UE21CS352B - Object Oriented Analysis & Design using Java

Mini Project Report

"Book On Rails"

Submitted by:

Shreeja Rajesh	**PES1UG21CS564**
Shreya Mishra	**PES1UG21CS574**
Sinchana	**PES1UG21CS596**
Siri M M	**PES1UG21CS600**

*6th Semester **J** Section*

Team Number : J6

Prof. Vinay Joshi
Assistant Professor

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING

PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Table of Contents

Problem Statement

With the increasing demand for convenient transportation options, there is a pressing need for a comprehensive Train Ticket Booking Management System that caters to the diverse needs of both users and administrators. The primary objective of this project is to develop a robust and user-friendly **Book On Rails** using Java, focused on providing seamless booking experiences and efficient management functionalities. It includes both user-facing features for booking tickets and managing profiles, as well as administrative functionalities for overseeing system operations and ensuring data integrity.

Major Features

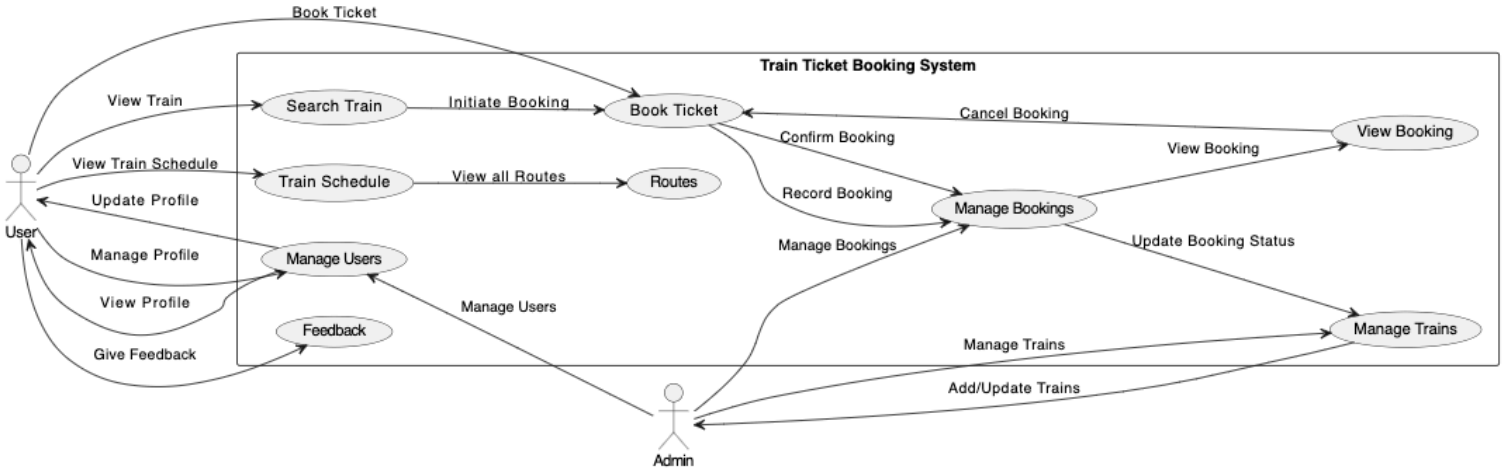
- **Search Trains**: Users can search for trains based on criteria such as class, station, and date, facilitating efficient journey planning.
- **Booking Ticket Online**: Seamless online booking experience featuring fare calculation, seat allocations, and instant booking confirmation.
- **User Management**: Allowing users to manage their profiles, view booking histories, and view their tickets as needed.
- **Admin Controls**: Empowers administrators to manage trains, user accounts, bookings, and seat availability, ensuring smooth system operations.

Minor Features

- **User Feedback and Ratings System**: Enables users to provide feedback and ratings for their travel experiences, enhancing overall user satisfaction and providing valuable insights for system improvement.
- **In-Train Food Menu**: Integrates an in-train food menu feature, allowing users to browse and select from the given food options (vegetarian and non-vegetarian) during their journey.
- **Viewing Train Schedules**: Allows users to easily access and view detailed schedules of trains, including information about all the stations they stop at, along with arrival and departure times, enhancing user convenience for trip planning.
- **Cancel Ticket**: Allows users to cancel their booked tickets, providing flexibility and convenience in adjusting travel plans.

