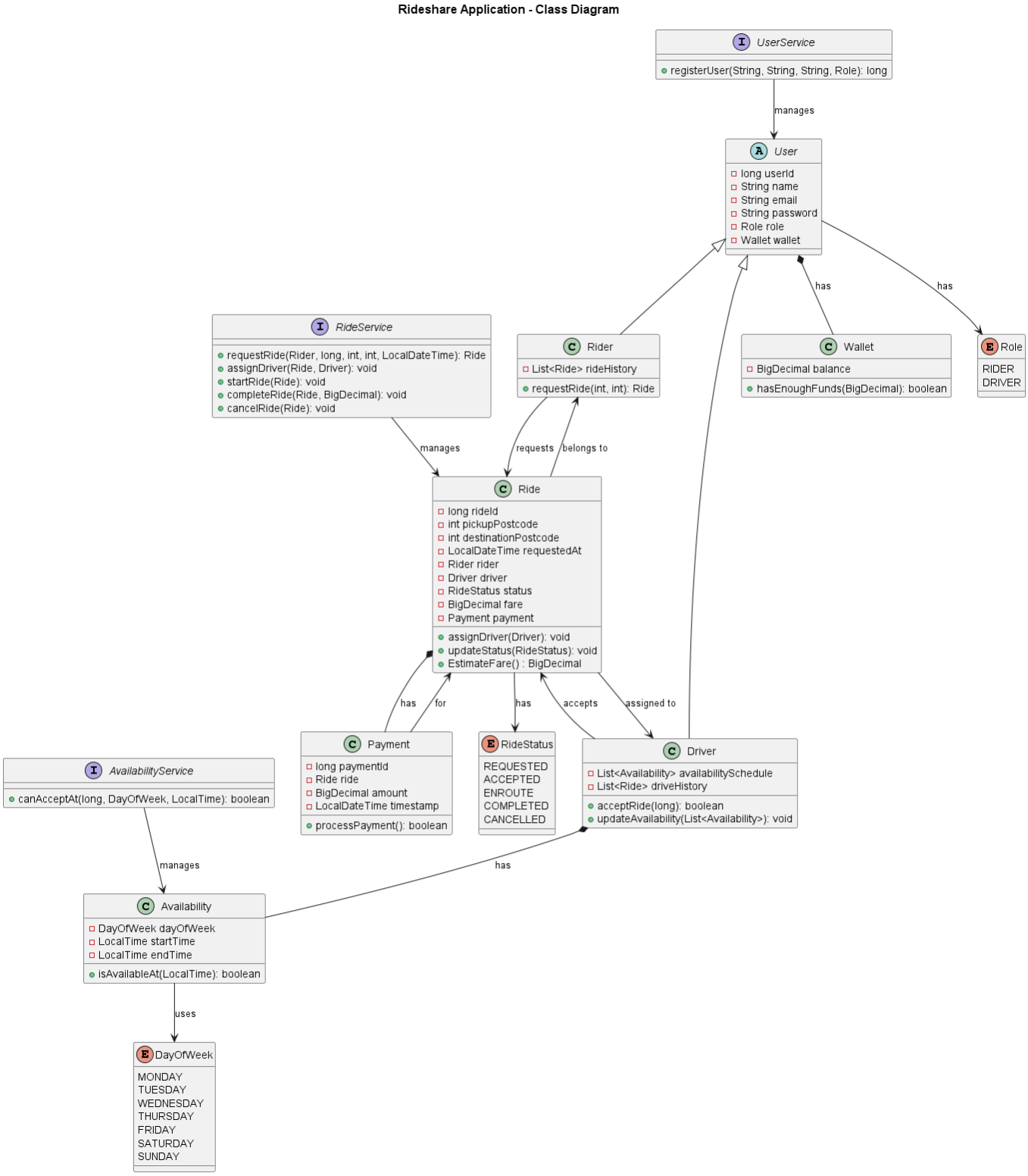
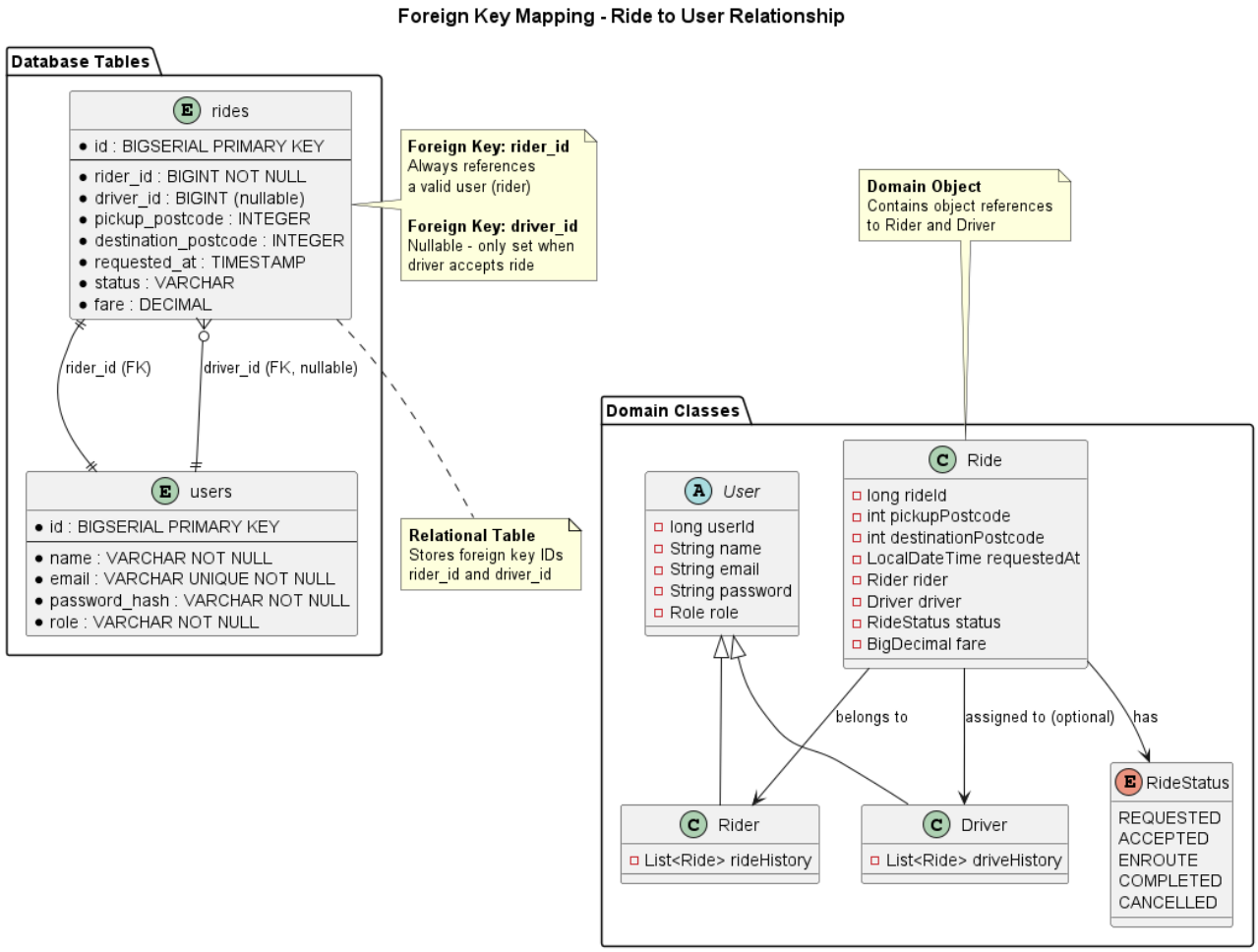
Part 2 Report

1 Class Diagram

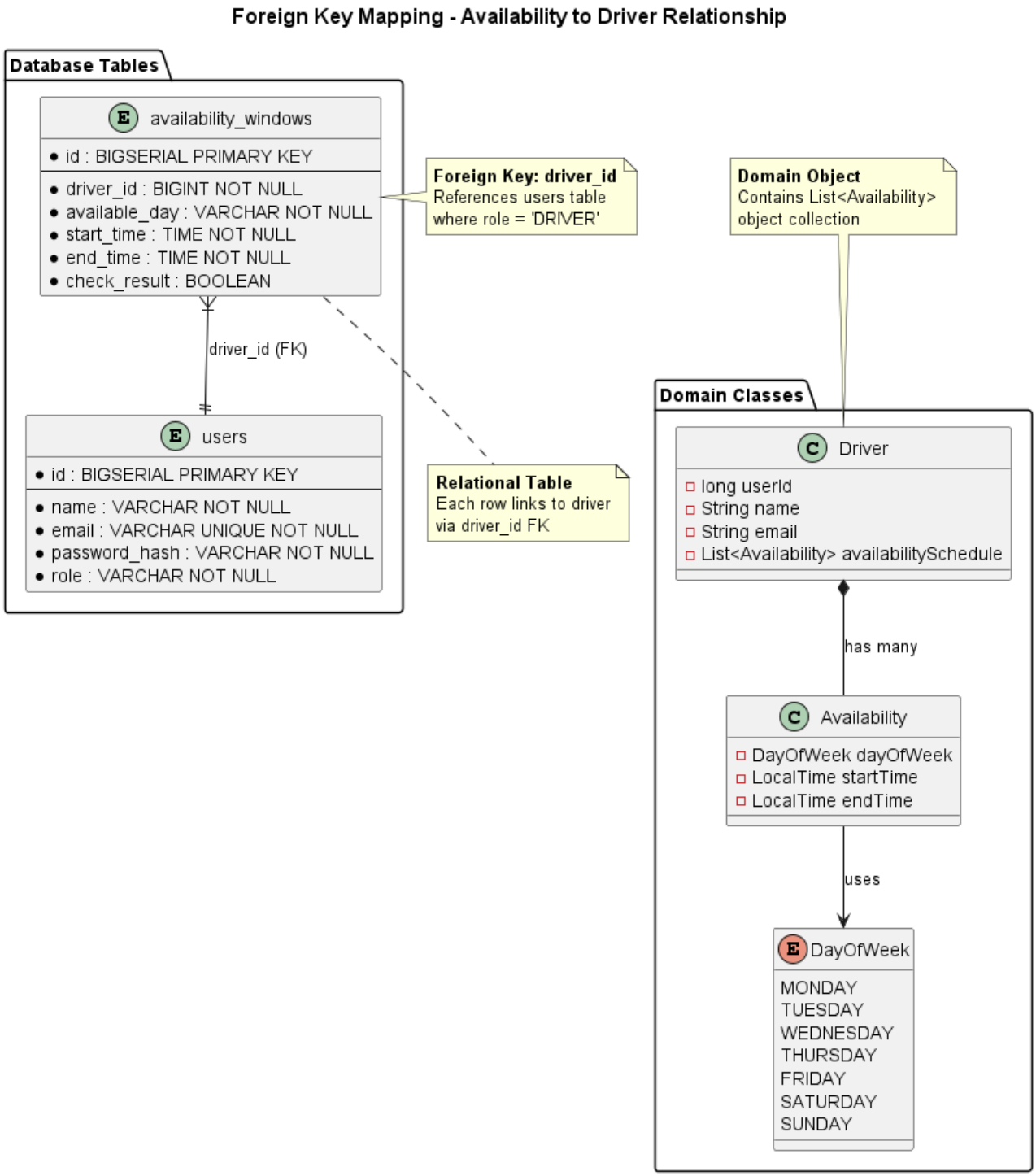


2 Foreign Key Mapping

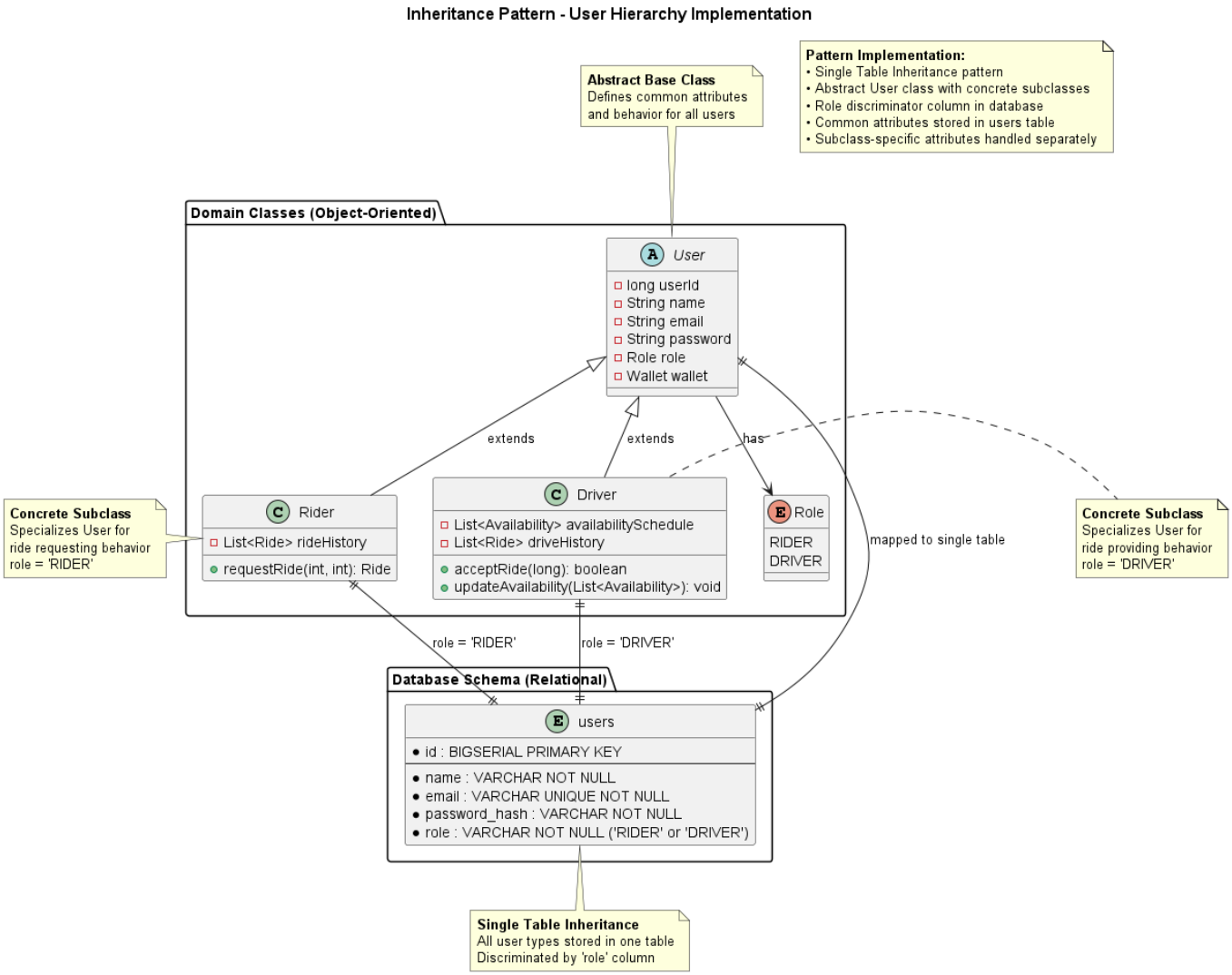
