

Ex No:6

Date:

## RAILWAY TICKET BOOKING

### AIM:

To design the E-Ticketing System, do testing and generate code using Argo UML Tools.

### PROBLEM ANALYSIS AND PROJECT PLANNING:

In the Railway Ticket Booking system, the primary process involves an applicant logging into the database. The database then verifies the entered username and password. After successful verification, the user is required to fill in their personal details. Subsequently, the user selects the desired train and the database reserves the ticket. Once the ticket is booked, it is sent to the applicant. Additionally, the user can search for trains or cancel the booking process.

### PROBLEM STATEMENT:

The Railway Ticket Booking system serves as the foundational requirement for developing a project that addresses the mechanism of online railway ticket booking.

- Requirements are analyzed and refined to ensure that end-users can efficiently navigate and use the Railway Ticket Booking system.
- The entire project is developed based on a comprehensive analysis, which includes defining the scope and preparing the project statement.
- The primary objective of this project is to enable applicants to reserve railway tickets.
- Initially, the applicant needs to log in to the database. Following that, they must provide their personal details.
- The database then searches for available tickets, or the user has the option to cancel the booking if they no longer require it.

### SYSTEM REQUIREMENT SPECIFICATION:

1. Introduction
1.1 Purpose
1.2 Scope
2. Functional Requirements
2.1 User Authentication
2.2 Train Search and Availability
2.3 Ticket Booking
2.4 Ticket Cancellation
2.5 Booking Management
3. Non-Functional Requirements

3.1 Performance
3.2 Security
3.3 Reliability
3.4 Scalability
4. Constraints
5. Glossary
6. Software Requirements
6.1 Server-side Software:
6.2 Client-side Software:
7. Hardware Requirements 7.1 Server-side Hardware:
7.2 Client-side Hardware:
8. Interface Requirements
8.1 User Interface (UI):
8.2 Application Programming Interface (API):
8.3 External Interfaces:
8.4 Accessibility Requirements:
9 Technologies to be Used:
10 System Functions:
11 Assumptions and Dependencies:
12 Tools to be Used:

## **1. Introduction**

The Railway E-Ticketing System is designed to facilitate the booking and management of train tickets electronically. This system aims to provide users with a seamless and convenient way to book, cancel, and manage their train tickets online.

### **1.1 Purpose**

The purpose of this document is to outline the functional and non-functional requirements of the Railway E-Ticketing System, providing a clear understanding of the system's features and constraints for developers, testers, and stakeholders.

### **1.2 Scope**

The Railway E-Ticketing System will allow users to perform the following functions:

- Register and authenticate as passengers.
- Search for trains and check seat availability.
- Book train tickets.
- Cancel booked tickets.
- View and manage booking history.

Receive email and SMS notifications for booking confirmations and cancellations.

## **2. Functional Requirements**

### **2.1 User Authentication**

The system shall allow users to register with valid credentials (name, email, phone number, etc.).

Users shall be able to log in securely using their registered email/phone number and password. Passwords shall be securely stored using encryption techniques.

### **2.2 Train Search and Availability**

Users shall be able to search for trains by entering source and destination stations, along with the preferred travel date.

The system shall display a list of available trains matching the search criteria. Users shall be able to view seat availability for each train.

### **2.3 Ticket Booking**

Users shall be able to select a train and book tickets by specifying the number of passengers and their details.

The system shall calculate the fare based on the selected train, class, and number of passengers. Users shall be able to make payment securely using various payment methods (credit/debit card, net banking, etc.).

Upon successful booking, the system shall generate a unique booking reference number and send confirmation via email and SMS.

### **2.4 Ticket Cancellation**

Users shall be able to cancel booked tickets before the departure time of the train. The system shall refund the appropriate amount to the user's account based on the cancellation policy.

### **2.5 Booking Management**

Users shall be able to view and manage their booking history.

The system shall allow users to reprint tickets if necessary.

Users shall have the option to update their profile information.

## **3. Non-Functional Requirements**

### **3.1 Performance**

The system shall be able to handle concurrent user requests efficiently without significant performance degradation.

Response time for booking and cancellation operations shall be within acceptable limits (e.g., < 5 seconds).

### **3.2 Security**

The system shall implement appropriate security measures to protect user data and transactions. Secure Socket Layer (SSL) encryption shall be used for all data transmissions. User authentication and authorization mechanisms shall be implemented to prevent unauthorized access.

### **3.3 Reliability**

The system shall be available 24/7 except during scheduled maintenance windows. Data integrity shall be maintained to ensure accurate booking and payment transactions.

### **3.4 Scalability**

The system architecture shall be designed to scale horizontally to accommodate increased user traffic.

The user interface shall be intuitive and user-friendly, catering to users with varying levels of technical expertise.

Error messages shall be informative and guide users to resolve issues effectively.

## **4. Constraints**

The system shall comply with the regulations and standards set by the railway authorities. Integration with payment gateways and SMS/email service providers is required for transaction processing and notifications.

## **5. Glossary**

SRS: Software Requirements Specification

SSL: Secure Socket Layer

API: Application Programming Interface

This Software Requirements Specification document outlines the key functionalities, performance requirements, security measures, and constraints of the Railway E-Ticketing System. It serves as a comprehensive guide for the development team to build a robust and user-friendly system that meets the needs of both passengers and railway authorities.

