Capstone Project: Bus Ticket Reservation Management

Problem Statement

Manual bus ticketing leads to overbooking, revenue leakages, and poor customer experience (long queues, phone-only support, no real-time seat visibility). A centralized **Bus Ticket Reservation System** is needed to manage **buses, routes, schedules, real-time seat inventory, online bookings, payments, cancellations, and e-tickets**, with **role-based access** and **secure JWT authentication**.

Scope of the System

Roles

- **Admin**: Manage buses, routes, trip schedules, pricing, seat layouts, and reports.
- **Customer**: Search trips, choose seats, book & pay, download e-tickets, cancel as per policy.

Security

- **JWT** authentication for login & API authorization.
- **Role-based access control** (Admin vs Customer).

Project Development Guidelines

Backend (Spring Boot + MySQL)

Technology Stack

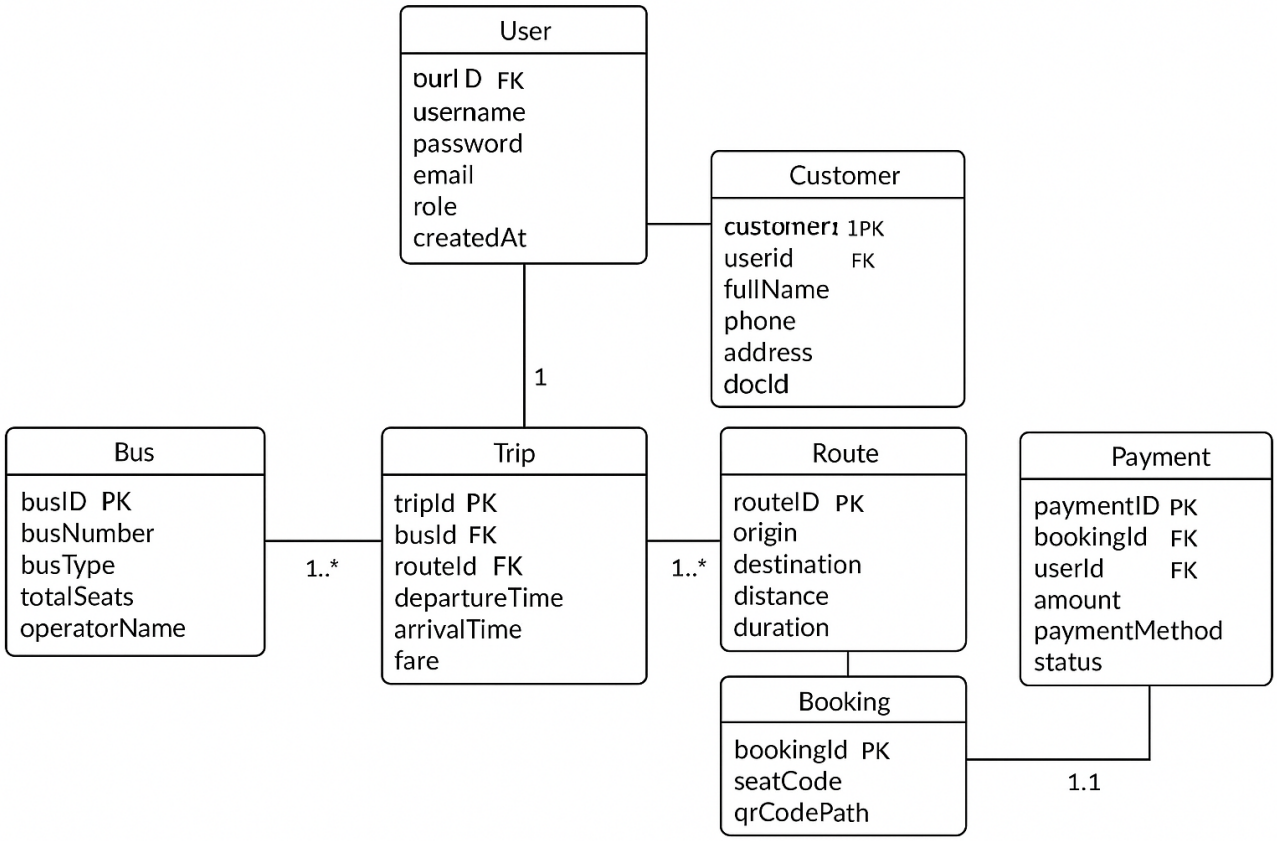
- Spring Boot (REST APIs)
- Spring Security + JWT
- MySQL (Users, Buses, Routes, Trips, Bookings, Payments)
- JPA/Hibernate (ORM)
- Swagger / OpenAPI (API docs)

