Prueba de ingreso al puesto Backend Developer (Ikatech Solutions)

Suponga que se desea construir un sistema de transporte similar a Uber donde existen dos

perfiles de usuarios, uno serán los usuarios que solicitan servicios y otros los conductores que recibirán y atenderán estos servicios, en este sistema se desea manejar dos aplicaciones distintas para soportar los dos perfiles que se plantearon anteriormente. El siguiente debe estar conformado bajo las siguientes directrices

1. Los usuarios podrán registrarse, loguearse y cerrar sesión

2. Los usuarios podrán solicitar servicios de transporte

3. Los usuarios conductores estarán emitiendo constantemente las coordenadas de su posición actual

4. Los servicios solicitados por usuarios solo le llegaran a los usuarios conductores que estén dentro de un rango en kms.

5. Cuando un usuario conductor acepte un servicio, en la aplicación de usuario debe mostrar un seguimiento en tiempo real del usuario conductor, para saber por dónde se dirige.

6. Cuando un usuario conductor acepte un servicio este no podrá recibir más servicios.

7. Cuando el usuario cancele el servicio este debe notificarle al usuario conductor la

cancelación del servicio y dejarlo disponible para recibir nuevos servicios

8. Cuando el usuario conductor finalice un servicio se le debe notificar el término del servicio a la app de usuario

Dadas estas directrices por favor plantear un esquema lógico y una arquitectura de software para solucionar a nivel de backend este proyecto, se debe especificar cada una de las herramientas tecnológicas utilizadas para llegar a la solución y la explicación del porqué el uso de estas herramientas.

**Solución Esquema lógico del backend**

App Usuario (Cliente)

App Conductor (Driver)

API Gateway

REST A/ WebSocket

Auth Svc

User Svc

Driver Svc

Trip Svc

Notification Svc

Location Svc

**WebSocket / RabbitMQ** **(mensajería)**

PostgreSQL

**JWT + OAuth2 + Firebase Auth**

**Node.js o Python**

**PostgreSQL con extensión PostGIS**

Redis

El proyecto debido a su utilidad presentará una alta concurrencia de peticiones, se puede presentar el caso en el que un usuario solicite un servicio y dos conductores acepten el servicio al mismo tiempo, lo cual desemboca en una doble asignación hacia un solo servicio, ¿Como se puede evitar este tipo de problemas, para garantizar que un servicio sólo podrá ser atendido por un solo conductor? (Ayuda: revisar el tema de condición de carrera).

**Solución**

Conductor A

Conductor B

RabbitMQ Queue

REST A/ WebSocket

Worker: Procesar Aceptación

Cola: aceptaciones\_servicio

Ambos publican mensaje de aceptación

PostgreSQL

viajes, estado

- Conductor (estado: asignado)

- Usuario (nombre y ubicación del conductor)

Conductor N

Suponga que se requiere crear un tercer módulo web donde solo se quiere conocer las ubicaciones en tiempo real de los usuarios conductores sobre un mapa; en este mapa se desea manejar dos iconos con dos estados para los usuarios conductores:

* Activo: Donde el conductor está emitiendo coordenadas constantemente
* Inactivo: Donde el conductor ha dejado de emitir coordenadas después de 1 minuto

Cómo puede usted plantear una solución para poder mostrar el estado inactivo en este módulo.

**Solución**:

Conductor App

Location Service

WebSocket / HTTP

Redis

PostgreSQL

Guardar lat/lng/timestamp

Guardar histórico

Verificación de estado



[Esta foto](https://www.pngall.com/es/clock-png) de Autor desconocido está bajo licencia [CC BY-NC](https://creativecommons.org/licenses/by-nc/3.0/)

Actualizar estado

API REST para Web

Devuelve ubicaciones y estado de conductor

Actualiza el estado a “inactivo” si no se tiene respuesta

Cron Job o microservicio

| **Componente** | **Tecnología** | **Descripción** |
| --- | --- | --- |
| **API Gateway / Backend** | **Node.js** | Framework rápido, modular y popular para construir APIs REST; ideal para microservicios. |
| **Autenticación** | **JWT + OAuth2 + Firebase Auth** | Manejo seguro de login/registro/cierre de sesión para usuarios y conductores. |
| **User & Driver Services** | **Node.js o Python** | Separar responsabilidades por roles mejora escalabilidad y mantenimiento. |
| **Trip Management Service** | **Node.js con WebSocket o Socket.IO** | Para asignación de servicios, seguimiento de estado y control de flujo del viaje. |
| **Location Service** | **PostgreSQL con extensión PostGIS** | Base de datos relacional con soporte para consultas geoespaciales (distancias, cercanías, etc). |
| **Mensajería interna** | **RabbitMQ** | Asegura la comunicación asincrónica entre servicios, especialmente eventos como aceptación o cancelación. |
| **Cache de sesión y estado** | **Redis** | Rápido y eficiente para almacenar datos volátiles como estados de disponibilidad y ubicaciones en tiempo real. |
| **Notificaciones Push** | **Firebase Cloud Messaging (FCM)** | Servicio gratuito para enviar notificaciones a las apps móviles. |
| **Base de datos** | **PostgreSQL + Redis** | PostgreSQL para datos persistentes y relaciones; Redis para información efímera (por ejemplo, si un conductor está activo). |