📘 Bus Ticket Reservation System (BTRS)

Name :Tulabandi chandana Lakshmi Priya

Batch : Java React (Batch 3)

Project Version: 1.0  
Date: September 2025

# 1. Introduction

The Bus Ticket Reservation System (BTRS) is a comprehensive full-stack web application designed to modernize the way bus tickets are booked and managed. Traditional ticket booking processes rely on manual systems or physical counters, which can result in delays, errors, and customer inconvenience. With BTRS, passengers are able to log in securely, search for buses based on their travel preferences, check seat availability in real time, and complete bookings seamlessly. On the other hand, administrators can manage buses, define routes, schedule trips, and generate reports for analysis.  
  
The application follows a three-tier architecture, ensuring scalability and maintainability. The \*\*frontend\*\* is built with React.js ,vite to provide a dynamic and responsive user interface. The \*\*backend\*\* is powered by Spring Boot, which implements REST APIs for handling all business logic including authentication, seat allocation, and ticketing. The \*\*database\*\* uses MySQL to persist data such as user profiles, buses, trips, and bookings.  
  
The key objectives of BTRS include:  
- Improving the passenger booking experience.  
- Providing real-time seat tracking to avoid conflicts.  
- Supporting secure authentication using JWT tokens.  
- Reducing the workload of administrators.  
- Ensuring scalability for future enhancements such as payment gateway integration or mobile applications.

# 2. Requirement Analysis

2.1 User Stories  
The requirements were captured in the form of user stories to ensure that both passengers and administrators are able to meet their needs effectively.  
  
Passenger Stories:  
- As a passenger, I want to create an account and log in securely.  
- As a passenger, I want to search buses by source, destination, and date.  
- As a passenger, I want to view available seats before confirming a booking.  
- As a passenger, I want to make secure payments and view my ticket.  
  
Admin Stories:  
- As an admin, I want to add new buses and manage their details.  
- As an admin, I want to define and update routes.  
- As an admin, I want to schedule trips and assign buses to routes.  
- As an admin, I want to generate reports to analyze bookings and revenue.  
  
2.2 Use Case Diagram (Figure 1)  
The use case diagram captures the primary interactions for both passengers and admins.

# 3. System Design

3.1 System Architecture (Figure 2)  
The architecture of BTRS follows the 3-tier design:  
- \*\*Presentation Layer (Frontend):\*\* React-based SPA for UI and user interactions.  
- \*\*Business Layer (Backend):\*\* Spring Boot REST API with services and controllers.  
- \*\*Data Layer (Database):\*\* MySQL relational schema storing entities like User, Trip, Booking.  
  
3.2 Database Schema (Figure 3)  
The ER diagram highlights relationships:  
- A User can make multiple Bookings.  
- A Trip is linked to a Bus and a Route.  
- Each Booking can have one Payment and one Ticket.  
  
3.3 UML Class Diagram (Figure 4)  
The UML diagram shows the structure of core classes and their associations, helping in understanding the static view of the system.

# 4. Implementation

4.1 Technology Stack  
Backend: Spring Boot 3.x, Spring Security, JWT, Spring Data JPA, Java 17, Maven.  
Frontend: React.js, React Router, Axios, Tailwind CSS, Vite.  
Database: MySQL.  
Testing: Postman, Swagger UI, JUnit 5.  
Collaboration: GitHub, npm, Maven.  
  
4.2 Flow Example – Ticket Booking Process (Figure 5)  
Step 1: Passenger logs in using credentials. Backend validates and issues JWT.  
Step 2: Passenger searches trips. Backend queries DB and returns results.  
Step 3: Passenger selects a seat. Backend checks availability.  
Step 4: Booking is created, seat is locked, ticket generated.  
Step 5: Payment information is stored, confirmation sent to user.

# 5. Team Roles & Contributions

This was developed as an individual project. Contributions include:  
- Backend: Implemented controllers, services, JWT-based security.  
- Frontend: Built React components for search, booking, seat selection.  
- Database: Designed schema and relationships.  
- Testing: Validated APIs with Swagger and Postman.

# 6. Testing

6.1 Backend Testing  
APIs were tested with Postman and Swagger. Various scenarios like login success/failure, trip search, and booking validation were covered.  
  
6.2 Frontend Validation  
React forms included validation logic to prevent invalid input. State management ensured smooth booking flow.  
  
