta errores o situaciones inesperadas en el desarrollo por este tiempo tan holgado en principio.

Otra convención que vamos a utilizar es el tiempo de desarrollo en días, va a ser traducido a jornada laboral colombiana, es decir, si mencionamos 8 horas se va a representar con 1 día de trabajo. Esto con el fin de poder calcular el costo del desarrollo en base a las horas trabajadas y un promedio del salario por hora de un desarrollador, es decir, para facilitarnos el trabajo de los costos.

Y aunque es poco importante en principio, vamos a utilizar un contrato de prestación de servicios que va a presentar un pago por hora donde serán verificados a través de una cuenta de cobro. Al ser un contrato de prestación de servicios reducimos costos en los trabajadores, para que resulte atractiva la oferta el precio de la hora debe superar en al menos un 10% el salario mínimo si trabaja la jornada completa.

Con esto en cuenta, esta sería la tabla de tiempos propuestos para las tareas del proyecto:

Tarea	Tiempo n (h)	Tiempo m (h)	Tiempo total (h)	Tiempo total (días)
Configuración del entorno de desarrollo y base de datos.	4	2	6	0,75
Registro y autenticación.	12	6	18	2,25
Publicación de viajes (conductor)	6	3	9	1,125
Publicación de viajes (pasajero)	6	3	9	1,125
Diseño responsivo de la interfaz para móvil y computadora	48	24	72	9
Pasarela de pagos (transacción en ambos sentidos)	6	3	9	1,125
Pasarela de pagos (medios de pago electrónicos y físicos)	6	3	9	1,125

<b>Comunicación entre usuarios (mensajería)</b>	16	8	24	3
<b>Comunicación entre usuarios (pacto de precio)</b>	4	2	6	0,75
<b>Filtros de búsqueda (origen, destino, fecha y horario)</b>	12	6	18	2,25
<b>Mapa interactivo (visualización rutas disponibles)</b>	24	12	36	4,5
<b>Mapa interactivo (seguimiento en tiempo real)</b>	12	6	18	2,25
<b>Sistema de reseñas y calificaciones (conductores)</b>	8	4	12	1,5
<b>Sistema de reseñas y calificaciones (usuario)</b>	8	4	12	1,5
<b>Seguridad y confianza (información del vehículo)</b>	4	2	6	0,75
<b>Seguridad y confianza (viaje compartido en tiempo real)</b>	12	6	18	2,25
<b>Notificaciones (Estado del viaje)</b>	6	3	9	1,125
<b>Notificaciones (Mensajes)</b>	4	2	6	0,75
<b>Usabilidad y personalización (Interfaz)</b>	16	8	24	3
<b>Usabilidad y personalización (modo oscuro)</b>	8	4	12	1,5
<b>Administración de viajes (Historial de viajes)</b>	4	2	6	0,75
<b>Administración de viajes (Opción de cancelar viaje)</b>	6	3	9	1,125
<b>Configuración y soporte (Perfil y preferencias de usuario)</b>	8	4	12	1,5
<b>Configuración y soporte (Apartado de preguntas)</b>	4	2	6	0,75
<b>Configuración y soporte (Integración de todos los módulos)</b>	24	12	36	4,5

<b>Configuración y soporte (pruebas y testing)</b>	72	36	108	13,5
--	----	----	-----	------

Entregandonos los siguientes totales:

<b>Tiempo total:</b>	510 horas	63,75 días
----------------------	-----------	------------

## Costo

Para el calculo de desarrollo se tienen 3 propuestas que consideramos viables en un caso realista:

- 1. Contratar 1 desarrollador Semi senior full stack:** Esta opción resulta atractiva para la reducción de costes, pero en tiempo estimado puede resultar bastante largo, ya que depende de una sola persona todo el desarrollo, a pesar de su experiencia el tiempo puede tender a alargarse, ya que desde la estructuración hasta el testeo depende de él. La gran ventaja es que solamente una persona se encarga del desarrollo y entiende perfectamente lo que se debe hacer en cada parte del código, pero si el proyecto escala la dependencia a una sola persona se vuelve un problema. Se le pagarán 35.000 COP la hora.
- 2. Contratar 2 desarrolladores backend junior y 2 frontend junior:** Esta opción representa eficiencia en el tiempo, ya que las tareas y el tiempo de estas estarían divididas entre un equipo considerable, lo que puede acelerar el tiempo de desarrollo desde ambos sectores. El problema puede recaer en que las tareas más complejas se pueden alargar inesperadamente, pero si mantenemos una exigencia correcta, el tiempo de desarrollo se debería reducir por lo menos a la mitad. Se les pagaría 18.000 COP a los desarrolladores frontend la hora, y 20.000 COP a los backend.
- 3. Contratar 1 desarrollador semi senior full stack, 1 desarrollador junior frontend y 2 junior backend:** Esta opción resulta siendo la opción óptima, donde el equipo está siendo liderado por una persona con conocimientos técnicos precisos que puede ayudar a potenciar a los integrantes menos experimentados, además de que este líder puede entrar a apoyar y desarrollar las tareas más complejas tanto de backend como de frontend, mientras que los desarrolladores junior se pueden enfocar en aspectos más sencillos, lo que aceleraría enormemente el desarrollo del software y mejoraría su calidad. Además, el desarrollador full stack podría impulsar a los junior a que apoyen el área de front en caso de los backend o viceversa, para así agilizar aún más el tiempo de desarrollo.

Teniendo en cuenta estas propuestas, vamos a hacer tablas donde vamos a suponer el costo total de desarrollo (sin haber desplegado) y tiempo, donde la opción 1 vamos a suponer que el tiempo está multiplicado por 1, en el caso 2 por un factor de 0,7 y en el caso 3 por un factor de 0,4, esto por las razones expuestas en cada una de las propuestas.

Propuesta 1			
Tiempo de desarrollo	Precio hora	Precio día	Precio total
63,75 días	\$35.000	\$280.000	\$17.850.000

Ahora vamos a realizar una tarea similar con la propuesta 2 de costos de desarrollo.

Propuesta 2			
Tiempo desarrollo	Precio hora backend	Precio por desarrollador backend	Precio total backend
44,625 días	\$20.000	\$5.100.000	\$10.200.000
	Precio hora frontend	Precio por desarrollador frontend	Precio total frontend
	\$18.000	\$4.590.000	\$9.180.000
<b>Precio total proyecto</b>	\$19.380.000		





Y finalmente lo haremos con la propuesta 3 de costos de desarrollo:





Propuesta 3			
Tiempo de desarrollo	Precio hora fullstack	Precio por desarrollador full stack	Coste full stack
25,5 días	\$35.000	\$8.925.000	\$8.925.000
	Precio hora backend	Precio por desarrollador backend	Coste total backend
	\$20.000	\$3.366.000	\$6.732.000
	Precio hora frontend	Precio por desarrollador frontend	Coste total frontend
	\$18.000	\$3.029.400	\$3.029.400
<b>Precio total proyecto</b>	\$18.686.400		

Teniendo en cuenta estas 3 propuestas, sentimos que depende mucho más del factor tiempo que del factor dinero, ya que los precios son bastante similares, pero su diferencia en este caso hipotético es el tiempo total de desarrollo que les va a tomar a las distintas configuraciones de equipo lograr el trabajo. Claramente hay muchas suposiciones y casos ideales atrás de estos números, pero en un escenario idílico lo mas optimo para este desarrollo seria la opcion 3.

Ahora que tenemos el equipo de desarrollo, vamos a optar por el uso de firebase y google cloud en general como el servicio de despliegue para el backend, RTD, storage, hosting y testing. Elegimos firebase por gusto, aunque existen alternativas bastante famosas como lo es AWS, oracle cloud, Azure, Digital Ocean, y un largo etcétera, google cloud se nos hace el servicio más cómodo y completo para manejar, ademas de que hay experiencia en el uso de este. Entonces, para este apartado vamos a usar la cotización que nos entrega gcloud. Para esto, vamos a suponer 2000 usuarios usando la app, y suponer las métricas.



 Cloud Firestore	GiB almacenado	502 GiB acerca de 10,040 mensajes de chat M	\$90.18
	Escrituras de documentos	600,000 operaciones de escritura cantidad de veces que se escriben los datos	No cost
	Lecturas de documentos	15,200,000 operaciones de lectura cantidad de veces que se leen los datos	\$8.22
	Eliminaciones de documentos	600,000 operaciones de borrado cantidad de veces que se borran los datos	No cost
	Don't forget to factor in Egress costs! See <a href="#">Google Cloud pricing</a>		
 Realtime Database	GB almacenados	3 GB acerca de 60 mensajes de chat M	\$10
	GB transferidos	10 GB acerca de 200 mensajes de chat M	No cost
 Autenticación	Phone Auth - All regions	Facturación por SMS enviado Consulta <a href="#">las tarifas actuales</a>	-
	Identity Platform Pricing Usuarios activos por mes (sin incluir SAML/OIDC)	50,000 MAU Usuarios activos por mes	No cost
	Usuarios activos por mes: SAML/OIDC	900 MAU Usuarios activos por mes	\$12.75
 Cloud Storage	GB almacenados	45 GB acerca de 22,500 fotos en alta resolución	\$1.04
	GB transferidos	40 GB acerca de 20,000 fotos en alta resolución	\$1.20
	Operaciones (cargas y descargas)	2,100,000 operaciones acerca de 210,000 cargas y; 1,890,000 descargas	No cost

 Cloud Functions	Invocaciones	2,000,000 invocaciones cantidad de veces que se invoca una función	No cost
	GB-segundo	400,000 GB-segundo tiempo con 1 GB de memoria aprovisionada	No cost ?
	CPU-segundo	200,000 CPU-segundo tiempo con una CPU de 1 GHz aprovisionada	No cost ?
	Herramientas de redes (salida)	5 GB transferencia de datos saliente	No cost
	Minutos de Cloud Build	170 min minutos empleados para crear Cloud Functions	\$0.15
	Almacenamiento en contenedores	0 MB cantidad de almacenamiento para los contenedores de función	No cost
 Hosting	GB almacenados	10 GB acerca de 5,000 páginas de contenido estático	No cost
	GB transferidos	10 GB acerca de 5,000 páginas de contenido estático	No cost
 Test Lab	Pruebas de dispositivos virtuales	3 horas al día acerca de 36 pruebas	\$60
	Pruebas de dispositivos físicos	60 minutos al día acerca de 12 pruebas	\$75
 Firebase ML	Llamadas a la API de Cloud Vision	1,000 llamadas por mes	No cost
			Estimated monthly cost \$259

Con el flujo de usuarios esperados, nos daría un coste mensual de 259 USD, lo que en precio Colombianos es alrededor de 1 '125.000 COP, dependiendo de la fluctuación del dólar.

Entonces, esto nos lleva a los siguientes costos totales los primeros 3 meses de funcionamiento de la aplicación junto al mes de desarrollo y los 5 días iniciales que estaría al aire la aplicación:

Mes	Concepto	Precio	Total
1	Desarrollo	\$18.686.400	\$18.868.654
	Despliegue y funcionamiento	\$182.254	
2	Gcloud	\$1.093.526	\$1.093.526
3	Gcloud	\$1.093.526	\$1.093.526
4	Gcloud	\$1.093.526	\$1.093.526
Coste total 3 meses			\$22.149.234