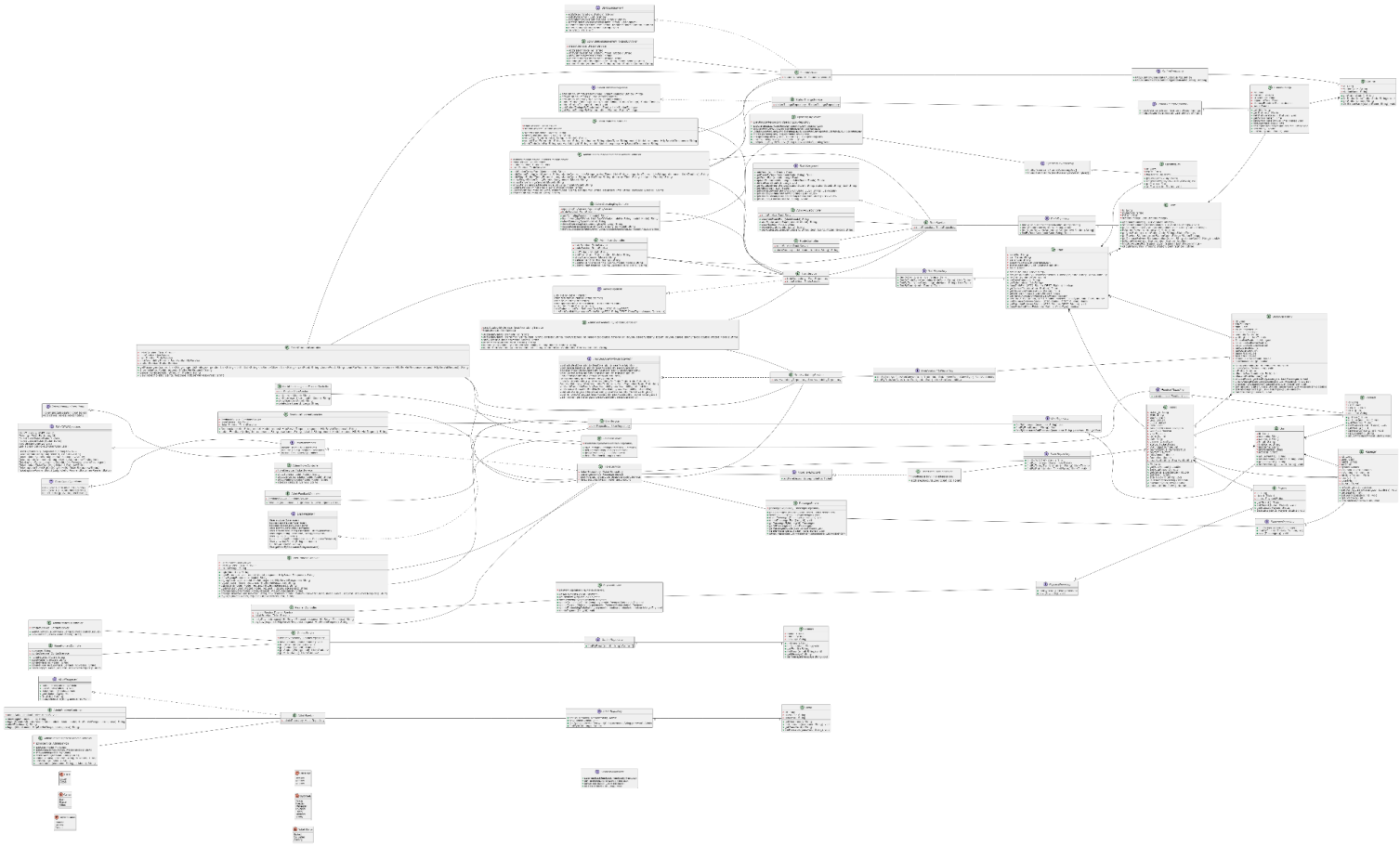
Models

Use Case Diagram



Class Diagram

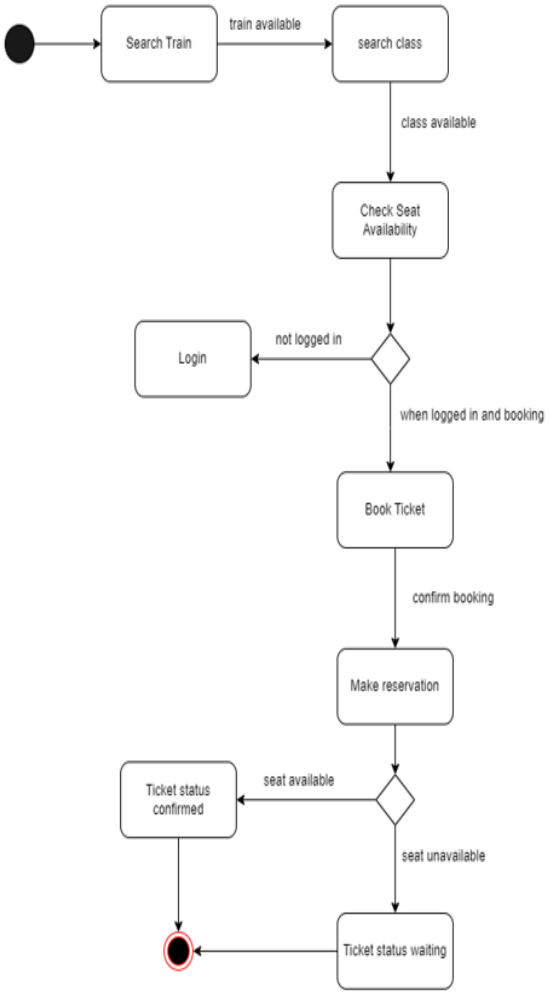
[ClassDiagram]



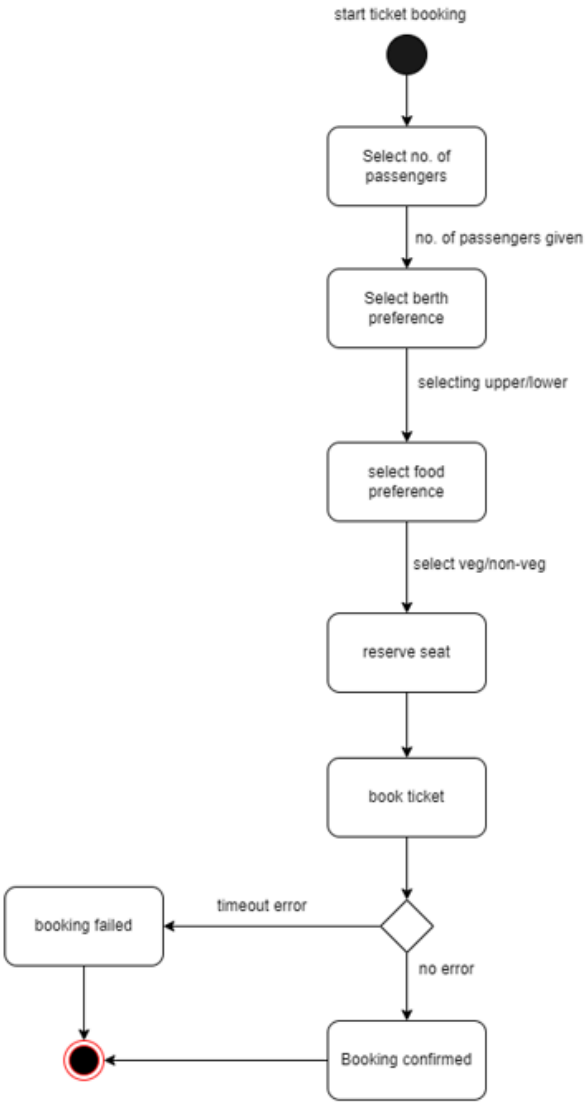
The class diagram illustrates a railway management system with various entities such as Admin, Train, Ticket, User, Feedback, and more, each encapsulating specific attributes and behaviors. These entities are interconnected through relationships like composition, aggregation, and association, facilitating data flow and system functionality. Additionally, the diagram outlines interfaces and services responsible for managing operations related to administration, feedback, ticketing, train management, and user interactions. Overall, it provides a comprehensive overview of the system's structure and functionalities.

State Diagrams:

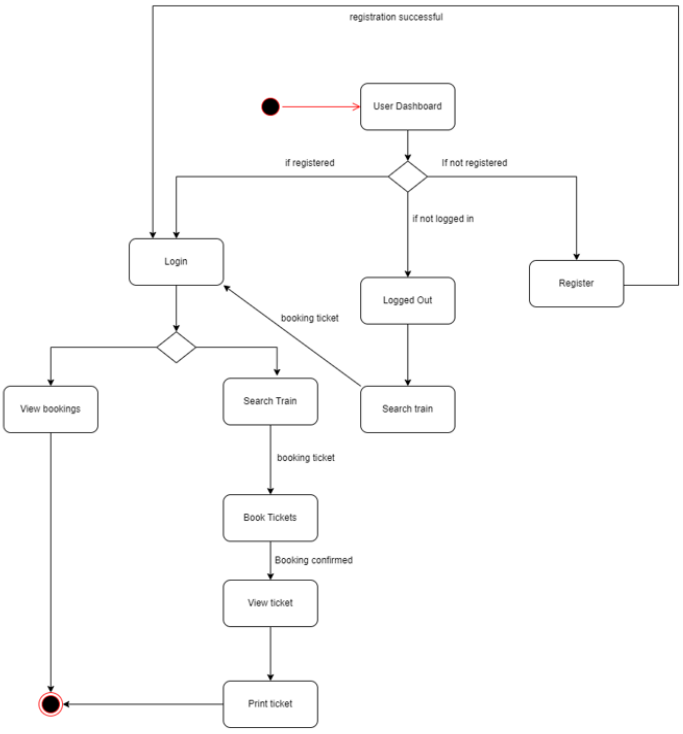
1. **Search Train:**



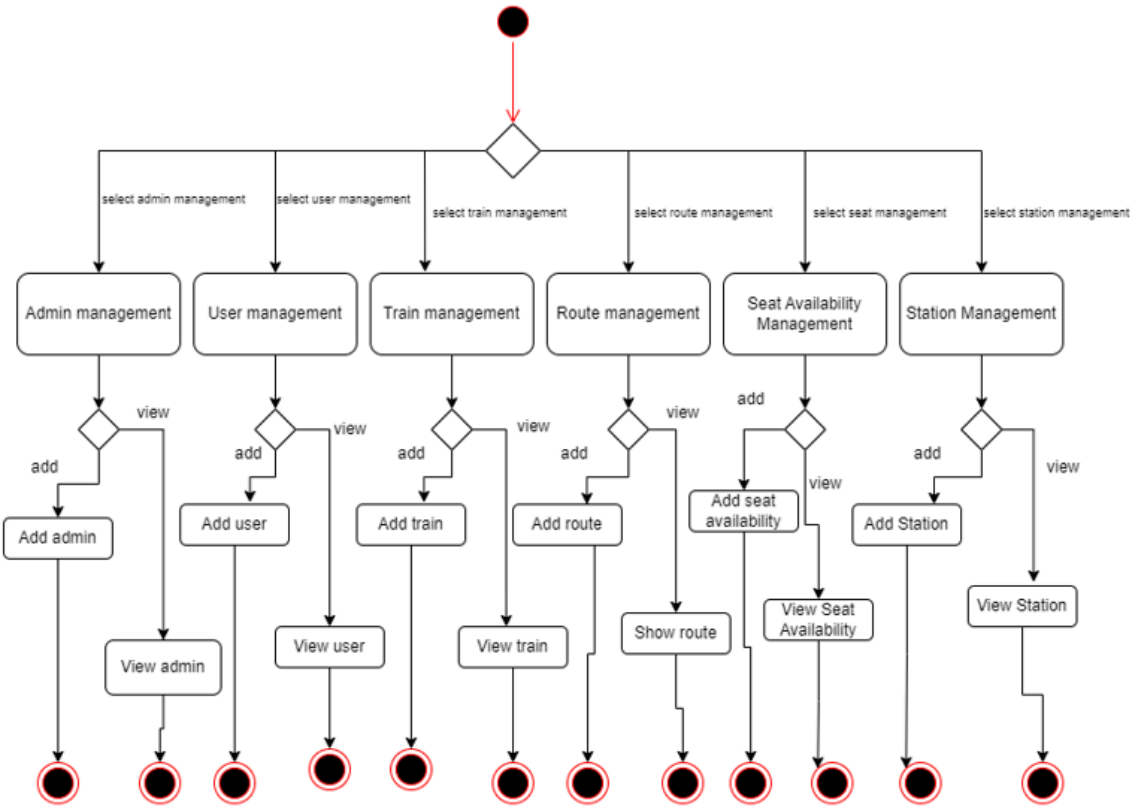
2. **Booking :**



3. User Management

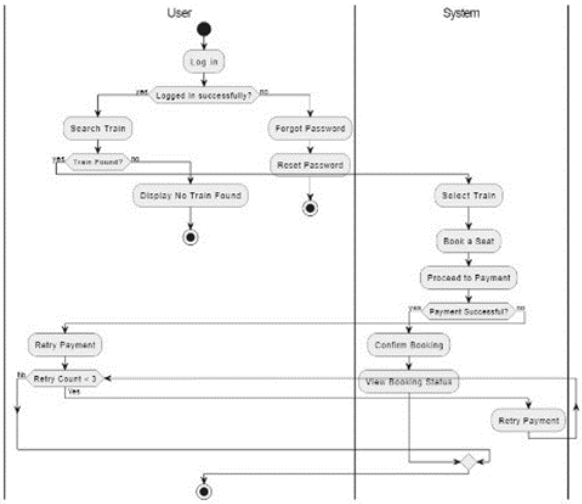


4. Admin Management

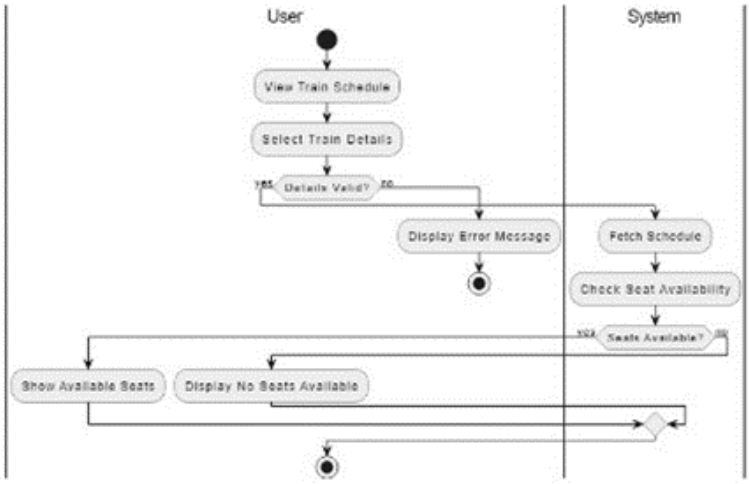


Activity Diagram

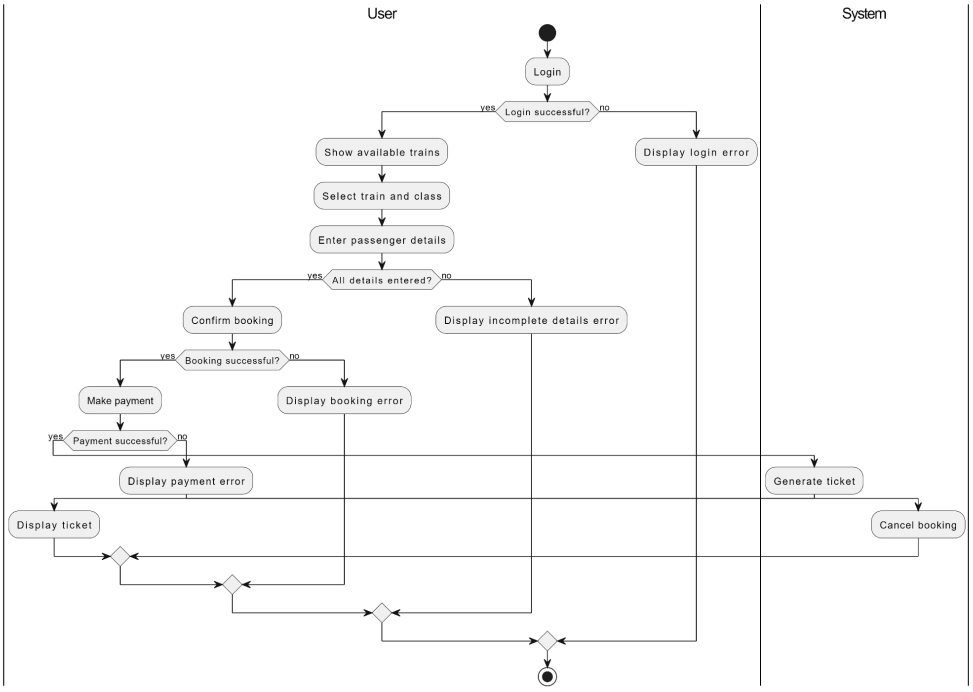
1. **Search Train:**



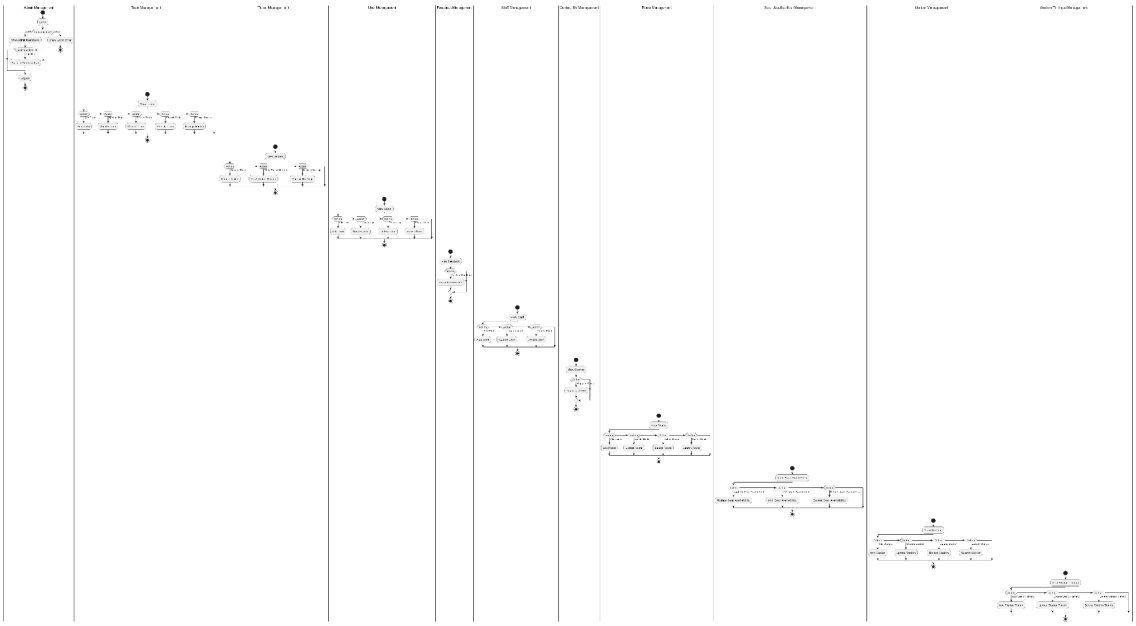
2.Train Schedule & Seat Availability:



3. **User Management**



4. **Admin Management**



Architecture Patterns

This project is a web application for managing train seat availability, we can map the MVC architecture as follows:

1. Model (Data):

- In this project, the Model represents the underlying data structure and business logic related to train seat availability. This includes entities like `SeatAvailability`, `Train`, and `Ticket`, along with their respective attributes and methods.
- The Model layer is responsible for managing data persistence, validation, and business rules related to seat availability, such as booking seats, managing waiting lists, and calculating fares.

2. View (User Interface):

- In this project, the View component utilizes JSP (JavaServer Pages) technology to create dynamic web pages that present data and interact with users. JSP pages contain a mix of HTML and Java code, allowing for the dynamic generation of content based on data retrieved from the Model layer.
- JSP pages are responsible for rendering the UI elements for viewing seat availability, booking tickets, and managing reservations. They integrate with server-side Java components, such as servlets and controllers, to retrieve data from the Model and process user requests.
- Views encapsulate the presentation logic and provide a user-friendly interface for interacting with the application.

3. Controller (Logic):

- The Controller component acts as an intermediary between the Model and View. It handles user requests, processes input data, interacts with the Model to retrieve or manipulate data, and determines which View to render in response to user actions.
- In this project, controllers like `AdminSeatAvailabilityFrontendController` receive HTTP requests, extract parameters from request payloads or URLs, invoke appropriate methods in the Model layer (e.g., `seatAvailabilityService`), and pass data to the View for rendering.

Example:

When a user accesses a JSP page to view seat availability (/admin/seat-availability/show.jsp), the server-side Controller (AdminSeatAvailabilityFrontendController) handles the request.
The Controller interacts with the Model layer (seatAvailabilityService) to retrieve seat availability data.
The retrieved data is passed to the JSP page, where it is dynamically rendered using HTML and JSP tags, presenting the available seats and allowing users to perform actions like booking or managing reservations.

Controller

```java
package com.bookonrails.ooad.FrontendController;

import java.sql.Date;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import com.bookonrails.ooad.Model.*;
import com.bookonrails.ooad.Service.SeatAvailabilityService;
import com.bookonrails.ooad.Service.TrainService;

@Controller
@RequestMapping(path = "/admin/seat-availability")
@CrossOrigin(origins = "*")
public class AdminSeatAvailabilityFrontendController {
    @Autowired
    private SeatAvailabilityService seatAvailabilityService;