Examples:  
- Valid login returns dashboard access.  
- Invalid login displays error.  
- Empty seat selection blocked by validation.

# 7. Integration & Workflow

Integration was ensured through consistent API contracts. Frontend consumed backend APIs via Axios with JWT in headers.  
  
Workflow (Figure 7):  
User → React Frontend → Spring Boot Controller → Service → Repository → Database  
Response flows back through the same path.

# 8. Functionalities

Passenger:  
- Signup/Login  
- Search buses and trips  
- View available seats  
- Book tickets and make payment  
- Access booking history  
  
Admin:  
- Add and manage buses  
- Define routes  
- Schedule trips  
- View bookings and generate reports

# 9. Challenges Faced

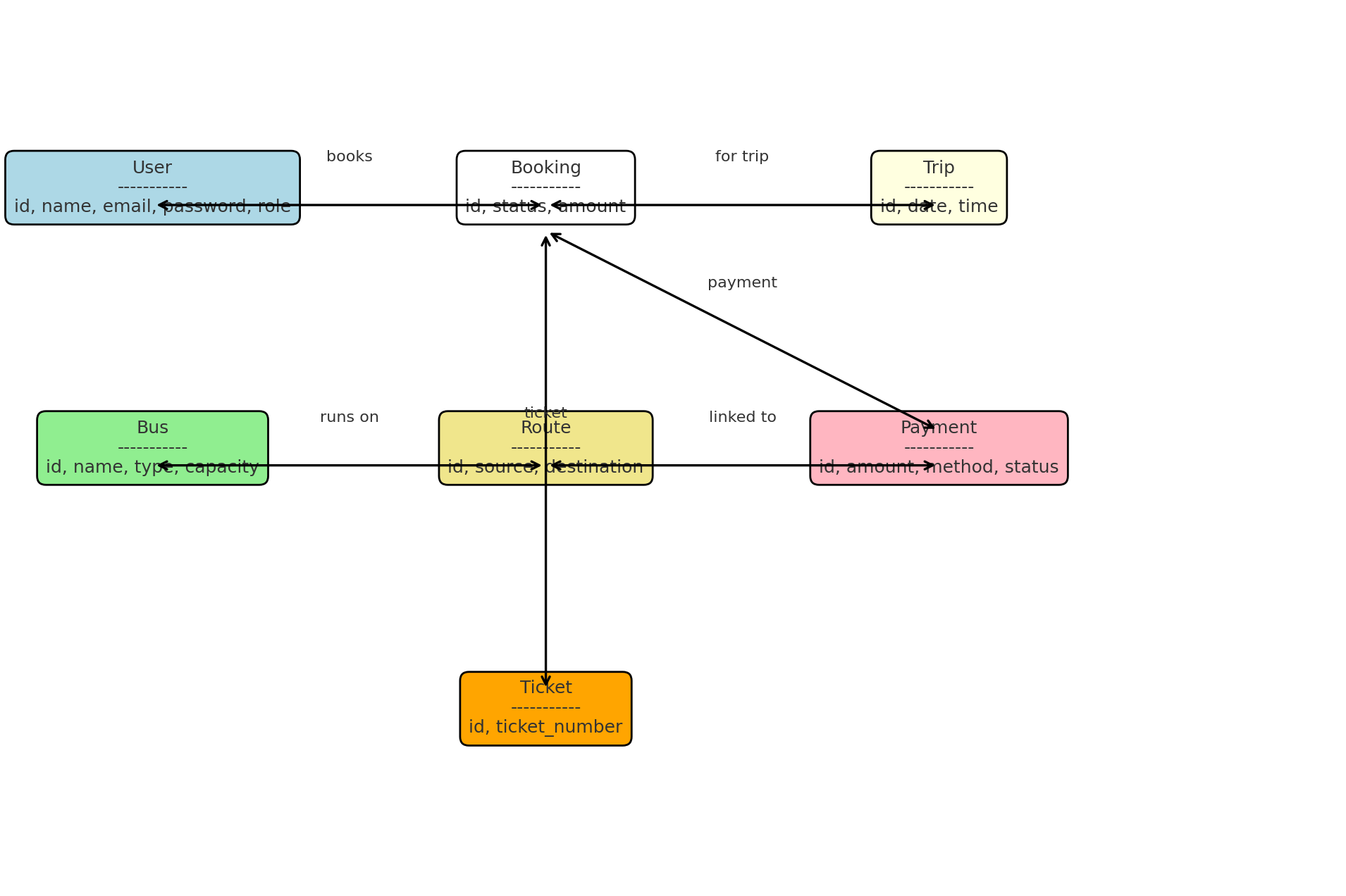
Controllers: Handling multiple endpoints for booking and trip logic.  
JWT: Issues with token validation and Spring Security filter.  
Frontend-Backend Connectivity: CORS issues between ports, async handling.  
Swagger: Integration difficulties with Spring Security, frequent unauthorized errors.

