Especificaciones de Arquitectura - Shared Ride.

En este documento se explican la arquitectura de desarrollo y las especificaciones técnicas de la aplicación Shared Ride. También es posible visualizar este documento en forma de wiki aquí.

Índice de temas de este Documento

Prerrequisitos para trabajar con el código de la aplicación

Se recomienda tener experiencia (al menos unas 200 líneas) escribiendo programas mediante el paradigma de Programación Orientada a Objetos en Python. De preferencia con Python 3.5 o superior.

También es muy recomendable conocer los comandos básicos en una terminal UNIX, desde el movimiento a través del sistema de archivos, hasta bash scripting y pipes.

Adicionalmente, las siguientes recomendaciones:

Entender con claridad los conceptos fundamentales del desarrollo web

- Modelos de arquitectura cliente-servidor
- Protocolos de internet, en especial HTTP
- Diferencias entre Back-end y Front-end

Django

- Ciclo de vida de una petición.
- Clases HttpRequest y HttpResponse
- Haber concluido y/o lanzado algún proyecto que vaya más allá de un CRUD básico
- Class-based views

Docker

- Contenedores, imágenes, volúmenes
- Docker-compose

Bonus

- Perderle el miedo al idioma **inglés**.
- **Leer** la documentación hasta el cansancio. Todo el conocimiento está ahí.
- **Practicar**.
- **Paciencia**.

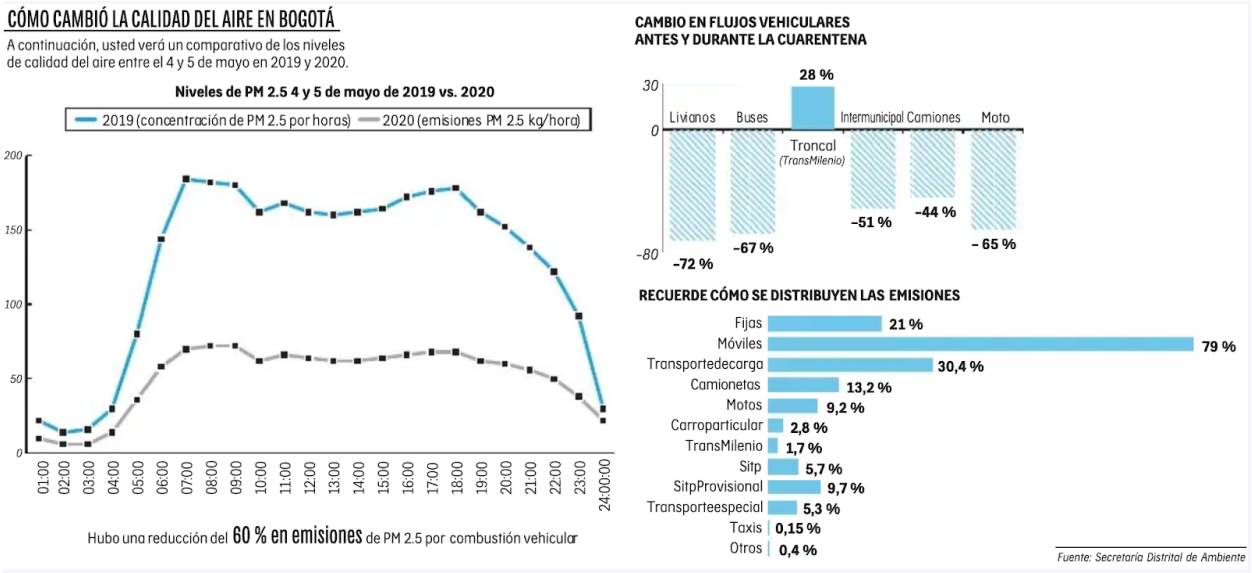
Especificaciones del proyecto

Definición del Problema

Los vehículos son considerados como agentes importantes de contaminación ambiental, en la actualidad, son indispensables en el funcionamiento de las sociedades modernas, circulando por las carreteras del mundo millones de vehículos que hacen que cada vez sean más los problemas ambientales que sufrimos.Según el periódico de EL TIEMPO ”Debido a la cuarentena los niveles de contaminación se redujeron en un 80%, pero con la vuelta a la presencialidad estos niveles volverán a subir” (Moreno et al., 2020). Por este motivo se implementará la medida del pico y placa en la capital. En la tarea de aliviar las dificultades presentadas por esta medida, crearemos una aplicación de carpooling donde los miembros puedan compartir vehículos dentro de comunidades privadas (con invitaciones) creadas por los mismos. Adicionalmente, se reunirá información y datos estadísticos sobre los viajes compartidos.

“Dentro de los diferentes contaminantes atmosféricos* , la Organización Mundial de la Salud (oms) considera que el material particulado (pm) † sobrepasa con mayor frecuencia los niveles críticos de concentración”,(Contaminación Del Aire y Vulnerabilidad de Individuos Expuestos: Un Caso de...: Sistema de Bibliotecas UTADEO - Mega Buscador, n.d.). De allí surge nuestra necesidad de reducir los niveles de estas partículas en la ciudad de Bogotá y para dicho caso la implementación del carpooling es una de las alternativas mayormente aceptadas y que cuentan ya con una implementación exitosa en ciudades como México.

Es importante destacar que la seguridad de un método como el carpooling no es totalmente confiable, debido a que estarás utilizando tu vehículo acompañado de personas que son desconocidas por lo que se implementa la funcionalidad de crear comunidades (círculos) en las que la confianza sea uno de los principales factores que permitan una aceptación en gran porcentaje de este medio que busca la reducción de la polución en la ciudad de Bogotá.



Objetivo General

Desarrollar una aplicación nativa que permita la gestión de viajes en automóviles compartidos mediante la modalidad de carpooling.

Objetivos Específicos

- Identificar los datos que se requiere recopilar sobre los usuarios, viajes, círculos, etc., así como la estructura sobre la cual se van a manejar.
- Determinar el diseño y arquitectura de la aplicación mediante metodologías y patrones de diseño de software.

- Implementar el prototipo funcional (*Minimum Viable Product*) para la aplicación, utilizando las metodologeias, fameworks o librerías elegidos para su implementación.
- Realizar testing a la aplicación mediante pruebas unitarias incluidas en el código fuente.

Requerimientos Funcionales

- El sistema debe permitir el registro usuarios sin estar vinculado a un círculo específico.
- El sistema debe realizar el envío automático de un correo de verificación en el momento en que se crea una cuenta nueva.
- El sistema debe verificar las cuentas creadas utilizando un token único para cada cuenta.
- El sistema debe permitir el log in de los usuarios con su username y password.
- El sistema debe permitir que los usuarios visualicen la información de sus perfiles.
- El sistema debe permitir que los usuarios actualicen la información de sus perfiles.
- El sistema debe listar todos los círculos, incluyendo opciones de filtrado y búsqueda.
- El sistema debe permitir crear círculos.
- El sistema debe permitir la visualización de los detalles de un círculo.
- El sistema debe permitir que los administradores de un círculo actualicen los datos del mismo.
- El sistema debe listar los miembros de un círculo.
- El sistema solo puede permitir el ingreso de usuarios dentro de los círculos con un código de invitación único generado por un miembro ya existente.
- El sistema debe mostrar los detalles de los miembros de un círculo.
- El sistema debe permitir a los administradores de los círculos eliminar miembros del mismo.
- El sistema debe permitir la generación de códigos únicos de invitación a los miembros de un círculo.
- El sistema debe mostrar la calificación de cada uno de los usuarios con base a la interacción con círculos y rides realizados.
- El sistema debe poder listar los rides realizados y guardados.
- El sistema debe permitir la creación de rides nuevos en un círculo.
- El sistema debe permitir a los miembros de un círculo unirse a un ride.
- El sistema debe permitir calificar un ride.
- El sistema debe enviar un recordatorio a los usuarios para calificar los rides tomados.
- El sistema debe permitir a sus administradores descargar un CSV con los usuarios con mejores calificaciones.

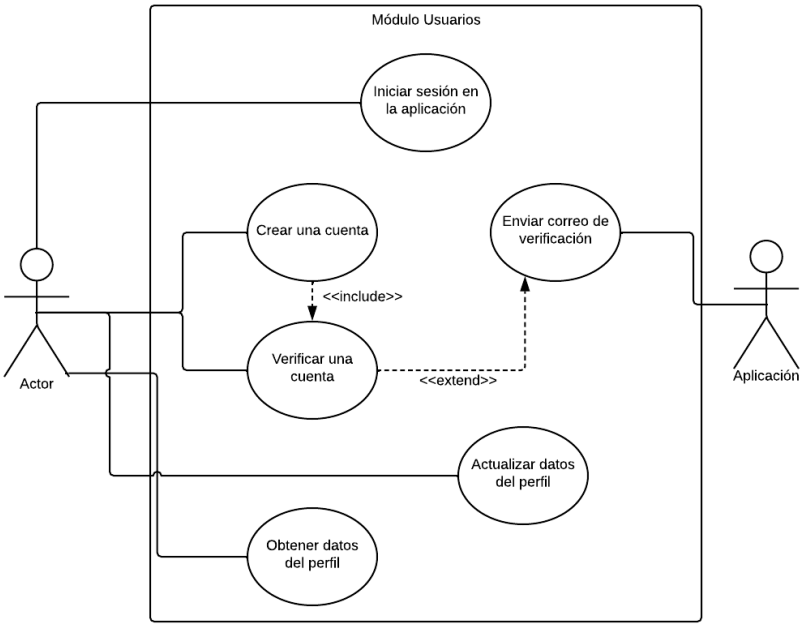
- El sistema debe enviar un reporte mensual a los adminstradores de la actividad del mismo.