## Alcance

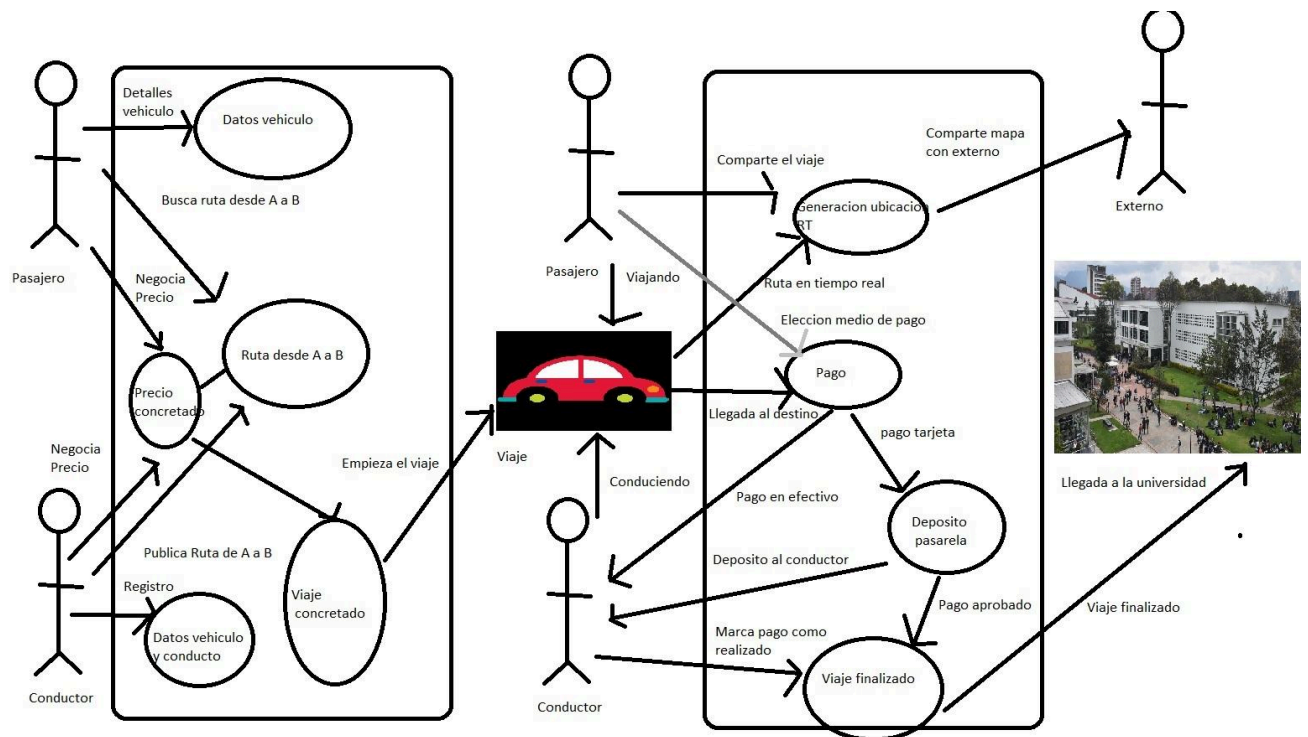
El alcance mínimo de una aplicación destinada a conectar a los usuarios que buscan transporte y los transportadores es que se puedan conectar ambas partes y concertar un precio, en este orden de ideas, el MVP debe poder permitir conectar a ambas partes, siendo una relación de uno a muchos la de conductor y pasajeros por cada viaje, con un máximo que determina el conductor por un precio. Claramente para esto debe existir la opción de registro e iniciar sesión, con una comunicación entre usuarios a través de la aplicación y la obtención de notificaciones del estado del viaje, con un mapa que al menos indique la ruta que se va a seguir y un seguimiento en tiempo real del viaje.





## PUNTO 5: Casos de uso.

Para los casos de uso vamos a utilizar el siguiente diagrama de casos de uso que nos orienta en la interacción mínima esperada entre todos los usuarios y el uso de la plataforma:



En este diagrama se evidencia la existencia de dos partes, la primera es la concertación del viaje y la segunda es la realización de este, ambas siendo igual de importantes pero mucho más compleja la segunda a la primera.

Como nota, en los casos de uso vamos a utilizar los comportamientos para el MPV, así que va a estar recortado a la interacción mínima de funcionamiento esperable.

### Caso de uso 1:

Nombre caso de uso: Registro de usuario	
<b>Actores involucrados</b> <ul style="list-style-type: none"> <li>Usuario(pasajero o conductor).</li> </ul>	<b>Descripción:</b> Proceso mediante el cual el conductor o el pasajero pueden crear una cuenta en la aplicación proporcionando la información básica necesaria para acceder a sus servicios.

<b>Flujo principal:</b> <ol style="list-style-type: none"> <li>1. El usuario accede a la pantalla de registro.</li> <li>2. Ingresa la información básica requerida (correo electrónico, nombre, rol, contraseña).</li> <li>3. Si es conductor, añade información adicional como licencia de conducción, modelo del vehículo y placas.</li> <li>4. El sistema valida los datos ingresados.</li> <li>5. Si los datos son correctos: <ul style="list-style-type: none"> <li>○ Se crea la cuenta.</li> <li>○ El usuario recibe una notificación de éxito.</li> </ul> </li> </ol>	
<b>Flujo alternativo:</b> <ul style="list-style-type: none"> <li>● En caso de que los datos sean incorrectos o incompletos: <ul style="list-style-type: none"> <li>○ El sistema muestra un mensaje indicando los campos que deben corregirse (ejemplo, contraseña débil, correo no válido).</li> <li>○ El usuario corrige los datos y vuelve a enviarlos.</li> <li>○ Si los datos siguen siendo incorrectos tras tres intentos, se bloquea temporalmente el registro por 15 minutos.</li> </ul> </li> </ul>	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>● El usuario debe poder abrir la pantalla de registro en la aplicación.</li> </ul>	<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>● El usuario queda registrado en el sistema.</li> <li>● Se genera un perfil de usuario.</li> </ul>

## Caso de uso 2:

Nombre caso de uso: Publicación de viajes	
<b>Actores involucrados</b> <ul style="list-style-type: none"> <li>● Conductor</li> </ul>	<b>Descripción:</b> El conductor puede publicar información sobre los viajes disponibles para que los pasajeros puedan unirse si así lo desean.

<b>Flujo principal:</b> <ol style="list-style-type: none"> <li>1. El conductor inicia sesión en la aplicación.</li> <li>2. Accede al módulo de publicación de viajes.</li> <li>3. Ingresa los datos del viaje: origen, destino, horario aproximado, número de asientos disponibles y precio sugerido.</li> <li>4. El sistema valida los datos ingresados.</li> <li>5. Si los datos son correctos: <ul style="list-style-type: none"> <li>○ El viaje queda publicado en la aplicación.</li> <li>○ Los pasajeros pueden visualizar la ruta en la sección correspondiente y solicitar unirse.</li> </ul> </li> <li>6. Si un pasajero desea cancelar su solicitud o el viaje: <ul style="list-style-type: none"> <li>○ Puede cancelar antes de la confirmación del conductor o bajo las condiciones establecidas para viajes confirmados.</li> <li>○ El sistema notifica al conductor sobre la cancelación.</li> </ul> </li> </ol>	
<b>Flujo alternativo:</b> <ul style="list-style-type: none"> <li>● En caso de que los datos del viaje sean inválidos: <ul style="list-style-type: none"> <li>○ El sistema muestra un mensaje de error especificando el problema (ejemplo, horario fuera del rango permitido, precio no válido).</li> <li>○ El conductor corrige los datos ingresados y vuelve a enviarlos.</li> <li>○ Si persiste el error tras tres intentos: <ul style="list-style-type: none"> <li>■ El conductor recibe una notificación para contactar soporte técnico.</li> </ul> </li> </ul> </li> </ul>	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>● El conductor debe estar registrado y autenticado en la aplicación.</li> <li>● El pasajero debe haber iniciado sesión y estar habilitado para realizar solicitudes o cancelaciones.</li> </ul>	<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>● La ruta queda publicada y disponible para los pasajeros.</li> <li>● Si un pasajero cancela, los asientos disponibles se actualizan en tiempo real.</li> <li>● El conductor es notificado en caso de cancelaciones por parte de los pasajeros.</li> </ul>

### Caso de uso 3:

Nombre caso de uso: Notificaciones	
<b>Actores involucrados</b> <ul style="list-style-type: none"> <li>● Usuario (pasajero o conductor).</li> </ul>	<b>Descripción:</b> El sistema envía notificaciones en tiempo real a los usuarios respecto al estado de los viajes y otros eventos importantes.



<b>Flujo principal:</b> <ol style="list-style-type: none"> <li>1. Un evento desencadena una notificación (ejemplo, un pasajero solicita unirse a un viaje, el conductor inicia el trayecto).</li> <li>2. El sistema genera la notificación con la información relevante.</li> <li>3. La notificación es enviada al usuario a través de la aplicación.</li> <li>4. El usuario recibe y visualiza la notificación en su dispositivo.</li> </ol>	
<b>Flujo alternativo:</b> <ul style="list-style-type: none"> <li>• En caso de que la notificación no sea recibida: <ul style="list-style-type: none"> <li>◦ El sistema intenta reenviar la notificación nuevamente.</li> </ul> </li> </ul>	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>• El usuario debe estar registrado y haber iniciado sesión.</li> </ul>	<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>• El usuario está informado sobre eventos relacionados con sus viajes.</li> </ul>

#### Caso de uso 4:

Nombre caso de uso: Comunicación entre usuarios	
<b>Actores involucrados</b> <ul style="list-style-type: none"> <li>• Pasajero.</li> <li>• Conductor.</li> </ul>	<b>Descripción:</b> Los pasajeros y conductores pueden comunicarse dentro de la aplicación para coordinar los detalles de los viajes.

<b>Flujo principal:</b> <ol style="list-style-type: none"> <li>1. El pasajero selecciona una ruta publicada por el conductor.</li> <li>2. Accede a la opción de chat o mensajería desde la aplicación.</li> <li>3. El pasajero o el conductor inician la conversación enviando un mensaje.</li> <li>4. El sistema notifica al destinatario que ha recibido un mensaje.</li> <li>5. Los mensajes son gestionados en tiempo real por el sistema.</li> <li>6. Se coordinan entre ambas partes los detalles necesarios para el viaje (ejemplo, punto de encuentro).</li> </ol>	
<b>Flujo alternativo:</b> <ul style="list-style-type: none"> <li>● <b>En caso de que el mensaje no sea enviado:</b> <ul style="list-style-type: none"> <li>○ El sistema muestra un error indicando el problema.</li> <li>○ Permite al usuario reenviar el mensaje.</li> <li>○ Si el problema persiste: <ul style="list-style-type: none"> <li>■ El usuario recibe una notificación indicando que el servicio está temporalmente inactivo.</li> </ul> </li> </ul> </li> <li>● En caso de que el mensaje no sea recibido por el destinatario: <ul style="list-style-type: none"> <li>○ El sistema genera un mensaje de estado indicando que el destinatario no está disponible.</li> </ul> </li> </ul>	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>● Ambos actores deben estar registrados y autenticados en la aplicación.</li> <li>● La ruta debe estar publicada y accesible para el pasajero.</li> </ul>	<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>● Los detalles del viaje quedan acordados entre el pasajero y el conductor.</li> <li>● El historial de la conversación queda registrado en el sistema para referencia futura.</li> </ul>

## Caso de uso 5:

<b>Nombre caso de uso:</b> Filtros de búsqueda	
<b>Actores involucrados:</b> <ul style="list-style-type: none"> <li>● Pasajero</li> </ul>	<b>Descripción:</b> El filtro de búsqueda le permitirá al pasajero poder encontrar un viaje según su ubicación de partida y ubicación de destino, buscando la ruta que se le acomode a sus necesidades.

<b>Flujo principal:</b> <ul style="list-style-type: none"> <li>• El usuario accede a la aplicación</li> <li>• En la pagina principal le da a la barra de búsqueda</li> <li>• Escribe su ubicación de partida y destino</li> <li>• El sistema busca viajes que le sean útiles al usuario             <ul style="list-style-type: none"> <li>○ En caso de no haber viajes le dirá al usuario que aún no hay ningún viaje programado</li> <li>○ En caso de haber algún viaje que le sea útil podrá interactuar con él desde la pantalla de programación de viajes.</li> </ul> </li> </ul>	
<b>Precondiciones:</b> <ol style="list-style-type: none"> <li>1. Estar registrado</li> <li>2. Acceder a la aplicación</li> </ol>	<b>Postcondiciones:</b> <ol style="list-style-type: none"> <li>1. Interacción con la persona que está ofreciendo el viaje, para poder pactar o no el viaje a través de la aplicación.</li> </ol>

## Caso de uso 6:

<b>Nombre caso de uso:</b> Pasarela de pagos	
<b>Actores involucrados:</b> <ul style="list-style-type: none"> <li>• Pasajero</li> <li>• Conductor</li> </ul>	<b>Descripción:</b> Momento clave donde el o los pasajeros deben hacer la transacción económica con el chofer asignado.
<b>Flujo principal:</b> <ul style="list-style-type: none"> <li>• Pasajeros ingresan su medio de pago en la aplicación.</li> <li>• Los conductores escriben una cuenta para depositar las transacciones.</li> <li>• Pasajeros y conductor pactan un precio a través de la app.</li> <li>• En caso de ser pago electrónico:             <ul style="list-style-type: none"> <li>○ Se hace la debitación automática a la hora de confirmar el precio.</li> <li>○ Se confirma el pago y el viaje.</li> </ul> </li> <li>• En caso de ser pago en efectivo:             <ul style="list-style-type: none"> <li>○ Se llega al destino</li> <li>○ La aplicación indica el precio a pagar</li> <li>○ Los pasajeros pagan el precio solicitado al conductor</li> </ul> </li> <li>• Termina el viaje.</li> </ul>	

