Name: V Kamakshi

Batch: Java React (Batch 3)

# Bus Ticket Reservation System

The Bus Ticket Reservation System is a web application developed using Spring Boot for the backend and React.js for the frontend. It allows customers to search, book, and manage bus tickets seamlessly while providing administrators with tools to manage buses, trips, and bookings.

---

## Problem Statement

Manual ticketing causes overbooking, revenue leakage, and poor customer experience. This system centralizes buses, routes, trips, live seat inventory, bookings, payments, cancellations, and e-tickets with role-based access and JWT-backed API security.

## Scope of the System

### Roles

- **Admin** – manages buses, routes, trips, pricing, reports.
- **Customer** – searches trips, selects seats, books and pays, downloads e-tickets, cancels per policy.

### Security

- Spring Security with stateless sessions, **JWT [JSON Web Token]** for authN/authZ, **BCrypt** for password hashing, and CORS for `http://localhost` origins.
- Public endpoints: auth, trip search and trip/seat read; everything else requires authentication; admin-only endpoints guard write/report operations (see Access in API tables below).

---

## Objectives:

• Develop a user-friendly interface for customers to book tickets online.

• Provide an admin panel for managing buses, trips, and bookings.

• Implement authentication and authorization using JWT.

• Ensure secure password encryption and role-based access.

## Tools and Technologies Used:

• Backend: Spring Boot (v3.5.5), Spring Security, JPA, MySQL

• Frontend: React.js, Bootstrap, CSS

• Database: MySQL

• Authentication: JWT (JSON Web Token)

• API Documentation: Swagger / OpenAPI

• Additional Libraries: Lombok, (QR code generation)

---

## The 6 Core Modules (implemented)

1. **Authentication & Users** – register, login, JWT issuance; role inferred from token and user profile.
2. **Bus & Route Management** – admin creates/reads buses and routes.
3. **Trip Scheduling & Seat Inventory** – admin creates trips; public GET for searching and seats listing.
4. **Booking & Payment** – hold then cancel/checkout; payment endpoint exposed; status persisted.
5. **Ticketing & Cancellations** – ticket retrieval, PDF export, cancel flow.
6. **Reports & Dashboards** – bookings and payments summaries; PDF exports.

---

## Extended API Guidelines

- **Base URL:** `/api/v1`
- **Auth:** `Authorization: Bearer <jwt>`
- **Swagger:** `/swagger-ui/`
- **Common errors:** 400 validation, 401 unauthorized, 403 forbidden, 409 seat conflict, 422 payment failure, 500 server.

### Actual Endpoints discovered (from controllers)

#### AuthController

| Method | Path | Access |
|---|---|---|
| POST | `/api/v1/auth/login` | Public |
| POST | `/api/v1/auth/register` | Public |

## BookingController

| Method | Path | Access |
|---|---|---|
| Method | Path | Access |
| POST | /api/v1/bookings/hold | Protected |
| POST | /api/v1/bookings/{id}/cancel | Protected |

## BusRouteController

| Method | Path | Access |
|---|---|---|
| GET | /api/v1/buses | Admin |
| POST | /api/v1/buses | Admin |
| GET | /api/v1/routes | Admin |
| POST | /api/v1/routes | Admin |

## PaymentController

| Method | Path | Access |
|---|---|---|
| POST | /api/v1/payments/checkout | Protected |

## ReportsController

| Method | Path | Access |
|---|---|---|
| GET | /api/v1/reports/bookings | Admin |
| GET | /api/v1/reports/payments | Admin |
| GET | /api/v1/reports/bookings/pdf | Admin |
| GET | /api/v1/reports/payments/pdf | Admin |

## RootController

| Method | Path | Access |
|---|---|---|
| GET | / | Public |

### TicketController

| Method | Path | Access |
|---|---|---|
| GET | `/api/v1/tickets/{bookingId}` | Protected |
| GET | `/api/v1/tickets/{bookingId}/pdf` | Protected |
| DELETE | `/api/v1/tickets/{bookingId}` | Protected |

### TripController

| Method | Path | Access |
|---|---|---|
| GET | `/api/v1/trips` | Admin |
| POST | `/api/v1/trips` | Admin |
| GET | `/api/v1/trips/{id}` | Public |
| GET | `/api/v1/trips/{id}/seats` | Public |
| GET | `/api/v1/trips/search` | Public |

---

## Database

- **Normalization:** ~3NF.
- **Users ↔ Bookings/Payments:** one user, many bookings and payments.
- **Buses/Routes/Trips:** bus→trips (1-M), route→trips (1-M).
- **Inventory:** seat availability derived from `Seat` and `BookingSeat` on a `Trip`.
- **Booking lifecycle:** `HOLD → (CANCEL|PAYMENT) → CONFIRMED → TICKET`; cancellations/refunds supported.

