Project: Cab Booking System

1. Introduction

This document outlines the Low-Level Design for a **Cab Booking System** that enables users to book rides, manage payments, and rate drivers.
This design supports both **Java (Spring Boot)** and **.NET (ASP.NET Core)** frameworks.

2. Module Overview

2.1 User Management
Handles user registration, login, and profile management.

2.2 Driver Management
Manages driver profiles, vehicle details, and availability status.

2.3 Ride Booking
Facilitates cab booking, ride status management, and driver assignment.

2.4 Payment Processing
Supports fare calculation, payment processing, and receipt generation.

2.5 Rating and Feedback
Allows users and drivers to rate and review each other post-ride.

3. Architecture Overview

3.1 Architectural Style

- **Frontend**: Angular or React
- **Backend**: REST API
- **Database**: MySQL/SQL Server

3.2 Component Interaction

- Frontend interacts with backend via secured REST APIs
- Backend performs business logic and handles data persistence
- All modules are decoupled for scalability and maintainability

4. Module-Wise Design

4.1 User Management

Entities

- User: UserID, Name, Email, Phone, PasswordHash, CreatedAt

Endpoints

- POST /api/users/register
- POST /api/users/login
- GET /api/users/profile

4.2 Driver Management

Entities

- Driver: DriverID, Name, Phone, LicenseNumber, VehicleDetails, Status

Endpoints

- POST /api/drivers/register
- GET /api/drivers/available
- PUT /api/drivers/status/{id}

4.3 Ride Booking

Entities

- Ride: RideID, UserID, DriverID, PickupLocation, DropoffLocation, Fare, Status

Endpoints

- POST /api/rides/book
- PUT /api/rides/status/{id}
- GET /api/rides/user/{userId}

4.4 Payment Processing

Entities

- Payment: PaymentID, RideID, UserID, Amount, Method, Status, Timestamp

Endpoints

- POST /api/payments/process
- GET /api/payments/receipt/{rideId}

4.5 Rating and Feedback

Entities

- Rating: RatingID, RideID, FromUserID, ToUserID, Score, Comments

Endpoints

- POST /api/ratings

- GET /api/ratings/user/{userId}

5. Deployment Strategy

- Frontend runs on localhost:4200 (Angular) or localhost:3000 (React)
- Backend runs on localhost:8080 (Spring Boot) or localhost:5000 (ASP.NET Core)
- Database hosted locally (MySQL/PostgreSQL/SQL Server)

6. Database Design

Table	Primary Key	Foreign Keys
User	UserID	–
Driver	DriverID	–
Ride	RideID	UserID, DriverID
Payment	PaymentID	RideID, UserID
Rating	RatingID	RideID, FromUserID

---

7. User Interface Design (Wireframes Suggested)

- Ride Booking Page
- Ride History Page
- Driver Availability Toggle
- Payment Confirmation Page
- Rating Submission Page

8. Non-Functional Requirements

- **Security**: Token-based authentication (JWT), password encryption
- **Performance**: Supports 500+ users in development setup
- **Scalability**: Modular backend services
- **Usability**: Mobile and web responsive frontend

9. Assumptions and Constraints

- Real-time location tracking is **not implemented**
- Payment gateway is simulated
- Notifications and alerts are excluded