## **6. Software Requirements**

### **6.1 Server-side Software:**

Operating System: Linux (e.g., Ubuntu Server)

Web Server: Apache or Nginx

Database Management System: MySQL or PostgreSQL

Application Framework: Node.js with Express.js

Programming Languages: JavaScript (Node.js), SQL Payment Gateway

Integration: Stripe, PayPal, etc.

Email Service Provider: SMTP (e.g., Gmail SMTP) for sending booking notifications

SMS Service Provider: Twilio, Nexmo, etc. for sending booking notifications via SMS

## **6.2 Client-side Software:**

Web Browser: Google Chrome, Mozilla Firefox, Safari, etc.

Programming Languages: HTML5, CSS3, JavaScript (React.js for dynamic front-end)

## **7. Hardware Requirements**

### **7.1 Server-side Hardware:**

Processor: Quad-core processor or higher

RAM: 8GB or more

Storage: 100GB SSD or higher for database and application files

Network: Ethernet connection with high-speed internet access

Backup System: Regular backups to ensure data integrity and availability

### **7.2 Client-side Hardware:**

Desktop/Laptop/Smartphone/Tablet with internet connectivity

Minimum screen resolution: 1024x768 pixels

## **8. Interface Requirements**

### **8.1 User Interface (UI):**

The user interface shall be responsive and compatible with various devices (desktops, laptops, tablets, smartphones).

Clear and intuitive navigation menus for easy access to different features.

Interactive forms for user registration, login, ticket booking, and cancellation.

Real-time updates for seat availability and booking status.

Consistent layout and design elements for a cohesive user experience across the application.

### **8.2 Application Programming Interface (API):**

RESTful API endpoints for communication between client-side and server-side components.

Well-defined API documentation specifying request/response formats, authentication mechanisms, and error handling procedures.

Secure communication using HTTPS protocol for data exchange between the client and server.

### **8.3 External Interfaces:**

Integration with payment gateways for secure payment processing.

Integration with email and SMS service providers for sending booking notifications to users. Integration with railway databases or APIs for retrieving train schedules, seat availability, and fare information.

#### **8.4 Accessibility Requirements:**

Compliance with Web Content Accessibility Guidelines (WCAG) for ensuring accessibility to users with disabilities.

Support for screen readers, keyboard navigation, and other assistive technologies. This section outlines the software, hardware, and interface requirements necessary for the development and deployment of the Railway E-Ticketing System. These requirements are essential for ensuring the system's functionality, performance, and accessibility across different platforms and devices.

#### **9 Technologies to be Used:**

Front-end: React.js for building interactive UI components.

Back-end: Node.js for server-side logic and Express.js as the web application framework.

Database: MongoDB for storing user profiles, bookings, and other relevant data in a NoSQL format.

Authentication: JSON Web Tokens (JWT) for secure authentication and session management.

Payment Integration: APIs provided by Stripe, PayPal for secure online transactions.

Communication: Twilio for SMS notifications and Gmail SMTP for email notifications.

Version Control: Git for source code management and collaboration.

#### **10 System Functions:**

User Registration: Allows new users to create accounts with necessary personal information.

Profile Management: Enables users to update their personal details and preferences.

Search and Book: Allows users to search for trains, check availability, and book tickets.

Cancellation: Provides a feature for users to cancel their booked tickets.

Notification System: Sends email and SMS notifications for booking status updates and cancellations.

Booking History: Displays a record of past and upcoming bookings for users. Ticket Reprinting: Allows users to reprint their booked tickets if needed.

#### **11 Assumptions and Dependencies:**

Users have a stable internet connection to access the system.

External APIs for train schedules and seat availability are reliable and up-to-date. Payment gateways and email/SMS service providers maintain their API endpoints and services without major interruptions.

#### **12 Tools to be Used:**

Development IDE: Visual Studio Code or IntelliJ IDEA for coding and debugging.

Testing Frameworks: Jest for unit testing and Selenium for end-to-end testing.  
API Documentation: Swagger for documenting API endpoints and specifications.  
Project Management: JIRA or Trello for task tracking and Agile project management.

## UML DIAGRAMS:

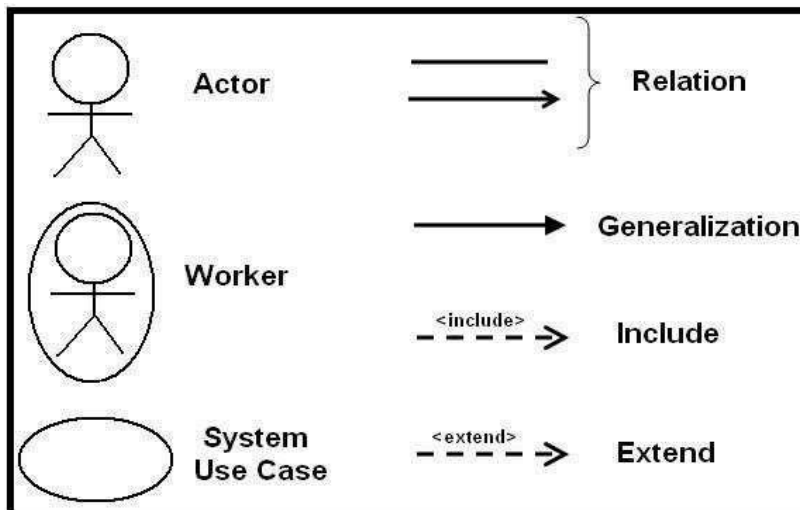
The following UML diagrams describe the process involved in the on-line recruitment System:

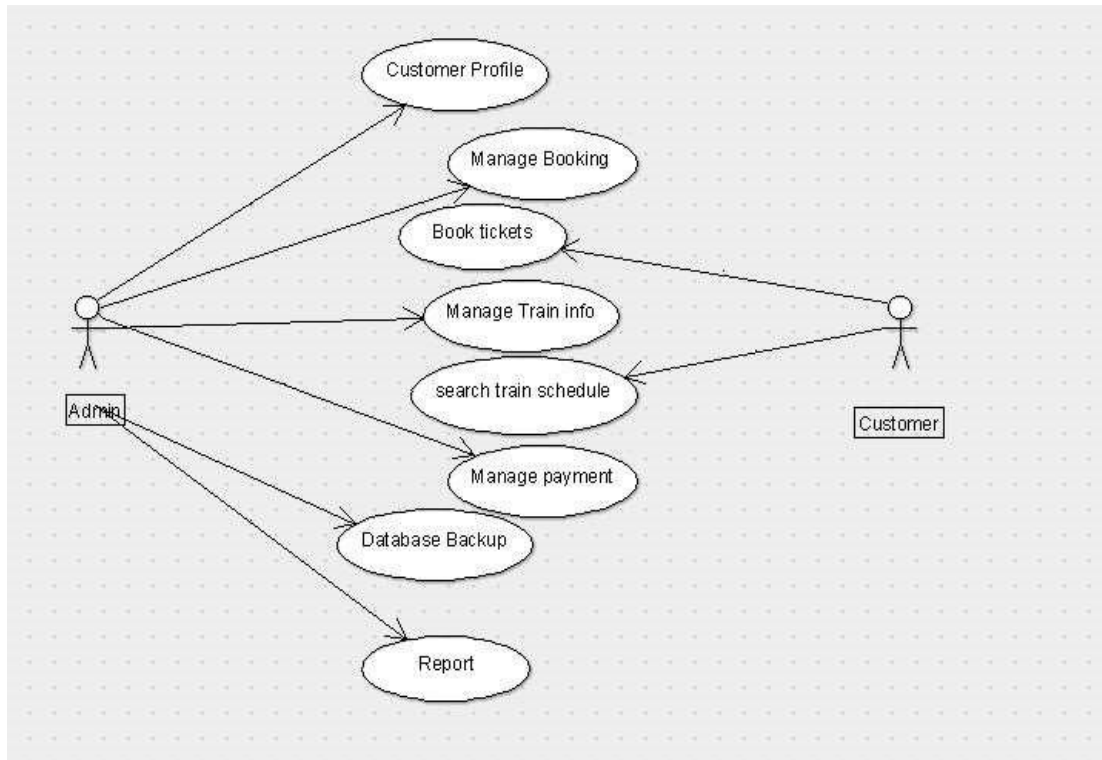
- Use case diagram
- Class diagram
- Sequence diagram
- Collaboration diagram
- State chart diagram
- Activity diagram
- Package diagram

### Use case diagram:

The railway ticket booking use case diagram depicts how passengers interact with the ticketing system. It involves two main components: the Passenger and the Ticketing System. The Passenger initiates the process by selecting a train, providing personal details, and making payment. The Ticketing System, in turn, checks seat availability, generates the ticket, and issues a confirmation. These actions are encapsulated in individual use cases like "Book Ticket," "Check Availability," "Make Payment," etc.