Requerimientos no Funcionales

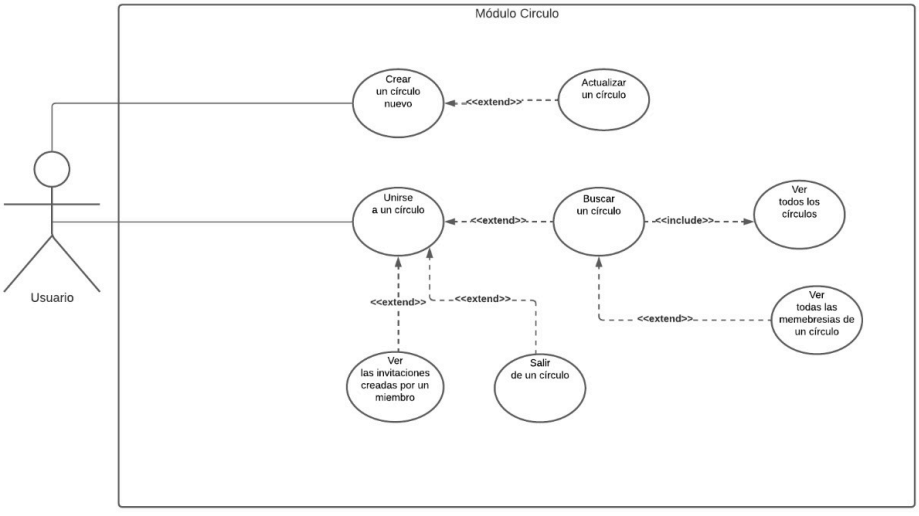
- Para el desarrollo de la aplicación se hará uso de una base de datos relacional.
- El sistema debe presentará una interfaz sencilla para que sea de fácil manejo para los usuarios.
- El sistema debe tener una disponibilidad continua con un nivel de servicio para los usuarios de 24 horas, los siete días de la semana.
- El backend de la aplicación contará con autenticación por medio de JWT, y con autenticación con password para los usuarios.
- El sistema deberá ser fácilmente escalable, con el fin de poder hacer crecer la aplicación a futuro e incorporar nuevas funcionalidades.
- El sistema debe funcionar en todos los navegadores web modernos.
- El sistema debe disponer de una buena documentación que permita realizar operaciones de mantenimiento con el menor esfuerzo posible.
- La aplicación móvil deberá garantizar un buen manejo en cuanto al acceso concurrente de usuarios, por lo que deberá contar con una arquitectura de servidor que permita la ampliación de los recursos de cómputo con base a la cantidad de usuarios realizando peticiones al servidor.
- La aplicación web debe ser responsive y la nativa debe estar enfocada a sistemas operativos Android y iOS.

Casos de uso

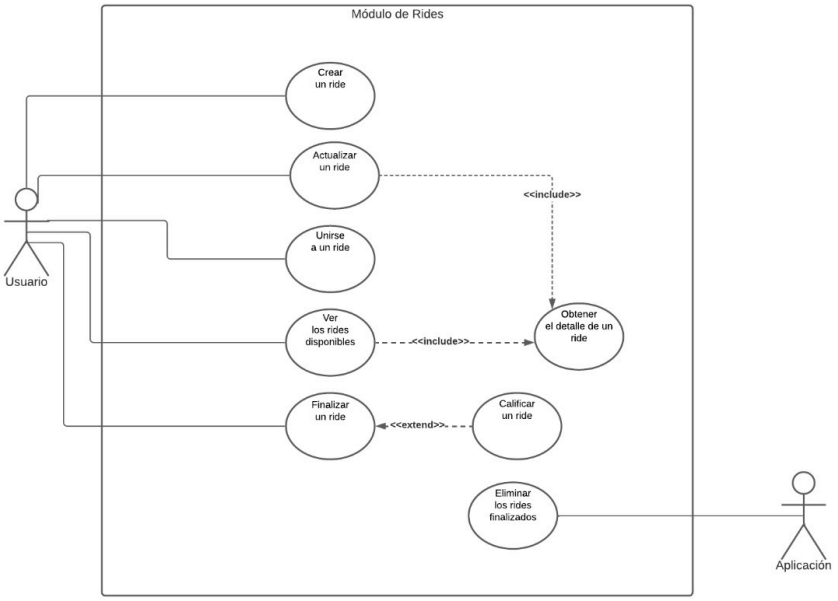
Módulo de usuarios



Módulo de círculos



Módulo de rides



Stack tecnológico

Es una lista de todos los servicios tecnológicos (lenguajes de programación, frameworks, librerías, etc.) utilizados para construir y desplegar la aplicación.

Para caso específico de Shared Ride, utilizamos el siguiente stack tecnológico:

Tecnologías Frontend

- React.JS
- React Native
- TypeScript
- Bootstrap (v.5.1)
- SASS

Tecnologías Backend

- Python 3.9
- Django
- Django REST Framework

Tecnologias de base de datos

- PostgreSQL

Framework de desarrollo

- Django
- React Native

Librerías especiales utilizadas

- Celery (Cola de tareas asíncronas)
- Redis (Message broker)
- PyJWT (Autenticación con JWT)
- Whitenoise (Gestión de archivos estáticos)
- django Anymail (Envío de correo electrónico)

Herramientas de despliegue

- Gunicorn (Servidor Web)
- Docker (Virtualización por contenedores)
- Docker Compose (Ejecución de múltiples contenedores)

Herramientas de desarrollo

- Visual Studio Code (https://code.visualstudio.com/download)
- Git (https://git-scm.com/downloads)

Al instalar Git por primera vez, se deben establecer las siguientes variables globales en línea de comandos:

```
$ git config --global user.name "John Doe"

$ git config --global user.email johndoe@example.com
```

¿Por qué Python y Django?

Django REST Framework fue escogido como la herramienta para construir el backend de la aplicación debido a su simplicidad, flexibilidad, calidad y cobertura para pruebas. Adicionalmente, su motor de serialización y sus clases genéricas para lógicas CRUD resultan en una gran optimización de los tiempos para el desarrollo.

JavaScript, ¿por qué no se usa?

Con la finalidad de mantener una mayor calidad en el código del frontend de la aplicación, se optó por la utilización de TypeScript, ya que el tipado estático reduce las posibilidades de error y hace que el código sea más legible, entendible y fácil de mantener. Ya que TypeScript se compila directamente en JavaScript, esto no representa ninguna limitación ni necesidad de dependencias adicionales para el despliegue de la aplicación.

Arquitectura de Shared Ride

En el siguiente diagrama, se muestra en forma simplificada, el diseño de arquitectura de la aplicación Shared Ride.

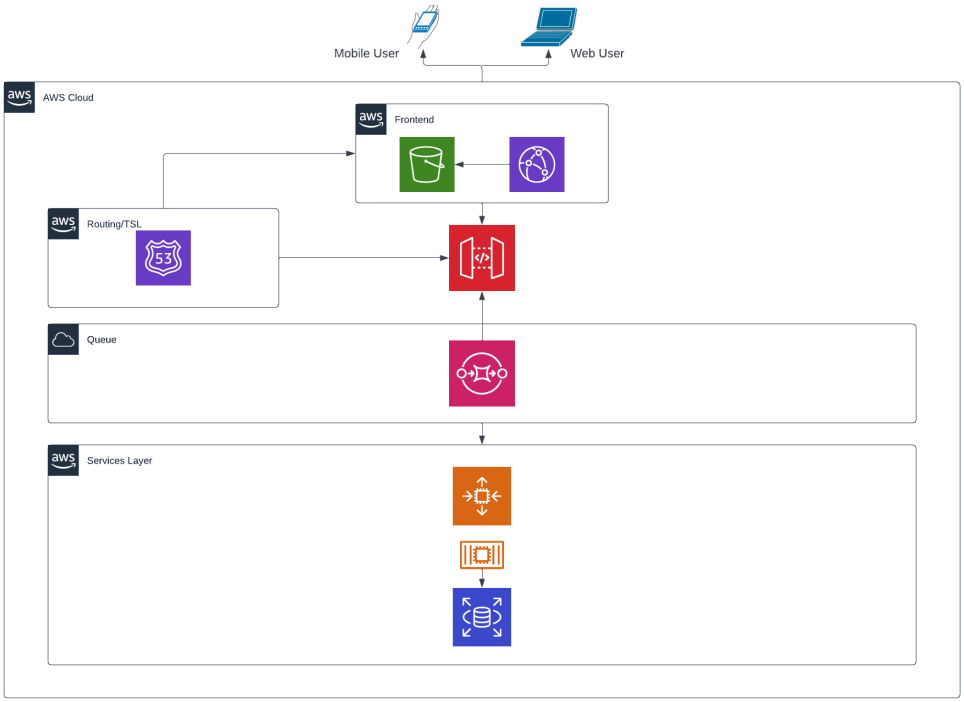
Para la arquitectura de la aplicación se ha escogido el estilo de arquitectura **SOA** (*Service-Oriented Architecture*). Por tanto, la aplicación:

- Es auto-contenida.
- Es una caja negra para sus consumidores.
- Representa una actividad de negocio con una finalidad específica.

La implementación de la aplicación se presenta como un *Web Service*, bajo el estándar RESTful HTTP. También se siguen los principios de la metodología The Twelve-Factor App y sus principios:

- Codebase. Todo el código fuente se encuentra en un sistema de control de versiones.
- Dependencias. La aplicación no depende de la existencia de ninguna dependencia en el sistema operativo anfitrión.
- Configuración. La configuración de la aplicación se almacena en su entorno.
- Backing services. Los servicios que utiliza la aplicación (bases de datos, cache, task queue, etc.) son recursos conectables.
- Build, release, run. Las etapas de construcción y ejecución están **estrictamente** separadas.
- Procesos. La aplicación se ejecuta como una serie de procesos sin estado, que no comparten nada entre ellos.
- Port-binding. Los procesos de la aplicación son auto-contenidos y exponen sus servicios a través de puertos específicos.
- Concurrencia. La aplicación es completamente escalable de manera horizontal. Solo basta con levantar otro contenedor o instancia.
- Desechabilidad. La aplicación se puede detener, y detener todos sus recursos, así como encenderlos.
- Paridad entre entornos. Los entornos de desarrollo y producción son lo más similar posible.

- Logs. Los logs son hilos de eventos. La aplicación no realiza enrutamiento ni almacenamiento de su hilo de salida. No se escriben o manejan archivos de log, pues cada proceso escribe su propio hilo de eventos a stdout.
- Procesos administrativos. Las tareas adminitrativas no están en la app y se realizan de manera independiente.



