RIDE ME

https://drive.google.com/drive/folders/1npFrGl2MNT-c_XZhsRn3R8tldOdpOxjA?uuspdrive_link

Modelo de datos (30 puntos)

Dado el contexto anterior:
1. (5 puntos) Enumere las entidades identificadas y mencione, de forma concisa, qué representa cada una de ellas.

- Ciclista: Personas que alquilan bicicletas en el sistema.
    - Medio de contacto: Puede tener uno o más. Indica el tipo (mail, teléfono, etc).

- Bicicleta: Son alquiladas por los viajeros. Se lleva trazabilidad de sus viajes y estaciones.
    - Estado: Las bicicletas poseen diferentes estados (disponible, no disponible, en reparación, en uso, etc).
    - Reparación: Indica cuándo se hizo y qué cosas se arreglaron.

- Viaje: Indica cada uso de las bicicletas por los ciclistas. Tiene un costo y tiempo determinado.
    - Tarifa: Gestionadas por los administradores. Se indica el tipo (básica, por km, especial, etc.) y el costo. Un viaje puede tener más de una tarifa (Ej. el extranjero paga la básica + tarifa por Km).

- Estación: Son los puntos donde se dejan y retiran las bicicletas.

- Administrador: Encargados de gestionar el sistema y fijar las tarifas.
    - Comuna: Localidades de la ciudad gestionadas por los administradores.

- Reporte: Las personas pueden indicar fallos en las bicicletas.
    - Motivo: Existen motivos predefinidos por el sistema
    - Imagen: Es opcional subir fotos de la bicicleta.

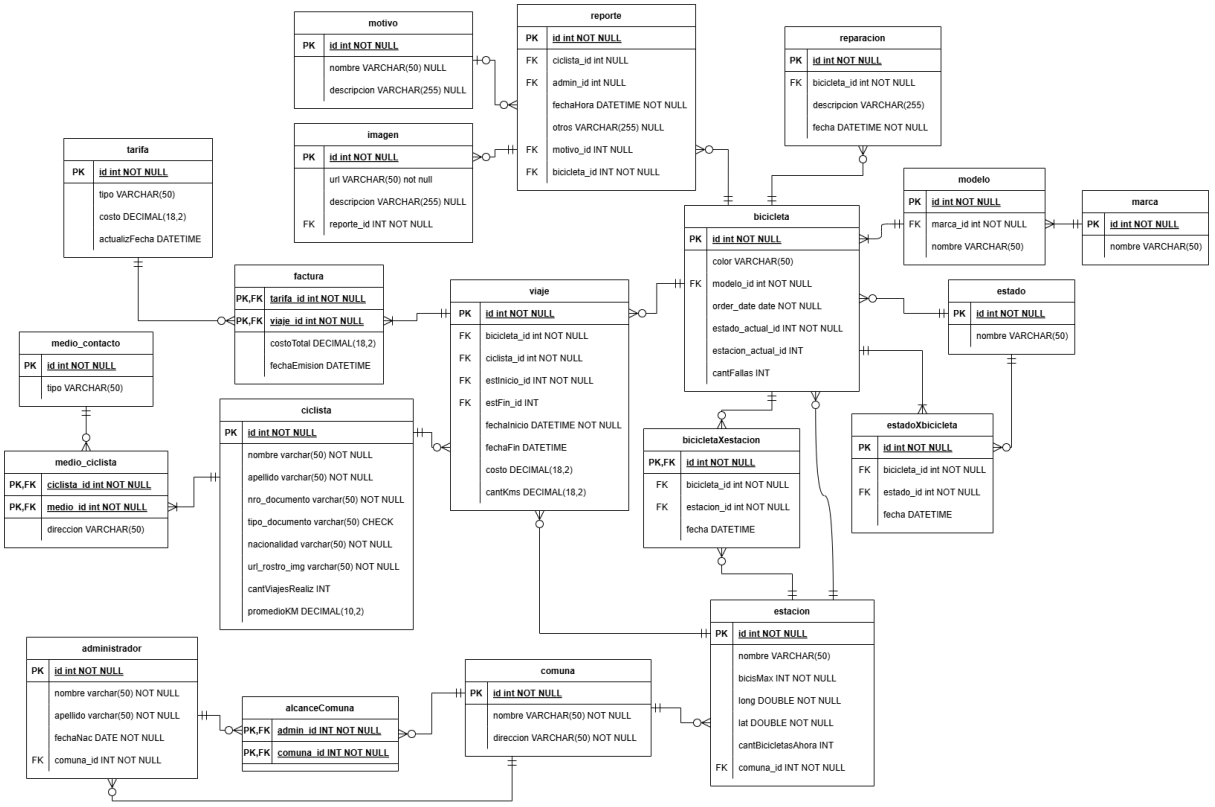
2. (25 puntos) Realice el modelo de datos del dominio presentado. Indique todos los supuestos que crea necesario considerar. Además, resulta de suma importancia que justifique todas las decisiones de diseño que tomó.

