# Bus Ticket Reservation System

## Introduction

Background

Public transportation plays a crucial role in connecting people across cities. Traditionally, bus ticket reservations were carried out at physical counters or over phone calls. These methods led to **long queues, manual errors, overbooking, and customer dissatisfaction**.

With the growth of internet-based applications, there is a need for a **digital bus reservation platform** where customers can search, book, and manage tickets online, while administrators can manage buses, routes, trips, and monitor sales.

### Problem Statement

The existing manual ticketing system suffers from:

- Lack of real-time seat availability tracking
- Error-prone cancellations & refunds
- Revenue losses due to poor management
- Time-consuming booking process for customers

### Objectives

- Develop a **centralized online bus reservation system**
- Enable **real-time seat booking & payment**
- Provide **role-based access** (Admin & Customer)
- Automate **ticket generation (QR/PDF)**
- Build **reports & analytics dashboards** for administrator

### Scope

- **Admin**: Manage buses, routes, trips, pricing, reports
- **Customer**: Search trips, book seats, make payments, download e-tickets
- **Security**: JWT authentication, password hashing, role checks
- **Scalability**: System designed for future enhancements (agents, promo codes, wallets)

# System Analysis & Design

## Functional Requirements

- **User Authentication & Role Management**
- **Bus & Route Management** (Admin)
- **Trip Scheduling**
- **Seat Booking & Payments**
- **Ticket Generation (QR/PDF)**
- **Cancellations & Refunds**
- **Reports & Analytics**

## Non-Functional Requirements

- Security: JWT authentication, BCrypt password hashing
- Performance: Seat booking <150 ms
- Reliability: Transaction rollback for booking failures
- Usability: Responsive frontend (React + Bootstrap)
- Scalability: Modular design ready for microservices

## Primary Users

- **Admin** → Manage buses, routes, trips, reports
- **Customer** → Book tickets, cancel tickets, payments