# 10. Conclusion & Next Steps

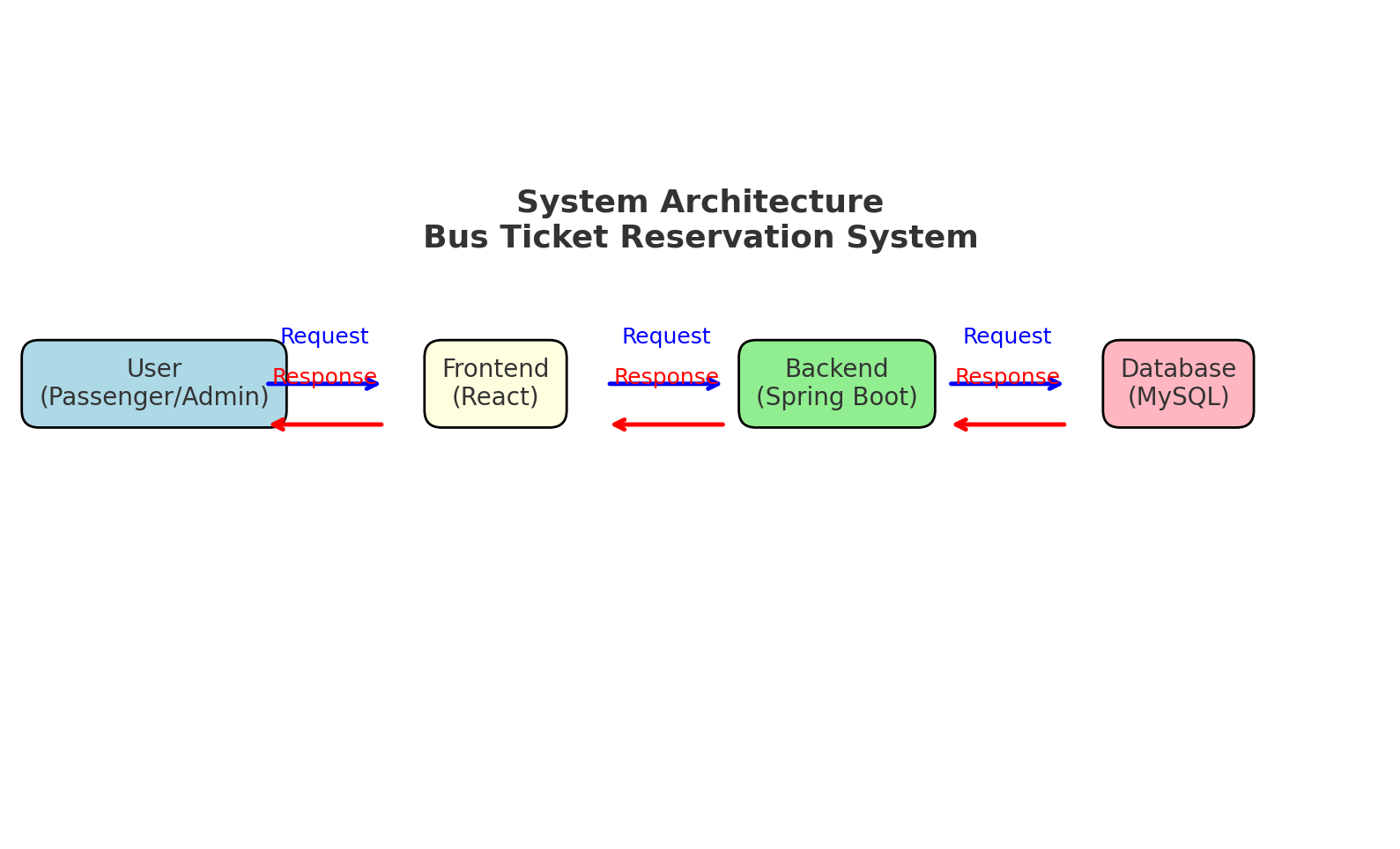
Achievements:  
- Functional ticket booking system with JWT security.  
- Real-time seat tracking.  
- Swagger API documentation.  
  