    @Autowired
    private TrainService trainService;

    @GetMapping("/add")
    public String addSeatAvailability(Model m) {
        m.addAttribute(attributeName:"SeatAvailability", new SeatAvailability());
        return "admin/seat-availability/add";
    }

    @PostMapping("/add")
    public String addSeatAvailability(
            @RequestParam("trainNo") String trainNo,
            @RequestParam("date") String date,
            @RequestParam("classes") String classes,
            @RequestParam("no_of_coaches") int noOfCoaches,
            @RequestParam("basePrice") double basePrice,
            @RequestParam("farePerKM") double farePerKM,
            @RequestParam("seniorCitizenDiscount") double seniorCitizenDiscount,
            @RequestParam("cancellationCharge") double cancellationCharge,
            Model model) {
        SeatAvailability seatAvailability = new SeatAvailability(noOfCoaches);
        // get train
```

model

```java
package com.bookonrails.ooad.Model;

import java.util.ArrayList;
import java.sql.Date;
import java.util.List;

import jakarta.persistence.*;

@Entity
public class SeatAvailability {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "trainNo", nullable = false)
    private Train train;

    @Temporal(TemporalType.DATE)
    private Date date;

    private int no_of_coaches;
    @Enumerated(EnumType.STRING) // to store the type of coach as a string
    private ClassType classes;
    private int availableSeats;

    @OneToMany(mappedBy = "seatAvailability", cascade = CascadeType.ALL)
    private List<Ticket> waitingList;

    @ElementCollection
    @CollectionTable(name = "CancelledSeats", joinColumns = @JoinColumn(name = "seatAvailability_id"))
    @Column(name = "seat_number")
    private List<Integer> CancelledSeats; // check n%2 = 1 -> Lower, n%2 = 0 -> Upper

    private int lastUnbookedLowerSeat;
    private int lastUnbookedUpperSeat;
    private int lastLowerSeat;
    private int lastUpperSeat;

    private double basePrice;
    private double farePerKM;
```

view

```
<!DOCTYPE html>
<!-- Base for all html pages -->
<html xmlns:th="http://www.thymeleaf.org" lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" th:href="@{/main.css}">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">
    <!-- <script src="https://cdn.tailwindcss.com"></script> -->

    <title>Book On Rails</title>
</head>

<body>
    <header th:insert="~{admin-base :: header}"></header>

    <div class="content-body container-fluid">
        <div class="container mt-5">

            <!-- Block Content will be here -->
            <h2 class="mb-4">Seat Availability Form</h2>
            <div id="info" class="text-bold text-danger">

            </div>
            <form action="/admin/seat-availability/add" method="post">
                <!-- Train Number Input -->
                <div class="form-group">
                    <label for="trainNo">Train Number:</label>
                    <input type="text" class="form-control" id="trainNo" name="trainNo" required />
                </div>

                <!-- Date -->
                <div class="form-group">
                    <label for="date">Date:</label>
                    <input type="date" class="form-control" id="date" name="date" min="" max="" required />
                </div>
```

