1. Resolucion de la Infraestructura/Arquitectura de Software

Para poder resolver los requerimientos pedido de la aplicación vamos a usar el paradigma de microservicios, en donde vamos a tener las siguientes tecnologias/paradigmas:

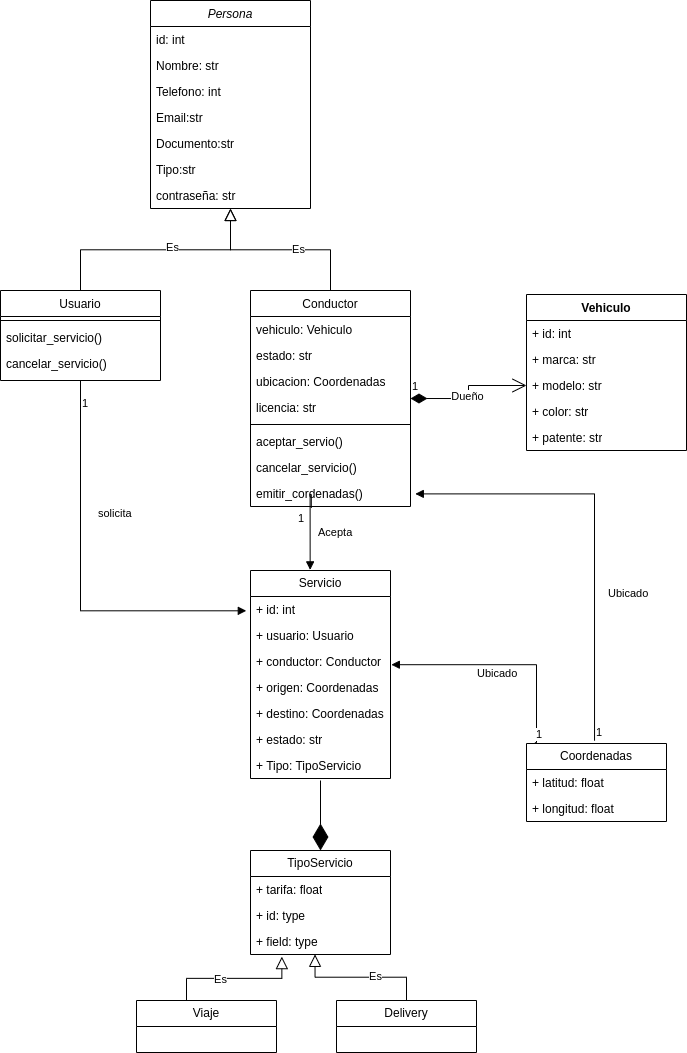
* API RESTful: esta va a ser usada tanto por usuarios como conductores, ya que es un estandard que funciona y es usado con éxito, permitiendo un robusto y rapido desarrollo. Esta API va a gestionar el registro, inicio y cierre de sesion, las solicitudes de viajes, tanto para pedir un viaje (usuario), como para aceptar uno(conductor).
* Backend: va a ser el que sea mas comodo para el equipo de desarrollo, y puede ser django si es python o Express/Node.js. En cualquiera de los lenguajes va a ser un desarrollo parecido y las entidades del problema van a ser las mismas
* Base de Datos: esta va PostgreSql, una base de datos relacional. La eleccion de la misma es porque tiene bastante soporte para trabajar con datos de posicionanmiento global, cuenta con librerias como postgis y H3, que permiten trabajar con geometrias en espacios de dos dimensiones y tres dimensiones (postgis), H3 divide el espacio en hexagonos, en este caso un mapa, permitiendo asi realizar calculos eficientes de distancia de una sesion de usuario por ejemplo.
* WebSockets: esta tecnologia nos permite tener un seguimiento en tiempo real de usuarios y conductores, permitiendo asi no tener un delay en la posicion de los conductores/usuarios.
* Redis: esta se utilizara como una base de datos en memoria, para poder mantener la posicion tanto de usuarios como de conductores y poder actualizar el estado activo/inactivo de los conductores.
* Geolocalizacion: se puede usar una librería como Google Maps API, o desarrollar algun modulo que permita tomar la ubicación del celular de usuario/conductor.