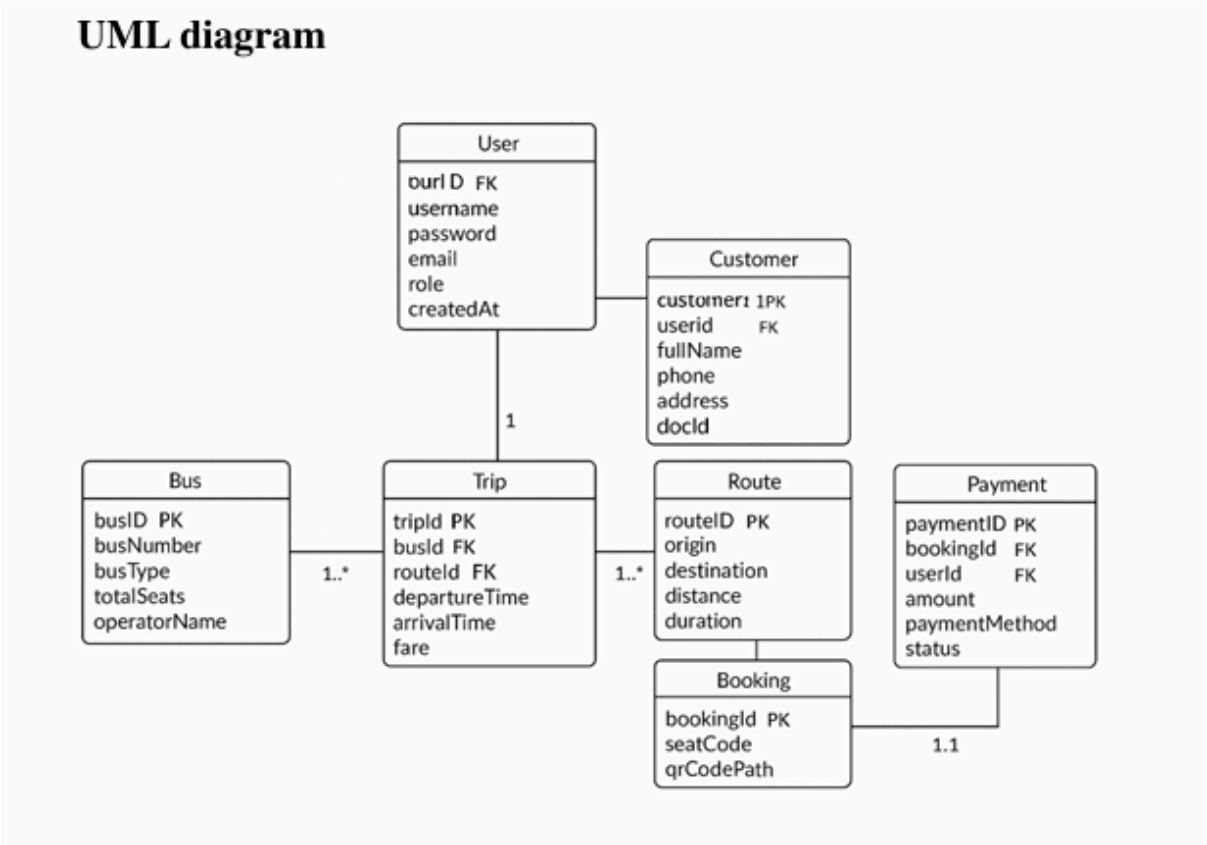
## UML Class Diagram

Entities:

- **User, Bus, Route, Trip, Seat, Booking, Payment, Ticket**

Relationships:

- User – Booking
- Bus – Trip
- Route – Trip
- Trip – Seat
- Booking – Payment
- Booking – Ticket



# System Architecture

### Technology Stack

- **Frontend**: React, Bootstrap, Axios
- **Backend**: Spring Boot, Spring Security, JPA/Hibernate
- **Database**: MySQL
- **Authentication**: JWT (JSON Web Token)
- **APIs**: REST APIs + Swagger Documentation

# Implementation

### Frontend (React + Vite)

- Components: Login, Registration, Trip Search, Seat Selection, Booking, Payment, Ticket
- React Router for navigation
- Axios for API communication
- LocalStorage for JWT token

### Backend (Spring Boot)

- Controllers for Authentication, Bus, Trip, Booking, Payment
- Service layer for business logic
- Repository layer for database operations
- Spring Security for authentication & authorization

### 5.3 Database Design

**Tables**:

- User (id, name, email, password, role)

- Bus (id, busNumber, busType, seats)
- Route (id, source, destination, distance, duration)
- Trip (id, busId, routeId, fare, time)
- Seat (id, tripId, seatNumber, status)
- Booking (id, userId, tripId, totalAmount, status)
- Payment (id, bookingId, amount, status)
- Ticket (id, bookingId, qrCode, pdfLink)

# Testing

## Testing Approaches

- **Unit Testing**: JUnit & Mockito for backend application services
- **Integration Testing**: API endpoints tested via Postman and swagger
- **Frontend Testing**: React Testing Library
- **User Acceptance Testing (UAT)**: Tested booking flow end-to-end

## Test Cases

- Successful login with valid credentials
- Failed login with invalid credentials
- Search trips by route & date
- Prevent double seat booking (concurrent requests)
- Ticket generated after successful payment
- Refund processed after cancellation

## Results & Discussion

- Customers can **book tickets online in <1 minute**
- Admins can easily **add/manage trips and generate reports**
- The system successfully prevents **double bookings**
- Automatic **QR/PDF ticket generation** improves efficiency
- Reports help admins track **sales, occupancy, and revenue**

# Challenges Faced and Solutions

### **1.** Challenge: JWT Authentication & Role-Based Access

- Problem: Implementing JWT tokens for authentication was tricky, especially differentiating between Admin and Customer access. Unauthorized users could still attempt to access admin APIs.
- Solution:
    - Configured Spring Security filters to validate JWT tokens before processing any API request.
    - Used role-based annotations (@PreAuthorize("hasRole('ADMIN')")) on sensitive endpoints.
    - Stored tokens in localStorage (frontend) and attached them to each API request using Axios interceptors.

