Railway Ticket Booking System

# 1. Introduction to Project

• Problem Statement: Define the issues with traditional ticket booking methods, such as long waiting times, lack of convenience, and limited accessibility.  
• Scope: The system will automate the booking and cancellation process, manage user information, and integrate payment options to provide a seamless experience.  
• Objectives: Outline the main goals, such as making ticket booking easier and faster, minimizing human intervention, and ensuring secure payment processing.

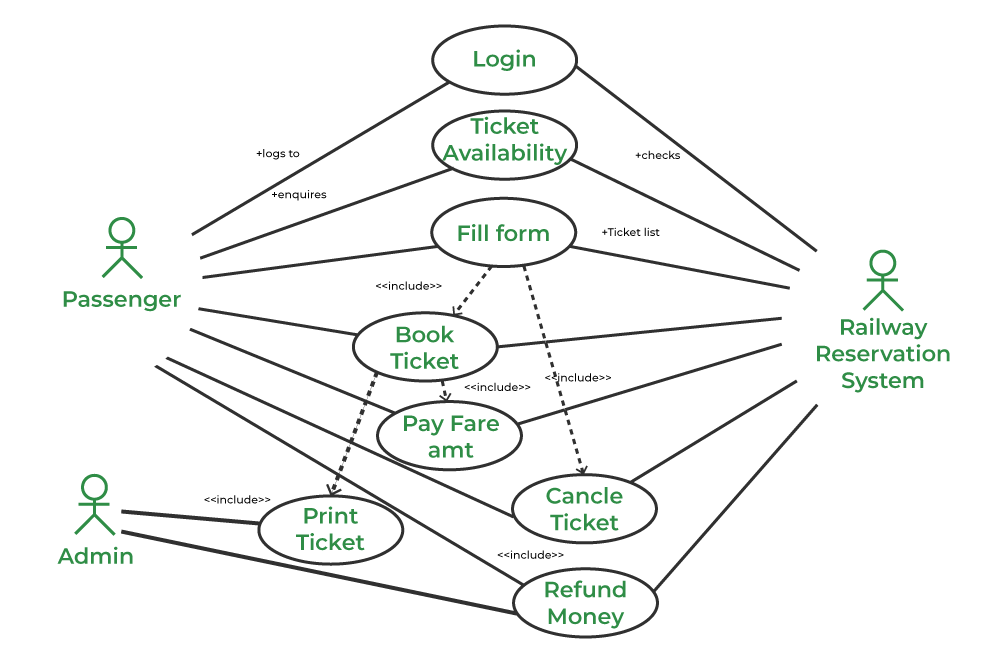
# 2. Requirements

• Hardware Requirements:  
 - Computer or server with internet access for managing the application.  
 - Printer (optional) for printing physical tickets.  
  
• Software Requirements:  
 - Operating System: Windows, Linux, or macOS.  
 - Database Management System: MySQL or PostgreSQL for storing user, ticket, and transaction data.  
 - Development Language: Python or Java,c++ for backend development, HTML/CSS/JavaScript for the frontend.  
  
• Functional Requirements:  
 - User Authentication: Registration and login for new and existing users.  
 - Ticket Booking and Cancellation: Allows users to search, book, and cancel tickets.  
 - Payment Processing: Integration with payment gateways for secure online payments.  
 - Admin Module: Admin functionalities for managing routes, schedules, and viewing transaction history.