Design Principles

The SOLID principles are followed throughout the project, guaranteeing a solid and flexible software architecture:

1. **Single Responsibility Principle(SRP)**: Classes and modules display distinct and well-defined roles across the project. As an example, service classes like 'TicketCRUDOperations', 'TicketAllocationOperation' and 'TicketQueryOperation' follow the SRP and are devoted to particular functionality like ticket booking operations. Repository classes, on the other hand, also emphasise maintainability and simplicity of understanding by concentrating only on data access and management using single responsibility principle.

```java
package com.bookonrails.ooad.Interface;

import java.sql.Date;
import java.util.List;

import com.bookonrails.ooad.Model.Passenger;
import com.bookonrails.ooad.Model.PaymentStatus;
import com.bookonrails.ooad.Model.Station;
import com.bookonrails.ooad.Model.Ticket;
import com.bookonrails.ooad.Model.TicketStatus;
import com.bookonrails.ooad.Model.Train;
import com.bookonrails.ooad.Model.User;

public interface TicketCRUDOperations {
    List<Ticket> getAllTickets();
    Ticket getTicketById(Long id);
    Ticket saveTicket(Ticket ticket);
    Ticket updateTicket(Ticket ticket);
    void deleteTicket(Long id);
    List<Ticket> getTicketByUser(User user);

    // update
    Ticket updatePNR(Long ticketId, String newPNR);
    Ticket updateTrain(Long ticketId, Train newTrain);
    Ticket updateSource(Long ticketId, Station newSource);
    Ticket updateDestination(Long ticketId, Station newDestination);
    Ticket updatePassengers(Long ticketId, List<Passenger> newPassengers);
    Ticket updateTicketDate(Long ticketId, Date newDate);
    Ticket updateTicketStatus(Long ticketId, TicketStatus newStatus);
    Ticket updatePaymentStatus(Long ticketId, PaymentStatus newPaymentStatus);
}
```

```java
package com.bookonrails.ooad.Interface;

import com.bookonrails.ooad.Model.Ticket;

public interface TicketAllocationOperations {
    Ticket allocateSeats(Ticket ticket);
    void cancelTicket(Ticket ticket);
}
```

```java
package com.bookonrails.ooad.Interface;

import com.bookonrails.ooad.Model.Ticket;

import java.util.List;

public interface TicketQueryOperations {
    List<Ticket> getCancelledTickets();
    List<Ticket> getConfirmedTickets();
    List<Ticket> getWaitingListTickets();
}
```

2. **Open/Closed Principle (OCP)**: The architecture of the project is intended to be closed to modification yet open to extension. It is possible to add new features or improvements without changing the current codebase. For example, by extending current interfaces or introducing new components, further capabilities like discount on fare can be smoothly added while preserving the stability and integrity of the Fare management system.

```java
package com.bookonrails.ooad.Interface;

import com.bookonrails.ooad.Model.Ticket;

//  for ocp
public interface TicketFareDiscount {
    Ticket addFareDiscount(Long ticketId);
}
```

```java
import com.bookonrails.ooad.Interface.TicketFareDiscount;
import com.bookonrails.ooad.Model.Ticket;
import com.bookonrails.ooad.Repository.TicketRepository;

@Service
public class TicketFareDiscountService implements TicketFareDiscount {

    @Autowired
    private TicketRepository ticketRepository;

    // @Override
    public Ticket addFareDiscount(Long ticketId) {
        Ticket ticket = ticketRepository.findById(ticketId).orElse(other:null);
        if (ticket != null) {
            double currentFare = ticket.getTotalAmount();
            double discountedFare = currentFare - 50.0;
            if (discountedFare < 0) {
                discountedFare = 0; // Ensure fare doesn't go negative
            }
            ticket.setFare(discountedFare);
            return ticketRepository.save(ticket);
        }
        return null;
    }
}

```

3. **Interface Segregation Principle (ISP)**: Interfaces are created to be consistent and narrowly focused, following the ISP, throughout the project. To ensure that clients only rely on the methods they need, each interface defines a defined set of methods related to a single component of the system. Our system has used a lot of interfaces to ensure this.

```java

import java.util.List;

import com.bookonrails.ooad.Model.Admin;

public interface AdminManagement {
    public Admin authenticate(Admin a);
    public void saveAdmin(Admin a);
    public void deleteAdmin(Long id);
    public Admin getAdmin(Long id);
    public List<Admin> findAll();
    public Admin findByUsername(String username);


}

```

```java
@Service
public class AdminService implements AdminManagement {

    @Autowired
    private AdminRepository adminRepository;

    public Admin authenticate(Admin a) {
        // Implement authentication logic by checking credentials in the database
        Admin admin = adminRepository.findByUsernameAndPassword(a.getUsername(), a.getPassword());
        return admin;
    }

    public void saveAdmin(Admin admin) {
        adminRepository.save(admin);
    }

    public void deleteAdmin(Long id) {
        adminRepository.deleteById(id);
    }
    public Admin getAdmin(Long id) {
        return adminRepository.findById(id).get();
    }

    public List<Admin> findAll() {
        return adminRepository.findAll();
    }

    @Override
    public Admin findByUsername(String username){
        return adminRepository.findByUsername(username);
    }
```

4. **Dependency Inversion Principle (DIP)**: DIP encourages the decoupling of high-level modules from low-level components by applying dependency injection and abstraction techniques. Interfaces are used to inject dependencies into classes, making it simple to switch out implementations. This improves the system's scalability, encourages code reuse, and makes testing easier.

The high-level module, represented by components like the `RouteService`, relies on access to seat availability data.

The low-level component corresponds to the data access layer, responsible for interactions with the database to manage seat availability information. This layer can be implemented using various technologies such as JDBC or JPA.