Clases del Dominio:

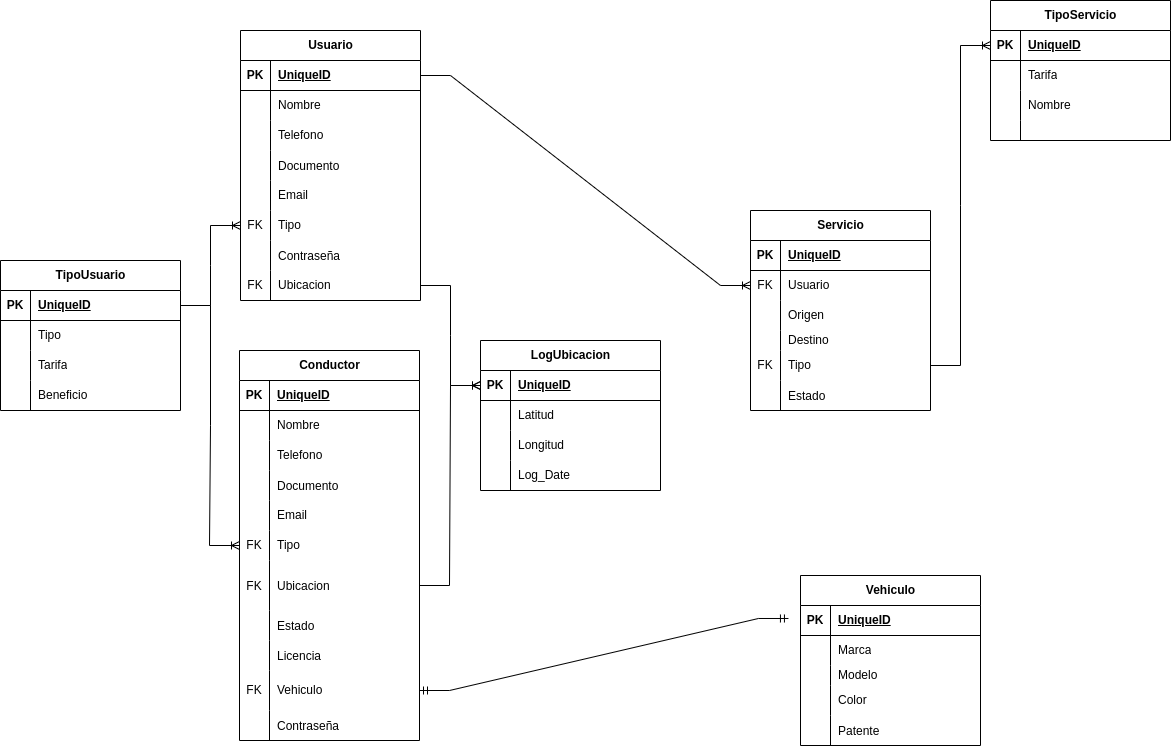
Las principales clases del dominio son:

* Usuario: esta representa a un usuario que solicita servicios de trasportes. Solicita y cancela servicios.
* Conductor: esta representar al conductor que acepta, cancela y emite sus coordenadas. Este se diferencia del usuario, ya que debe tener un vehiculo y una licencia para poder ofrecer servicios.
* Vehiculo: esta representa al vehiculo que posee el conductor. Tiene las variables principales que lo definen. Es necesaria para poder hacer la verificaciones de seguridad que necesitemos, asi como poder ofrecerle al usuario la patente, el modelo y el color del mismo, para poder identificarlo.
* Servicio: esta representa el servicio que involucra tanto al usuario y al conductor. Contiene la principal informacion sobre el mismo: usuario, conductor, origen y destino, estado y el tipo de servicio que se esta ofreciendo
* Coordenadas: Esta representa la ubicación en latitud y longitud.

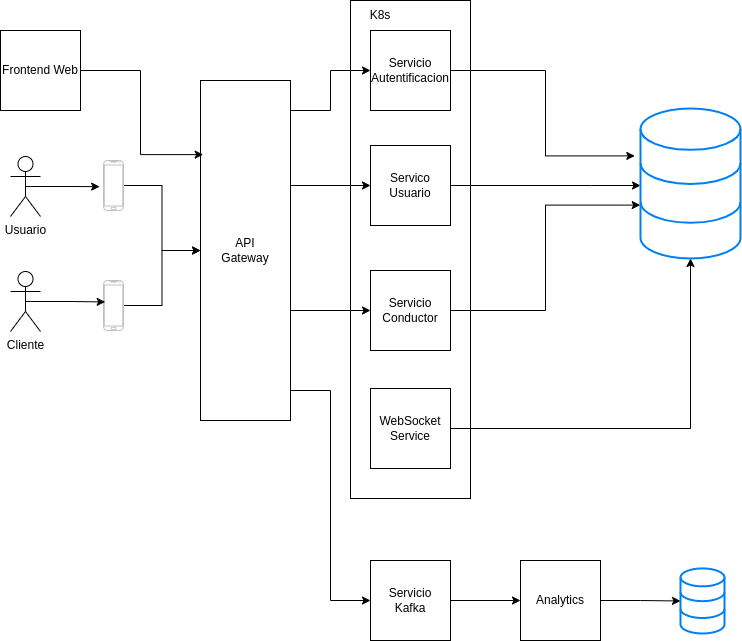
A continuacion esta el Diagrama de Clases:



Digrama de Bases de Datos:



Al principio hablabamos de el paradigma de microservicios, esto esta mas orientado a la infraestructura, que al desarrollo de software visto anteriormente, a continuacion podemos ver el diseño de la misma:



En este diagama podemos ver que tenemos 4 solicitantes a nuestro api gateway, depende de quien este solicitando y que se esta solicitando, lo va a redirigir a alguno se los microservicios conectados la API gateway. Las apps solicitantes son, las apps mobile, tanto del usuario como del cliente, nuestro frontend web, el cual es solo para usuarios y nuestro servicio de ubicación en tiempo real con websockets, el cual permite actualizar la ubicación de los autos en tiempo real. Ya pasados el API gateway tenemos nuestros servicos de autentificacion, servico de usuario y servicio de conductor, cada uno encargado de proveer soluciones distintas para nuestra empresa. Por ultimo tenemos un servicio de Kafka, que nos permite trabajar bajo el concepto de event-driven, en este caso para llevar adelanta los procesos de analisis de demanda, precio dinamico y estadisticas en tiempo real. Esta es una aproximacion de lo que podria ser nuestra estructura.

2. Problemas de concurrencia y doble asignaciòn

Para evitar un conflictos en la asignacion de conductores al pedido de servicio emitido por el cliente, lo resolveria usando un semaforo, es necesario implementar un sistema de bloqueo en el proceso de aceptacion de servicios. A continuacion detallo una posible aplicación:

1. Hay que introducir un semaforo en el microservicio de conductores. Este se utilizara para controlar el acceso a los metodos de asignacion de servicios.
2. Al momento en el que un conductor quiere tomar un servicio, en primera medida debe adquirir el semaforo. Este evitaraque otros conductores tomen el servicio.
3. Una vez adquirido el semaforo por el conductor, verifica si el servicio se asigno a otro conductor. Si el servicio ya habia sido adquirido por otro conductor, este no puede aceptar el servicio y libera el semaforo:
4. Si el servicio no fue tomado, el conductor acepta el mismo y libera el semaforo

A continuacion un pequeño ejemplo, a modo ilustrativo:

import threading

# Crear un semáforo con un valor máximo de 1 (mutex)

service\_assignment\_semaphore = threading.Semaphore(1)

def accept\_service(conductor, service):

# Intentar adquirir el semáforo

service\_assignment\_semaphore.acquire()

# Verificar si el servicio ya ha sido asignado a otro conductor

if service.conductor is None:

# Asignar el servicio al conductor y actualizar el estado del servicio

service.conductor = conductor

service.estado = "aceptado"

print(f"Servicio {service.id} asignado al conductor {conductor.id}")

else:

print(f"El servicio {service.id} ya fue asignado al conductor {service.conductor.id}")

# Liberar el semáforo

service\_assignment\_semaphore.release()

3. Respuesta modulo web, ubicación en tiempo real y estado  
  
Para tener un modulo web que permita mostrar la ubicación y el estado de los vehiculos en tiempo real se podria utilizar la siguiente logica:

1. Podemos utilizar para la parte grafica la librería leaflet, OpenLayers o Google Maps API para poder renderizar un mapa e iconos en javascript.
2. Este microservicio tiene que estar conectado al servicio de websocket, para poder recibir en tiempo real, las actualizaciones de ubicación y estado.
3. El microservico de websocket tiene un temporizador, este permite actualizar cada un minuto la ubicación del conductor. Si no se reciben actualizaciones dentro del minuto, el conductor se considera inactivo y se envia la notificacion al microservicio del mapa para actualizar el estado del conductor.