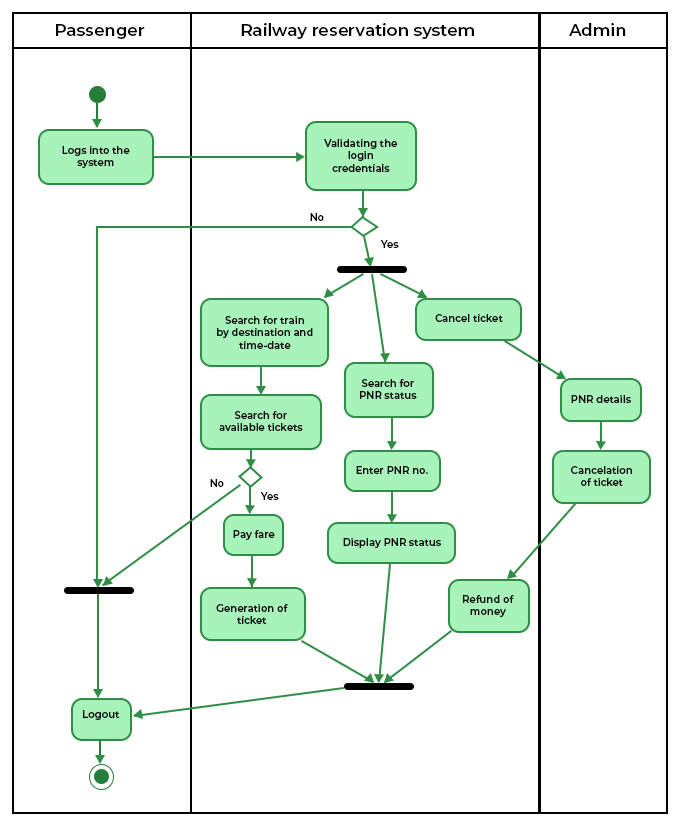
# 3. Diagrams

• Use Case Diagram: Visual representation of user interactions with the system (e.g., login, book ticket, cancel ticket, make payment).  
• Modular Diagram: Depicts different modules like user management, ticket management, and payment processing.  
• Activity Diagram: Shows the flow of activities in the system, such as the ticket booking process from search to payment confirmation.

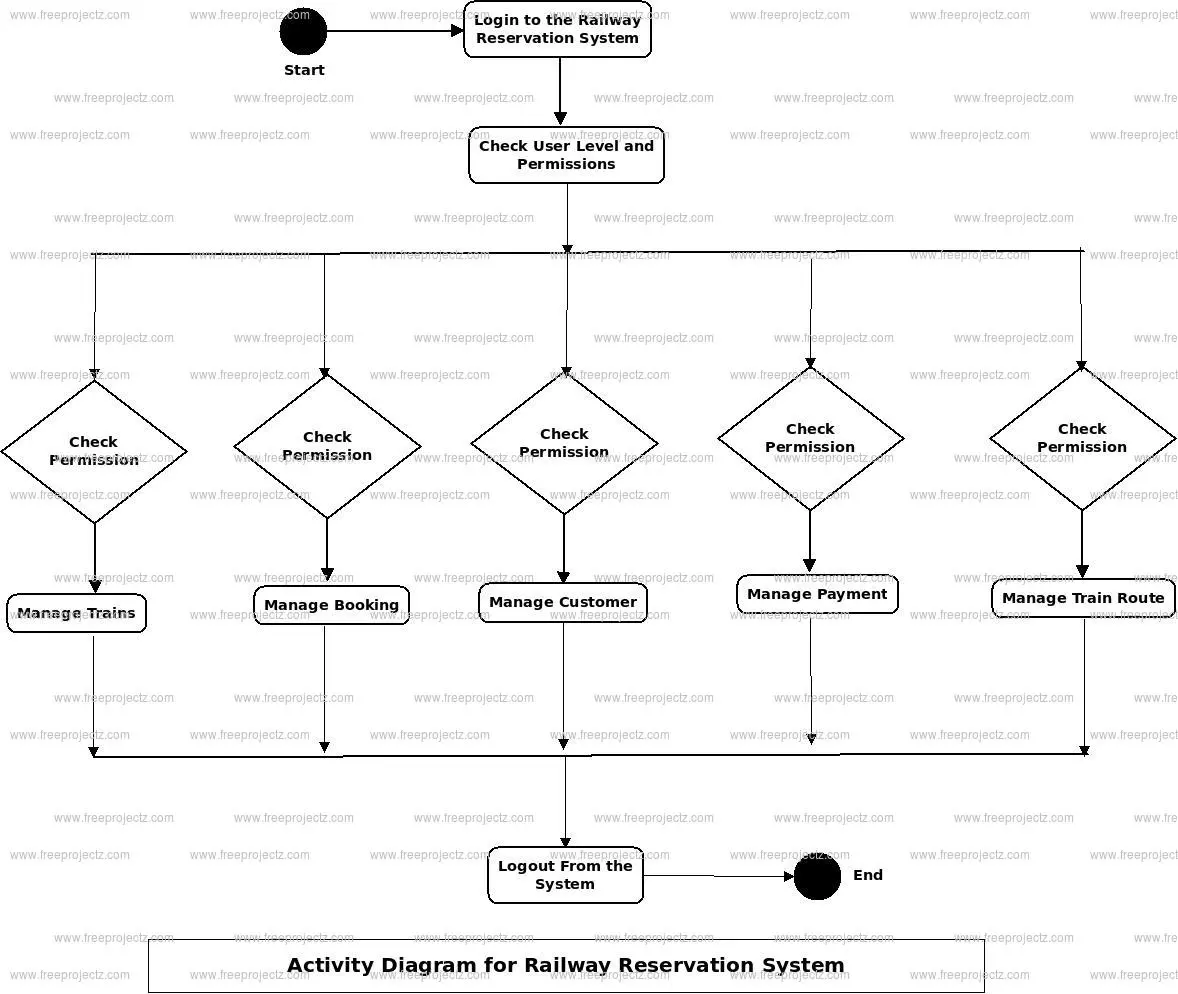
1.Use case diagram



2.modular diagram



3.Activity diagram



# 4. Design / Interface

• User Interface Design: Create mockups for key interfaces like the login page, ticket search page, booking confirmation page, and admin dashboard.  
• Navigation Flow: Plan the navigation between pages/modules (e.g., after login, the user is directed to the home page with search and booking options).