<b>Flujo alternativo:</b> <ul style="list-style-type: none"> <li>● En caso de que el pago electrónico sea rechazado <ul style="list-style-type: none"> <li>○ Se intenta automáticamente por segunda vez. <ul style="list-style-type: none"> <li>■ Si es rechazado, muestra pantalla para ingresar otros medios de pago</li> <li>■ Si es aprobado pasa al flujo normal</li> </ul> </li> <li>○ Se elige otro medio de pago <ul style="list-style-type: none"> <li>■ Pasa al flujo normal.</li> </ul> </li> </ul> </li> <li>● En caso de no pagar en efectivo <ul style="list-style-type: none"> <li>○ Se genera un cobro a través de la tienda de aplicaciones (app store, play store) <ul style="list-style-type: none"> <li>■ Se banea la cuenta hasta que se realice el pago.</li> </ul> </li> <li>○ Se genera un saldo a favor del chofer en la aplicación, como si fuera una transacción electrónica.</li> <li>○ Se pasa a verificación manual el caso.</li> </ul> </li> </ul>	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>● Estar registrado</li> <li>● Tener un medio de pago asociado</li> </ul>	<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>● Tener el pago y cobro como aprobado en la pasarela de pagos/</li> </ul>

## Caso de uso 7:

<b>Nombre caso de uso:</b> Mapa interactivo	
<b>Actores involucrados:</b> <ul style="list-style-type: none"> <li>● Pasajeros</li> <li>● Conductores</li> <li>● Externos a la app</li> </ul>	<b>Descripción:</b> A través del mapa interactivo, se puede conocer la ruta a tomar, la ubicación en tiempo real del vehículo y el estado del viaje para cada uno de los pasajeros.
<b>Flujo principal:</b> <ul style="list-style-type: none"> <li>● Se concreta el viaje</li> <li>● Se inicia el viaje</li> <li>● Se notifica a los pasajeros que el viaje a iniciado y les dice donde encontrarse con el conductor</li> <li>● Los pasajeros se suben al vehículo y se confirma el inicio de su ruta</li> <li>● Se comparte la ubicación del viaje y se monta en un mapa de viaje.</li> <li>● Cuando termina el viaje se genera un mapa del trayecto.</li> </ul> <p><b>Nota:</b> Se puede compartir la ubicación del viaje en cualquier momento y la persona a la que se le compartió va a poder visualizarla desde la app, sin interferir en el flujo, se verá más adelante en seguridad y confianza.</p>	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>● Haber iniciado el viaje</li> </ul>	<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>● Culminar el viaje al llegar al destino.</li> </ul>

## Caso de uso 8:

<b>Nombre caso de uso:</b> Seguridad y confianza	
<b>Actores involucrados:</b> <ul style="list-style-type: none"> <li>● Conductor</li> </ul>	<b>Descripción:</b> El chofer deberá subir los datos tanto suyos como del vehículo, adjuntando papeles que le certifiquen como alguien apto para conducir como imágenes y datos del vehículo.
<b>Flujo principal:</b> <ul style="list-style-type: none"> <li>● Usuario se registra</li> <li>● Usuario registrado entra al apartado de conductores</li> <li>● Usuario registrado carga los archivos solicitados <ul style="list-style-type: none"> <li>○ Placas</li> <li>○ Modelo del vehículo</li> <li>○ Fotos del vehículo</li> <li>○ Foto por ambas caras de la licencia de conducción</li> </ul> </li> <li>● Los documentos y fotos son verificadas por el sistema</li> <li>● El usuario es aprobado.</li> </ul>	
<b>Flujo alternativo:</b> En caso de no subir todos los archivos solicitados: <ul style="list-style-type: none"> <li>● No se encuentran los archivos</li> <li>● El usuario es rechazado como conductor</li> </ul> En caso de que haya algún problema con los documentos del usuario: <ul style="list-style-type: none"> <li>● El usuario es rechazado como conductor</li> <li>● Se le notifica el inconveniente y se le abre la opción de resubir los archivos.</li> </ul>	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>● Estar registrado</li> </ul>	<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>● Ser aprobado como conductor</li> </ul>

## Caso de uso 9:

<b>Nombre caso de uso:</b> Usabilidad y personalización	
<b>Actores involucrados:</b> <ul style="list-style-type: none"> <li>● Todos los usuarios de la aplicación</li> </ul>	<b>Descripción:</b> Los usuarios pueden modificar sus perfiles y personalizarlos a placer, los únicos datos inmutables serán sus identificadores como lo es numero de telefono y correo

<b>Flujo principal:</b> <ul style="list-style-type: none"> <li>● El usuario entra a la parte de ajustes</li> <li>● El usuario da click en mi perfil</li> <li>● Si el usuario da click en su foto <ul style="list-style-type: none"> <li>○ Puede cargar una foto</li> <li>○ Puede tomarse una foto en ese momento</li> <li>○ Actualizó su foto de perfil <ul style="list-style-type: none"> <li>■ Puede cancelar la operación en cualquier momento.</li> </ul> </li> </ul> </li> <li>● Si el usuario da click en nombre de usuario <ul style="list-style-type: none"> <li>○ Puede cambiar su nombre de usuario por uno único.</li> </ul> </li> </ul>	
<b>Flujo alternativo:</b> Si el nombre de usuario ya está tomado: <ul style="list-style-type: none"> <li>● Alerta al usuario que ya existe un usuario con ese mismo username</li> <li>● Se cancela la transacción</li> </ul>	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>● Estar registrado</li> <li>● Tener la cuenta verificada</li> </ul>	<b>Postcondiciones:</b>

## Caso de uso 10:

<b>Nombre caso de uso: Administración de viajes</b>	
<b>Actores involucrados</b> <ul style="list-style-type: none"> <li>● Conductores</li> <li>● Pasajeros</li> </ul>	<b>Descripción:</b> Los usuarios pueden ver un historial de viajes (realizados o programados) y pueden también realizar la cancelación de un viaje programado de manera que la otra parte sea avisada.

<b>Flujo principal:</b> <ul style="list-style-type: none"> <li>● El usuario selecciona su perfil.</li> <li>● El usuario puede ver su historial con los viajes anteriores y programados: <ul style="list-style-type: none"> <li>○ Puede seleccionar un viaje programado e ir al perfil de la persona con quien se pactó el viaje y revisar su historial.</li> <li>○ Puede seleccionar un viaje propio programado y cancelarlo.</li> </ul> </li> <li>● En caso de que se cancele un viaje: <ul style="list-style-type: none"> <li>○ El otro usuario involucrado es notificado.</li> </ul> </li> </ul>	
<b>Flujo alternativo:</b> <ul style="list-style-type: none"> <li>● En caso de que no se tengan viajes programados : <ul style="list-style-type: none"> <li>○ No es posible realizar una cancelación, solo revisar el historial de viajes hechos y los perfiles de las personas con quienes se pactaron.</li> </ul> </li> </ul>	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>● Haber realizado al menos un viajes para poderlo visualizar</li> <li>● Tener al menos un viaje programado para poder cancelarlo.</li> </ul>	<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>● La cancelación se debe confirmar y se le debe avisar al otro usuario involucrado.</li> </ul>

## Caso de uso 11:

Nombre caso de uso: Configuración y soporte	
<b>Actores involucrados</b> <ul style="list-style-type: none"> <li>● Conductores</li> <li>● Pasajeros</li> </ul>	<b>Descripción:</b> Se puede configurar el perfil y actualizarlo después de haberlo creado en un inicio. Además, se ofrece un apartado de preguntas frecuentes y soporte técnico básico.

<b>Flujo principal:</b> <ul style="list-style-type: none"> <li>● Una vez registrado el usuario, en la página principal se entra a ajustes</li> <li>● Si el usuario da click en mi perfil <ul style="list-style-type: none"> <li>○ Puede actualizar la foto de perfil</li> <li>○ Puede actualizar el nombre de usuario</li> <li>○ Puede cambiar las preferencias del usuario. <ul style="list-style-type: none"> <li>■ Puede cancelar la operación en cualquier momento.</li> </ul> </li> </ul> </li> <li>● Si el usuario da click en preguntas frecuentes <ul style="list-style-type: none"> <li>○ Puede revisar una lista de preguntas y respuestas que surgen frecuentemente dentro de la comunidad.</li> </ul> </li> <li>● Si el usuario da click en soporte técnico básico <ul style="list-style-type: none"> <li>○ Se le brinda asistencia intelectual, tecnológica y material para prevenir y resolver problemas, además de optimizar y mejorar el rendimiento de los servicios.</li> </ul> </li> </ul>	
<b>Precondiciones:</b> <ul style="list-style-type: none"> <li>● Estar registrado en la plataforma</li> </ul>	<b>Postcondiciones:</b> <ul style="list-style-type: none"> <li>● Los cambios generados deben guardarse.</li> </ul>



## PUNTO 6: Historias de usuario

### Historia de usuario #1

#### Anexo de Documentos Relacionados:

- [REST API URL - Best Practices and Examples](#)

#### Descripción conceptual

<b>Módulo</b>	Gestión de Usuarios
<b>Descripción de la(s) funcionalidad(es) requerida(s):</b>	Permitir a los pasajeros y conductores registrarse en la aplicación proporcionando información básica. Los conductores deben ingresar datos adicionales relacionados con su licencia de conducción y vehículo.

#### Backend

<b>URL</b> https://localhost:8080/api/users/register	<b>Método</b> POST	<b>Código html</b> <ul style="list-style-type: none"><li>• <b>201</b> (Creado)</li><li>• <b>400</b> (Solicitud Incorrecta)</li></ul>
<b>Caso de uso tecnico</b> Este endpoint permite a los usuarios registrarse en la aplicación. Si los datos son válidos, retorna 201 con el ID del usuario creado. Si hay errores, retorna 400 con un mensaje indicando los campos incorrectos.		
<b>Datos de entrada</b> <pre>{   "email":     "usuario@example.com",   "name": "Juan Pérez",   "role": "pasajero",   "password":     "ContraseñaSegura123",   "driver_info": {     "license_number":       "ABC12345",     "vehicle_model": "Toyota       Corolla",     "license_plate": "XYZ789"   } }</pre>	<b>Datos de salida</b> <b>201:</b> <pre>{   "status": "success",   "data": {     "user_id": 1,     "name": "Juan Pérez",     "role": "pasajero"   } }</pre> <b>404:</b> <pre>{   "status": "error",   "message": "El correo electrónico no es válido" }</pre>	

## Frontend

El formulario de registro debe permitir al usuario ingresar su información básica (correo, nombre, contraseña, rol) de manera clara y estructurada. Si el usuario elige el rol de conductor, se habilitarán campos adicionales para ingresar los datos de su licencia de conducción, modelo de vehículo y placa.

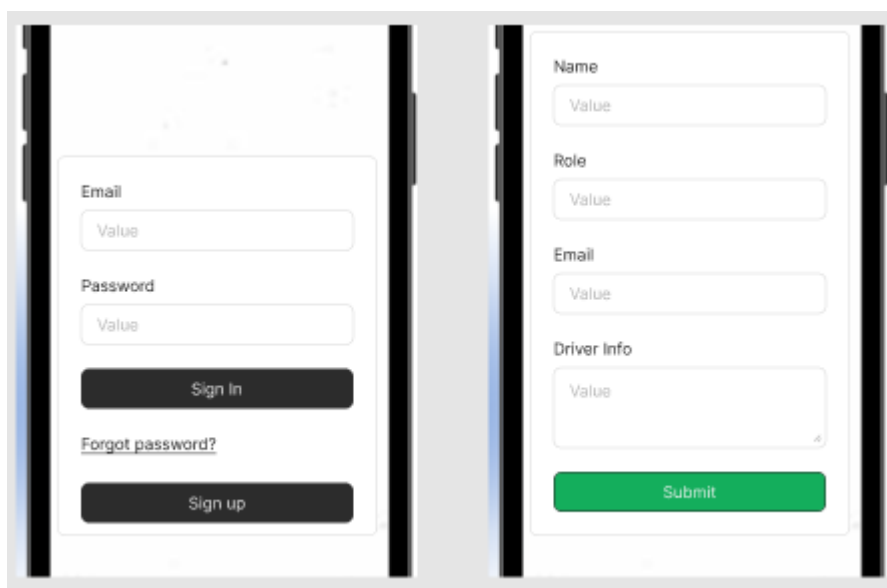
El botón "Registrarse" debe permanecer deshabilitado hasta que todos los campos obligatorios sean completados correctamente. La validación debe realizarse en tiempo real, resaltando en rojo los campos con errores y mostrando mensajes explicativos (ejemplo: "La contraseña debe tener al menos 8 caracteres").

Si el registro es exitoso, se mostrará un mensaje de confirmación en verde y el usuario será redirigido a la pantalla de inicio de sesión. En caso de error, los campos incorrectos se marcarán en rojo con mensajes detallados.