The 6 Core Modules

1. **Authentication & User Management** – JWT login, registration, roles, profile.
2. **Bus & Route Management** – Buses, seat layouts, routes, route stops.

3. **Trip Scheduling & Seat Inventory** – Create date-wise trips, fares, live seat availability.
4. **Booking & Payment Processing** – Seat hold, booking confirmation, payment capture.
5. **Ticketing, Cancellations & Notifications** – E-ticket PDF, QR code, cancel/refund policy, email/SMS.
6. **Reports & Dashboards** – Sales, occupancy, route performance, daily settlement.

UML diagram



Frontend (React.js + Bootstrap + CSS)

Technology Stack

- React (Functional Components + Hooks)
- React Router
- Axios
- JWT storage: `localStorage/sessionStorage`

- **Bootstrap** for UI, **custom CSS** for branding

Key Screens

- Auth: Login/Register
- Customer: Search Trips → Seat Selection → Checkout → Payment → Ticket
- Admin: Manage Buses/Routes/Trips, Pricing, Reports

Frontend Flow

1. Login/Register → Store JWT → Decode role → Route to dashboard.
2. Attach JWT in `Authorization: Bearer <token>`.
3. Handle `401/403` → Auto-logout and redirect to Login.

Extended API Guidelines

- **Base URL:** `/api/v1`
- **Auth:** Bearer JWT in headers
- **Swagger UI:** `http://localhost:8080/swagger-ui/`
- **Common Errors:** 400 (validation), 401 (unauthorized), 403 (forbidden), 409 (seat conflict), 422 (payment failure), 500 (server)

✅ Swagger API Examples

Module	Endpoint	Method	Access	Description
Authentication	`/api/v1/auth/login`	POST	Public	Authenticates a user and returns an `accessToken` and user details.
	`/api/v1/auth/register`	POST	Public	Registers a new user with their details.
Bus & Route Management	`/api/v1/buses`	POST	Admin	Creates a new bus entry with its details, including seat layout.
	`/api/v1/routes`	POST	Admin	Creates a new route with origin, destination, and stops.

Module	Endpoint	Method	Access	Description
Trip Scheduling & Seat Inventory	`/api/v1/trips`	POST	Admin	Schedules a new trip for a specific bus and route.
	`/api/v1/trips/search`	GET	All	Searches for available trips between an origin and destination on a specific date.
	`/api/v1/trips/:id/seats`	GET	All	Retrieves the seat layout and shows which seats are booked for a specific trip.
Booking & Payment Processing	`/api/v1/bookings/hold`	POST	Customer	Places a temporary hold on selected seats for a trip.
	`/api/v1/payments/checkout`	POST	Customer	Processes payment for a held booking.
Ticketing, Cancellations & Notifications	`/api/v1/tickets/:id`	GET	Customer	Retrieves ticket details, including a QR code and a link to a PDF.
	`/api/v1/bookings/:id/cancel`	POST	Customer	Cancels a confirmed booking and processes any eligible refund.
Reports & Dashboards	`/api/v1/reports/sales`	GET	Admin	Provides a summary of sales data, including total revenue, bookings, and top routes.