Dependency Injection is utilized by the `RouteService`. Rather than directly instantiating a specific implementation of the `RouteRepository`, it declares a dependency on the `RouteRepository` interface. Through Spring's `@Autowired` annotation, an appropriate implementation of the repository interface is injected at runtime. This approach decouples the service from the underlying data access mechanism, promoting flexibility and ease of maintenance.

```java
package com.bookonrails.ooad.Service;

import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.bookonrails.ooad.Interface.RouteManagement;
import com.bookonrails.ooad.Model.Route;
import com.bookonrails.ooad.Repository.RouteRepository;

@Service
public class RouteService implements RouteManagement {
    @Autowired
    private RouteRepository routeRepository; // dependency injection

    public Route addRoute(Route route) {
        return routeRepository.save(route);
    }

    public Route getRouteByRouteCode(String routeCode) {
        return routeRepository.findByRouteCode(routeCode);
    }

    public Route getRouteById(Long routeId) {
        return routeRepository.findById(routeId).orElse(other:null);
    }
```

```java
package com.bookonrails.ooad.Repository;

import java.util.List;

// import org.springframework.data.jdbc.repository.query.Query;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.query.Procedure;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import com.bookonrails.ooad.Model.Route;

@Repository
public interface RouteRepository extends JpaRepository<Route, Long> {
    Route findByRouteCode(String routeCode);

    @Procedure(name = "FindRouteBetweenStations") // returns route codes
    public List<String> findRouteBetweenStation(@Param("stationCode1") String stationCode1, @Param("stationCode2") String stationCode2);

    List<Route> findByTrainTrainName(String trainName);
    Route findByTrainTrainNo(String trainNo);

    List<Route> findByStationTimingsStationStationCode(@Param("stationCode") String stationCode);

}
```

Design Patterns

Facade Pattern

The Facade design pattern is a structural design pattern that provides a unified interface to a set of interfaces in a subsystem. It hides the complexities of the subsystem and provides a simpler interface to the client or user.

Facade: This is the main interface that the client interacts with. It provides a simplified interface to the more complex subsystem. The facade delegates client requests to appropriate objects within the subsystem.

```
@Controller
@RequestMapping(path = "/ticket")
@CrossOrigin(origins = "*")
public class TicketFrontendController {
    @Autowired
    private TicketService ticketService;

    @Autowired
    private UserService userService;

    @Autowired
    private TrainService trainService;

    @Autowired
    private SeatAvailabilityService seatAvailabilityService;

    @Autowired
    private StationService stationService;

    @Autowired
    private GeneralTicket generalTicket;

    @Autowired
    private TatkalTicket tatkalTicket;

    private boolean CheckTatkal (Date todayDate,Date ticketDate){
        // check if ticket is tatkal
        // ticket is tatkal if booked one day before
        // if ticket is booked today, it is also tatkal
        if(todayDate.compareTo(ticketDate)<=1){
            return true;
        }
        return false;

    }
```

Subsystems: These are the individual components or classes that make up the subsystem. Each subsystem class has its own responsibilities and methods. The facade does not encapsulate these subsystems but rather provides a simplified interface to work with them.

```java
@Service
public class TrainService implements TrainManagement {

    @Autowired
    private TrainRepository trainRepository;

    @Autowired
    private RouteRepository routeRepository;



    // Method to retrieve all trains
    public List<Train> getAllTrains() {
        return trainRepository.findAll();
    }

    // Method to retrieve a train by train number
    public Train getTrainByTrainNo(String trainNo) {
        return trainRepository.findByTrainNo(trainNo);
    }

    // Method to save a train
    public Train saveTrain(Train train) {
        return trainRepository.save(train);
    }
```

```java
@Service
public class SeatAvailabilityService implements TrainSeatAvailibilityManagement
{
    @Autowired
    private SeatAvailabilityRepository seatAvailabilityRepository;

    public SeatAvailability findById(Long sa){
        return seatAvailabilityRepository.findById(sa).orElse(other:null);
    }
    public void addSeatAvailabitity(SeatAvailability seatAvailability){
        seatAvailabilityRepository.save(seatAvailability);
    }


    public void deleteSeatAvailabitity(SeatAvailability seatAvailability){
        if(seatAvailability==null){
            throw new IllegalArgumentException(s:"Seat availability object is null or has no ID.");
        }
        seatAvailabilityRepository.delete(seatAvailability);
    }


    public boolean checkSeatAvailability(SeatAvailability seatAvailability){
        return seatAvailability.isFull();
    }


    public void updateSeatAvailibity(SeatAvailability seatAvailability){
        if (seatAvailability == null || seatAvailability.getId() == null) {
            throw new IllegalArgumentException(s:"Seat availability object is null or has no ID.");
        }
        seatAvailabilityRepository.save(seatAvailability);
    }

    public List<SeatAvailability> getAllSeatAvailabilities(){
        return seatAvailabilityRepository.findAll();
    }
```

Template Pattern:

The Template Method pattern is a behavioral design pattern that defines the skeleton of an algorithm or operations in a superclass (often abstract) and leaves the details to be implemented by the child classes. It allows subclasses to customize specific parts of the algorithm without altering its overall structure. In our code, we create TicketAbstractService (abstract class) which has two ticket types, General Ticket and Tatkal Ticket (child classes), which implement the fare discount in detail.

```
@Service
public abstract class TicketAbstractService implements TicketManagement {

    @Autowired
    private TicketRepository ticketRepository;

    @Autowired
    private PassengerService passengerService;

    @Autowired
    private SeatAvailabilityService seatAvailabilityService;

    @Autowired
    private TicketFareDiscount fareDiscountStrategy; // Inject the FareDiscountStrategy interface


    public List<Ticket> getAllTickets() {
        return ticketRepository.findAll();
    }

    public String generateUniquePNR() {

        String uniqueID = UUID.randomUUID().toString();
        String pnr = uniqueID.substring(beginIndex:0, endIndex:6);
        pnr = "PNR"+ pnr;
        return pnr;
    }

public Ticket addFareDiscount(Long ticketId,double discount) {
    return fareDiscountStrategy.addFareDiscount(ticketId,discount);
}
```

```
package com.bookonrails.ooad.Service;

import org.springframework.beans.factory.annotation.Autowired;

import com.bookonrails.ooad.Model.Ticket;
import com.bookonrails.ooad.Repository.TicketRepository;

public class TatkalTicket extends TicketAbstractService {

    @Autowired
    private TicketRepository ticketRepository;

    TatkalTicket(){
        super();
    }

    @Override
    public Ticket addFareDiscount(Long ticketId,double discount) {
        // If ticket is Tatkal, we give discount for the ticket price and add tax and a double the base fare
        Ticket t= ticketRepository.findById(ticketId).orElse(other:null);
        if(t!=null) {
            double currentFare = t.getTotalAmount();
            double doubleFare = 2* currentFare;
            double discountedFare = doubleFare - discount;
            if (discountedFare < 0) {
                discountedFare = 0; // Ensure fare doesn't go negative
            }
            // t.setFare(discountedFare);
            t.setFare(discountedFare + (discountedFare * 0.18)); // Add 18% tax
            return ticketRepository.save(t);
        }
        return null;
    }


}
```

```java
import org.springframework.beans.factory.annotation.Autowired;

import com.bookonrails.ooad.Model.Ticket;
import com.bookonrails.ooad.Repository.TicketRepository;

public class GeneralTicket extends TicketAbstractService {

    @Autowired
    private TicketRepository ticketRepository;

    GeneralTicket(){
        super();
    }

    @Override
    public Ticket addFareDiscount(Long ticketId,double discount) {
        // If ticket is General, we give discount for the ticket price and add tax
        Ticket t= ticketRepository.findById(ticketId).orElse(other:null);
        if(t!=null) {
            double currentFare = t.getTotalAmount();
            double discountedFare = currentFare - discount;
            if (discountedFare < 0) {
                discountedFare = 0; // Ensure fare doesn't go negative
            }
            // t.setFare(discountedFare);
            t.setFare(discountedFare + (discountedFare * 0.18)); // Add 18% tax
            return ticketRepository.save(t);
        }
        return null;
    }


}
```