### Entities and Relationships

| Entity | Attributes (type) | Relationships |
|---|---|---|
| Booking | id:Long, user:User, trip:Trip, status:String, totalAmount:Double, createdAt:Instant | ManyToOne, ManyToOne, OneToMany |
| BookingSeat | id:Long, booking:Booking, seat:Seat | ManyToOne, ManyToOne |

| Entity | Attributes (type) | Relationships |
|---|---|---|
| Bus | id:Long, busNumber:String, busType:String, totalSeats:Int, operatorName:String | - |
| Payment | id:Long, booking:Booking, status:String, reference:String, amount:Double, createdAt:Instant | ManyToOne |
| Route | id:Long, source:String, destination:String, distance:Double, duration:String | - |
| Seat | id:Long, trip:Trip, seatNumber:String, seatType:String, booked:boolean | ManyToOne |
| Trip | id:Long, bus:Bus, route:Route, departureTime:Instant, arrivalTime:Instant, fare:Double | ManyToOne, ManyToOne |
| User | id:Long, email:String, password:String, name:String, role:String, createdAt:Instant | - |

## Non-Functional Requirements

- **Security:** BCrypt password storage, signed JWT, input validation.
- **Performance:** seat hold/conflict checks optimized at repository/service layers.
- **Reliability:** transactional boundaries around booking and payment status updates.
- **Scalability:** clear seams for splitting Search/Booking/Payments into services later.
- **Auditability:** persist payment references and ticket numbers.

## UX Guidelines → Implementation

- **Consistency:** common colors/typography/components via Tailwind; shared `NavBar`.
- **Clarity & Simplicity:** minimal search fields (source, destination, date) and straightforward seat/checkout flow.

- **Feedback & Responsiveness:** seat availability shown on trip details; post-actions confirm states.
- **Error Prevention & Handling:** frontend validates inputs; backend returns precise status codes; 401/403 handled by router guard and interceptor.

---

# Execution Notes

## Backend

1. Ensure MySQL is running and schema is reachable.
2. `mvn clean package -DskipTests` then `java -jar target/*.jar` or `mvn springboot:run`.
3. Visit Swagger at `http://localhost:8085/swagger-ui/index.html`.

## Frontend

1. `npm install`
2. `npm run dev` → `http://localhost:5173`
3. Set `VITE_API_BASE_URL` if backend is not `http://localhost:8085/api/v1`.

---

# Appendix – Dependency Highlights

- **JWT:** `io.jsonwebtoken:jjwt-*`
- **OpenAPI UI:** `org.springdoc:springdoc-openapi-starter-webmvc-ui`
- **DB:** `com.mysql:mysql-connector-j`
- **Test:** JUnit 5, Mockito