### Notation:

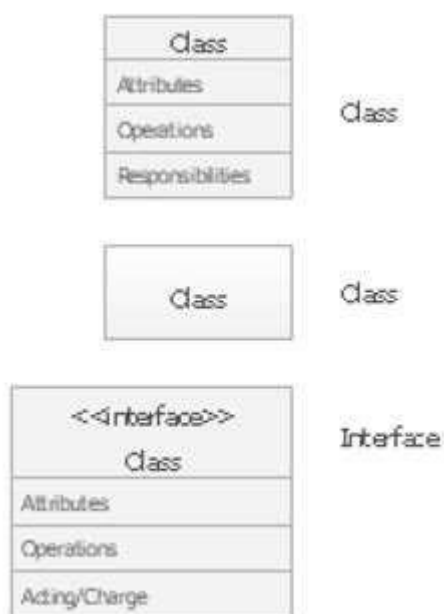




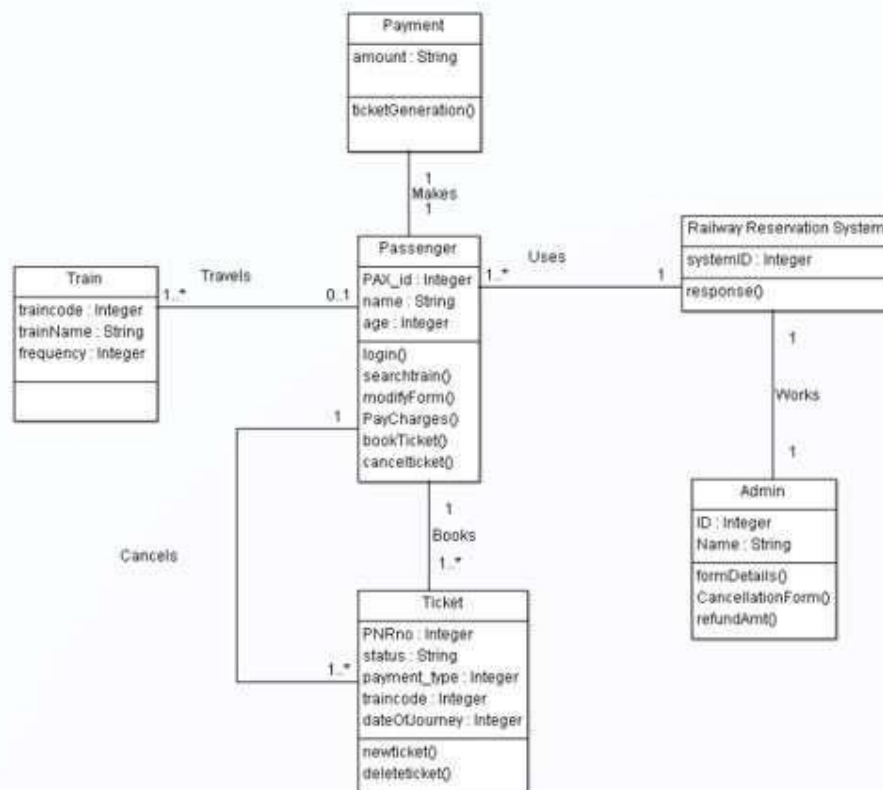
### Class diagram:

The class diagram for railway ticket booking includes classes like Passenger, Train, Ticket, and TicketingSystem. Each class represents a key entity in the system, such as passengers, trains, tickets, and the ticketing system itself. These classes have attributes and methods that define their properties and behaviors. Relationships like associations and aggregations illustrate how these classes interact, forming the structure of the ticket booking system.

### Notation:





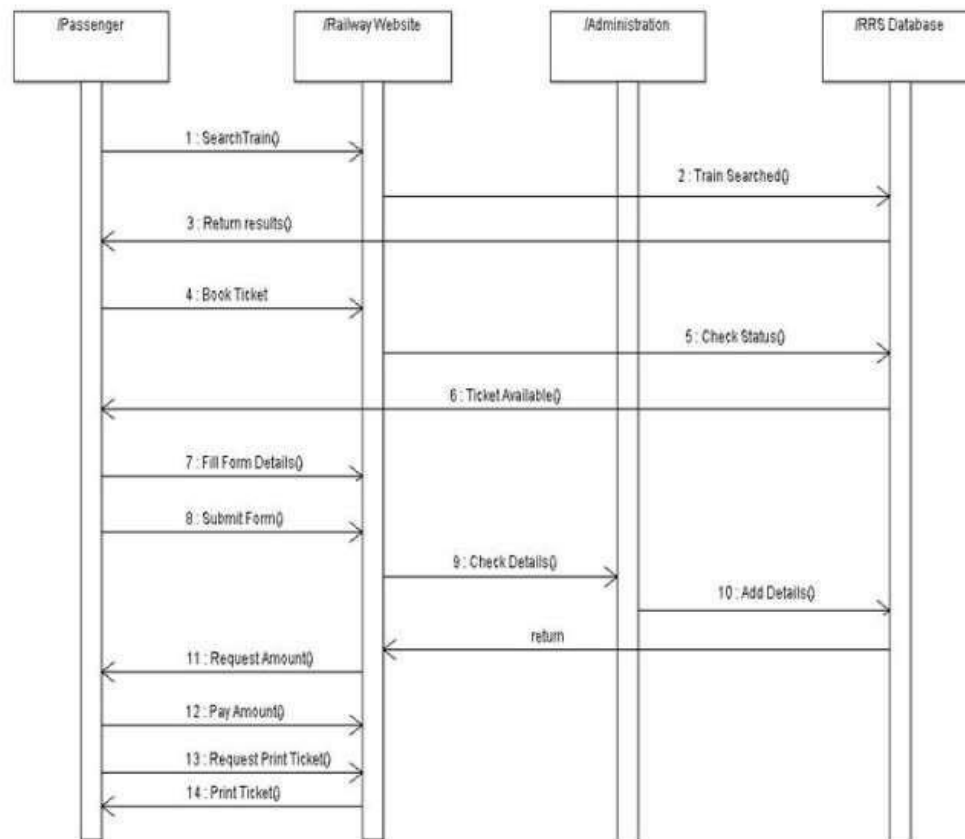


### Sequence diagram:

The sequence diagram for railway ticket booking outlines the step-by-step interactions between the Passenger and the Ticketing System. It illustrates how the Passenger requests a ticket, the Ticketing System checks availability, the Passenger provides personal details, makes payment, and finally, the Ticketing System issues the ticket. Each interaction is depicted as a message exchanged between the Passenger and the Ticketing System, highlighting the sequence of events in the booking process.

