Bus Ticket Reservation – System Summary

This document mirrors the structure and level of detail used in the sample “Bus Ticket Reservation Management” PDF and consolidates what’s actually implemented in your codebase (backend fixed.zip + frontend fixed.zip).

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).

Project Development Guidelines

Backend (Spring Boot + MySQL)

Tech stack: Java 17, Spring Boot 3.5.x, Spring Security + JWT, JPA/Hibernate, MySQL, springdoc-openapi (Swagger UI).
Key modules: Auth, Bus/Route, Trip + Seat Inventory, Booking + Payment, Ticketing/Cancellation, Reports.

Notable configuration (scrubbed): - `spring.datasource.url` to a local MySQL schema; `ddl-auto=update`. - `app.jwt.secret` and `app.jwt.expiration` set for token signing and TTL. - Swagger: `/swagger-ui/index.html`.

Frontend (React + Vite + Tailwind + React Router)

Tech stack: React 18, Vite 5, React Router 6, Axios, jwt-decode, Tailwind CSS.
Flow: Login/Register → store JWT → infer role → route to Customer or Admin areas; Axios interceptor attaches `Authorization: Bearer <token>`; 401 triggers local sign-out.

Key screens and routes (from `App.jsx`): - `/` Home, `/login`, `/register` - `/search` Trip search, `/trips/:id` Trip details + seats - `/checkout/:bookingId` Checkout and payment - `/ticket` My ticket, `/cancel` Cancel booking - `/admin` Dashboard with nested `trips`, `buses-routes`, `reports` (protected via `ProtectedRoute`)

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
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

Method	Path	Access
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 Guidelines (Conceptual)

- **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 (from `model` package)

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
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 spring-boot:run`.
3. Visit Swagger at `http://localhost:8080/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:8080/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 (if present)