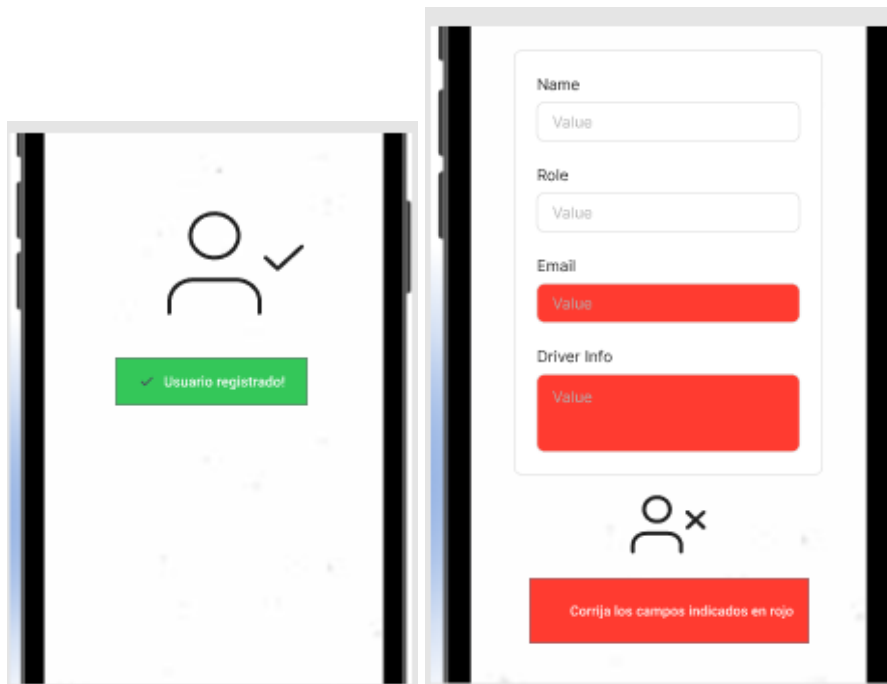
### *Interacción esperada:*

- *Formulario dinámico con campos para ingresar los datos de registro.*
- *Validación en tiempo real con resaltado de errores en los campos incorrectos.*
- *Botón "Registrarse" habilitado solo cuando todos los campos sean válidos.*
- *Mensajes de éxito o error según el resultado del proceso.*

### *Mockups/Prototipos:*



The image displays two side-by-side mobile application mockups. The left mockup represents a login screen with a light gray background. It features a white rounded rectangle containing an 'Email' field with a placeholder 'Value', a 'Password' field with a placeholder 'Value', a dark gray 'Sign in' button, a blue link for 'Forgot password?', and a dark gray 'Sign up' button. The right mockup represents a registration screen with a similar light gray background. It features a white rounded rectangle containing a 'Name' field with a placeholder 'Value', a 'Role' field with a placeholder 'Value', an 'Email' field with a placeholder 'Value', and a 'Driver Info' section with a placeholder 'Value'. A green 'Submit' button is positioned at the bottom of the form area.



*Flujo visual y eventos:*

1. *El usuario abre la pantalla de registro.*
2. *Ingresa su información básica (correo, nombre, contraseña, rol).*
3. *Si selecciona el rol de conductor, aparecen los campos adicionales.*
4. *El usuario presiona "Registrarse".*
5. *Si los datos son correctos:*
  - *Se muestra un mensaje de éxito y el usuario es redirigido a la pantalla de inicio de sesión.*
6. *Si los datos son incorrectos:*
  - *Se resaltan los campos erróneos y se muestra un mensaje indicando qué corregir.*

## Historia de usuario #2

### Anexo de Documentos Relacionados:

- [REST API URL - Best Practices and Examples](#)

### Descripción conceptual

<b>Módulo</b>	<i>Gestión de Viajes.</i>
<b>Descripción de la(s) funcionalidad(es) requerida(s):</b>	<i>Permitir a los conductores publicar viajes con detalles como origen, destino, horario, precio y número de asientos.</i>

### Backend

URL	Método	Código html
https://localhost:8080/api/trips	POST	201 (Creado), 400 (Solicitud Incorrecta)
<b>Caso de uso tecnico</b> Este endpoint permite a los conductores publicar viajes. Si los datos son válidos, retorna 201 con el ID del viaje creado. Si hay errores, retorna 404 con un cuerpo vacío.		
<b>Datos de entrada</b> { "origin": "Universidad Nacional", "destination": "Calle 45", "schedule": "2025-02-01T08:00:00", "seats": 3, "price": 10000 }	<b>Datos de salida</b>  <b>201:</b> { "status": "success", "data": { "trip_id": 1, "origin": "Universidad Nacional", "destination": "Calle 45", "seats": 3 } }  <b>404:</b> { "status": "error", "message": "Missing origin or destination" }	

### Frontend

El formulario para publicar un viaje debe permitir al conductor ingresar los detalles necesarios de manera clara y organizada. Debe incluir campos para especificar el origen y destino del viaje, un selector de fecha y hora para el horario, un campo numérico para el precio, y un control deslizante o selector para el número de asientos, limitado entre 1 y 5. Además, el formulario debe mostrar validaciones dinámicas en tiempo real, resaltando en rojo cualquier campo con errores o vacío.

El botón "Publicar" debe permanecer deshabilitado hasta que todos los campos estén correctamente llenados, cambiando su estado a viaje publicado con éxito cuando la información sea válida. De lo contrario se mostraran en rojo los campos que faltan por completar

#### *Interacción esperada:*

- *Formulario para que el conductor ingrese los detalles del viaje.*
- *Botón "Publicar" para enviar los datos al servidor.*
- *Visualización del mensaje de estado para indicar el estado final de la publicación.*

#### *Mockups/Prototipos:*

The mockup shows a mobile app interface for publishing a new trip. The screen has a white background with a black border. At the top, the title "PUBLICAR UN NUEVO VIAJE" is displayed in bold black text. Below the title is a table with five rows and two columns. The first column contains labels: "ORIGEN", "DESTINO", "HORARIO", "PRECIO", and "#ASIENTOS". The second column contains empty input fields. Below the table is a large green button with the word "PUBLICAR" in white capital letters. The entire form is centered on the screen.

ORIGEN	
DESTINO	
HORARIO	
PRECIO	
#ASIENTOS	

PUBLICAR

## PUBLICAR UN NUEVO VIAJE

ORIGEN	UNAL
DESTINO	CALLE 45
HORARIO	28/1/2025 - 6:00 PM
PRECIO	10.000
#ASIENTOS	3

VIAJE PUBLICADO CON  
EXITO

**PUBLICAR UN NUEVO VIAJE**

ORIGEN	UNAL
DESTINO	
HORARIO	28/1/2025 - 6:00 PM
PRECIO	10.000
#ASIENTOS	

**CAMPOS "DESTINO" Y  
"# ASIENTOS"  
INCOMPLETOS**

*Flujo visual y eventos:*

- 1. El conductor llena los campos del formulario y presiona "Publicar".*
- 2. Si la publicación es exitosa, aparece un mensaje de éxito.*
- 3. Si hay errores, se resaltan los campos incorrectos.*

## Historia de usuario #3

### Anexo de Documentos Relacionados:

- [\*REST API URL - Best Practices and Examples\*](#)

### Descripción conceptual

<b>Módulo</b>	<i>Gestión de Notificaciones.</i>
<b>Descripción de la(s) funcionalidad(es) requerida(s):</b>	<i>Enviar notificaciones en tiempo real a los usuarios sobre eventos relacionados con los viajes (ejemplo, solicitud aceptada, cancelación, mensajes recibidos).</i>

### Backend

<b>URL</b> https://localhost:8080/api/notifications	<b>Método</b> POST	<b>Código html</b> 200 (OK), 500 (Error del servidor)
<b>Caso de uso tecnico</b> Este endpoint envía notificaciones en tiempo real a los usuarios. Si el proceso es exitoso, retorna 200 con el mensaje enviado. Si ocurre un error, retorna 500 con un cuerpo vacío.		
<b>Datos de entrada</b> { "user_id": 1, "message": "Tu solicitud de viaje ha sido aceptada." }	<b>Datos de salida</b>  <b>201:</b> { "status": "success", "message": "Notificación enviada." }  <b>500:</b> { "status": "error", "message": "Failed to send notification" }	

### Frontend

*El apartado de notificaciones debe mostrar un listado de eventos recientes relacionados con los viajes del usuario, organizados cronológicamente y con la opción de expandir cada notificación para ver detalles y acceder al módulo correspondiente.*

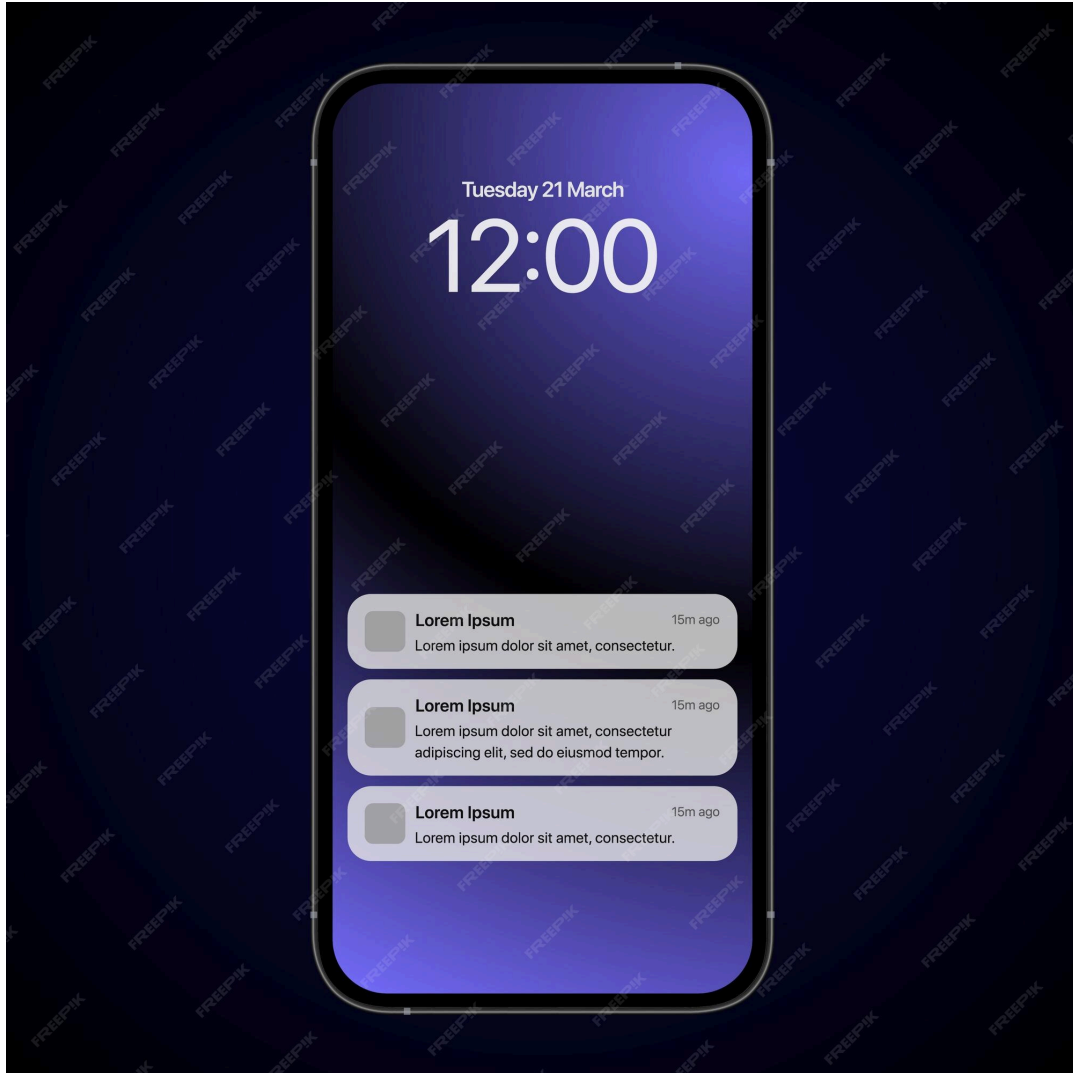
### Interacción esperada:

- *Notificaciones emergentes (pop-ups o banners) aparecen automáticamente en la interfaz del usuario cuando se generan eventos en el sistema.*



- *El usuario puede interactuar con la notificación para redirigirse al módulo correspondiente (ejemplo, detalles del viaje).*

*Mockups/Prototipos:*



*Flujo visual y eventos:*

1. *El sistema genera una notificación cuando ocurre un evento (ejemplo., aceptación de solicitud).*
2. *La notificación se muestra al usuario como un banner emergente en tiempo real.*
3. *El usuario puede hacer clic en la notificación para navegar al módulo correspondiente.*

## Historia de usuario #4

### Anexo de Documentos Relacionados:

- [REST API URL - Best Practices and Examples](#)

### Descripción conceptual

<b>Módulo</b>	<i>Mensajes</i>
<b>Descripción de la(s) funcionalidad(es) requerida(s):</b>	<i>Facilitar la comunicación en tiempo real entre pasajeros y conductores mediante mensajes dentro de la aplicación.</i>