1. Ride to User Relationship



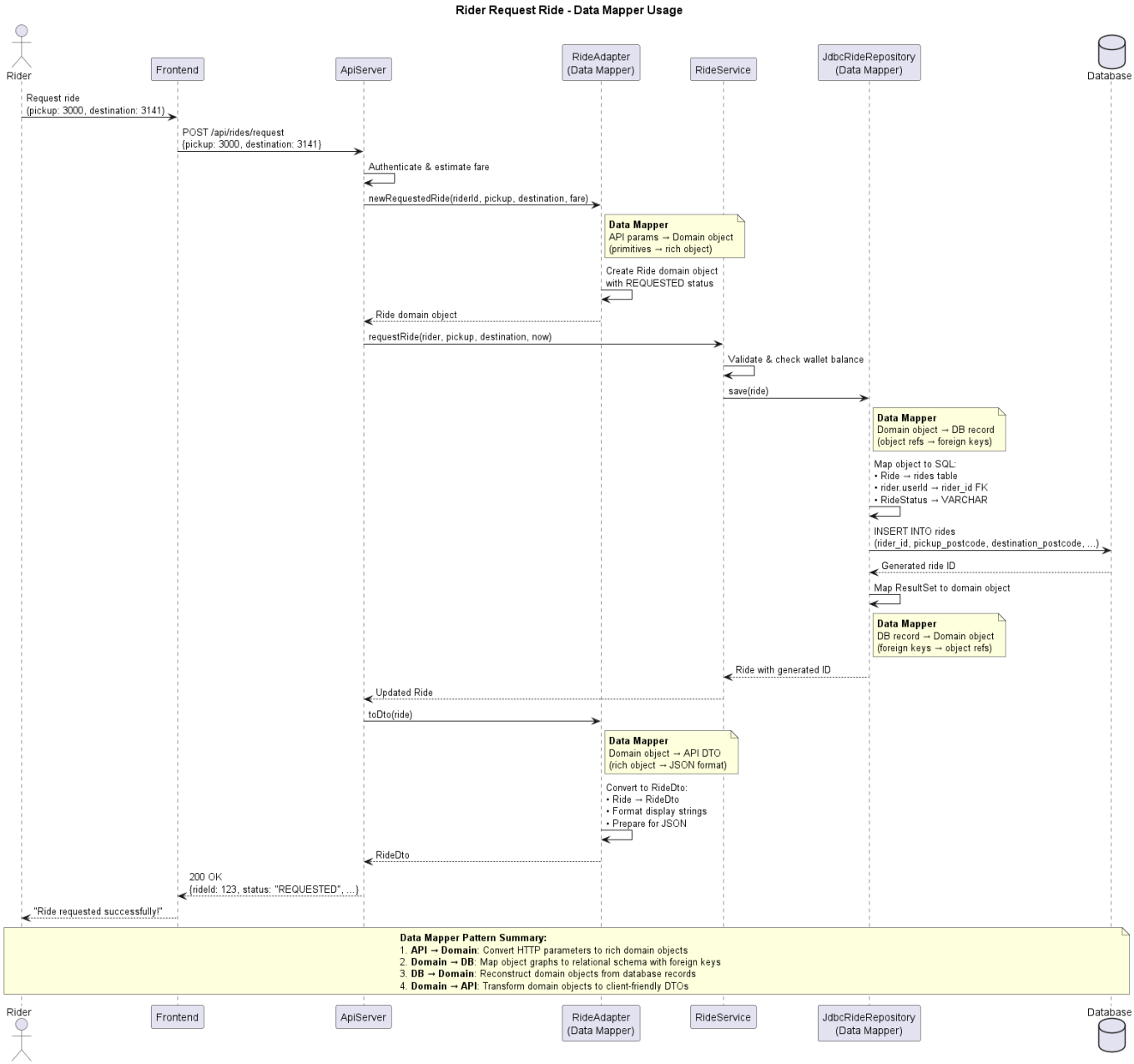
2. Availability to Driver Relationship



3 Inheritance



4 Sequence Diagram (Rider Request Ride)


Rider Request Ride - Data Mapper Usage
Rider
Frontend
ApiServer
RideAdapter (Data Mapper)
RideService
JdbcRideRepository (Data Mapper)
Database
Request ride (pickup: 3000, destination: 3141)
POST /api/rides/request {pickup: 3000, destination: 3141}
Authenticate & estimate fare
newRequestedRide(riderId, 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
save(ride)
Data Mapper
Domain object → DB record (object refs → foreign keys)
Map object to SQL:
• Ride → rides table
• rider.userId → rider_id FK
• RideStatus → VARCHAR
INSERT 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 {rideId: 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