A continuación se describen cada uno de los contenedores que se muestran en la figura anterior:

Servicios

En este contenedor residen los servicios de la aplicación. Se entiende por servicio una API REST que gestiona los datos de la aplicación y contiene los m´ttodos de lógica de negocio necesarios para la operación. Así mismo, contiene uno más gestores de bases de datos (PostgreSQL) que se encargan de almacenar los datos gestionados. Cada uno de estos componentes se encuentra desplegado en un contenedor independiente, para facilitar la escalabilidad de la aplicación.

Cola de mensajes

En este contenedor reside la cola de mensajes Celery y el message broker Redis, para la gestión de tareas asíncronas.

Routing/TSL

En este contenedor se emplea el servicio Route 53 para lograr el correcto enrutamiento interno de las peticiones realizadas por la aplicación, las cuales son enviadas a través de un Gateway hacia el backend.

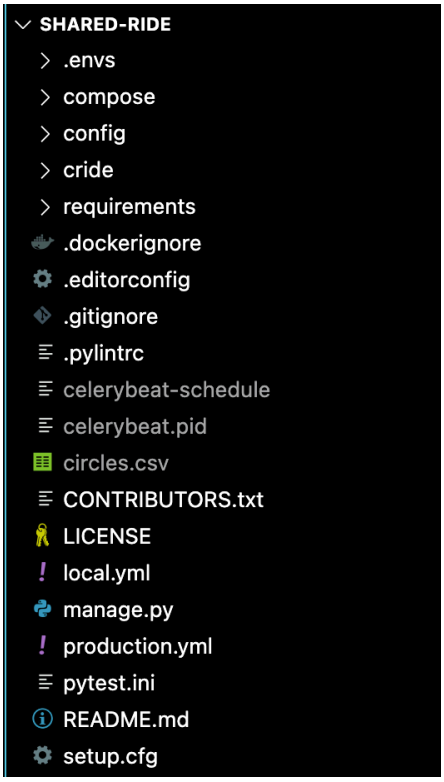
Frontend

También denominada *Capa de Presentación* en modelos de arquitectura por capas. En esta capa se contiene el CDN que distribuye contenidos estáticos y el frontend de la aplicación.

Organización del proyecto en Visual Studio Code

Actualmente usamos Visual Studio Code como IDE de desarrollo.

A continuación se muestra una imagen del proyecto en Code:

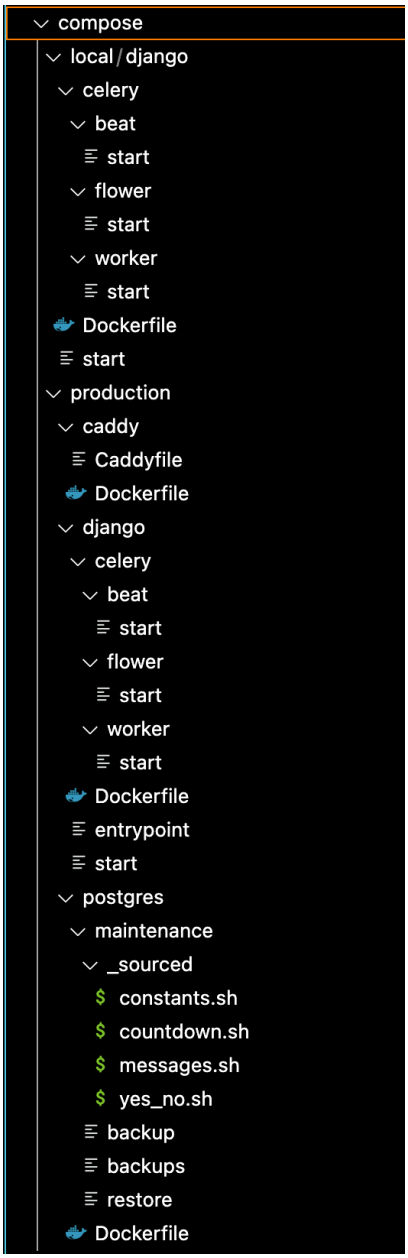


Se observa que la solución está organizada en varios directorios, con varios archivos con finalidades específicas. A continuación se explica el contenido de cada una de ellos:

01. ./.env

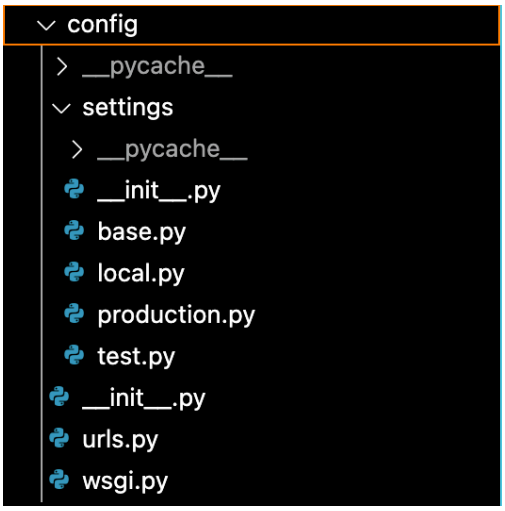
En este directorio se encuentran almacenadas las variables de entorno para ser utilizadas en los ambientes de desarrollo.

02. ./compose



En este directorio se encuentran los archivos de configuración y ejecución para los contenedores de los componentes de la aplicación.

03. ./config



En este directorio se encuentran los archivos principales de configuración de Django, separados en dos módulos:

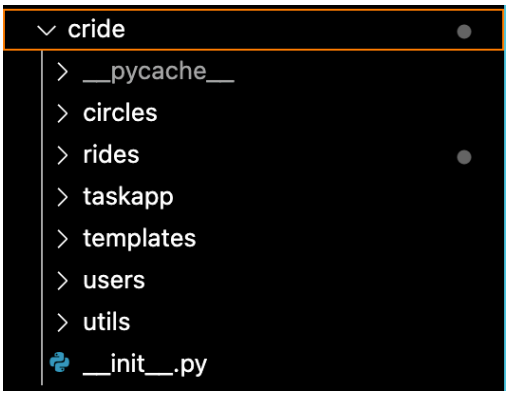
- Módulo config.urls: Contiene la configuración global de las URLs creadas para el backend de la aplicación.
- Módulo config.wsgi: En este módulo se encuentra el punto de entrada de la aplicación WSGI utilizada tanto por el servidor de desarrollo de Django, como por los despliegues WSGI en producción. Expone una variable a nivel del módulo denominada application. Los comandos runserver y runfcgi de Django encuentran esta aplicación por medio de la variable de configuración global WSGI_APPLICATION.
- Módulo config.settings: Contiene las variables de configuración globales de la aplicación. Se compone de tres archivos:

    - base.py contiene las variables base, que son comunes para todos los ambientes.
    - local.py contiene las variables que permiten la ejecución local en ambientes de desarrollo.
    - production.py contiene las variables para la ejecución de la aplicación en ambientes de producción.
    - test.py contiene variables especiales para el *testing* de la aplicación, con la finalidad de acelerar la ejecución de las pruebas.

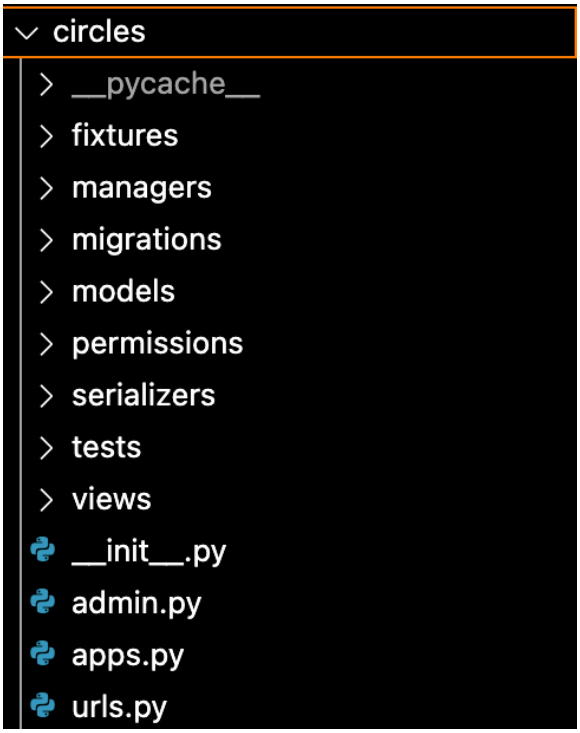
04. ./cride

En este directorio se encuentra el núcleo principal de la aplicación. Contiene los módulos (*apps*) que a su vez contienen las reglas de negocio que la aplicación debe cumplir, aquí es dónde se procesan los datos pasados a través de peticiones desde el frontend, integraciones, etc. Con esto en mente, la aplicación Shared Ride se compone de tres apps: **circles**, **rides** y **users**; así

como de un módulo de tareas asíncronas (**taskapp**), un directorio con la plantilla de correo electrónico (**templates**) y un módulo de utilidades (**utils**).



App de Círculos (*circles*): Un círculo es un grupo privado en el cual los miembros toman y ofrecen viajes (*rides*). Para unirse a un círculo, los usuarios deben recibir un código de invitación único de un miembro ya existente.

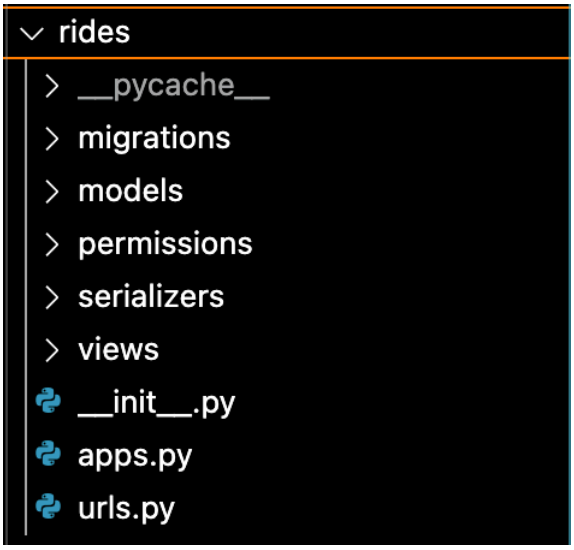


Esta app se compone de los siguientes módulos:

- circles.managers: Contiene el gestor personalizado para los modelos de las invitaciones.
- circles.migrations: Contiene las migraciones de la app, para ser reflejadas en la base de datos.