Se normaliza el Medio de Contacto para dar extensibilidad a futuro. Lo mismo con los estados de las bicicletas, marca y modelo, motivos de reclamo, tarifas y comunas.

EstadoActual, EstacionActual, CantBicicletasAhora son campos desnormalizados para mejorar la performance al evitar JOINs costosos.

El Costo en el viaje se desnormaliza por consistencia de datos.

https://drive.google.com/file/d/1-80ARL6LvDVE5YDIT5rV4v0iEs8BqRR2/view?usp=sharing



Arquitectura (40 puntos)

1. (15 puntos) Entendiendo que nuestro Sistema debe poseer una capa de presentación gráfica, ¿qué tipo de interfaz propondría implementar para los ciclistas? Justifique adecuadamente su respuesta entendiendo que se poseen 4 meses en total para lanzar el Sistema a producción.

a. Aplicación mobile nativa.
b. Aplicación mobile híbrida.
c. Cliente liviano.
d. Cliente liviano desacoplado de la lógica de negocio.

Debido al poco tiempo de desarrollo, se descartan las opciones nativas, ya que implican desarrollarlo para cada SO (Android e IOS).

Propondría una interfaz **mobile híbrida cliente pesado**.

Usabilidad: Nuestros usuarios van a usar esta aplicación en exteriores (ciclismo), por lo que debemos ofrecerles un sistema al que puedan acceder desde sus celulares fácilmente. Al instalarse como cliente pesado puede acceder a funcionalidades móviles nativas mediante plugins, ofreciendo geolocalización, sensores, notificaciones y almacenamiento local, por lo que la experiencia es equivalente a una app nativa.

Disponibilidad: El cliente pesado es más resistente a fallas del servidor, ya que parte del procesamiento y renderizado se realiza en el dispositivo del usuario. Además, puede ofrecer capacidades de funcionamiento parcial incluso sin conexión, permitiendo que ciertas funciones sigan operativas aun cuando no haya conectividad.

Mantenibilidad: La solución híbrida permite que un único proyecto se despliegue en múltiples sistemas operativos móviles, minimizando el esfuerzo de desarrollo y mantenimiento. Las actualizaciones se realizan una sola vez y se reflejan en todas las plataformas, además de ofrecer tiempos de compilación generalmente más rápidos que en el desarrollo nativo tradicional.

2. (5 puntos) ¿Mantendría la elección de la capa de presentación visual del punto anterior para todas las tareas que tienen que realizar los administradores (empleados del Gobierno de la Ciudad)? Justifique su respuesta adecuadamente y genere otra propuesta en caso negativo.

Debido al poco tiempo de desarrollo, no considero que la usabilidad del lado del Administrador sea fundamental. El nivel de tareas que tienen es muy básico: Actualizar tarifas cada tanto y gestionar reclamos.

Propondría un Cliente Liviano, simple de desarrollar y que se use desde la PC en una oficina. No requiere instalar nada y las actualizaciones se hacen solo en el servidor y están disponibles instantáneamente para el usuario.

3. (10 puntos) Teniendo en cuenta que es necesario que no se demore más de 20 milisegundos en el proceso de reconocimiento facial de un ciclista, y que los procesos de fichadas de retiro y devolución no deben involucrar más de 45 segundos en total; mencione de qué forma abordaría estos requerimientos, justificando adecuadamente y detallando el mecanismo completo, además de considerar que las estaciones podrían sufrir cortes de conexión a internet en ciertos casos.

El reconocimiento facial debe ejecutarse localmente en la estación, sin depender del servidor central. Cada estación mantiene una copia local de la base de usuarios habilitados. Esto reduce la latencia a milisegundos y evita fallas por falta de conectividad.

No permite que usuarios sin registrar accedan al sistema. Cada tótem de registro debe estar sincronizado a un servidor central, para que cada cierto tiempo actualice la información de los usuarios registrados.