### Notation:

Select		Initiator
Object		Active object
Timing constraint		Note
Vertex		Nested message
Flat message		Asynchronous message
Return message		Note connector
In scope region		Object termination

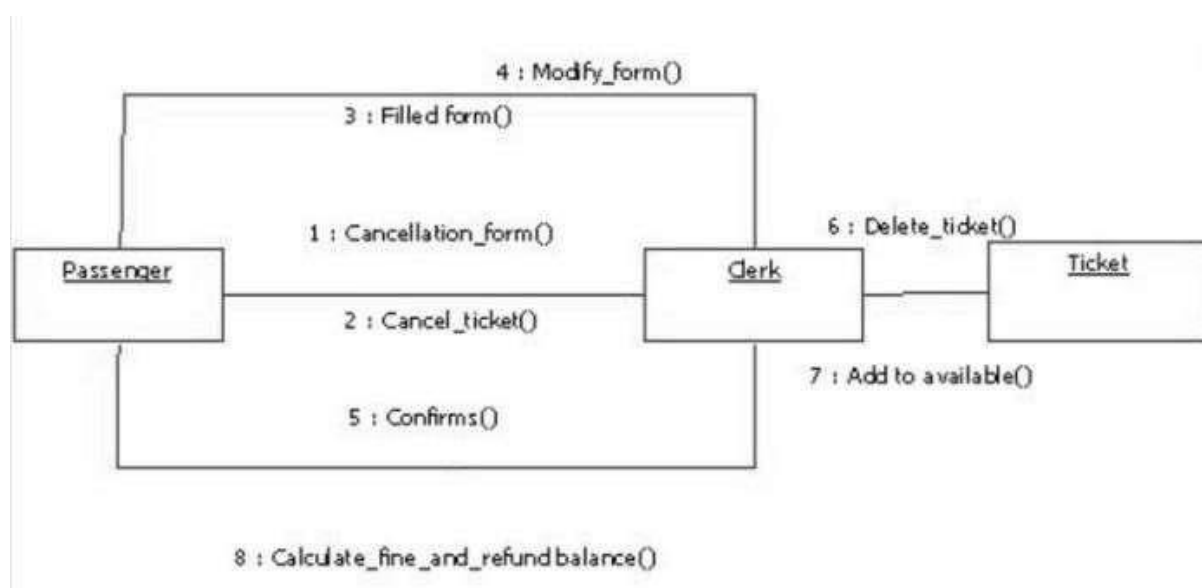


### Collaboration diagram:

In a collaboration diagram for a train ticket booking system, several objects interact to facilitate the booking process. The User selects a Train, providing necessary details to the Ticket object. This Ticket object collaborates with Payment to reserve a seat and initiate payment. Once payment is completed, Confirmation confirms the booking, prompting the Ticket object to issue the ticket to the User. This sequence illustrates how the User, Train, Ticket, Payment, and Confirmation objects collaborate to achieve successful train ticket booking.

### Notation:

Select			Actor
Instance			Multi object
Active object			N-ary link
Note			Note connector
N-ary link connector			Vertex
Link			Qualified link
Aggregation link			Qualified aggr. link
Composite link			Qualified compos. link
Nested msg.			Nested msg.
Flat msg.			Flat msg.
Asynchronous msg.			Asynchronous msg.



### State diagram:

In the state chart diagram for the train ticket booking system, the process begins in an Idle state. Upon user interaction, it moves to Train Selection, followed by Details Entry for user information. The system then transitions to Seat Reservation and subsequently to Payment Process. If payment is successful, it moves to Confirmation for ticket issuance. However, payment failure or cancellation leads to the Cancelled state. After ticket issuance, the system returns to Idle, ready for the next booking.

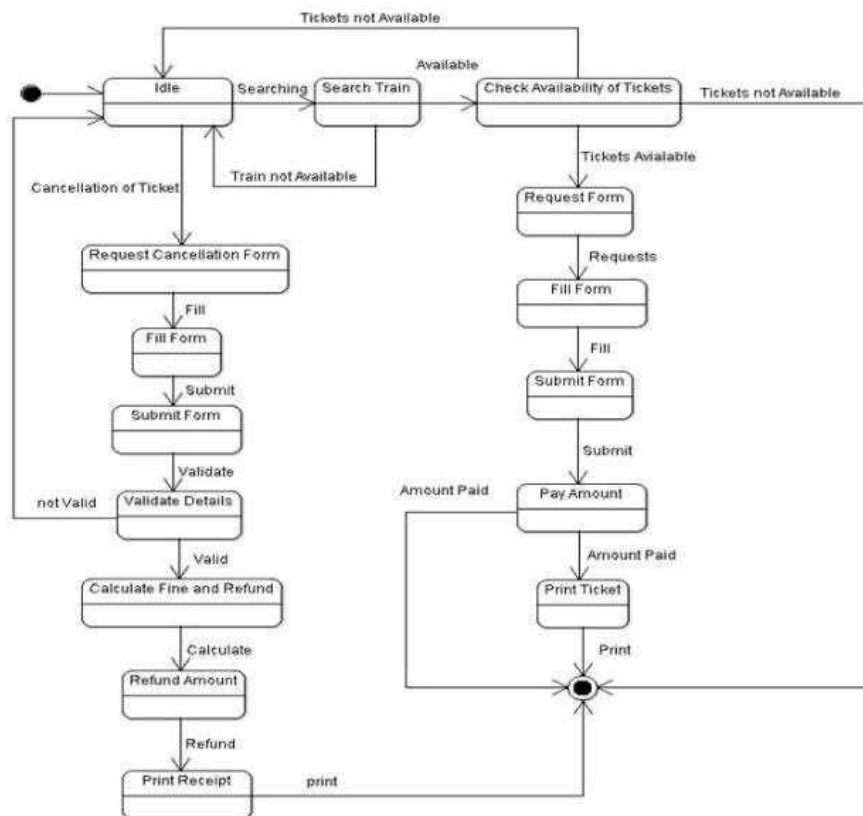
### Notation:

● initial state

State1 state-box

◇ decision-box

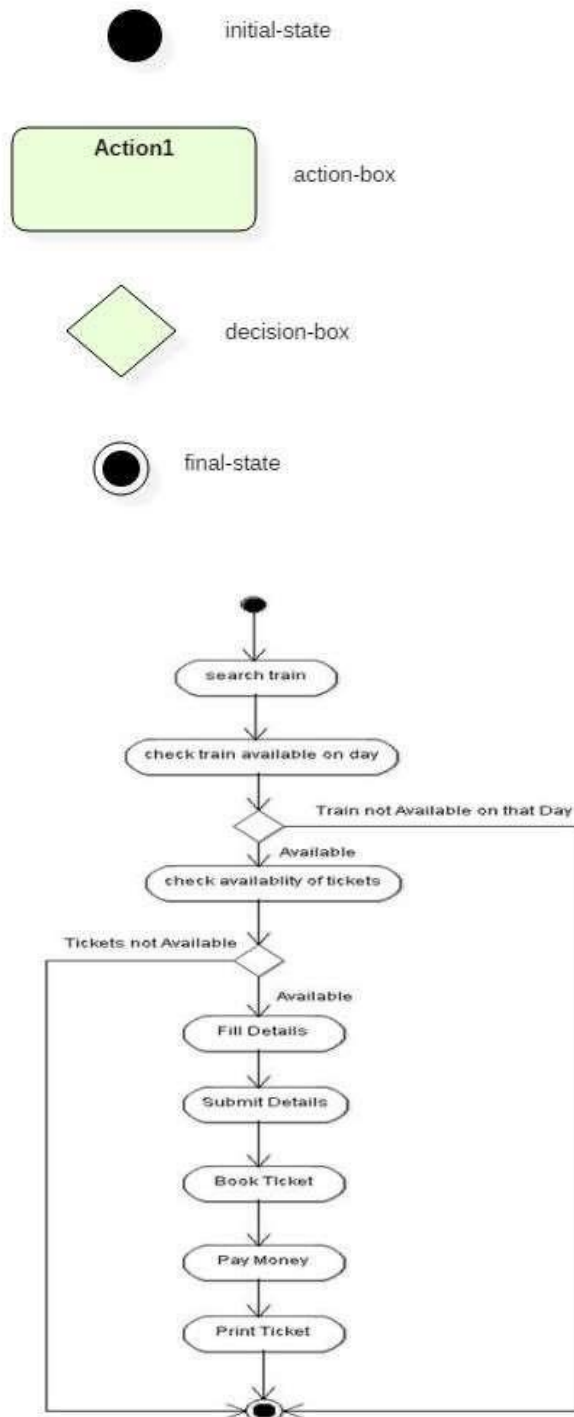
⦿ final-state



### Activity diagram:

In an activity diagram for railway ticket booking, the process starts with the user selecting a train. This is followed by entering personal and journey details. Once details are provided, the system reserves a seat or berth. Then, the user proceeds to make a payment. If the payment is successful, the booking is confirmed, and a ticket is issued. If payment fails, the process ends, or the user can retry payment. After ticket issuance or in case of cancellation, the activity concludes, and the system is ready for a new booking.

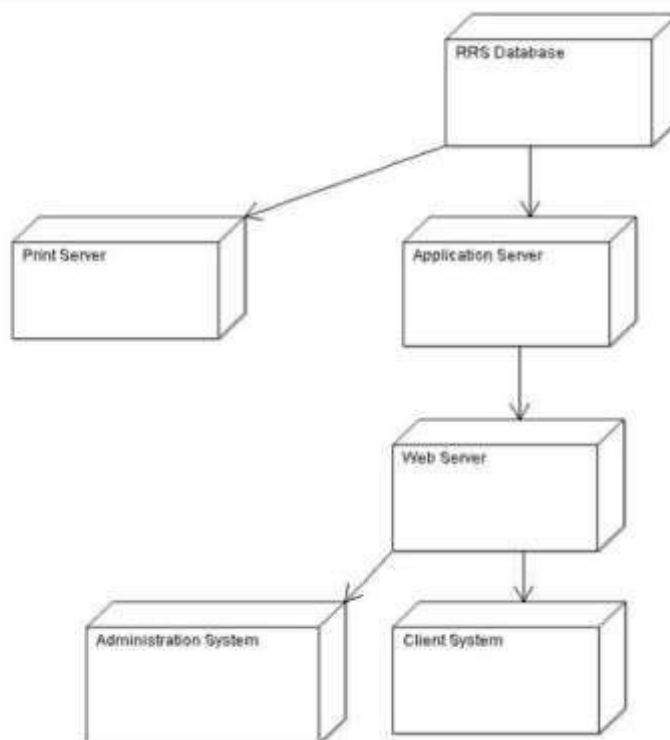
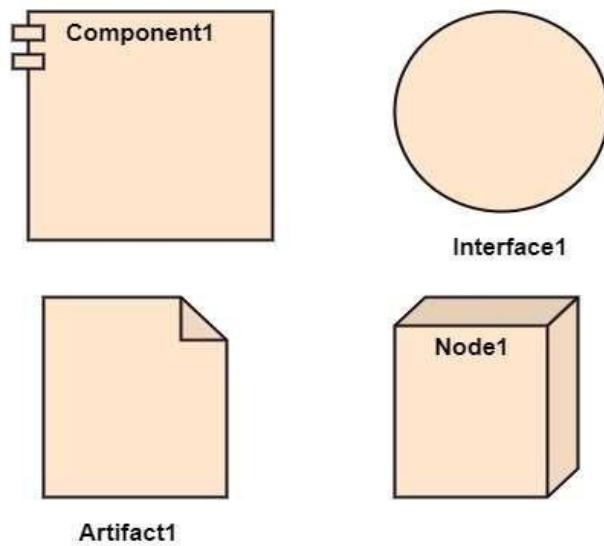
### Notation:



### Deployment Diagram:

In a deployment diagram for railway ticket booking, the system is typically deployed across multiple nodes or servers. The User Interface node handles user interactions, allowing users to browse trains and enter details. The Application Server node processes these requests, handles seat reservations, and initiates payment processes. The Database Server node stores train schedules, user details, and booking information. Lastly, the Payment Gateway node facilitates secure transactions. All these nodes are interconnected, ensuring smooth communication and data flow to enable efficient railway ticket bookings.

### Notification:



### Code Generation:

STEPS TO GENERATE CODE USING ARGO UML:

Step 1: Draw Class diagram with all its notations.

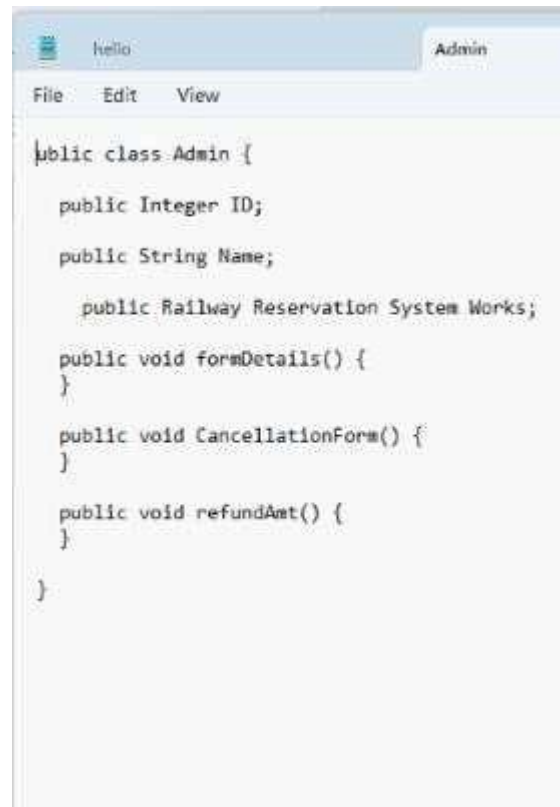
Step 2: Select all the Classes.

Step 3: Choose Generation menu for code generation.

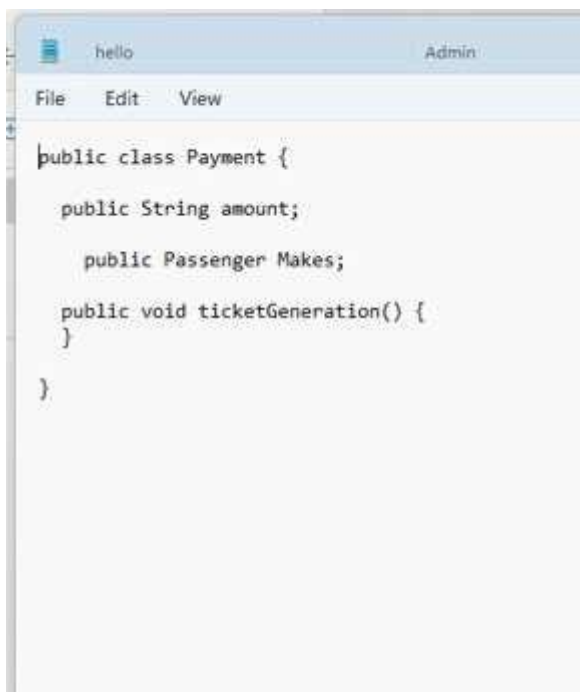
Step 4: Choose the required classes    Step 5:  
Browse the location to locate the code.  
Step 6: Apply generate.



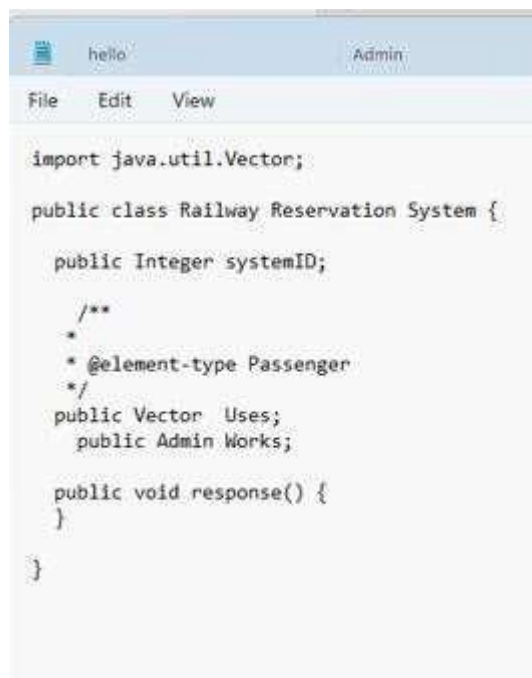
```
public class Train {  
    public Integer traincode;  
    public String trainName;  
    public Integer frequency;  
    public Passenger Travels;  
}
```



```
public class Admin {  
    public Integer ID;  
    public String Name;  
    public Railway Reservation System Works;  
    public void formDetails() {  
    }  
    public void CancellationForm() {  
    }  
    public void refundAmt() {  
    }  
}
```



```
public class Payment {  
    public String amount;  
    public Passenger Makes;  
    public void ticketGeneration() {  
    }  
}
```



```
import java.util.Vector;  
public class Railway Reservation System {  
    public Integer systemID;  
    /**  
     * @element-type Passenger  
     */  
    public Vector Uses;  
    public Admin Works;  
    public void response() {  
    }  
}
```