# 5. Implementation

• Setup: Explain the installation and configuration steps for the development environment (e.g., installing Python/Java, setting up MySQL, etc.).  
• Database Structure: Design tables for storing user information, ticket details, transaction logs, and schedules.  
• Code Overview: Briefly describe how each module functions, such as user authentication, booking logic, and integration with payment APIs.

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class Train {

public:

int trainNumber;

string trainName;

string source;

string destination;

int seatsAvailable;

Train(int trainNumber, string trainName, string source, string destination, int seatsAvailable) {

this->trainNumber = trainNumber;

this->trainName = trainName;

this->source = source;

this->destination = destination;

this->seatsAvailable = seatsAvailable;

}

};

class Ticket {

public:

int ticketId;

string passengerName;

int trainNumber;

Ticket(int ticketId, string passengerName, int trainNumber) {

this->ticketId = ticketId;

this->passengerName = passengerName;

this->trainNumber = trainNumber;

}

};

vector<Train> trains;

vector<Ticket> tickets;

int ticketCounter = 1;

void addTrain(int trainNumber, string trainName, string source, string destination, int seatsAvailable) {

Train newTrain(trainNumber, trainName, source, destination, seatsAvailable);

trains.push\_back(newTrain);

cout << "Train added successfully!" << endl;

}

void bookTicket(string passengerName, int trainNumber) {

for (auto& train : trains) {

if (train.trainNumber == trainNumber) {

if (train.seatsAvailable > 0) {

Ticket newTicket(ticketCounter++, passengerName, trainNumber);

tickets.push\_back(newTicket);

train.seatsAvailable--;

cout << "Ticket booked successfully! Ticket ID: " << newTicket.ticketId << endl;

return;

} else {

cout << "No seats available on this train." << endl;

return;

}

}

}

cout << "Train not found." << endl;

}

void viewBookedTickets() {

if (tickets.empty()) {

cout << "No tickets booked." << endl;

return;

}

for (const auto& ticket : tickets) {

cout << "Ticket ID: " << ticket.ticketId << ", Passenger Name: " << ticket.passengerName << ", Train Number: " << ticket.trainNumber << endl;

}

}

int main() {

int choice;

while (true) {

cout << "\n1. Add Train\n2. Book Ticket\n3. View Booked Tickets\n4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

int trainNumber, seatsAvailable;

string trainName, source, destination;

cout << "Enter Train Number: ";

cin >> trainNumber;

cout << "Enter Train Name: ";

cin.ignore();

getline(cin, trainName);

cout << "Enter Source: ";

getline(cin, source);

cout << "Enter Destination: ";

getline(cin, destination);

cout << "Enter Seats Available: ";

cin >> seatsAvailable;

addTrain(trainNumber, trainName, source, destination, seatsAvailable);

break;

}

case 2: {

string passengerName;

int trainNumber;

cout << "Enter Passenger Name: ";

cin.ignore();

getline(cin, passengerName);

cout << "Enter Train Number: ";

cin >> trainNumber;

bookTicket(passengerName, trainNumber);

break;

}

case 3:

viewBookedTickets();

break;

case 4:

return 0;

default:

cout << "Invalid choice. Please try again." << endl;

}

}

}

# 6. Output

1. Add Train

2. Book Ticket

3. View Booked Tickets

4. Exit

Enter your choice: 1

Enter Train Number: 101

Enter Train Name: Express 1

Enter Source: City A

Enter Destination: City B

Enter Seats Available: 50

Train added successfully!

1. Add Train

2. Book Ticket

3. View Booked Tickets

4. Exit

Enter your choice: 2

Enter Passenger Name: John Doe

Enter Train Number: 101

Ticket booked successfully! Ticket ID: 1

1. Add Train

2. Book Ticket

3. View Booked Tickets

4. Exit

Enter your choice: 2

Enter Passenger Name: Jane Smith

Enter Train Number: 101

Ticket booked successfully! Ticket ID: 2

1. Add Train

2. Book Ticket

3. View Booked Tickets

4. Exit

Enter your choice: 3

Ticket ID: 1, Passenger Name: John Doe, Train Number: 101

Ticket ID: 2, Passenger Name: Jane Smith, Train Number: 101

1. Add Train

2. Book Ticket

3. View Booked Tickets

4. Exit

Enter your choice: 4

# 7. Conclusion

• Summary: Summarize the functionalities implemented in the system and how it addresses the problem statement.  
• Future Scope: Discuss potential enhancements, such as adding more payment options, providing mobile app support, or incorporating real-time train tracking.

# 8. References

• List any references used for research, such as books, online tutorials, and documentation for tools and technologies used (e.g., MySQL, Python libraries).