**EXPERIMENT 5**

**Document the SRS for the Railway Reservation System**

A **Software Requirements Specification (SRS)** for a **Railway Reservation System (RRS)** is a formal document that defines the functional and non-functional requirements of the system. It provides a detailed description of the system's purpose, scope, features, constraints, and user interactions.

**1. Introduction**

**1.1 Purpose**

The purpose of the Railway Reservation System (RRS) is to provide an efficient, user-friendly platform for passengers to search for trains, book tickets, manage reservations, and receive real-time updates. This system aims to enhance the operational efficiency of railway stations and operators by automating the ticketing and reservation process while ensuring a seamless travel experience for passengers.

**1.2 Scope**

The RRS will facilitate:

* Train search and booking for passengers.
* Real-time seat availability updates.
* Secure payment processing for ticket reservations.
* Reservation management for passengers and administrators.
* Train schedule and pricing management for administrators.
* Notifications regarding booking status, train delays, and cancellations.

The system will support web and mobile platforms, ensuring accessibility across different devices.

**1.3 Definitions, Acronyms, and Abbreviations**

* **RRS**: Railway Reservation System
* **PNR**: Passenger Name Record
* **UI**: User Interface
* **API**: Application Programming Interface
* **CDN**: Content Delivery Network
* **SSL**: Secure Socket Layer

**1.4 References**

* ISO/IEC 25010: Software Quality Requirements and Evaluation
* Payment Gateway API Documentation (Stripe, PayPal)

**1.5 Overview**

This document outlines the functional and non-functional requirements of the RRS, including design considerations, software dependencies, and system architecture.

**2. Overall Description**

**2.1 Product Perspective**

The RRS is a standalone web and mobile-based system that integrates with external payment gateways and notification services. It follows a client-server architecture where users interact through the client interface while backend servers manage data processing and storage.

**2.2 User Classes and Characteristics**

* **Passengers**: Search trains, book tickets, manage reservations, and receive notifications.
* **Administrators**: Manage train schedules, ticket pricing, monitor bookings, and oversee system performance.

**2.3 Operating Environment**

* **Frontend**: Web (React.js, Angular) and Mobile (React Native, Flutter)
* **Backend**: Java (Spring Boot), Python (Django/Flask), or Node.js (Express)
* **Database**: MySQL, PostgreSQL, or MongoDB
* **Hosting**: AWS, Google Cloud, Microsoft Azure

**2.4 Design and Implementation Constraints**

* The system must be scalable to handle peak traffic loads.
* Payment transactions should comply with security standards (PCI DSS).
* Authentication should be implemented using OAuth 2.0.

**2.5 Assumptions and Dependencies**

* Users have access to an internet connection.
* Third-party payment gateways and notification services are available.

**3. Functional Requirements**

**3.1 Passenger-Side Features**

* **Train Search**: Search for trains based on source, destination, date, and class.
* **Seat Availability**: Display real-time seat availability.
* **Payment Gateway Integration**: Secure transactions through Stripe/PayPal.
* **Ticket Issuance**: Generate an e-ticket with a unique PNR upon payment.
* **Notifications**: Send booking confirmations, cancellations, and train updates via SMS/Email.

**3.2 Administrator-Side Features**

* **Train Schedule Management**: Add, update, or delete train schedules.
* **Ticket Pricing Management**: Modify pricing based on class, route, and availability.
* **Reservation Monitoring**: Manage passenger bookings and waitlist.

**4. Non-Functional Requirements**

* **Scalability**: Handle a large number of users and concurrent bookings.
* **Security**: Protect user data using SSL encryption and OAuth authentication.
* **Performance**: Ensure low-latency responses for booking transactions.
* **Availability**: 99.9% system uptime with cloud hosting.
* **Usability**: Intuitive UI with minimal booking steps.

**5. System Design**

**5.1 Architecture**

* **Client-Server Model**: Frontend communicates with backend servers via RESTful APIs.
* **Modular Design**: Divided into modules for user management, train schedules, bookings, payments, and notifications.

**5.2 Database Design**

* **Relational Database**: Tables for users, trains, bookings, payments, and schedules.

**5.3 UI Design Considerations**

* **Responsive Design**: Works across desktop and mobile.
* **Notifications**: Display pop-ups for booking confirmations and train delays.

**6. External Interface Requirements**

**6.1 User Interfaces**

* **Web Interface**: HTML, CSS, JavaScript (React.js, Angular)
* **Mobile App**: React Native or Flutter

**6.2 Hardware Interfaces**

* Standard computing devices (PCs, smartphones, kiosks)

**6.3 Software Interfaces**

* **Payment Gateways**: Stripe, PayPal
* **Notification Services**: Twilio, Firebase Cloud Messaging

**7. Algorithm and Workflow**

**7.1 Booking Algorithm**

1. **Start**
2. User logs in.
3. Inputs search criteria (source, destination, date).
4. System fetches available trains.
5. User selects a train and checks seat availability.
6. If seats are available, proceeds to enter passenger details.
7. User chooses a payment method and completes payment.
8. If payment is successful, an e-ticket with a unique PNR is generated.
9. System sends a booking confirmation notification.
10. End

**Booking Flowchart:**