- circles.models: Contiene la definición de los modelos Circle, Invitation y Membership.
- circles.permissions: Contiene la definición de los permisos necesarios para operar con círculos y membresías.
- circles.serializers: Contiene la definición de los serializadores relacionados con círculos y membresías.
- circles.tests: Contiene las pruebas unitarias de la app de círculos.
- circles.views: Contiene los *views* y *ViewSets* de la app de Círculos.
- circles.admin: Contiene la configuración del sitio de administración de Django para la app.
- circles.apps: Provee la configuración de la app de Círculos para su instalación dentro del proyecto principal.
- circles.urls: Provee la configuración de URLs para la app de Círculos.

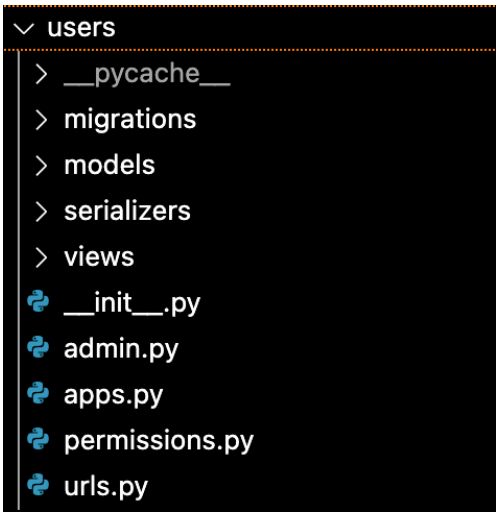
App de Rides: Un Ride es un viaje ofrecido por un usuario miembro de un círculo dentro del mismo.



Esta app se compone de los siguientes módulos:

- rides.migrations: Contiene las migraciones de la app, para ser reflejadas en la base de datos.
- rides.models: Contiene la definición de los modelos Ride y Rating.
- rides.permissions: Contiene la definición de los permisos necesarios para operar con rides y puntuaciones.
- rides.serializers: Contiene la definición de los serializadores relacionados con rides y puntuaciones.
- rides.views: Contiene los *views* y *ViewSets* de la app de Rides.
- rides.apps: Provee la configuración de la app de Rides para su instalación dentro del proyecto principal.
- rides.urls: Provee la configuración de URLs para la app de Rides.

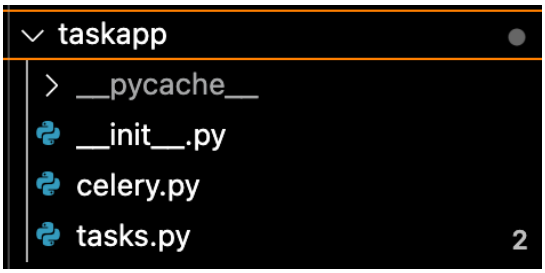
App de Usuarios



Esta app se compone de los siguientes módulos:

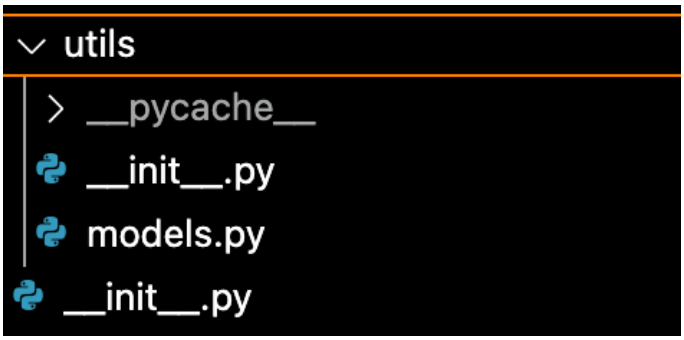
- users.migrations: Contiene las migraciones de la app, para ser reflejadas en la base de datos.
- users.models: Contiene la definición de los modelos User y Profile.
- users.permissions: Contiene la definición de los permisos necesarios para operar con usuarios y perfiles.
- users.serializers: Contiene la definición de los serializadores relacionados con usuarios y perfiles.
- users.views: Contiene los *views* y *ViewSets* de la app de Rides.
- users.apps: Provee la configuración de la app de Usuarios para su instalación dentro del proyecto principal.
- users.urls: Provee la configuración de URLs para la app de Usuarios.

Módulo taskapp: Gestor de tareas asíncronas.



Este módulo contiene dos componentes: El archivo celery.py que inicializa la configuración de la cola de tareas Celery, y el archivo tasks.py, que contiene las tareas que van a ser ejecutadas por la misma.

Módulo utils: Utilidades para la aplicación en general.



Este módulo contiene una sola definición de clase:

- CRideModel: Este es el modelo base de Shared Ride. Actúa como una clase base abstracta de la cual heredan todos los otros modelos de base de datos del proyecto. Esta clase hereda de django.db.models.Model y provee a todas las tablas de la base de datos con los atributos created y modified.

05. ./requirements

En este directorio se encuentran los archivos con las dependencias a ser instaladas por pip en el momento de la creación de los contenedores de la aplicación.

06 ./.

Los otros archivos presentes en el directorio de la aplicación presentan funcionalidades diversas, a saber:

- .dockerignore contiene estructuras de nombres de archivo y/o directorios que deben ser ignorados por Docker.
- .editorconfig contiene variables de configuración para diversos editores de texto.
- .gitignore contiene estructuras de nombres de archivo y/o directorios que deben ser ignorados por Git.
- .pylintrc y setup.cfg contienen variables de configuración para Flake 8, el linter usado para el proyecto.
- celerybeat-schedule y celerybeat.pid son archivos de metadatos generados por Celery.
- circles.csv contiene datos para ser importados durante las pruebas de la aplicación.
- CONTRIBUTORS.txt contiene el listado de las personas que contribuyeron con el desarrollo del proyecto.
- LICENSE contiene la licencia del proyecto (MIT).
- local.yml y production.yml son los archivos utilizados por docker-compose para construir y desplegar las imágenes del proyecto.

- manage.py constituye el archivo de punto de entrada de los comandos de Django.
- pytest.ini contiene variables de configuración para las pruebas del proyecto.
- README.md contiene el Read Me del proyecto.

Control de versiones

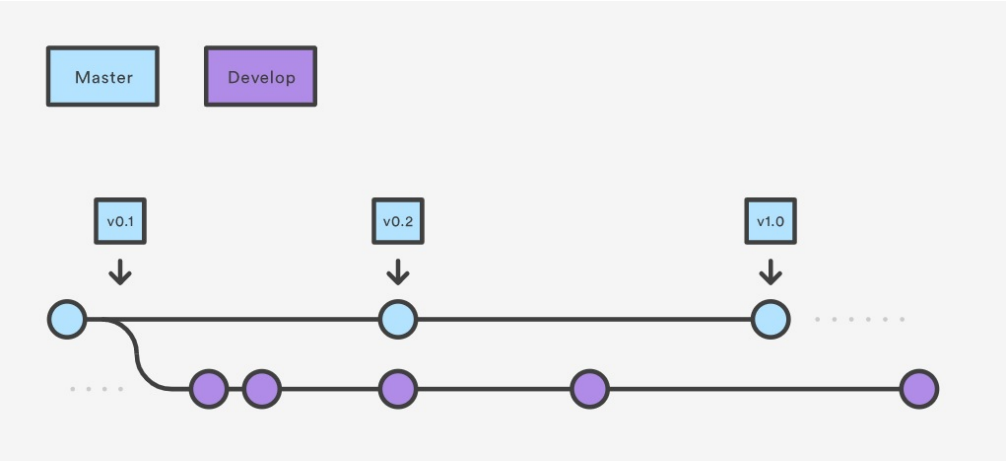
Para el control de versiones en la aplicación, usamos el sitema GIT, utilizando diferentes ramas según el flujo de trabajo de *Gitflow*.

¿Qué es Gitflow y cómo se usa?

El flujo de trabajo de Gitflow es un diseño de flujo de trabajo de Git que define un modelo estricto de ramificación diseñado alrededor de la publicación del proyecto.

Ramas de desarrollo y maestras

En lugar de una única rama maestra, este flujo de trabajo utiliza dos ramas para registrar el historial del proyecto. La rama maestra almacena el historial de publicación oficial y la rama de desarrollo sirve como rama de integración para nuevas funcionalidades. Conviene etiquetar todas las confirmaciones de la rama maestra con un número de versión.

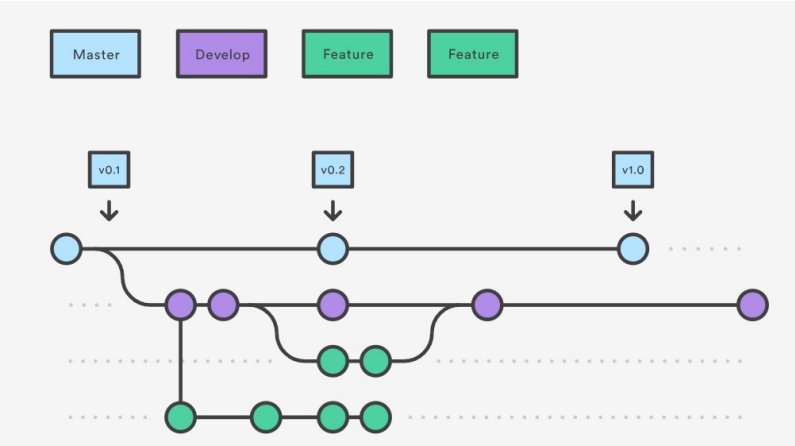


La rama de desarrollo contendrá el historial completo del proyecto, mientras que la maestra contendrá una versión abreviada. Para empezar, los desarrolladores deberían clonar el repositorio central y crear una rama de seguimiento para desarrollo.