### Backend

URL	Método	Código html
https://localhost:8080/api/messages	POST	200 (OK), 500 (Error del servidor)
<b>Caso de uso tecnico</b> Este endpoint permite enviar mensajes en tiempo real entre usuarios. Si el mensaje se envía correctamente, retorna 200 con los detalles del mensaje. Si ocurre un error, retorna 500 con un cuerpo vacío.		
<b>Datos de entrada</b> { "sender_id": 1, "receiver_id": 2, "message": "¿Dónde nos encontramos?" }	<b>Datos de salida</b>  <b>200:</b> { "status": "success", "data": { "message_id": 1, "message": "¿Dónde nos encontramos?", "timestamp": "2025-01-27T10:15:00Z" } }  <b>500:</b> { "status": "error", "message": "Failed to send message" }	

### Frontend

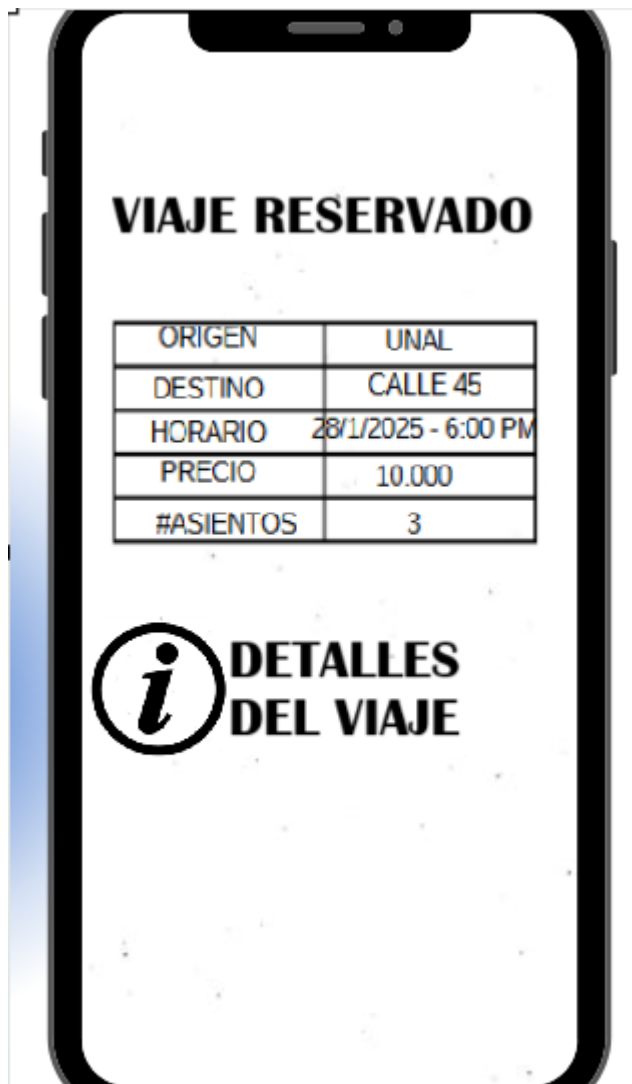
El módulo de comunicación entre usuarios debe incluir una pantalla de chat en tiempo real donde los pasajeros y conductores puedan intercambiar mensajes relacionados con el viaje. La interfaz debe listar las conversaciones activas y, al seleccionar una, expandirse para mostrar el historial de mensajes en formato de burbujas. Los mensajes enviados se mostrarán con la fecha y hora correspondiente, diferenciando visualmente los mensajes del usuario y del interlocutor. Si un

mensaje no se envía correctamente, se debe mostrar una alerta con la opción de reintento. El diseño debe ser responsivo y estar integrado con la estética general de la aplicación.

*Interacción esperada:*

- Chat en tiempo real con un área de texto para redactar mensajes y un botón "Enviar".
- Los mensajes enviados aparecen en forma de burbujas en el historial del chat.

*Mockups/Prototipos:*





*Flujo visual y eventos:*

1. El usuario abre el módulo de chat desde la pantalla de detalles del viaje.
2. Redacta un mensaje en el área de texto y presiona "Enviar".
3. Si el mensaje se envía correctamente, aparece en el historial del chat con la fecha y hora.
4. En caso de error, se muestra un mensaje de advertencia como "No se pudo enviar el mensaje".

## Historia de Usuario #5

### Descripción conceptual

<b>Módulo</b>	<i>Filtros de búsqueda.</i>
<b>Descripción de la(s) funcionalidad(es) requerida(s):</b>	<i>Los usuarios podrán buscar según el punto de partida y destino rutas que se acoplen a sus necesidades.</i>

### Backend

<b>URL</b> <b>https://localhost:8080/route?initialPoint=PointA&amp;endPoint=PointB</b>	<b>Método</b> <b>GET</b>	<b>Código http</b> <b>200, 404</b>
<b>Caso de uso técnico</b> <i>Este método ingresa 2 coordenadas, una de salida y otra de llegada, para retornar viajes los cuales se adapten a la solicitud.</i>		
<b>Datos de entrada</b>	<b>Datos de salida</b>	

- **PointA:** 4.7325° N, 74.2642° W
- **PointB:** 4.7329° N, 74.2602° W

**200:**

JavaScript

```
{
  "status": "Succesful",
  "statusCode": 200,
  "data": [
    {
      "driver": "Jhon Doe",
      "pointA": "4.7324° N,
74.2641° W",
      "PointB": "4.7329° N,
74.2602° W"
    }
  ]
}
```

**400:**

JavaScript

```
{
  "status": "Succesful",
  "statusCode": 200,
  "data": [
    {
      "driver": "Jhon Doe",
      "pointA": "4.7324° N,
74.2641° W",
      "PointB": "4.7329° N,
74.2602° W"
    }
  ]
}
```

## Frontend

*El usuario escribirá en la barra de búsqueda su destino y punto de partida, si no está con la ubicación actual por defecto, y le saldrán rutas las cuales se ajustan a su trayecto.*

*Interacción esperada:*

*El usuario escribe su punto de partida y punto de destino, después elige el viaje que mas le convenga y/o llame la atención,*

*Flujo visual y eventos:*

1. En el momento de búsqueda va a haber un focus en la escritura de los detalles del viaje, tanto el punto de partida como de salida.
2. Tras la búsqueda va a haber un espacio de carga.
3. Se deben listar los trayectos disponibles.
4. Al darle click a alguno pasa al módulo de mapa interactivo como pop up.

*Mockups/Prototipos:*

The image displays two mobile application mockups side-by-side. The left mockup represents a search interface, featuring a header bar, two input fields labeled 'Salida' and 'Destino', and a 5x4 grid of buttons at the bottom. The right mockup represents a results interface, showing three travel cards. Each card includes a placeholder icon, a title ('Viaje 1', 'Viaje 2', 'Viaje 3'), and a label 'Datos viaje'.

## Historia de usuario #2

### Descripción conceptual

<b>Módulo</b>	<i>Módulo de Gestión de Usuarios</i>
<b>Descripción de la(s) funcionalidad(es) requerida(s):</b>	<i>Los usuarios podrán registrarse en la aplicación como usuarios o conductores.</i>

### Backend

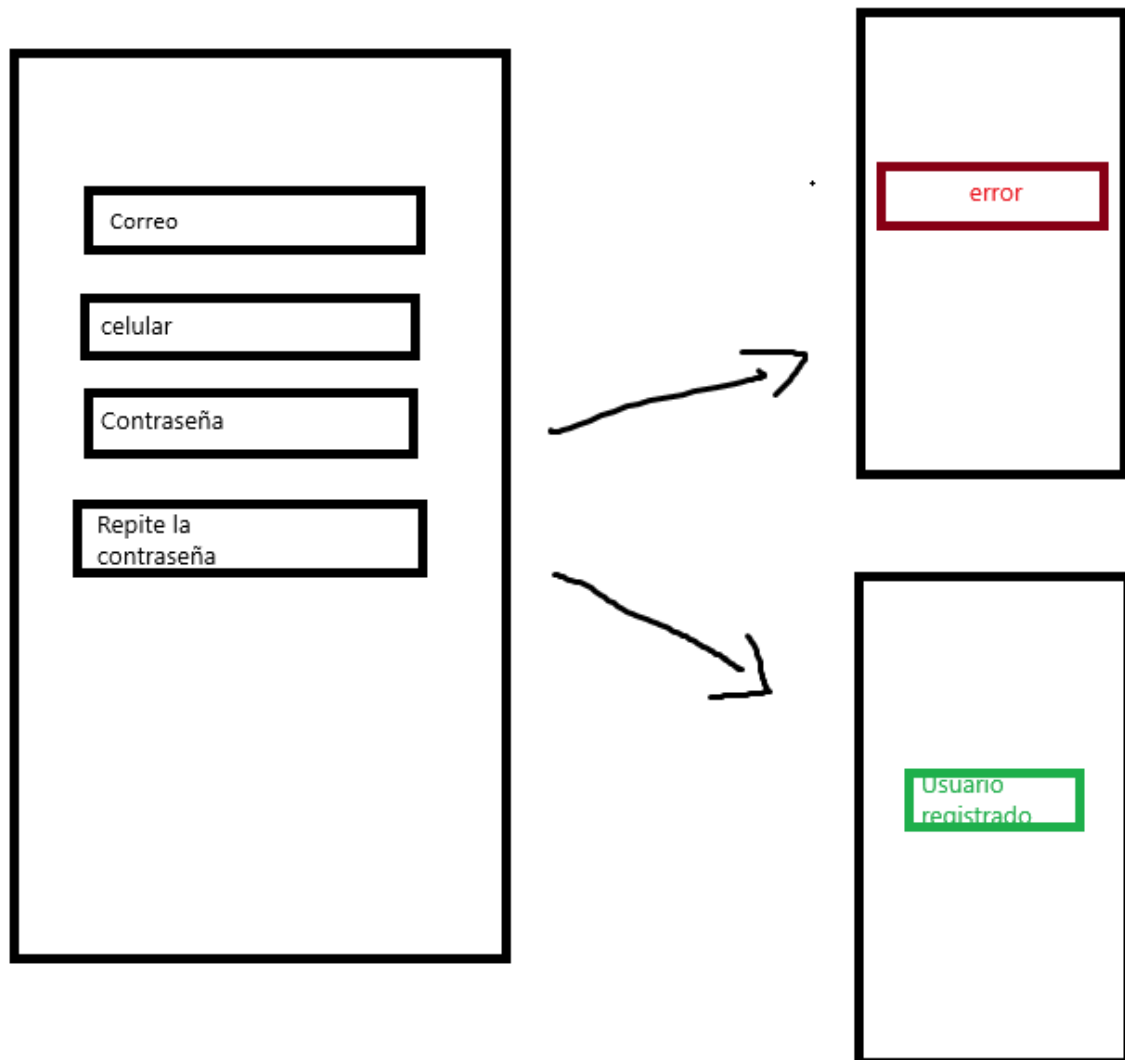
*El usuario envía sus datos de registro al servidor para posteriormente ser almacenado en la base de datos.*

URL	Método	Código http
<a href="#"><u>&lt;url a consultar&gt;</u></a>	POST	201, 400
<b>Caso de uso técnico</b>  <i>Este método se encarga de registrar al usuario. Si la operación es exitosa regresa el código 201, si no fue exitoso regresa el código 400</i>		
<b>Datos de entrada</b>  { "nombre": "Ana López", "email": "ana@unal.edu.co", "password": "*****" }	<b>Datos de salida</b>  201: { "mensaje": "Usuario registrado" }  400: { "error": "Corrige los errores y vuelve a intentarlo" }	

### Frontend

**El usuario escribe en los campos correspondientes su correo, nombre, celular y contraseña (la cuál deberá escribir 2 veces en otro campo que se llame “repetir contraseña”) para posteriormente darle**

click a un botón llamado “Registrarse”. Si hay un error, saldrá un mensaje en rojo abajo que diga el error que sucedió, en caso contrario saldrá un mensaje en verde diciendo Usuario registrado. Finalmente se redirecciona a la página de inicio con el usuario ya logueado.



5.



## Historia de usuario #6

### Descripción conceptual

<b>Módulo</b>	<i>Módulo de Pagos</i>
<b>Descripción de la(s) funcionalidad(es) requerida(s):</b>	<i>Realizar pagos mediante tarjetas de crédito o débito de forma virtual.</i>

### Backend

URL	Método	Código http
<a href="#"><u>&lt;url a consultar&gt;</u></a>	POST	200, 402, 422

### Caso de uso técnico

*Este método se encarga de procesar un pago y retorna un comprobante en caso de que haya sido procesado correctamente. El código 402 es para los casos cuando la tarjeta es declinada por insuficiencia de fondos y el 422 es para cuando hubo un error en la información de la tarjeta.*

Datos de entrada	Datos de salida
{ "monto": 20000, "metodo": "tarjeta", "detalles": { "numero": "4242...", "cvv": "123" } }	200: { "comprobante": "TX-7890", "estado": "aprobado" }  402: { "error": "Tarjeta declinada" }  422: { "error": "CVV inválido" }

### Frontend

- **interacción esperada:**
  - El usuario selecciona "Reservar asiento" en una ruta.
  - Ingresa al formulario de pago y elige método (tarjeta o billetera digital).
  - Confirma los detalles y completa la transacción.
- **Flujo visual:**
  - Formulario con campos de tarjeta validados en tiempo real (ej: máscara para número de tarjeta).
  - Spinner de carga durante el procesamiento.
  - Mensaje de éxito con comprobante descargable o error con sugerencias (ej: "Verifique su saldo").