Mecanismo:

- Cada estación tiene un módulo local de inferencia (pequeño motor de IA o modelo comprimido).
- El totem recibe la imagen y realiza la verificación facial en el dispositivo.
- Sólo envía al backend un “evento de fichada”, no la imagen.
- Esto asegura 20 ms porque el procesamiento es edge computing.

Para las fichadas, se debe implementar un mecanismo asíncrono. Esto permite que el cliente no se quede esperando respuestas del servidor, el cual podría procesarlo luego.

Mecanismo propuesto:

- Se utiliza una cola de mensajes (RabbitMQ, Kafka o SQS).
- El totem actúa como productor:
    1. Ejecuta localmente el reconocimiento facial.
    2. Valida la bicicleta.
    3. Genera un mensaje con el evento (retiro/devolución + datos del usuario + timestamp).
    4. El usuario continúa su operación sin esperar respuesta del servidor.

Los mensajes quedan en una cola local si no hay internet, y luego se sincronizan en cuanto el enlace vuelva a estar disponible.

4. (10 puntos) Para poder realizar el cobro por uso del Servicio, a los ciclistas, debemos integrarnos con RiuPagos. Este Sistema ofrece la posibilidad de integración sincrónica, mediante llamada a una API REST, o asincrónica mediante el mecanismo de WebHook. Considerando que los cobros hay que efectuarlos de forma mensual, ¿qué mecanismo escogería? ¿Por qué? Justifique adecuadamente su respuesta.

Implementaría el siguiente mecanismo:

1. Implementar un CronJob que barra las facturas aún pendientes de generar.
2. Mediante HTTP REST, generar los cobros en RiuPagos. Adjuntarles el endpoint de nuestro Webhook.
3. Actualizar las facturas con la fecha actual de emisión y estado PENDIENTE.
4. El cliente es notificado y paga.
5. Cuando se recibe el pago, RiuPagos nos notifica a través del WebHook el depósito del dinero correctamente.
6. Se marca en nuestro sistema la factura como PAGADA.

Esto permite combinar un sistema síncrono, dónde nos aseguramos de cada mes crear la orden de pago, con un sistema push based asíncrono, dónde no tenemos que consultar constantemente al sistema si el pago ya llegó.

Brinda una consistencia fuerte, porque al generar la orden recibimos respuesta inmediata del sistema externo para saber si se creó correctamente. Luego, el webhook ahorra performance y evita consultas constantes.

Persistencia (20 puntos)

```
@Entity @Table(name="deportista")
class Deportista {
  @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer id;

  @Column(name = "altura")
  private Double altura;

  @Column(name = "apellido")
  private String apellido;

  @Column(name = "pesoInicial")
  private Double pesoInicial;

  @Column(name = "nombre")
  private String nombre;

  @ElementCollection
  @CollectionTable(name = "deportista_contactos",
        joinColumns= @JoinColumn(name=”deportista_id”))
  @Column(name = "contacto")
  private List<String> contactos;

  @Convert(converter = MotivacionConverter.class)
  @Column(name = "motivacion")
  private Motivacion motivacion;
}

@Entity @Table(name="rutina")
class Rutina {
  @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer id;

  @ManyToOne @JoinColumn(name = "deportista_id", referencedColumnName=”id”)
  private Deportista deportista;

 @ManyToMany @JoinTable(name = "rutina_dia",
        joinColumns= @JoinColumn(name=”rutina_id”,
        referencedColumnName=”id”),
        inverseJoinColumns= @JoinColumn(name=”dia_entrenamiento_id”,
        referencedColumnName=”id”)
  private List<DiaEntrenamiento> dias;
}
```

```
@Entity @Table(name="dia_entrenamiento")
class DiaEntrenamiento {
  @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer id;

  @Column(name = "numero")
  private Integer numero;

   @ManyToMany @JoinTable(name = "dia_ejercicio",
         joinColumns= @JoinColumn(name=”dia_entrenamiento_id”)
         referencedColumnName=”id”,
         inverseJoinColumns= @JoinColumn(name=”ejercicio_id”)
         referencedColumnName=”id”)
  private List<Ejercicio> ejercicios;
}

@Entity @Table(name="ejercicio")
class Ejercicio {
  @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer id;

  @Column(name = "detalle")
  private String detalle;

  @Column(name = "nombre")
  private String nombre;

   @ManyToMany
   private List<Ejercicio> ejerciciosQueCombinan;
}

@Converter
public class MotivacionConverter implements AttributeConverter<Motivacion,String>{
  . . .
}
```