Ramas de función

Cada nueva funcionalidad debe estar en su propia rama, que se puede enviar al repositorio central para copia de seguridad/colaboración. Sin embargo, en vez de ramificarse de la maestra, las ramas de función utilizan la de desarrollo como rama primaria. Cuando una función

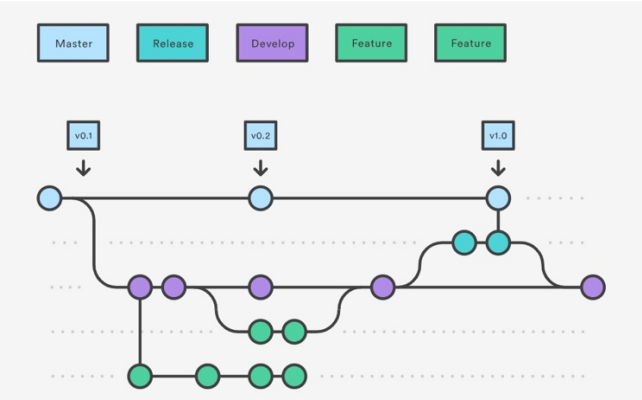
está completa, se vuelve a fusionar en la de desarrollo. Las funciones nunca deben interactuar directamente con la maestra.



Las ramas de función normalmente se crean a partir de la última rama de desarrollo.

Ramas de publicación (release)

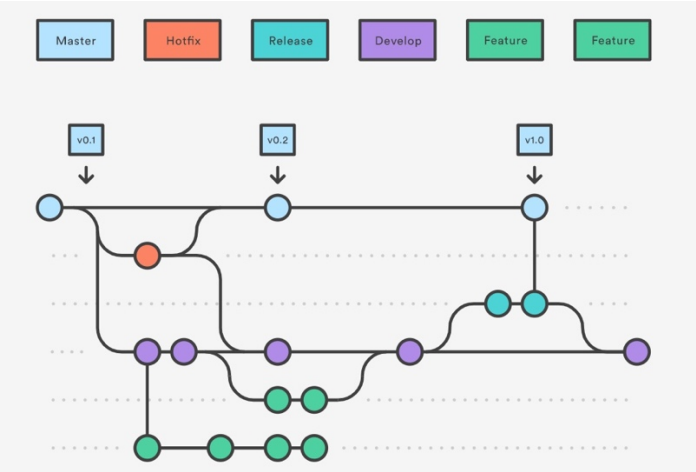
Una vez que el desarrollo ha adquirido suficientes funciones para una publicación (o se está acercando una fecha de publicación predeterminada), bifurcas una rama de versión a partir de una de desarrollo. Al crear esta rama, se inicia el siguiente ciclo de publicación, por lo que no pueden añadirse nuevas funciones tras este punto; solo las soluciones de errores, la generación de documentación y otras tareas orientadas a la publicación deben ir en esta rama. Una vez que esté lista para el lanzamiento, la rama de publicación se fusiona en la maestra y se etiqueta con un número de versión. Además, debería volver a fusionarse en la de desarrollo, que podría haber progresado desde que se inició la publicación.



Utilizar una rama específica para preparar publicaciones hace posible que un equipo perfeccione la publicación actual mientras otro equipo sigue trabajando en las funciones para la siguiente publicación.

Ramas de corrección

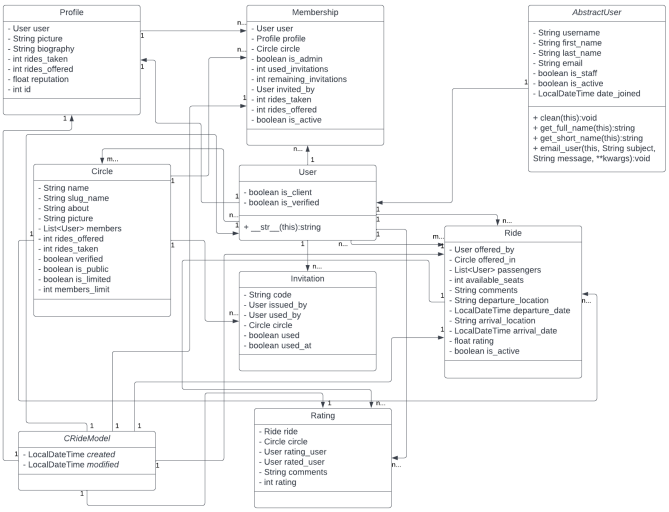
Las ramas de mantenimiento o "corrección" (hotfix) se utilizan para reparar rápidamente las publicaciones de producción. Las ramas de corrección son muy similares a las ramas de publicación y a las de función, salvo porque se basan en la maestra en vez de la de desarrollo. Es la única rama que debería bifurcarse directamente a partir de la maestra. Una vez que la solución esté completa, debería fusionarse en la maestra y la de desarrollo (o la rama de publicación actual), y la maestra debería etiquetarse con un número de versión actualizado.



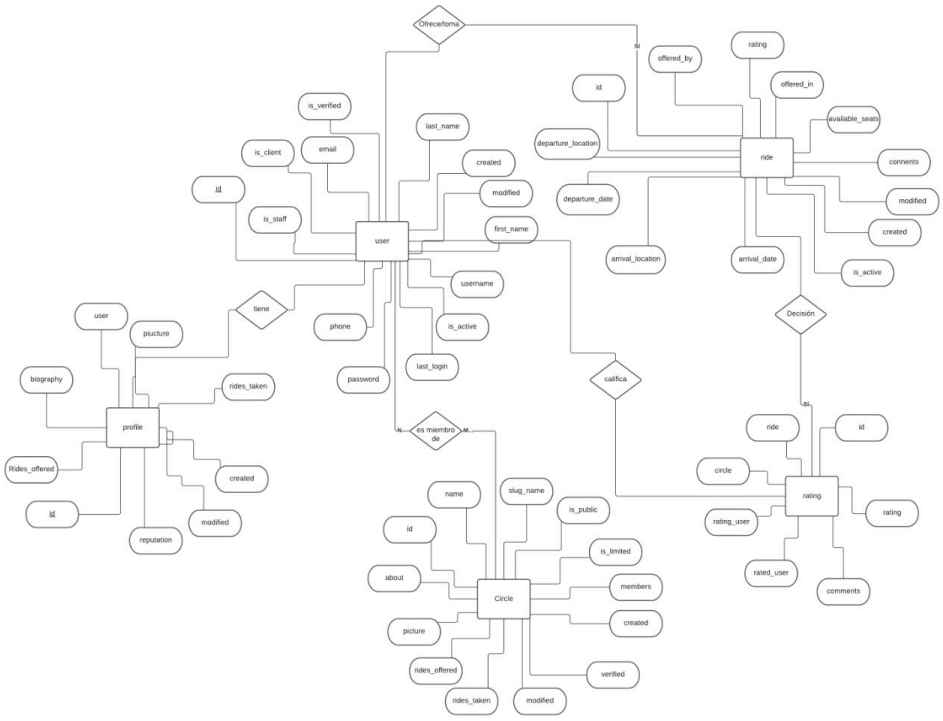
Modelado de la aplicación

1. Diagrama de clases del núcleo de la aplicación.

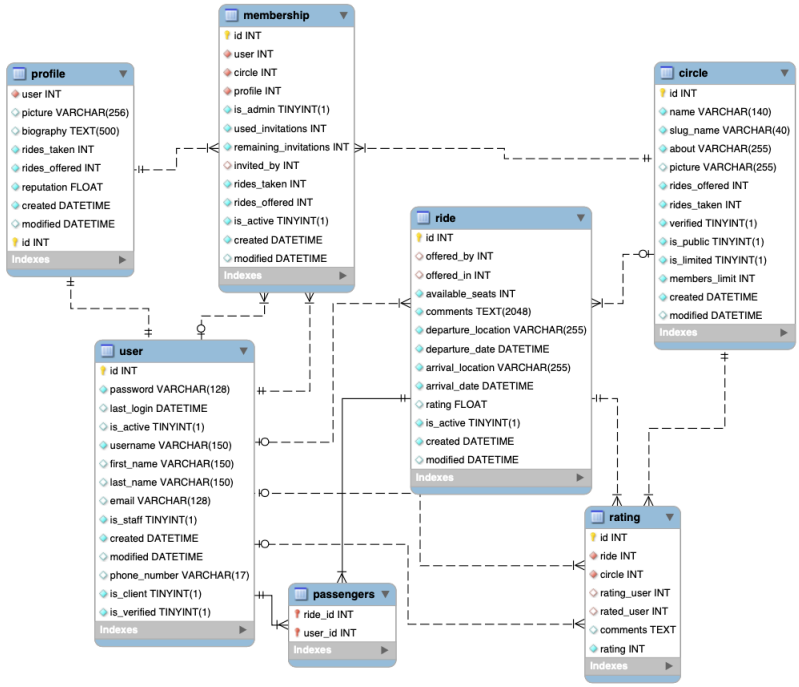
En este diagrama se explica la relación entre las diversas clases que intervienen en el core de la aplicación, las cuales se ven mapeadas en las entidades de base de datos.



2. Modelo E/R y Modelo relacional.



Se evidencia el mapeo entre clases y tablas de base de datos.



2.1 Modelo de usuarios

El modelo de usuarios de Shared Ride se compone de tres sub-modelos, a saber:

Base

- Email
- Username
- Número de teléfono
- Nombre
- Apellido

Perfil

- Usuario (PK)
- Imagen
- Biografía (about me)
- Rides tomados (total)
- Rides ofrecidos (total)

Miembro

- Usuario (PK), Perfil (PK)
- Círculo (PK)
- ¿Es administrador del círculo?
- Invitaciones (usadas/restantes)
- ¿Quién lo invitó?
- Rides tomados (dentro del círculo)
- Rides ofrecidos (dentro del círculo)

2.2 Modelo de círculos

El modelo de círculos de la aplicación comprende la interacción entre los modelos de **Círculo** y de **Invitación**:

Círculo

- Nombre
- Slug name
- Acerca de
- Imagen
- Miembros
- Rides tomados

- Rides ofrecidos
- ¿Es un círculo oficial?
- ¿Es público?
- ¿Tiene límite de miembros? ¿Cuál es?