### Historia de usuario #7

#### Descripción conceptual

Módulo	Módulo de Seguimiento y Navegación
Descripción de la(s) funcionalidad(es) requerida(s):	Visualizar en tiempo real, la ubicación del vehículo para asegurar una ruta conocida y estimar el tiempo de llegada.

### Backend

URL	Método	Código http
<a href="#">&lt;url a consultar&gt;</a>	GET	200, 404

Caso de uso técnico	
<i>Este método retorna el punto de inicio y final de la ruta los cuales mediante el uso de un WebSocket se genera una conexión persistente que enviará la ubicación del conductor cada 5 segundos.</i>	
Datos de entrada	Datos de salida
<b>WebSocket:</b> { "id_viaje": "789", "token": "abc123" }	200: { "origen": "Calle 80", "destino": "UNAL", "waypoints": ["Calle 72", "Avenida 30"] }  404: { "error": "Viaje no encontrado" }  Flujo continuo (WebSocket): { "lat": 4.7110, "lng": -74.0721, "timestamp": "2023-10-25T14:30:00Z" }

### Frontend

- **Interacción esperada:**
  - Al iniciar un viaje, el conductor activa el compartir ubicación → Establece conexión WebSocket.
  - Los pasajeros ven el mapa con:
    - **Marcador dinámico** del vehículo (actualizado cada 5 segundos).
    - **Ruta planificada** (cargada vía HTTP GET /api/viajes/{id}/ruta).
    - **Tiempo estimado de llegada** (calculado con API de tráfico como Google Maps).
- **Flujo visual:**
  - Animación suave del marcador del vehículo en el mapa.
  - Notificación de "Conexión perdida" si el WebSocket se interrumpe por más de 30 segundos.

### Historia de usuario #8

#### Descripción conceptual

Módulo	Módulo de Seguridad y Reputación
Descripción de la(s) funcionalidad(es) requerida(s):	seguridad al interactuar con otros usuarios (conductores o pasajeros), sabiendo que la app valida identidades, protege los datos y ofrece herramientas para reportar comportamientos inapropiados.

### Backend

URL	Método	Código http
<a href="#">&lt;url a consultar&gt;</a>	POST, GET	201, 403, 200,
<p align="center"><b>Caso de uso técnico</b></p> <p><i>Este método autentifica los documentos presentados por el conductor. También brinda información acerca del conductor (su calificación y comentarios). Y también permite reportar a un conductor.</i></p>		
<p align="center"><b>Datos de entrada</b></p> <pre>{ "id_usuario_reportado": "123", "motivo": "incumplió ruta" }</pre>	<p align="center"><b>Datos de salida</b></p> <pre>200: { "id_usuario_reportado": "123", "motivo": "incumplió ruta" }</pre>	
	<pre>200: { "puntaje": 4.8, "comentarios": ["Puntual", "Amable"] }</pre>	
<pre>{ "tipo": "documento", "archivo": "base64..." }</pre>	<pre>{ "status": "verificado", "nivel": "alto" }</pre>	

## Frontend

- **Interacción esperada:**
  - **Verificación de identidad:**
    - El usuario sube su documento
    - Un sello de "Verificado" aparece en su perfil.
  - **Perfil transparente:**
    - Al seleccionar un conductor/pasajero, se muestra su reputación, viajes completados y comentarios.
  - **Reportes y bloqueos:**
    - Opción para reportar un usuario durante o después del viaje.

- *Confirmación: "Tu reporte será revisado en 24 horas".*
- **Flujo visual:**
  - **Chat seguro:** *Mensajes encriptados y opción para no compartir número telefónico.*

**Badge de verificación:** *Icono de escudo junto al nombre de usuarios verificados.*

## Historia de Usuario #10

### Descripción conceptual

<b>Módulo</b>	<i>Administración de viajes.</i>
<b>Descripción de la(s) funcionalidad(es) requerida(s):</b>	<i>Los usuarios podrán consultar el historial de viajes realizados o programados, así como cancelar viajes programados, notificando al otro usuario involucrado</i>

### Backend

<b>URL</b> <b>https://localhost:8080/trips</b>	<b>Métodos</b> <b>GET</b> <b>DELETE</b>	<b>Código http</b> <b>GET: 200, 404</b> <b>DELETE: 200, 404, 403</b>
<b>Caso de uso técnico</b> <b>GET: consultar historial de viajes</b> <i>Este método permite a los usuarios obtener un historial de sus viajes realizados y programados.</i>		

<p><b>Datos de entrada</b></p> <ul style="list-style-type: none"> <li>• <code>userId</code> (ID del usuario que realiza la solicitud).</li> </ul>	<p><b>Datos de salida</b></p> <p><b>200:</b></p> <pre>{   "status": "Successful",   "statusCode": 200,   "data": [     {       "tripId": "1234",       "role": "Driver",       "status": "Completed",       "pointA": "4.7325° N, 74.2642° W",       "pointB": "4.7329° N, 74.2602° W",       "date": "2025-01-25",       "partner": "Jane Doe"     },     {       "tripId": "5678",       "role": "Passenger",       "status": "Scheduled",       "pointA": "4.7325° N, 74.2642° W",       "pointB": "4.7329° N, 74.2602° W",       "date": "2025-01-29",       "partner": "John Smith"     }   ] }</pre> <p><b>400:</b></p> <pre>{   "status": "Not Found",   "statusCode": 404,   "message": "No trips found for the user." }</pre>
<p align="center"><b>Caso de uso técnico</b>  <b>DELETE: Cancelar viaje programado</b>  <i>Este método permite a los usuarios cancelar un viaje programado, notificando al otro usuario involucrado.</i></p>	
<p><b>Datos de entrada</b></p> <ul style="list-style-type: none"> <li>• <code>tripId</code> (ID del viaje que se desea cancelar).</li> <li>• <code>userId</code> (ID del usuario que realiza la solicitud).</li> </ul>	<p><b>Datos de salida</b></p> <p><b>200:</b></p> <pre>{   "status": "Successful",   "statusCode": 200,   "message": "The trip has been successfully canceled.",   "notifiedUser": "John Smith" }</pre> <p><b>404:</b></p> <pre>{   "status": "Not Found",   "statusCode": 404,   "message": "The trip could not be found or does not exist." }</pre>

	<b>403:</b> <pre>{   "status": "Forbidden",   "statusCode": 403,   "message": "The user does not have permission to cancel this trip." }</pre>
--	---

## Frontend

### Interacción esperada:

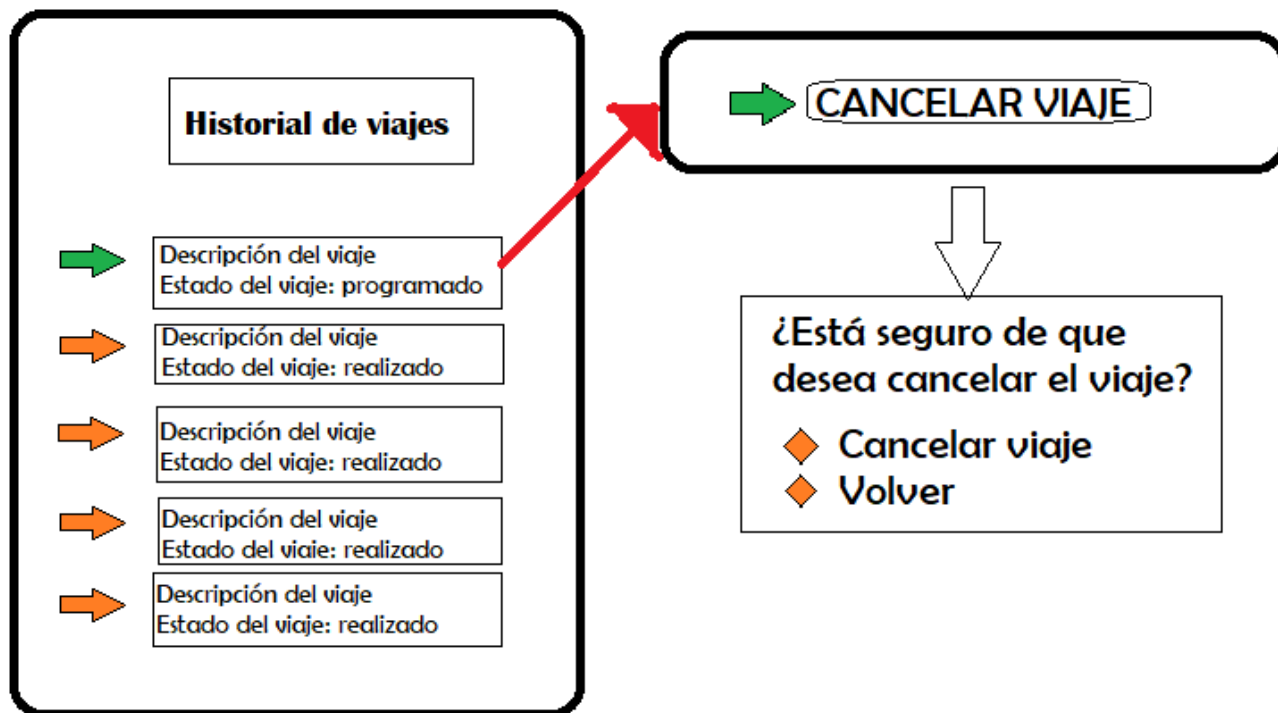
1. El usuario accede a su perfil y selecciona la opción "Mis viajes".
2. Aparece una lista con el historial de viajes realizados y programados.
3. El usuario puede:
  - Revisar detalles de un viaje realizado o programado.
  - Cancelar un viaje programado.
4. Si selecciona "Cancelar viaje", se mostrará una confirmación antes de proceder.

### Flujo visual y eventos:

1. **Acceso a "Mis viajes":**
  - En el perfil del usuario, habrá una opción visible para acceder a "Mis viajes".
  - Al seleccionar esta opción, se cargará una lista de los viajes.
2. **Visualización del historial:**
  - Se mostrará una lista dividida en dos secciones:
    - **Viajes realizados:** Listados con fecha, rol (conductor/pasajero), origen, destino, y nombre del usuario con quien se pactó.
    - **Viajes programados:** Listados con la misma información, pero también incluirán un botón para cancelarlos.
3. **Cancelación de un viaje programado:**
  - Al hacer clic en "Cancelar viaje":
    - Aparecerá un pop-up de confirmación con el mensaje: *"¿Estás seguro de que deseas cancelar este viaje? El otro usuario será notificado."*
    - El usuario podrá confirmar o cancelar la acción.
4. **Mensajes de retroalimentación:**
  - **En caso de éxito:** Aparecerá un mensaje en pantalla: *"Viaje cancelado exitosamente. Se ha notificado al otro usuario."*
  - **En caso de error (por ejemplo, si no hay viajes programados):** Se mostrará: *"No tienes viajes programados para cancelar."*
5. **Animaciones y transiciones:**

- La lista de viajes debe cargarse con una animación de entrada (desvanecimiento o desplazamiento).
- Al cancelar un viaje, el elemento correspondiente en la lista desaparecerá con una animación suave para indicar que ya no está disponible.

Mockups/Prototipos:



## Historia de Usuario #11

### Descripción conceptual

Módulo	Configuración y soporte.
Descripción de la(s) funcionalidad(es) requerida(s):	Los usuarios podrán configurar y actualizar su perfil, acceder a una lista de preguntas frecuentes, y solicitar soporte técnico básico para resolver problemas o mejorar su experiencia en la plataforma.