Database Guidelines (Conceptual)

- **Normalization:** 3NF or better.
- **User–Customer Mapping:** `users` (auth) ↔ `customers` (profile) 1–1.
- **Bus/Route/Trip:** `buses` 1–M `trips`; `routes` 1–M `trips`.
- **Inventory:** Seats are derived from `seat_layout` + `bookings_seats`.
- **Booking Lifecycle:** hold → payment → confirmed → ticket; supports cancel/refund.
- **Payments:** store status, gateway ref, audit.

- **Notifications:** store email/SMS logs.

Entities and Relationships

Entity	Attributes	Relationships
`User`	`id (PK), name, email, phone, password, role (admin/user), createdAt`	`1 user can have many bookings and many payments.`
`Bus`	`id (PK), busNumber, busType (AC/Non-AC/Sleeper), totalSeats, operatorName`	`1 bus can have many trips.`
Route	`id` (PK), `source`, `destination`, `distance`, `duration`	1 route can have many **trips**.
Trip	`id` (PK), `busId` (FK), `routeId` (FK), `departureTime`, `arrivalTime`, `fare`	1 trip can have many **seats** and many **bookings**.
Seat	`id` (PK), `tripId` (FK), `seatNumber`, `seatType`(window/aisle), `isBooked`	Each seat belongs to one **trip**. A seat may be part of one **booking**.
Booking	`id` (PK), `userId` (FK), `tripId` (FK), `bookingDate`, `totalAmount`, `status`	1 booking can include many **seats** and is associated with one **payment**.

Non-Functional Requirements

- **Security:** BCrypt for passwords, HTTPS, input validation, rate limiting for search/hold.
- **Performance:** Seat hold & conflict checks < 150 ms under normal load.
- **Reliability:** Idempotent payment callbacks; transaction boundaries around booking.
- **Scalability:** Separate modules/services for Search, Booking, Payments (future microservices).

- **Auditability:** Persist payment & status changes; immutable ticket numbers.

UX guidelines with **principles** and **implementation in a Bus Ticket Reservation System**

UX Guideline	UX Principle	Implementation in Bus Ticket System
Consistency	Maintain uniformity in visuals and interactions	Use the same colors, fonts, button styles, and layouts across all pages (search, seat selection, payment)
Clarity & Simplicity	Reduce cognitive load and prioritize information	Simplify search filters (Source, Destination, Date, Bus Type) and avoid cluttered screens
Feedback & Responsiveness	Provide immediate responses to user actions	Show real-time seat availability, loading indicators, and booking confirmation pop-ups
Error Prevention & Handling	Prevent mistakes and provide clear guidance	Validate user inputs like email, phone number, and payment details before submission; show helpful error messages if incorrect

✅ Expected Outcomes

1. **Real-time availability** with reliable seat locking to prevent double booking.
2. **Frictionless booking** flow from search to e-ticket with secure payments.
3. **Operational control** for admins: route profitability, occupancy, schedule management.
4. **Customer satisfaction** via instant e-tickets (PDF/QR), notifications, and easy cancellations.
5. **Extensibility** for agents, promo codes, wallets, and multi-operator support.

General Guidelines

1. **Project Structure**
    - Maintain a consistent folder structure for frontend, backend, and documentation.
    - Separate configuration files, source code, and assets for clarity.
2. **Coding Standards**
    - Use meaningful variable, function, and class names.
    - Maintain consistent indentation, spacing, and commenting.
3. **Version Control with Git**
    - **Repository Setup:** Initialize a Git repository at the project root (`git init`).

- **Branching Strategy:**
  - Use `main` (or `master`) for production-ready code.
  - Create feature branches (`feature/<feature-name>`) for new features.

4. **Documentation**
   - Keep API specs, UI/UX guidelines, and system diagrams updated.
   - Include README with project setup, dependencies, and instructions.
5. **Testing & Validation**
   - Test features before committing.
   - Document test cases and outcomes for major modules.