Invitación

- Código
- Círculo (PK)
- ¿Quién invita?
- ¿Quién la usó?
- ¿Cuándo se usó?

2.3 Modelo de Rides

El modelo de rides comprende las entidades correspondientes a los **Rides** y sus **Calificaciones**:

Ride

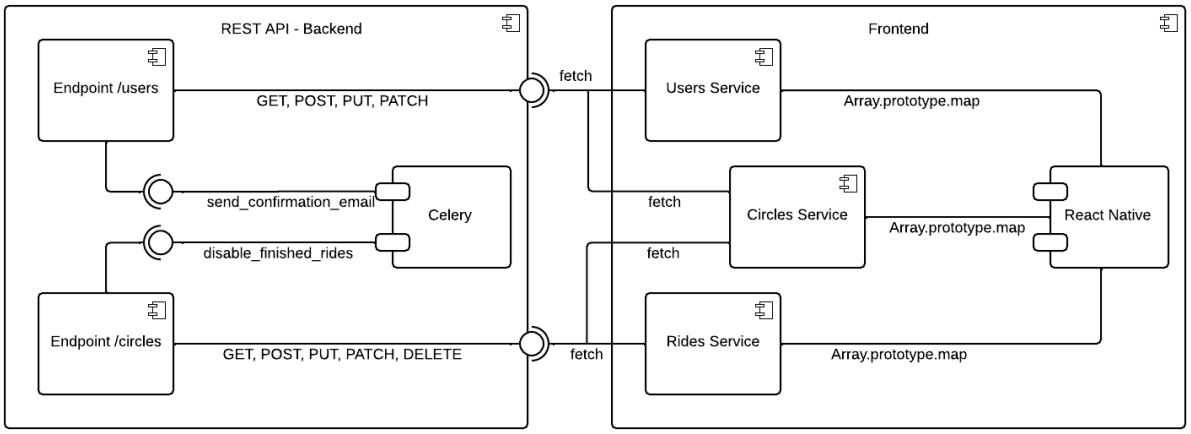
- ¿Quién lo ofreció?
- ¿Dónde se ofreció? (Círculo)
- ¿Cuándo? ¿Dónde?
- Asientos disponibles
- Comentarios adicionales
- Calificación (promedio)
- ¿Sigue activo?

Calificación

- Ride (PK)
- Círculo (PK)
- ¿Quién califica?
- ¿A quién califica?
- Calificación (1-5)

3. Diagrama de componentes.

Se evidencia la interacción entre los diferentes componentes de la aplicación.



4. Diccionario de datos.

PROFILE						
Atributo	Tipo de dato	Longitud	Restriccion	Valor inicial	Nulos	Descripcion
user	INT		FK		NO	id del usuario
picture	VARCHAR	256			SI	Foto del cliente
biography	TEXT	500			SI	Descripción del cliente
rides_taken	INT	11			NO	Viajes tomados
rides_offered	INT				NO	viajes ofrecidos
reputation	FLOAT				NO	reputación del ususario
created	DATETIME				NO	fecha de creación del ususario
modified	DATETIME				SI	fecha de modificación
id	INT		PK		NO	id del perfil

USER						
Atributo	Tipo de dato	Longitud	Restriccion	Valor inicial	Nulos	Descripcion
Id	INT		PK		NO	id del usuario
password	VARCHAR	128			NO	contraseña del usuario
last_login	DATETIME				SI	ultimo inicio de sesión
is_active	TINYINT	1			SI	estado del usuario
username	VARCHAR	150			NO	nombre de usuario
first_name	VARCHAR	150			SI	nombres
last_name	VARCHAR	150			SI	apellidos
email	VARCHAR	128			SI	correo
is_staff	TINYINT	1			NO	para el staff de shired ride
created	DATETIME				NO	fech de creación
modified	DATETIME				SI	fecha de modificación
phone_number	VARCHAR	17			SI	número de telefono
is_client	TINYINT	1			NO	para inidica los clientes
is_verified	TINYINT	1			NO	para validar si el ususario está verificado

MEMBERSHIP						
Atributo	Tipo de dato	Longitud	Restriccion	Valor inicial	Nulos	Descripcion
id	INT		PK		NO	codigo de afiliación
user	INT		FK		NO	usuario afiliado
circle	INT		FK		NO	circulo de afiliación
profile	INT		FK		NO	perfil del usuario
is_admin	TINYINT	1			NO	designado para administradores
used_invitatioins	INT				NO	invitaciones utlizadas
remaining_invitations	INT				NO	invitaciones restantes
invited_by	INT		FK		SI	usuario que recomienda
rides_taken	INT				NO	viajes tomados
rides_offered	INT				NO	viajes ofrecidos
is_active	TINYINT	1			NO	actividad del ususario
create	DATETIME				NO	fecha de creación
modified	DATETIME				SI	fecha de modificación

CIRCLE						
Atributo	Tipo de dato	Longitud	Restriccion	Valor inicial	Nulos	Descripcion
int	INT		PK		NO	id del circulo
name	VARCHAR	140			NO	nombre del circulo
slug_name	VARCHAR	40			NO	string amigable con las URL
about	VARCHAR	255			NO	información del grupo
picture	VARCHAR	255			SI	imagen del grupo
rides_offered	INT				NO	viajes ofrecidos
rides_taken	INT				NO	viajes tomados
verified	TINYINT	1			NO	verificación del circulo
is_public	TINYINT	1			NO	circulo visible u oculto al público
is_limited	TINYINT	1			NO	restricción de cantidad de miembros del circulo
members_limit	INT				NO	limite de miembros
create	DATETIME				NO	fecha de creación
modified	DATETIME				SI	fecha de modificación

PASSENGERS						
Atributo	Tipo de dato	Longitud	Restriccion	Valor inicial	Nulos	Descripcion
ride_id	INT		FK		NO	referencia id del viaje
user_id	INT		FK		NO	referencia al usuario que toma el viaje

RIDE						
Atributo	Tipo de dato	Longitud	Restriccion	Valor inicial	Nulos	Descripcion
id	INT		PK		NO	id del viaje
offered_by	INT		FK		SI	id del usuario que ofrece el viaje
offered_in	INT		FK		SI	id del círculo en el que se realizará el viaje
available_seats	INT				NO	asientos disponibles
comments	TEXT	2048			NO	comentarios del viaje
departure_location	VARCHAR	255			NO	punto de salida
departure_date	DATETIME				NO	hora de salida
arrival_location	VARCHAR	255			NO	punto de destino
arrival_date	DATETIME				NO	hora de llegada
rating	FLOAT				SI	calificación del viaje
is_active	TINYINT	1			NO	indica si el viaje esta activo o inactivo
created	DATETIME				NO	fecha de creación del viaje
modified	DATETIME				SI	fecha de modificación del viaje

RATING						
Atributo	Tipo de dato	Longitud	Restriccion	Valor inicial	Nulos	Descripcion
id	INT		PK		NO	id de la calificación
ride	INT		FK		NO	viaje a calificar
circle	INT		FK		NO	círculo en el que fue ofrecido el viaje
rating_user	INT		FK		SI	usuario que califica
rated_user	INT		FK		SI	usuario calificado
comments	TEXT				SI	comentarios de la calificacion
rating	INT				NO	puntuación del viaje

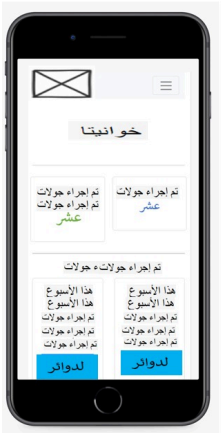
5. Diagrama de Gantt

Nombre de la tarea	Fecha de inicio	Fecha de finalización	Asignado	Estado	17.03.2022	24.03.2022	31.03.2022	07.04.2022	14.04.2022	21.04.2022	28.04.2022	05.05.2022	12.05.2022	19.05.2022
Identificar las variables que intervienen en la situación presentada, así como la estructura en la que se van a manejar dichos datos mediante metodologías y patrones de diseño de software	17.03.2022	31.03.2022	Todos	Abierto										
Definir el lenguaje y tecnologias para la implementación	17.03.2022	24.03.2022	Todos	Abierto										
Establecer la arquitectura general de la aplicación.	17.03.2022	24.03.2022	Todos	Abierto										
Diseñar los modelos de base de datos de la aplicación.	24.03.2022	31.03.2022	Todos	Abierto										
Determinar el diseño y arquitectura de la aplicación	31.03.2022	14.04.2022	Todos	Abierto										
Establecer la arquitectura específica del frontend de la aplicación (frameworks a utilizar, librerías, metodologías, etc.)	31.03.2022	07.04.2022	Todos	Abierto										
Construir los contenedores e imágenes correspondientes a los servicios de la aplicación en ambiente de desarrollo. (Backend, Base de Datos, Cache, Message broker, Frontend).	07.04.2022	14.04.2022												
Implementar el prototipo funcional (MVP) para el desarrollo de la aplicación utilizando las metodologías, frameworks o librerías elegidos para su implementación	14.04.2022	28.04.2022	Todos	Abierto										
Implementar el módulo de usuarios (users) del backend de la aplicación.	14.04.2022	21.04.2022	Todos	Abierto										
Implementar el módulo de círculos (circles) del backend de la aplicación.	14.04.2022	21.04.2022	Todos	Abierto										
Implementar el módulo de viajes (rides) del backend de la aplicación.	21.04.2022	28.04.2022	Todos	Abierto										
Implementar el módulo de tareas asíncronas del backend de la aplicación.	21.04.2022	28.04.2022	Todos	Abierto										
Realizar testing a la aplicación, tanto en ambientes de desarrollo como en ambientes de producción	28.04.2022	12.05.2022	Todos	Abierto										
Realizar pruebas unitarias e integrales sobre los módulos del backend.	28.04.2022	05.05.2022	Todos	Abierto										
Realizar pruebas unitarias e integrales sobre los módulos del frontend.	05.05.2022	12.05.2022	Todos	Abierto										