### Backend

URL	Métodos	Código http
https://localhost:8080/user/profile/update (PUT)	PUT GET POST	PUT: 200, 400, 404 GET (FAQ): 200, 404 POST (Soporte): 200, 400
https://localhost:8080/help/faq (GET)		



https://localhost:8080/help/support (POST)		
<div>Caso de uso técnico</div> <div>PUT: Configuración del perfil</div> <div>Permite a los usuarios actualizar su foto de perfil, nombre de usuario y preferencias.</div>		
<div>Datos de entrada</div> <pre>{  "userId": "12345",  "profilePicture": "base64_image_string",  "username": "NuevoNombre",  "preferences": {    "notifications": true,    "darkMode": false  }  }</pre>	<div>Datos de salida</div> <div>200:</div> <pre>{  "status": "Successful",  "statusCode": 200,  "message": "Profile updated successfully."}</pre> <div>400:</div> <pre>{  "status": "Bad Request",  "statusCode": 400,  "message": "Invalid input data."}</pre> <div>404:</div> <pre>{  "status": "Not Found",  "statusCode": 404,  "message": "User not found."}</pre>	
<div>Caso de uso técnico</div> <div>GET: Preguntas frecuentes (FAQ)</div> <div>Devuelve una lista de preguntas y respuestas frecuentes.</div>		
<div>Datos de entrada</div> <div>No hay datos necesarios</div>	<div>Datos de salida</div> <div>200:</div> <pre>{  "status": "Successful",  "statusCode": 200,  "data": [    {      "question": "¿Cómo actualizo mi perfil?",      "answer": "Puedes actualizarlo desde la sección 'Ajustes' en tu cuenta."    },    {      "question": "¿Qué hago si no encuentro un viaje?",      "answer": "Asegúrate de configurar bien"<!--    }  ]}</pre--></pre>	

	<pre>tu punto de partida y destino."     }   ] }</pre> <p><b>404:</b></p> <pre>{   "status": "Not Found",   "statusCode": 404,   "message": "No FAQs available at the moment." }</pre>
<p align="center"><b>Caso de uso técnico</b>  <b>POST: Soporte técnico básico</b>  Permite a los usuarios enviar solicitudes de soporte técnico.</p>	
<p align="center"><b>Datos de entrada</b></p> <pre>{   "userId": "12345",   "issue": "No puedo actualizar mi foto de perfil.",   "priority": "High" }</pre>	<p align="center"><b>Datos de salida</b></p> <p><b>200:</b></p> <pre>{   "status": "Successful",   "statusCode": 200,   "message": "Your support request has been submitted. We will contact you shortly." }</pre> <p><b>404:</b></p> <pre>{   "status": "Bad Request",   "statusCode": 400,   "message": "Incomplete support request data." }</pre>

## Frontend

### Interacción esperada:

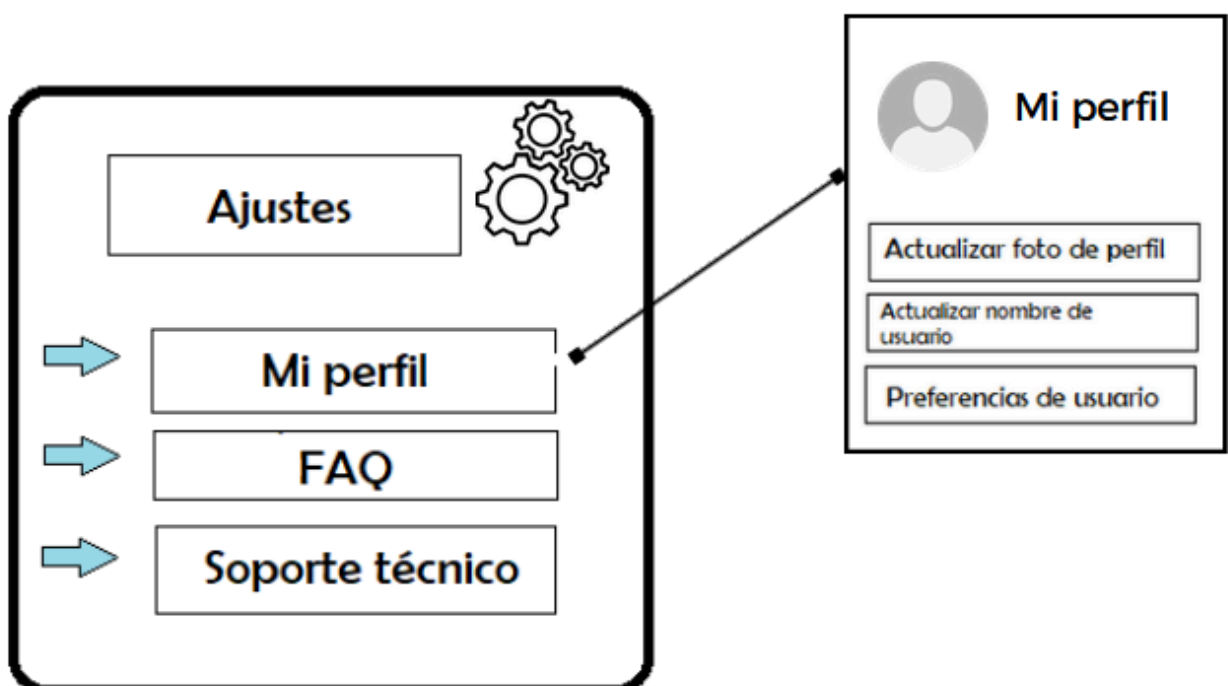
1. El usuario accede a la sección de **Ajustes** desde la página principal.
2. En la sección de ajustes, hay tres opciones disponibles:
  - **"Mi perfil"**: Permite actualizar la foto de perfil, nombre y preferencias.
  - **"Preguntas frecuentes"**: Presenta una lista interactiva con preguntas y respuestas frecuentes.
  - **"Soporte técnico"**: Incluye un formulario para describir problemas y enviar solicitudes de ayuda.

### Flujo visual y eventos:

1. **Acceso a la sección de ajustes:**
  - El botón de "Ajustes" será accesible desde el menú de navegación o la página principal.

- Al hacer clic, redirige a una pantalla con las tres opciones principales: *Mi perfil*, *Preguntas frecuentes* y *Soporte técnico*.
- 2. **Mi perfil:**
  - Los datos actuales del usuario (foto de perfil, nombre, preferencias) estarán precargados en campos editables.
  - Los eventos incluyen:
    - Cambiar la foto de perfil (al hacer clic, se abrirá un selector de archivos).
    - Editar el nombre o preferencias.
    - Hacer clic en "Guardar cambios" para enviar los datos al servidor.
    - Hacer clic en "Cancelar" para deshacer los cambios y restaurar el estado original.
  - Feedback visual:
    - Una barra de progreso o animación al guardar cambios.
    - Mensajes de éxito o error tras el intento de actualización.
- 3. **Preguntas frecuentes:**
  - Se mostrará una lista de preguntas con un diseño de acordeón (desplegable).
  - Cada pregunta se expandirá al hacer clic, mostrando la respuesta asociada.
  - Opcional: incluir una barra de búsqueda para filtrar preguntas por palabras clave.
- 4. **Soporte técnico:**
  - Habrá un formulario sencillo con los siguientes campos:
    - Descripción del problema (campo de texto obligatorio).
    - Selección de prioridad (baja, media, alta).
  - Botón de "Enviar solicitud" que:
    - Muestra un indicador de carga mientras se procesa.
    - Al completarse, da un mensaje de confirmación como *"Tu solicitud ha sido enviada. Nos pondremos en contacto contigo pronto."*
    - En caso de error, muestra un mensaje como *"No se pudo enviar la solicitud. Intenta nuevamente más tarde."*

Mockups/Prototipos:



## PUNTO 7: Clean code

Para la implementación del clean code estamos utilizando las herramientas de [Prettier](#) y [ESlinter](#), pero el uso de estas herramientas no resulta siendo más allá de anecdótico esto porque el framework que hemos usado ([NestJS](#)) no presenta necesidad de generar una configuración ya que este paquete viene listo para ser usado solamente con la invocación de un comando si no se tiene instalada la extensión de dichos linters, por tanto generalmente el código al menos en forma de todos los integrantes del equipo va a ser similar por las restricciones que generan estos linters (por ejemplo ESlinter no permite ejecutar si no está todo según la configuración dada por defecto). A pesar de ello, se adjunta ejemplo de la sencilla implementación a continuación:

```
— class-transformer@0.5.1
— class-validator@0.14.1
— eslint-config-prettier@9.1.0
— eslint-plugin-prettier@5.2.1
— eslint@8.57.1
— jest@29.7.0
— prettier@3.4.1
— reflect-metadata@0.2.2
— rxjs@7.8.1
— source-map-support@0.5.21
— supertest@7.0.0
— ts-jest@29.2.5
— ts-loader@9.5.1
```

```
PS C:\Users\backe\Documents\testingnestapp> npx prettier --write .
```

```
>>
```

```
.eslintrc.js 45ms
.prettierrc 17ms
.vscode/settings.json 1ms
nest-cli.json 2ms
package-lock.json 114ms
package.json 2ms
README.md 45ms
src/app.controller.spec.ts 58ms (unchanged)
src/app.controller.ts 9ms (unchanged)
src/app.module.ts 3ms (unchanged)
```

```
PS C:\Users\backe\Documents\testingnestapp> npm install --save-dev eslint prettier eslint-config-prettier eslint-plugin-prettier
>>
```

```
changed 2 packages, and audited 697 packages in 4s
```

```
112 packages are looking for funding
  run `npm fund` for details
```

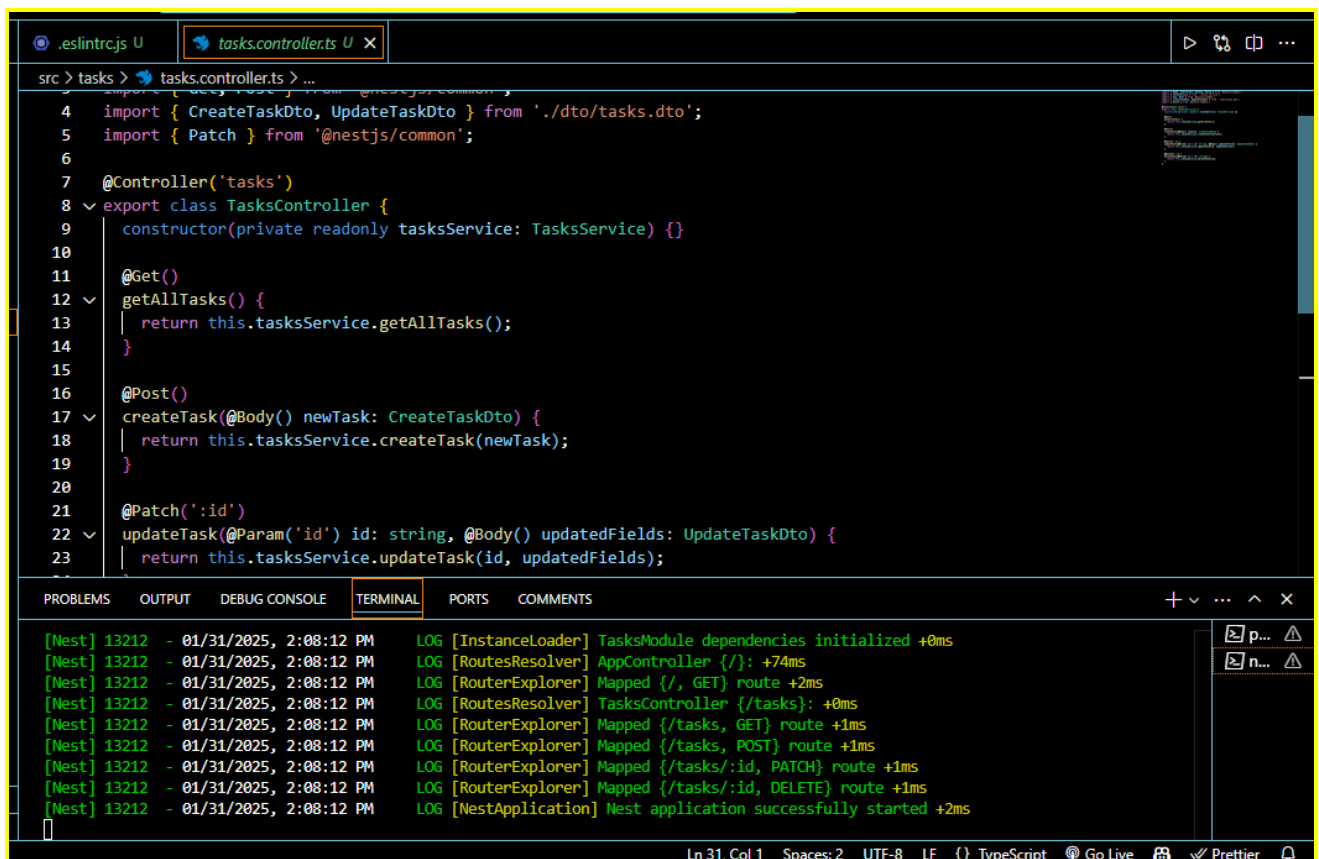
```
3 high severity vulnerabilities
```

```
To address all issues, run:
  npm audit fix
```

**.eslintrc.js** U X

**.eslintrc.js** > <unknown> > rules

```
1  module.exports = {
2
3    parserOptions: {
4      project: 'tsconfig.json',
5      tsconfigRootDir: __dirname,
6      sourceType: 'module',
7    },
8    plugins: ['@typescript-eslint/eslint-plugin', 'prettier'],
9    extends: [
10     'plugin:@typescript-eslint/recommended',
11     'plugin:prettier/recommended',
12   ],
13   root: true,
14   env: {
15     node: true,
16     jest: true,
17   },
18   ignorePatterns: ['.eslintrc.js'],
19   rules: {
20     '@typescript-eslint/interface-name-prefix': 'off',
21     '@typescript-eslint/explicit-function-return-type': 'off',
22     '@typescript-eslint/explicit-module-boundary-types': 'off',
23     '@typescript-eslint/no-explicit-any': 'off',
24     'prettier/prettier': [
25       'error',
26       {
27         endOfLine: 'auto',
28       },
29     ],
30   },
31 }
```