A screenshot of a Java IDE window. The title bar shows 'hello', 'Admin', and 'Pas'. The menu bar has 'File', 'Edit', and 'View'. The code editor contains the following Java code:

```
import java.util.Vector;

public class Ticket {

    public Integer PNRno;
    public String status;
    private Integer payment_type;
    public Integer traincode;
    public Integer dateOfJourney;

    public Passenger Books;
    public Vector myPassenger;
    public Passenger Cancels;

    public void newticket() {
    }

    public void deleteticket() {
    }

}
```

### Software testing:

Software testing is a crucial process in the development lifecycle of any software application, encompassing various techniques and methodologies to ensure its quality, reliability, and functionality. It involves systematically evaluating the software to identify defects, errors, or bugs that could impact its performance, security, or user experience. Software testing aims to validate that the software meets the specified requirements, works as intended, and delivers value to its users. Types of

Testing: • Integration testing

- Unit testing

- Black box testing

- White box testing

- Functional testing

- System testing

- Stress testing

- Performance testing

- Acceptance testing

- Regression testing

- Beta testing



**Integration Testing:**

It is conducted to evaluate the interaction between integrated units or components of a software system. The main aim is to detect any interface defects between these units or components.

**White Box Testing:**

Unlike black-box testing, in white-box testing, the tester has knowledge of the internal workings of the software being tested. It is also known as clear-box or transparent testing.

**Functional Testing:**

This type of testing verifies that each function of the software application operates in conformance with the requirement specification.

**System Testing:**

System testing tests the entire software system, as a whole. It ensures that all components integrated together as expected, and the system meets the specified requirements.

**Stress Testing:**

Stress testing is a type of software testing that determines the robustness of software by testing it under extreme conditions. It involves testing beyond normal operational capacity, often to a breaking point, to observe the results.

**Performance Testing:**

Performance testing evaluates the speed, responsiveness, and stability of a system under a particular workload. The main goal is to determine the speed, responsiveness, and stability of a system under a specific workload.

**Acceptance Testing:**

This testing is performed to determine if the software system has met the required specifications and is accepted by the customer or end-users.

**Regression Testing:**

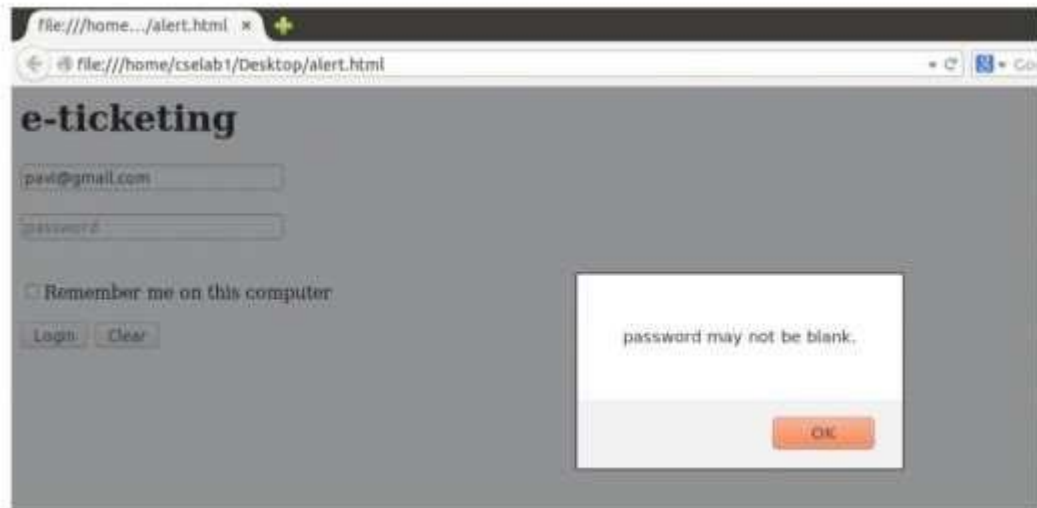
Regression testing is carried out to ensure that a recent program or code change has not adversely affected existing features.

**Beta Testing:**

Beta testing is the final testing done by end-users. It allows users to provide feedback about the product before the full launch.

**Unit Testing:**

Unit testing involves testing individual units or components of a software application in isolation from the rest of the system. The goal is to validate that each unit of the software performs as designed.



### **Black Box Testing:**

This testing technique focuses solely on the functionality of the software. Testers are not concerned with the internal workings of the system; they test the software based on its specifications.



### **RESULT:**

Thus the domain E-Ticketing System was designed, tested and the code was generated successfully using Argo UML tools.