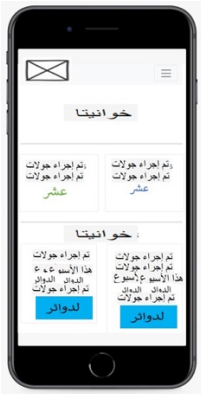
Mockups y Wireframes

inicio

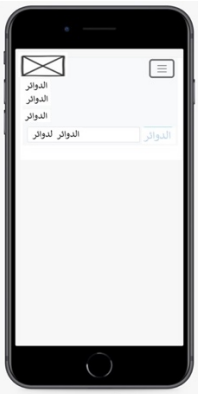
perfil



circulo

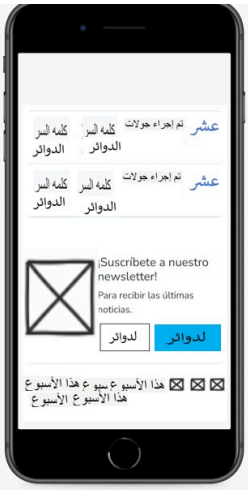


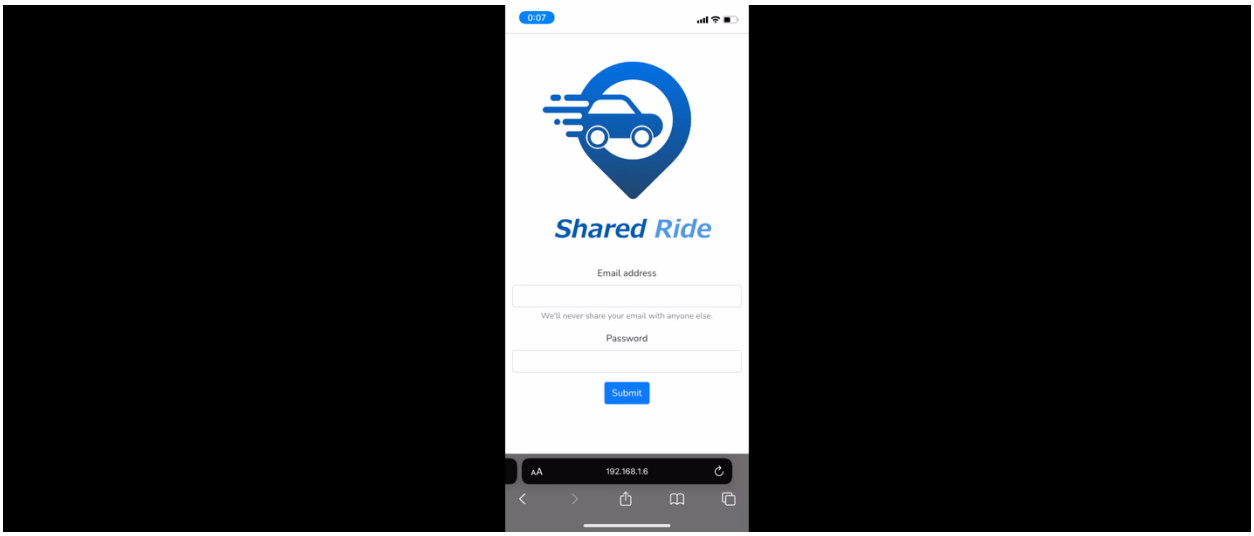
rides

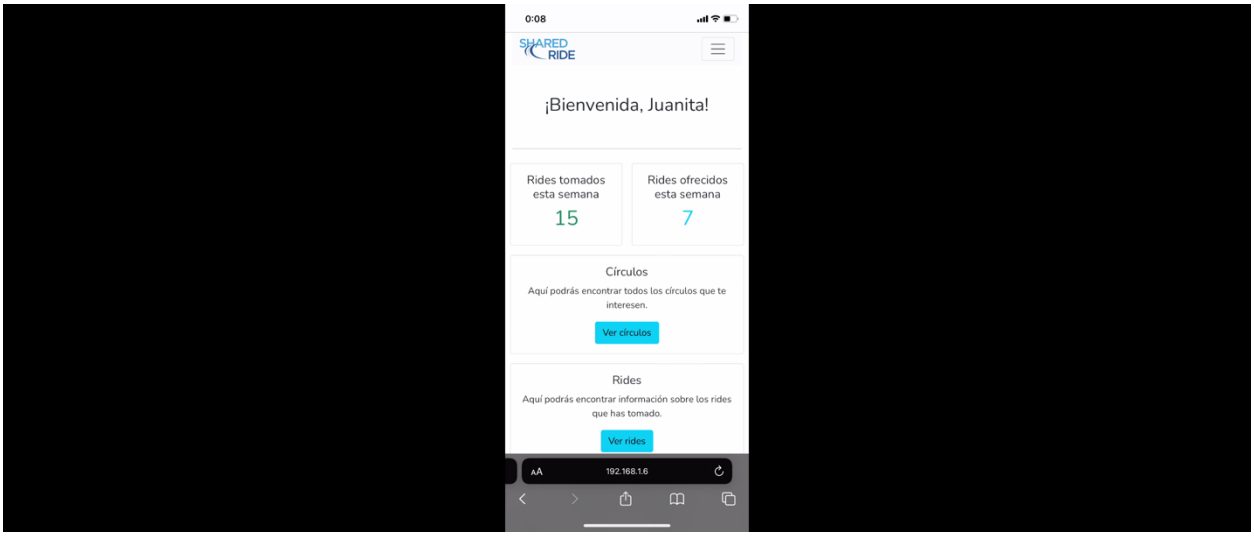


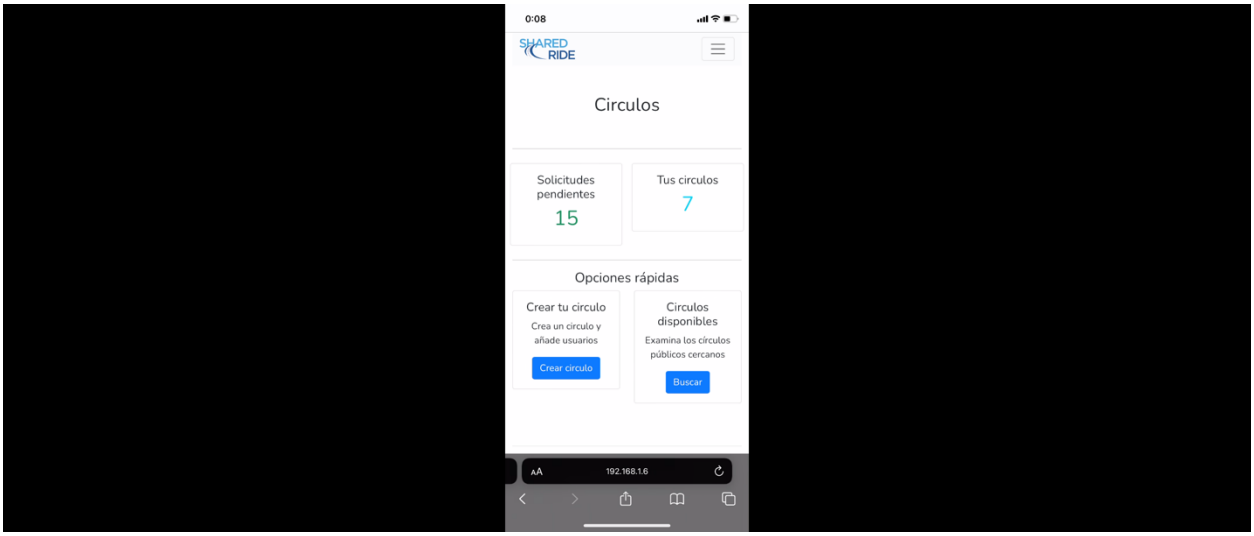
Opciones
de menú

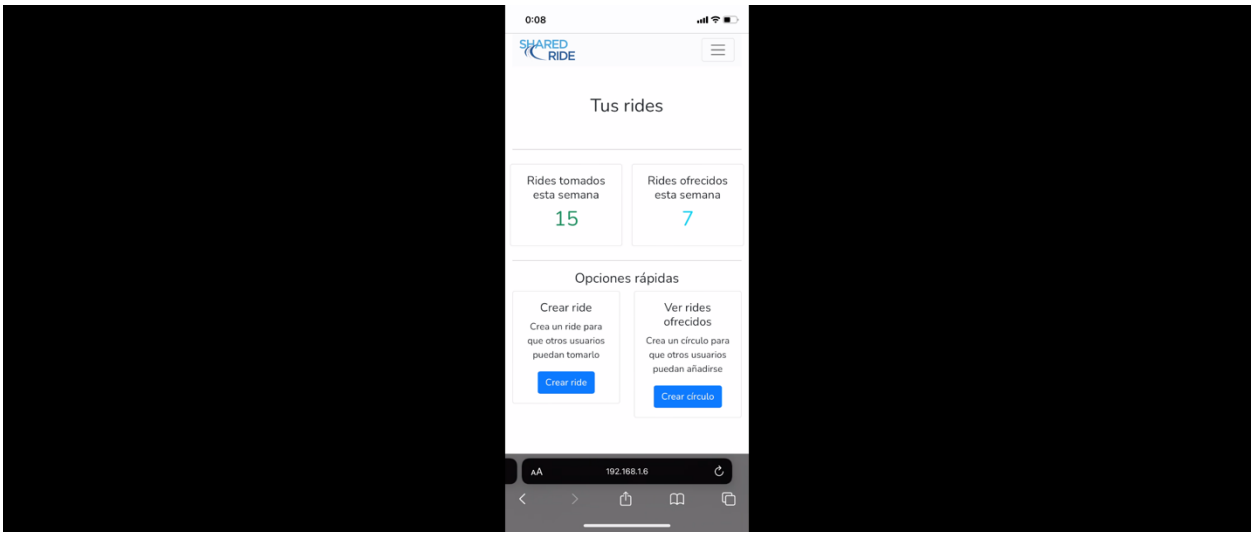
home

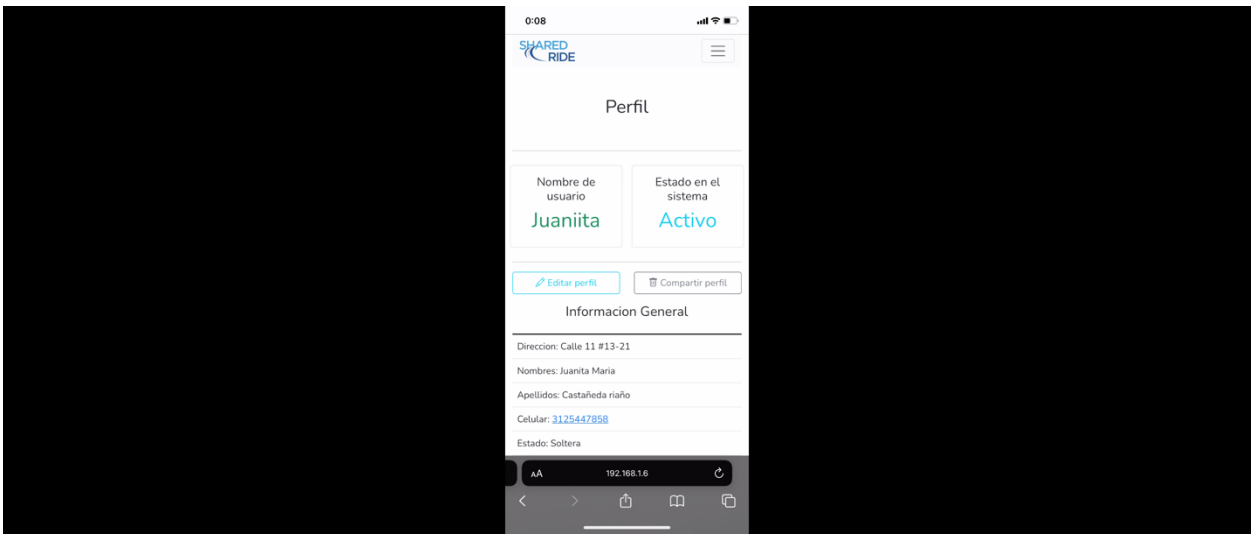


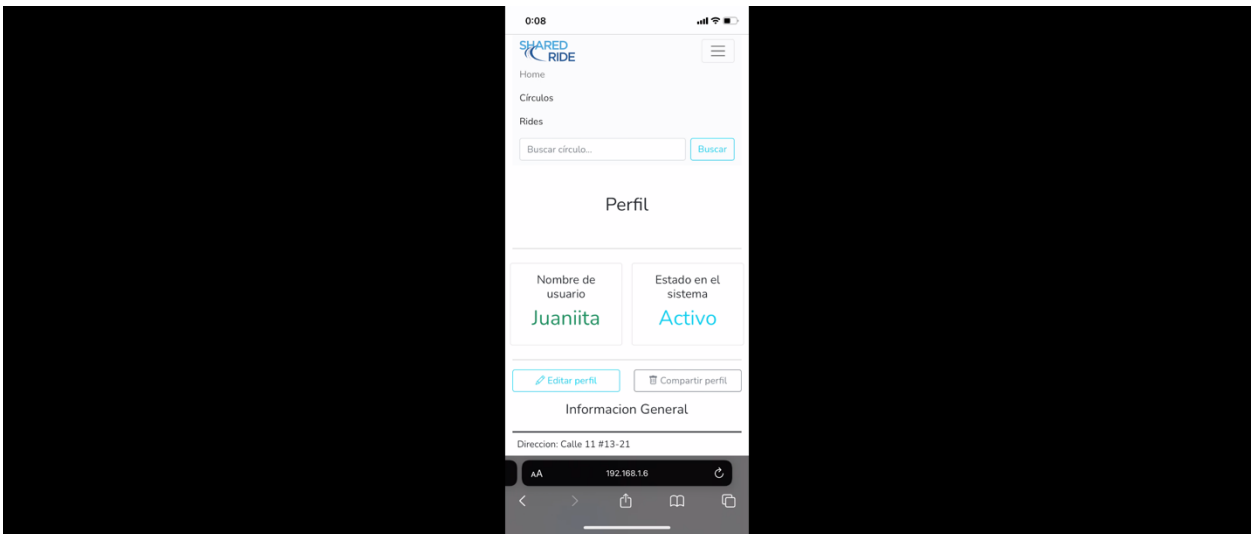

0:07
Shared Ride
Email address
We'll never share your email with anyone else.
Password
Submit
AA
192.168.1.6

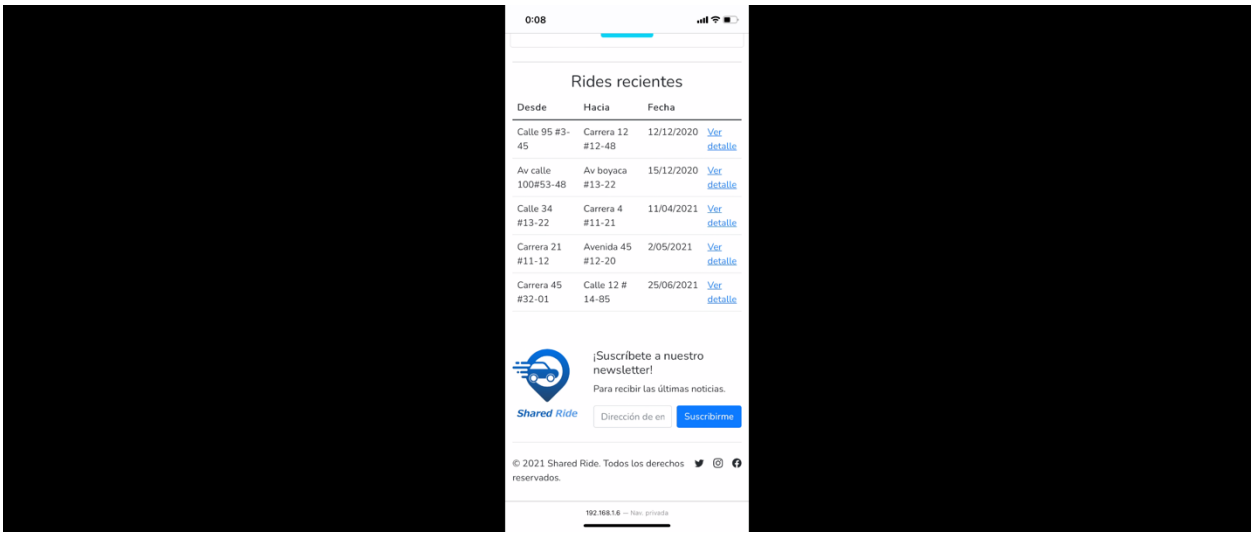

0:08
SHARED RIDE
¡Bienvenida, Juanita!
Rides tomados esta semana
15
Rides ofrecidos esta semana
7
Círculos
Aquí podrás encontrar todos los círculos que te interesen.
Ver círculos
Rides
Aquí podrás encontrar información sobre los rides que has tomado.
Ver rides
AA
192.168.1.6


0:08
SHARED RIDE
Circulos
Solicitudes pendientes
15
Tus circulos
7
Opciones rápidas
Crear tu circulo
Crea un circulo y añade usuarios
Crear circulo
Circulos disponibles
Examina los círculos públicos cercanos
Buscar
AA
192.168.1.6


0:08
SHARED RIDE
Tus rides
Rides tomados esta semana
15
Rides ofrecidos esta semana
7
Opciones rápidas
Crear ride
Crea un ride para que otros usuarios puedan tomarlo
Crear ride
Ver rides ofrecidos
Crea un círculo para que otros usuarios puedan añadirse
Crear círculo
AA
192.168.1.6


0:08
SHARED RIDE
Perfil
Nombre de usuario
Juaniita
Estado en el sistema
Activo
Editar perfil
Compartir perfil
Informacion General
Direccion: Calle 11 #13-21
Nombres: Juanita Maria
Apellidos: Castañeda riaño
Celular: 3125447858
Estado: Soltera
AA
192.168.1.6


0:08
SHARED RIDE
Home
Círculos
Rides
Buscar círculo...
Buscar
Perfil
Nombre de usuario
Juaniita
Estado en el sistema
Activo
Editar perfil
Compartir perfil
Informacion General
Direccion: Calle 11 #13-21
AA
192.168.1.6



Glosario de la aplicación