Next Steps:  
- Integrate real payment gateways.  
- Add email/SMS notifications.  
- Extend with mobile apps.  
- Cloud deployment for scalability.

# Figures

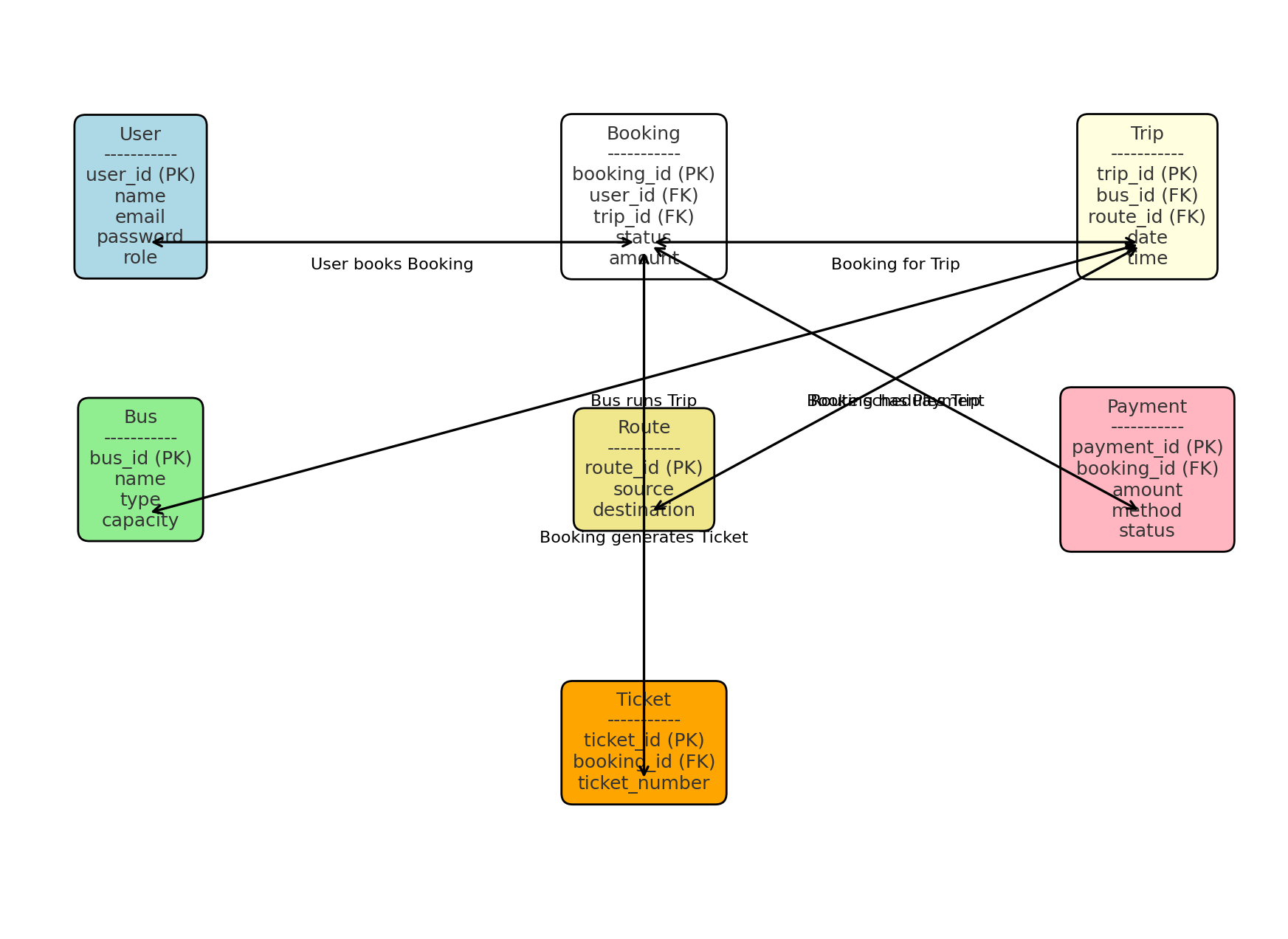
