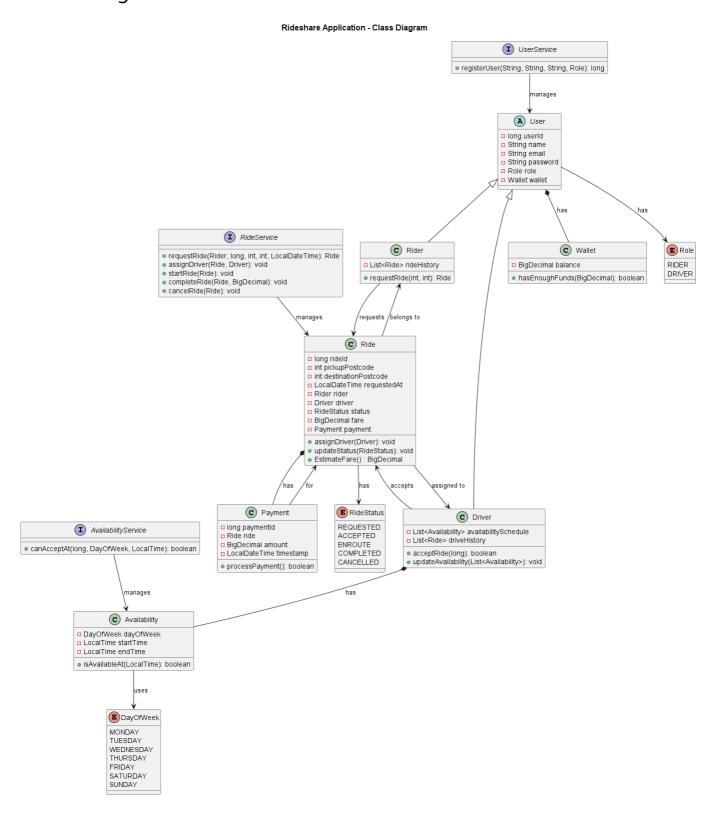
# Part 2 Report

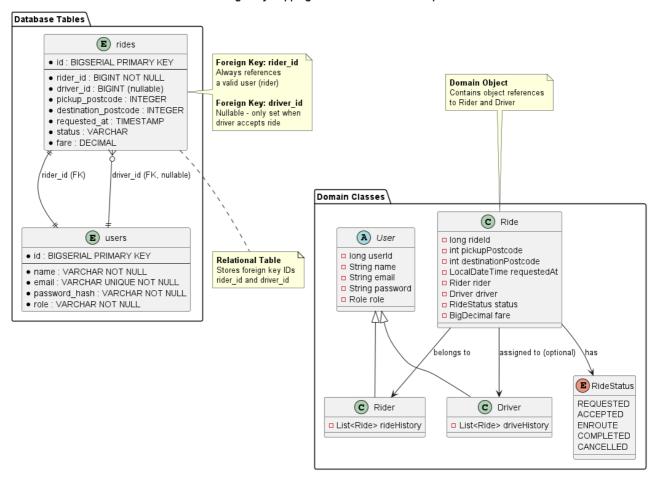
## 1 Class Diagram



# 2 Foreign Key Mapping

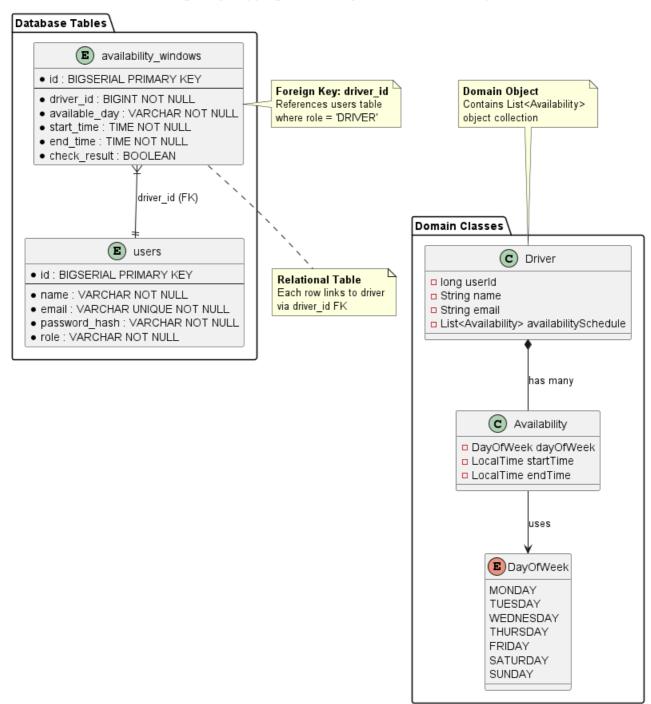
#### 1. Ride to User Relationship

#### Foreign Key Mapping - Ride to User Relationship



#### 2. Availability to Driver Relationship

#### Foreign Key Mapping - Availability to Driver Relationship



### 3 Inheritance

#### Inheritance Pattern - User Hierarchy Implementation Pattern Implementation: Single Table Inheritance pattern Abstract User class with concrete subclasses Abstract Base Class Defines common attributes Role discriminator column in database Common attributes stored in users table and behavior for all users Subclass-specific attributes handled separately Domain Classes (Object-Oriented) A User □ long userId □ String name String emailString password □ Role role ■ Wallet wallet extends extends C Driver C Rider **E** Role Concrete Subclass Specializes User for Concrete Subclass Specializes User for □ List<Availability> availabilitySchedule □ List<Ride> driveHistory List<Ride> rideHistory RIDER mapped to single table ride requesting behavior role = 'RIDER' ride providing behavior role = 'DRIVER' DRIVER requestRide(int, int): Ride acceptRide(long): boolean updateAvailability(List<Availability>): void ole = 'RIDER' role = 'DRIVER' Database Schema (Relational) users • id : BIGSERIAL PRIMARY KEY name : VARCHAR NOT NULL email : VARCHAR UNIQUE NOT NULL • password\_hash : VARCHAR NOT NULL • role : VARCHAR NOT NULL ('RIDER' or 'DRIVER') Single Table Inheritance All user types stored in one table Discriminated