The screenshot shows a Visual Studio Code editor with a TypeScript file named `tasks.controllers.ts` open. The file contains a `TasksController` class with methods `getAllTasks`, `createTask`, and `updateTask`. The terminal at the bottom displays the output of the application, showing logs for the initialization of the `TasksModule` and the mapping of routes for the `TasksController`.

```
src > tasks > tasks.controllers.ts > ...
import { Get, Post, Patch } from '@nestjs/common';
import { CreateTaskDto, UpdateTaskDto } from '../dto/tasks.dto';
import { Patch } from '@nestjs/common';

@Controller('tasks')
export class TasksController {
  constructor(private readonly tasksService: TasksService) {}

  @Get()
  getAllTasks() {
    return this.tasksService.getAllTasks();
  }

  @Post()
  createTask(@Body() newTask: CreateTaskDto) {
    return this.tasksService.createTask(newTask);
  }

  @Patch('/:id')
  updateTask(@Param('id') id: string, @Body() updatedFields: UpdateTaskDto) {
    return this.tasksService.updateTask(id, updatedFields);
  }
}
```

TERMINAL

```
[Nest] 13212 - 01/31/2025, 2:08:12 PM LOG [InstanceLoader] TasksModule dependencies initialized +0ms
[Nest] 13212 - 01/31/2025, 2:08:12 PM LOG [RoutesResolver] AppController {/}: +74ms
[Nest] 13212 - 01/31/2025, 2:08:12 PM LOG [RouterExplorer] Mapped {/, GET} route +2ms
[Nest] 13212 - 01/31/2025, 2:08:12 PM LOG [RoutesResolver] TasksController {/tasks}: +0ms
[Nest] 13212 - 01/31/2025, 2:08:12 PM LOG [RouterExplorer] Mapped {/tasks, GET} route +1ms
[Nest] 13212 - 01/31/2025, 2:08:12 PM LOG [RouterExplorer] Mapped {/tasks, POST} route +1ms
[Nest] 13212 - 01/31/2025, 2:08:12 PM LOG [RouterExplorer] Mapped {/tasks/:id, PATCH} route +1ms
[Nest] 13212 - 01/31/2025, 2:08:12 PM LOG [RouterExplorer] Mapped {/tasks/:id, DELETE} route +1ms
[Nest] 13212 - 01/31/2025, 2:08:12 PM LOG [NestApplication] Nest application successfully started +2ms
```

```
{
  "extends": ["plugin:prettier/recommended"],
  "rules": {
    "prettier/prettier": ["error", { "endOfLine": "auto" }]
  }
}
```

*Con esta configuración, Prettier formatea automáticamente el código al guardar los archivos, asegurando consistencia entre los integrantes del equipo.*

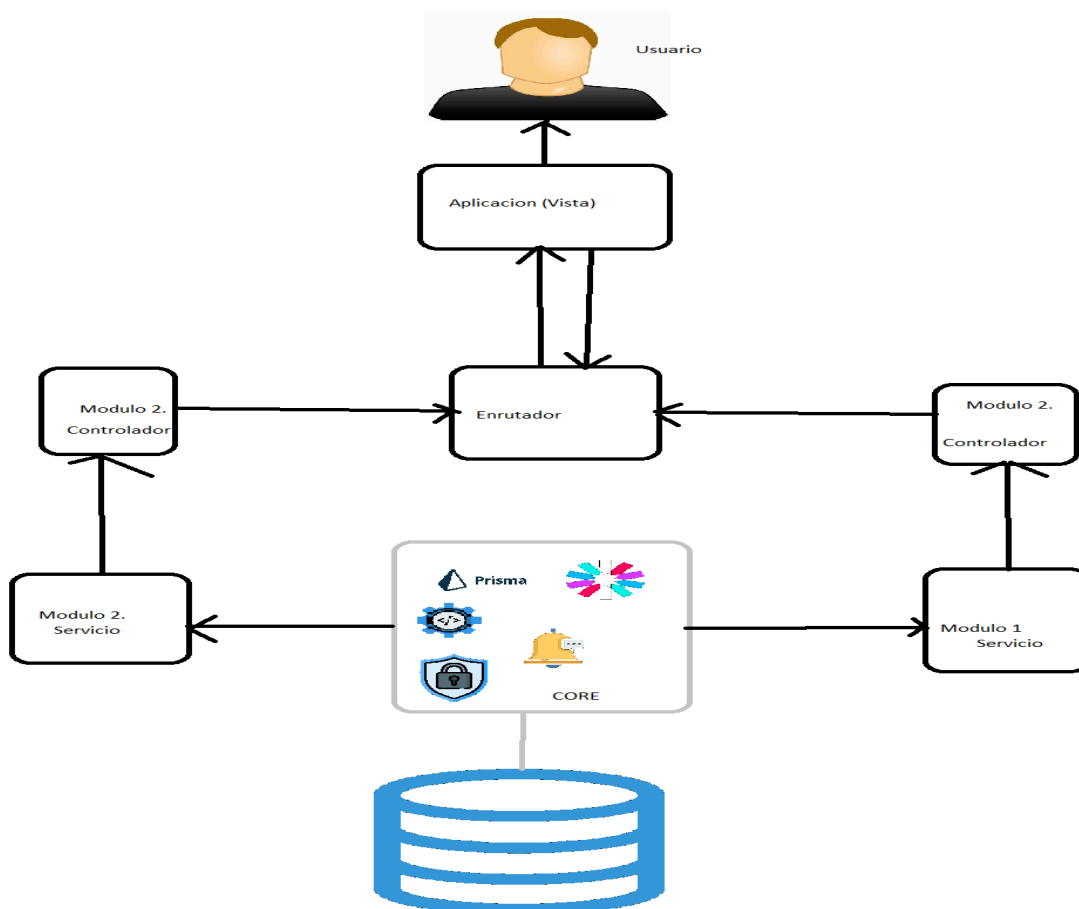
## PUNTO 8: Diseño y Arquitectura

Para el proyecto hemos elegido usar la arquitectura propuesta por el framework (NestJS es flexible frente a la arquitectura a ser usada, así que le hicimos un pequeño cambio para poder aplicar el patrón de diseño singleton de una manera eficaz sin afectar completamente la arquitectura general del proyecto) la cual es el MVC (modelo vista controlador). La elección va encaminada a la elección del framework, ya que es la arquitectura por defecto y no requiere ningún cambio en la configuración del entorno de desarrollo, además presenta la ventaja del manejo individual de cada uno de los módulos lo que se traduce en una mayor reutilización del código y facilidad en el trabajo colaborativo asignando a cada programador un módulo en particular. Pero su desventaja reside en sus ventajas, ya que el manejo individual de modelos puede hacer que unos queden muchísimo más cargados que otros debido a la variedad de interacciones existentes, lo que puede hacer que en ciertas partes el código

resulte más extenso y difícil de mantener si no sigue parámetros de Clean Code.

Para poder controlar un poco este problema de sobrecarga de tareas y responsabilidades se crea un módulo el cual contiene aquellos puntos de la aplicación que más van a ser utilizados en la app, por poner un ejemplo la base de datos va a ser un factor fundamental de la aplicación y llamada casi por todos los módulos, entonces vamos a implementarla en core para así poder tener una lógica menos pesada y un menor consumo de recursos por la app, cosa que también va a ocurrir con la RTDB (Real time database), notificaciones, seguridad y configuración.

Finalmente, esta aplicación en NestJS se va a volver el backend, que se va a comunicar con una interfaz gráfica construida en React Native (que sigue los mismos principios que el MVC si se plantea de la forma correcta), para que finalmente tengamos la conexión entre lógica e interfaz de usuario.



Para la base de datos elegimos una relacional, esto ya que a la hora de hacer un análisis de nuestras necesidades desde la aplicación lo que más sentido tenía era el manejo de una base de datos relacional que nos permitiera manejar datos en formato JSON y datos como generalmente se suelen manejar en este tipo de implementación como lo es bien organizados, claramente lo JSON que vamos a utilizar tienen un ENUM para así poder tener controlados los datos del JSON. Además de que buscamos poder hacer consultas rápidas a partir de una llave o una relación. En este orden de ideas elegimos la base de datos postgresSQL. A pesar de esto seguimos aprendiendo a utilizar la RTDB, para poder tener un control de la posición en tiempo real de los viajes, la cual no entra ni en relacional ni en no relacional.

El [diagrama se encuentra en este vinculo](#). Las decisiones de diseño buscaron normalizar al máximo pero sin perder información importante, como pueden ser las dos claves primarias de un vehículo, para poder mantener una sola persona por vehículo y además por razones de seguridad poder hacer filtros de control en caso de emergencias.





## PUNTO 9: Patrones de diseño

Para el desarrollo de nuestra aplicación, hemos decidido utilizar una serie de **patrones de diseño** que nos permitirán mejorar la **escalabilidad, modularidad y mantenibilidad del código**. A continuación, se detallan los principales patrones implementados en la arquitectura del sistema, junto con ejemplos específicos de su aplicación dentro del proyecto.

Dentro de los **patrones creacionales**, implementaremos el **Singleton** para garantizar que ciertos servicios críticos, como la conexión a la base de datos (PostgreSQL) y la gestión de notificaciones en tiempo real (Firebase RTDB), mantengan una única instancia a lo largo de la ejecución de la aplicación. Esto evitará múltiples conexiones innecesarias y asegurará la eficiencia en la comunicación. Asimismo, utilizaremos el **Factory Method** en la creación de objetos clave como usuarios y viajes, lo que permitirá mayor flexibilidad en la inicialización y facilitará la integración de nuevos tipos de usuarios en el futuro. También aplicaremos el **Builder**, particularmente en la construcción de viajes, permitiendo configurar de manera estructurada múltiples atributos como origen, destino, asientos disponibles, conductor y precio, asegurando así claridad en la creación de cada instancia.

En cuanto a los **patrones estructurales**, utilizaremos la arquitectura **Model-View-Controller (MVC)**, ya que viene por defecto en **NestJS y React Native**, permitiendo separar la lógica de negocio, la interfaz de usuario y la gestión de datos de forma modular. Para facilitar la integración con servicios externos como la pasarela de pagos y la API de Google Maps, aplicaremos el patrón **Adapter**, asegurando que los cambios en estos servicios no afecten directamente el código de nuestra aplicación. Además, emplearemos el patrón **Facade**, el cual encapsulará servicios como la autenticación de usuarios, la base de datos y las notificaciones, proporcionando una única interfaz de acceso y reduciendo la complejidad del sistema. Adicionalmente, usaremos **Decorator** en la validación de datos dentro de los controladores de NestJS, lo que permitirá agregar validaciones dinámicas sin alterar la estructura base de los controladores, además de facilitar la personalización de perfiles de usuario al añadir roles y permisos específicos.

Para mejorar la interacción entre módulos, aplicaremos diversos **patrones de comportamiento**. El patrón **Observer** será fundamental en el sistema de notificaciones en tiempo real, permitiendo que pasajeros y conductores reciban actualizaciones instantáneas sobre el estado del viaje. Para la gestión de pagos, utilizaremos el patrón **Command**, garantizando que las transacciones sean procesadas de manera desacoplada, lo que facilitará la posibilidad de reintentar, cancelar o almacenar pagos en caso de fallos. En la búsqueda de viajes, implementaremos el patrón **Strategy**, ofreciendo distintas estrategias de filtrado según ubicación, precio y horario, lo que permitirá mayor flexibilidad en las opciones de búsqueda. Para validar usuarios y pagos, usaremos **Chain of Responsibility**, asegurando que diferentes módulos del sistema puedan procesar y validar datos antes de aprobar una transacción o un registro. Finalmente, implementaremos **Mediator** en el sistema de mensajería, lo que permitirá coordinar la comunicación entre usuarios sin generar acoplamiento excesivo entre los módulos.

En lo que respecta a los **patrones arquitectónicos**, consideramos la posibilidad de escalar la aplicación en el futuro mediante el uso de **microservicios**, separando módulos clave como autenticación, viajes y pagos en servicios independientes para mejorar la escalabilidad y modularidad del sistema. Para manejar la persistencia de datos de manera eficiente, aplicaremos el patrón **Repository**, proporcionando una capa de abstracción entre la base de datos y la lógica de negocio, lo que facilitará el mantenimiento y optimización de las consultas en PostgreSQL. Adicionalmente, en la gestión de pagos y reservas, utilizaremos el patrón **Unit of Work**, garantizando la consistencia de las transacciones en la base de datos, evitando errores en caso de fallos y asegurando que todas las operaciones se completen correctamente o se reviertan en caso de problemas.