## Figure 1: Use Case Diagram



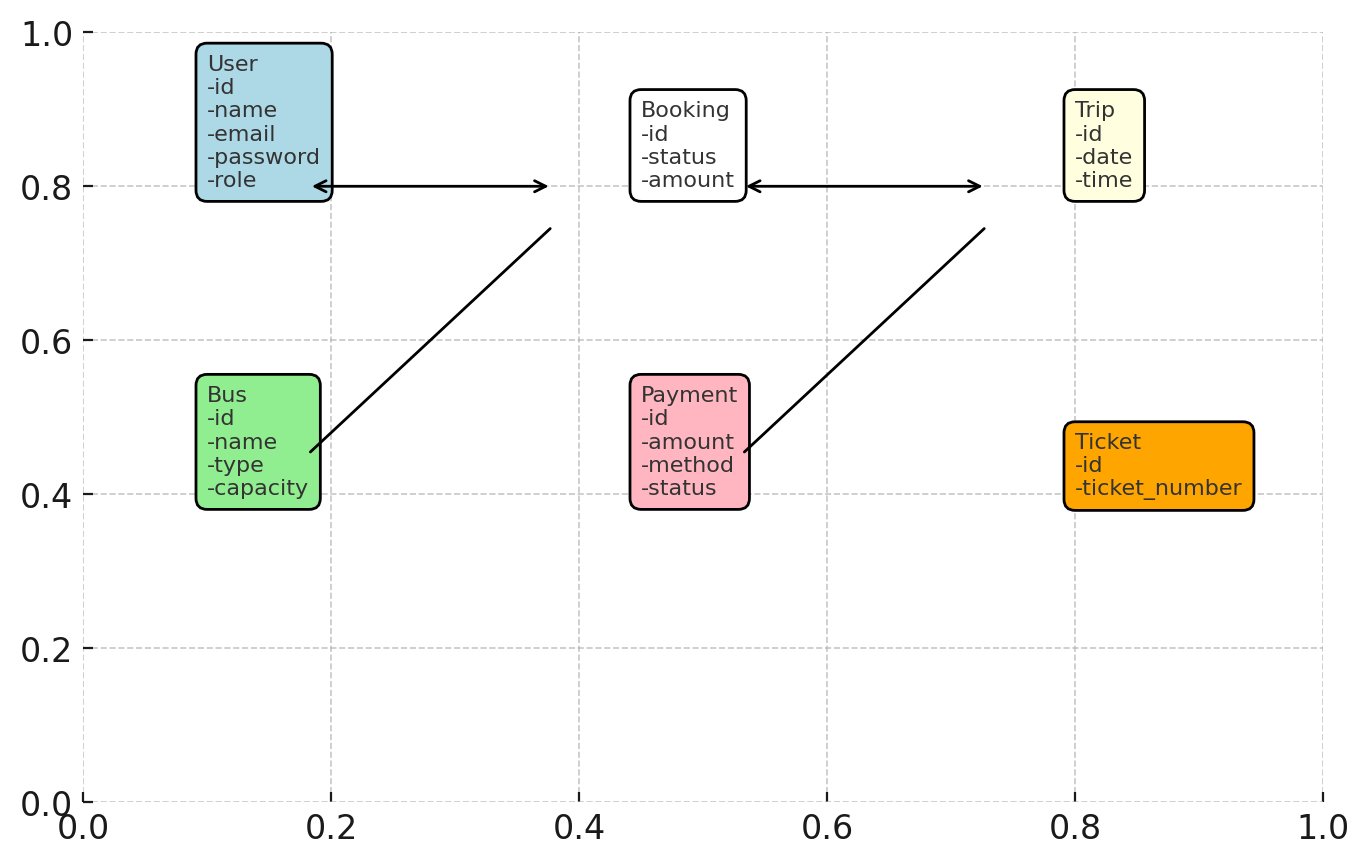
## Figure 2: System Architecture



## Figure 3: Entity-Relationship Diagram



## Figure 4: UML Class Diagram



## Figure 5: Booking Flow Sequence Diagram