by 'role' column

## 4 Sequence Diagram (Rider Request Ride)

### Rider Request Ride - Data Mapper Usage 9 JdbcRideRepository (Data Mapper) RideAdapter (Data Mapper) Frontend ApiServer RideService Rider Request ride (pickup: 3000, destination: 3141) POST /api/rides/request {pickup: 3000, destination: 3141} Authenticate & estimate fare newRequestedRide(riderld, pickup, destination, fare) Data Mapper API params → Domain object (primitives → rich object) Create Ride domain object with REQUESTED status Ride domain object requestRide(rider, pickup, destination, now) Validate & check wallet balance Data Mapper Domain object → DB record (object refs → foreign keys) Map object to SQL: Ride – rides table rider userld – rider ider ider RidsStatus – VARCHAR NSERT INTO rides (rider\_id. pickup\_postcode, destination\_postcode, ...) Generated ride ID Map ResultSet to domain object Data Mapper DB record → Domain object (foreign keys → object refs) Ride with generated ID Updated Ride toDto(ride) Data Mapper Domain object → API DTO (rich object → JSON format) Convert to RideDto: Ride — RideDto Format display strings Prepare for JSON RideDto 200 OK {rideld: 123, status: "REQUESTED",...} "Ride requested successfully!" Data Mapper Pattern Summary: 1. API → Domain: Convert HTTP parameters to rich domain objects 2. Domain → DB. Map object graphs to relational schema with foreign keys 3. DB → Domain: Reconstruct domain objects from database records 4. Domain → API: Transform domain objects to client-friendly DTOs

RideAdapter (Data Mapper)

RideService

JdbcRideRepository (Data Mapper) Database

Rider

Frontend

ApiServer