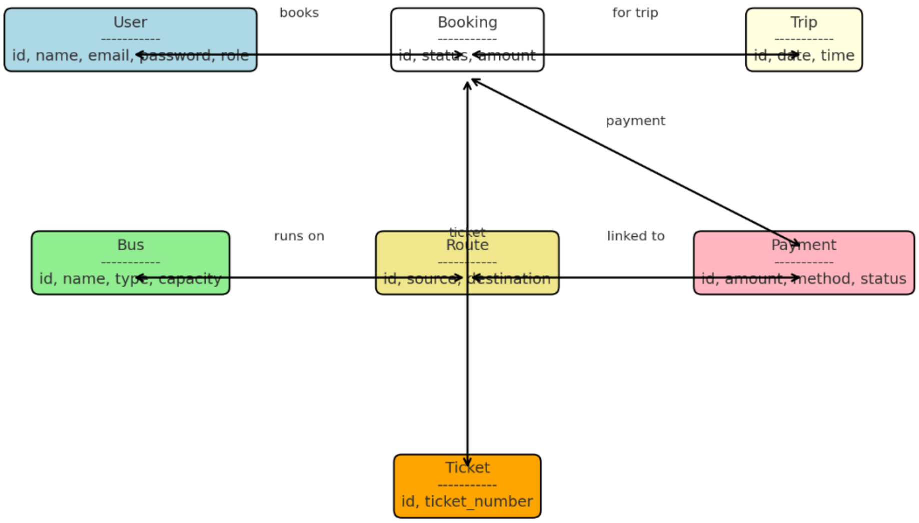
---

# Key Challenges and Resolutions

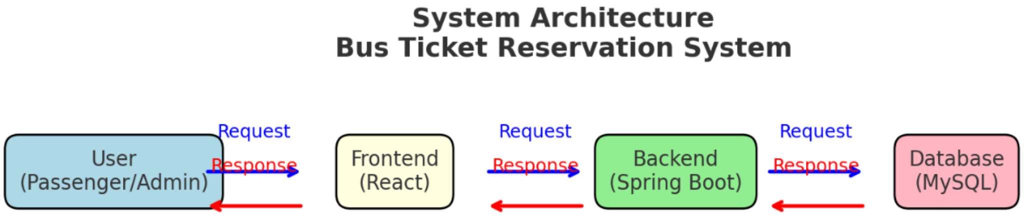
- **Day 1 —** setup and plumbing Frontend wouldn't start: missing package.json and Vite, so npm scripts failed. I rebuilt the project, added the right dev dependencies, and fixed the scripts. API calls broke until I set VITE_API_BASE_URL and handled CORS [Cross-Origin Resource Sharing]. Maven flagged duplicate dependencies (ZXing and OpenAPI), which I cleaned up. MySQL [Structured Query Language] tables didn't appear because spring.jpa.hibernate.ddl-auto and credentials were wrong. I kept guessing IDs like busId until I fixed the datasource and let Hibernate create the schema.
- **Day 2 —** auth, logic, and polish Auth worked but role checks still returned 401/403. I corrected the Spring Security filter chain, enabled method security, and ensured the Axios interceptor always sent the JWT [JSON Web Token] in Authorization: Bearer <token>. To prevent double-booking, I added transactional seat-locking and conflict checks. PDF [Portable Document Format] exports for tickets and reports needed proper content types and stream handling. Finally, I synced Swagger/OpenAPI with the actual DTOs so the API [Application Programming Interface] docs matched the UI [User Interface] behavior end-to-end.