Terminología general

CRUD (Cread, Read, Update, Delete)

La creación, lectura, actualización y eliminación son las cuatro operaciones básicas del almacenamiento persistente. Respectivamente, son mapeadas con las sentencias INSERT, SELECT, UPDATE y DELETE en SQL.

REST (Representational state transfer)

Definición de arquitectura que enfatiza la escalabilidad de las interacciones entre componentes, las interfaces uniformes, el despliegue independiente de componentes y la creación de arquitecturas que faciliten el cacheo de componentes para reducir la latencia, mejorar la seguridad y encapsular sistemas antiguos.

Consiste en una serie de guías y recomendaciones para la creación de APIs web que sean *stateless*. Es decir, el estado de una transacción no afecta el estado de las demás.

Slug

Una etiqueta corta para un objeto, que contiene solamente números, letras, guiones bajos o guiones. Generalmente se utilizan en URLs. Por ejemplo, el slug de Shared Ride sería shared-ride.

STDOUT

La salida estándar es un concepto propio de los sistemas operativos UNIX. Está constituida por un hilo sobre el cual un programa escribe sus datos de salida. El programa solicita la transferencia de datos con la operación *write*.

Terminología específica

Círculo

Los círculos son comunidades formadas por miembros de la aplicación, las cuales son cerradas (solo se pueden unir miembros nuevos con una invitación de un miembro existente).

Invitación

Los usuarios se unen a los círculos mediante una invitación. Cada usuario tiene un número **limitado** de invitaciones dentro de cada círculo.

Ride

Dentro de los círculos, los miembros pueden **ofrecer** o **sumarse** a un viaje. Solo un número definido de usuarios se pueden sumar a un viaje.

Calificación

Al finalizar el viaje, los usuarios **califican su experiencia**. Cada usuario tiene una **reputación** general, la cual es pública a través de todos sus círculos.