### 2. Challenge: Preventing Double Seat Booking

- Problem**:** When multiple customers tried to book the same seat at the same time, there was a risk of overbooking due to race conditions.
- Solution**:**
    - Implemented seat hold mechanism before payment confirmation.
    - Added synchronized database transactions in Spring Boot with optimistic locking (@Version in JPA).
    - Performed real-time seat availability checks before finalizing a booking.

## 3. Challenge: API Security & Error Handling

- Problem**:** Without proper error handling, customers saw generic 500 errors instead of helpful messages.
- Solution**:**
    - Used Spring Boot Exception Handlers (@ControllerAdvice) to return custom error responses:
        - 400 – Validation errors
        - 401 – Unauthorized
        - 403 – Forbidden (wrong role)
        - 409 – Seat conflict
        - 422 – Payment failure
    - Displayed clear error messages on the frontend.

## 4. Challenge: Query Performance in Reports

- Problem**:** Generating sales and occupancy reports with multiple joins (User → Booking → Trip → Route) was slow.
- Solution**:**
    - Created materialized views (pre-aggregated tables) for daily sales.
    - Scheduled a batch job to refresh report data during off-peak hours.
    - Used SQL GROUP BY + SUM/COUNT to generate efficient reports.

# Work Division

## Day 1 – Backend & Database Setup

**Tasks Completed:**

- Installed and configured **Spring Boot + MySQL** environment.
- Designed the **UML diagram** and finalized database schema (Users, Buses, Routes, Trips, Seats, Bookings, Payments, Tickets).
- Implemented **User Authentication & JWT security** in Spring Boot.
- Created basic **REST APIs** for:
    - User Registration & Login
    - Adding Buses & Routes (Admin)
- Tested database connectivity and verified initial API responses using **Postman and Swagger**.

**Outcome:**

- A working backend setup with secure authentication and a connected database.
- Able to create users and buses through API calls.

## Day 2 – Frontend & Core Features

**Tasks Completed:**

- Set up **React (Vite)** frontend with React Router and Axios.
- Built **Login & Register pages** with JWT storage in localStorage.
- Implemented **Customer flow**:
    - Trip Search (source, destination, date)
    - Seat Selection screen with real-time seat availability
- Implemented **Admin flow**:

    - Add new trips with bus and route assignment
    - View all trips and schedules
- Handled **protected routes** based on user role (Admin vs Customer).

**Outcome:**

- A functional frontend where customers can search for trips and admins can manage buses/routes.
- JWT-based login working across backend & frontend.

# Day 3 – Booking, Payments, Ticketing & Testing

**Tasks Completed:**

- Implemented **Booking flow**: Seat Hold → Payment → Confirmed Booking.
- Added a **dummy payment gateway** (simulated success/failure).
- Integrated **Ticket Generation** (PDF with QR code).
- Implemented **Cancellations & Refund Policy**.
- Added **Reports Module** for Admin (Sales, Occupancy, Top Routes).
- Conducted **testing**:
    - Unit tests (JUnit for backend)
    - Frontend tests (React Testing Library)
    - End-to-end booking flow tested on Postman & browser.
- Wrote **project documentation** (Problem Statement, UML, API docs, Challenges & Solutions).

**Outcome:**

- Fully working system:

    - Customers can book, pay, and download e-tickets.
    - Admins can manage trips and view reports.
- Complete documentation prepared for submission.

# Conclusion

This project successfully demonstrates a **Bus Ticket Reservation System** with real-time seat booking, payments, and ticket generation. It solves key problems of manual systems such as overbooking, inefficiency, and customer dissatisfaction.