# Figures

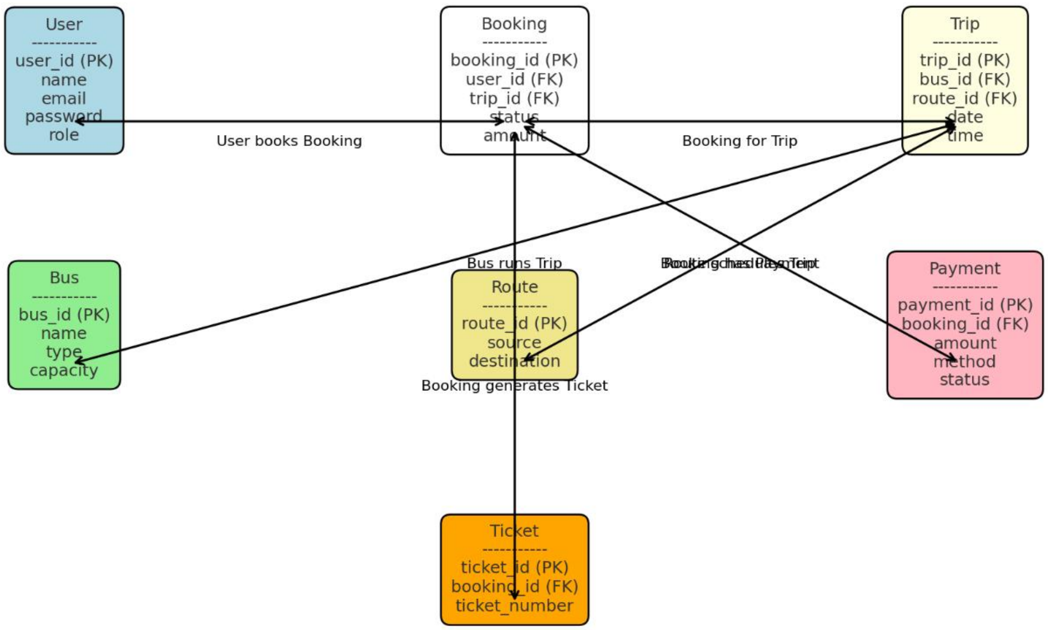
# Figure 1: Use Case Diagram



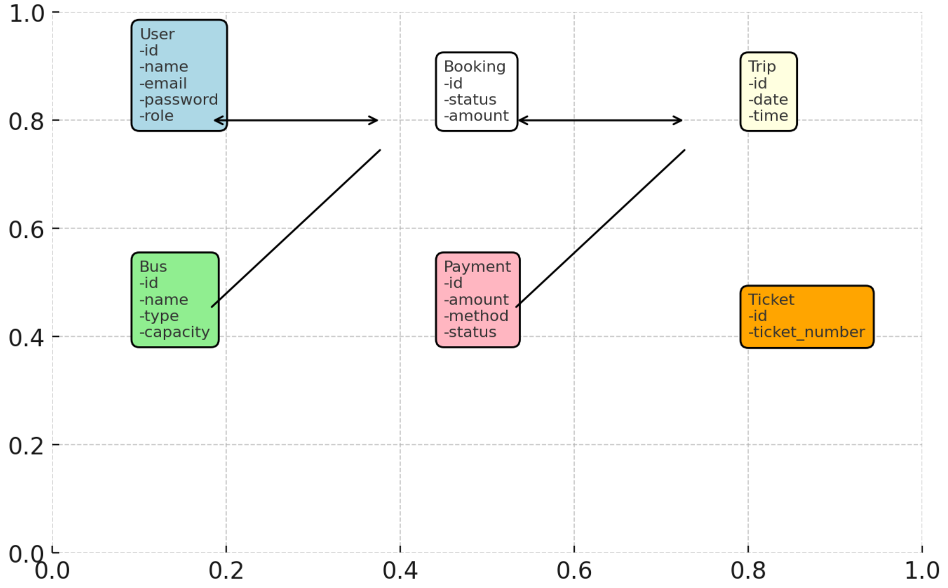
## Figure 2: System Architecture



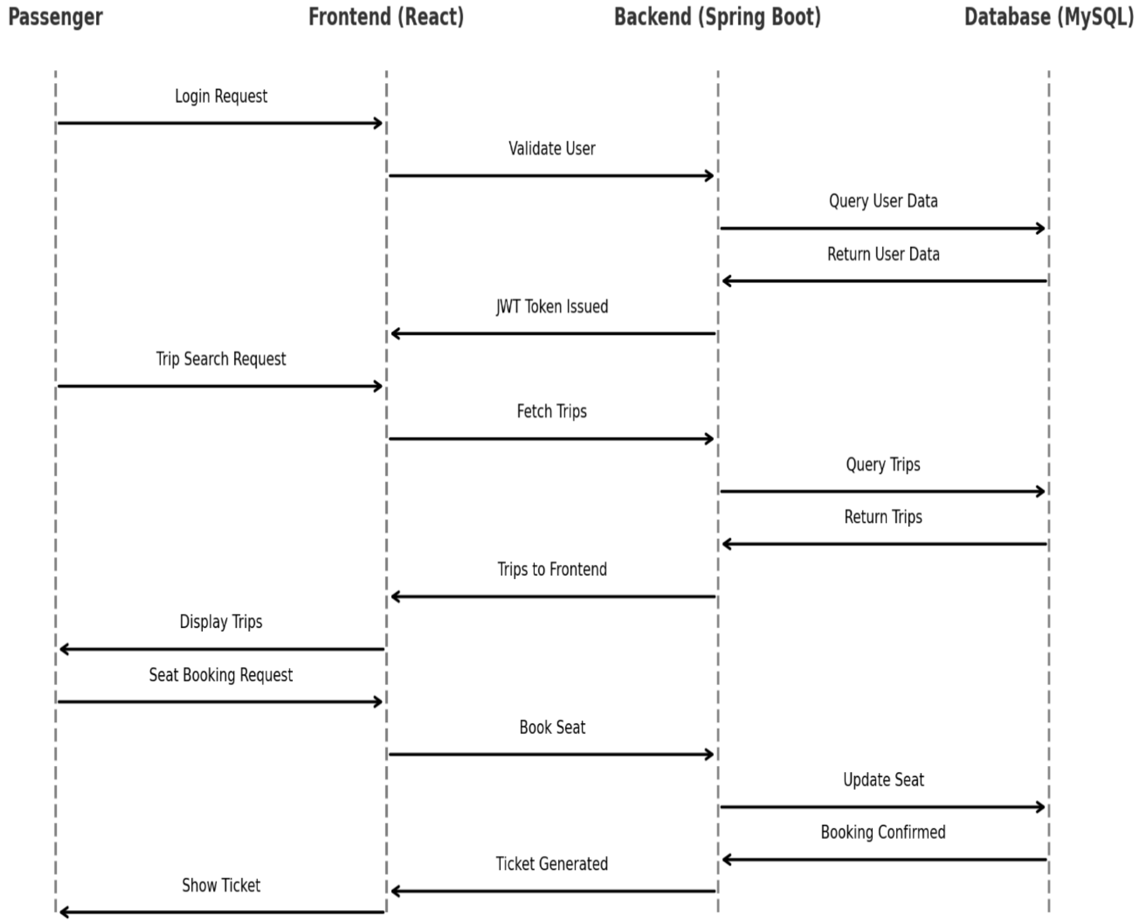
## Figure 3: Entity-Relationship Diagram



## Figure 4: UML Class Diagram



## Figure 5: Booking Flow Sequence Diagram


Passenger
Frontend (React)
Backend (Spring Boot)
Database (MySQL)
Login Request
Validate User
Query User Data
Return User Data
JWT Token Issued
Trip Search Request
Fetch Trips
Query Trips
Return Trips
Trips to Frontend
Display Trips
Seat Booking Request
Book Seat
Update Seat
Booking Confirmed
Ticket Generated
Show Ticket