Proxy Pattern:

The proxy pattern is a technique that allows one object — the proxy — to control access to another object — the subject or service. In Spring, beans are proxied to control access to the underlying bean. We see this approach when using transactions in TrainService:

```java

    @Transactional
    public List<Train> searchTrainBySrcAndDest(String SRC,String DEST){
        List<Train> t = new ArrayList<>();
        List<String> r= routeRepository.findRouteBetweenStation(SRC,DEST );
        for(String s:r){
            Route route= routeRepository.findByRouteCode(s);
            t.add(trainRepository.findByRoute(route));
            System.out.println(route.getRouteCode());

        }
        return t;
    }

    @Transactional
    public List<SeatAvailability> searchTrain(String SRC,String DEST, ClassType classes, Date date){
        List<Train> t=searchTrainBySrcAndDest(SRC,DEST);
        // now we need to filter trains based on if they run on the given date
        // and then filter them based on class type
        List<SeatAvailability> newList= new ArrayList<>();
        for(int i=0;i<t.size();i++){
            if(t.get(i).doesDateAndClassExist(date,classes)){
                newList.add(t.get(i).getSeatAvailabilityBasedOnClassAndDate(date, classes));
            }
        }
        return newList;
    }

```

Singleton Pattern:
The singleton pattern is a mechanism that ensures only one instance of an object exists per application. This pattern can be useful when managing shared resources or providing cross-cutting services, such as logging. The Spring @Autowired annotation demonstrates autowired singleton pattern, which creates only one object for a repository and achieves loose coupling between classes.

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

// import com.bookonrails.ooad.Interface.TrainSeatAvailibilityManagement;
import com.bookonrails.ooad.Model.ClassType;
import com.bookonrails.ooad.Model.SeatAvailability;
import com.bookonrails.ooad.Model.Ticket;
import com.bookonrails.ooad.Model.Train;
import com.bookonrails.ooad.Repository.SeatAvailabilityRepository;

@Service
public class SeatAvailabilityService // extends TrainSeatAvailibilityManagement
{
    @Autowired
    private SeatAvailabilityRepository seatAvailabilityRepository;

    public SeatAvailability findById(Long sa){
        return seatAvailabilityRepository.findById(sa).orElse(other:null);
    }
    public void addSeatAvailabitity(SeatAvailability seatAvailability){
        seatAvailabilityRepository.save(seatAvailability);
    }
```

```java
@Service
public abstract class TicketAbstractService implements TicketManagement {

    @Autowired
    private TicketRepository ticketRepository;

    @Autowired
    private PassengerService passengerService;

    @Autowired
    private SeatAvailabilityRepository seatAvailabilityRepository;

    @Autowired
    private TicketFareDiscount fareDiscountStrategy; // Inject the FareDiscountStrategy interface


    public List<Ticket> getAllTickets() {
        return ticketRepository.findAll();
    }

    public String generateUniquePNR() {

        String uniqueID = UUID.randomUUID().toString();
        String pnr = uniqueID.substring(beginIndex:0, endIndex:6);
        pnr = "PNR"+ pnr;
        return pnr;
    }

    public List<Ticket> getTicketByUser(User user) {
```

Here TicketAbstractService and SeatAvailabilityService are using SeatAvailabilityRepository, but only one object of SeatAvailabilityRepository is created, following Singleton Pattern.

Github

https://github.com/Mystery510/Book-On-Rails

Individual Contributions :

Shreya Mishra
Shreya Mishra made significant contributions across both backend and frontend aspects of the project. She took a lead role in integrating the backend services with the database, ensuring seamless data storage and retrieval. Shreya also worked extensively on the frontend, where she leveraged her skills in HTML, CSS, and Thymeleaf to create dynamic web pages that displayed data fetched from the backend. Her holistic approach to development bridged the gap between the frontend and backend components, resulting in a cohesive and functional application.

Siri
Siri's contributions were predominantly in the backend development of the application. She took the lead in designing and implementing the core backend logic of the system. Siri's responsibilities included developing RESTful APIs, managing database interactions, and handling server-side functionalities. Her expertise in Java and Spring Boot framework enabled the creation of robust backend services supporting features such as user authentication, ticket booking, and data retrieval.

Shreeja
Shreeja's primary focus was on backend development, working closely with Siri. Together, they collaborated on designing and implementing the core logic of the system. Shreeja's responsibilities included developing RESTful APIs, managing database interactions, and implementing server-side functionalities. Her expertise in Java and Spring Boot framework was instrumental in creating the backend services supporting various application features. Shreeja's contributions ensured the backend was reliable, efficient, and scalable.

Sinchana
Sinchana played a crucial role in the frontend development of the application. Her primary focus was on creating intuitive and visually appealing user interfaces. She worked extensively on crafting responsive and dynamic web pages using HTML, CSS, and Bootstrap. Sinchana's attention to detail ensured that the frontend components were not only functional but also aesthetically pleasing. Additionally, she contributed to the implementation of client-side JavaScript functionality to enhance user interactivity.

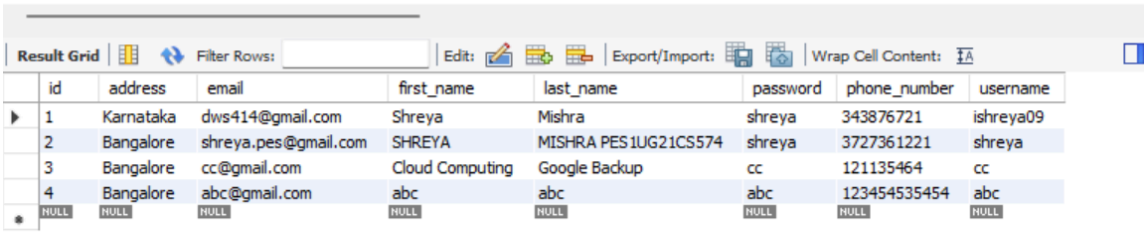
Overall Collaboration
While each team member had their primary areas of focus, we all worked collaboratively on various aspects of the project. Regular meetings and discussions facilitated a smooth integration of frontend and backend components. Sinchana's frontend designs seamlessly integrated with the APIs developed by Siri and Shreeja, thanks to Shreya's efforts in database integration. This collaborative approach ensured that the final product met the project requirements and delivered a user-friendly experience.

Screenshots

Database Before Changes

User Table:

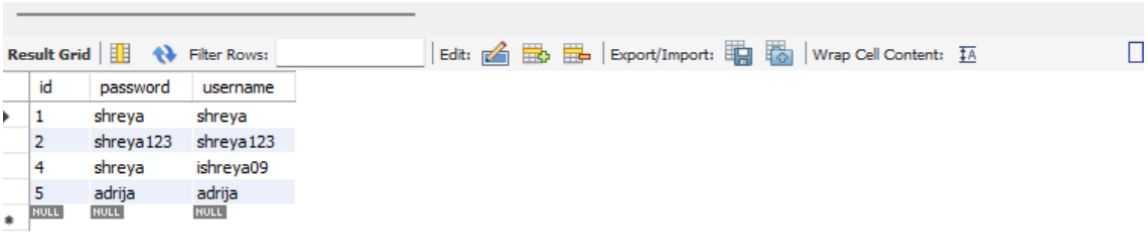
```
1 • use bookonrails;
2 • select * from user;
```



id	address	email	first_name	last_name	password	phone_number	username
1	Karnataka	dws414@gmail.com	Shreya	Mishra	shreya	343876721	ishreya09
2	Bangalore	shreya.pes@gmail.com	SHREYA	MISHRA PES1UG21CS574	shreya	3727361221	shreya
3	Bangalore	cc@gmail.com	Cloud Computing	Google Backup	cc	121135464	cc
4	Bangalore	abc@gmail.com	abc	abc	abc	123454535454	abc
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Admin Table:

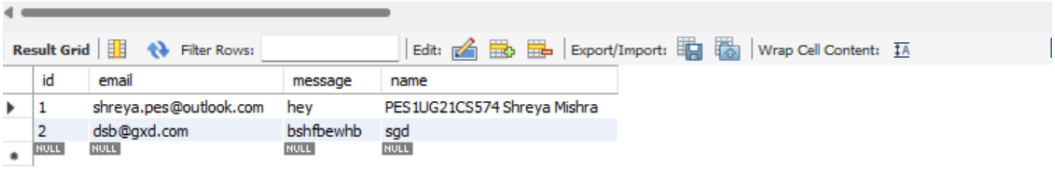
```
4 • select * from admin;
```



id	password	username
1	shreya	shreya
2	shreya123	shreya123
4	shreya	ishreya09
5	adrija	adrija
NULL	NULL	NULL

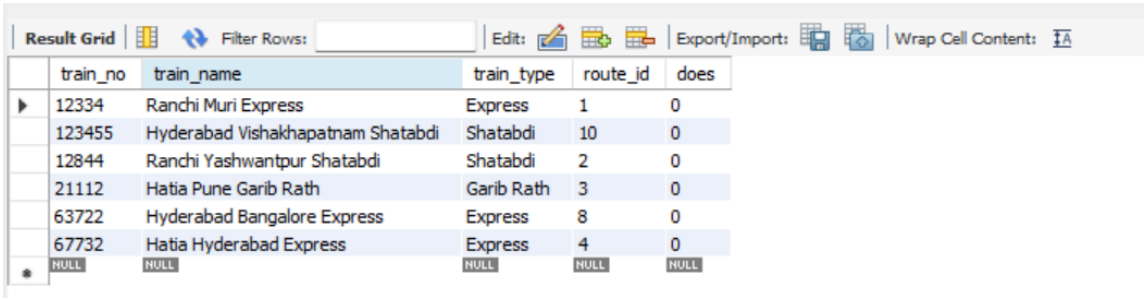
Contact Table:

```
5 • select * from contact;
```



id	email	message	name
1	shreya.pes@outlook.com	hey	PES1UG21CS574 Shreya Mishra
2	dsb@gxd.com	bshfbewhb	sgd
NULL	NULL	NULL	NULL

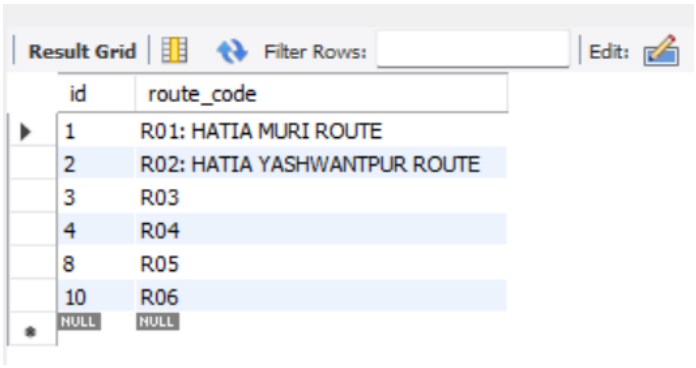
Train :



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

train_no	train_name	train_type	route_id	does
12334	Ranchi Muri Express	Express	1	0
123455	Hyderabad Vishakhapatnam Shatabdi	Shatabdi	10	0
12844	Ranchi Yashwantpur Shatabdi	Shatabdi	2	0
21112	Hatia Pune Garib Rath	Garib Rath	3	0
63722	Hyderabad Bangalore Express	Express	8	0
67732	Hatia Hyderabad Express	Express	4	0
NULL	NULL	NULL	NULL	NULL

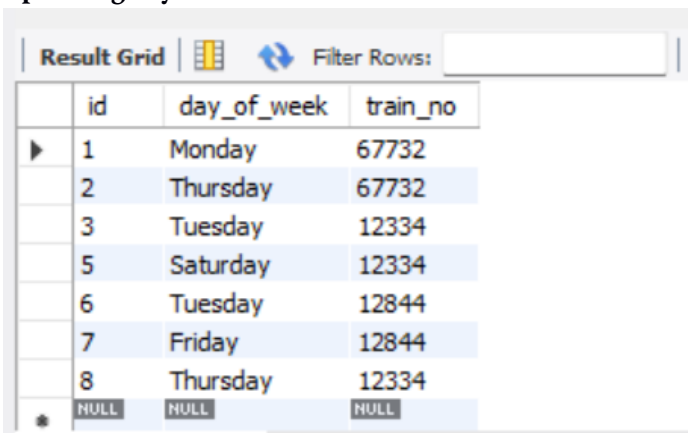
Route:



Result Grid | Filter Rows: | Edit:

id	route_code
1	R01: HATIA MURI ROUTE
2	R02: HATIA YASHWANTPUR ROUTE
3	R03
4	R04
8	R05
10	R06
NULL	NULL

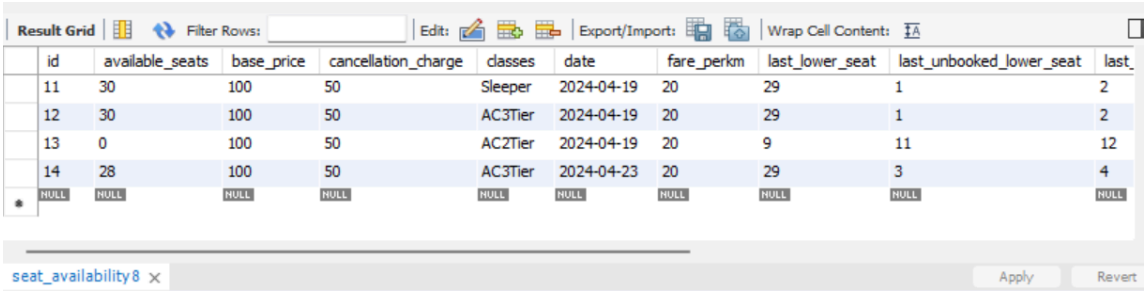
Operating day:



Result Grid | Filter Rows:

id	day_of_week	train_no
1	Monday	67732
2	Thursday	67732
3	Tuesday	12334
5	Saturday	12334
6	Tuesday	12844
7	Friday	12844
8	Thursday	12334
NULL	NULL	NULL

Seat Availability:

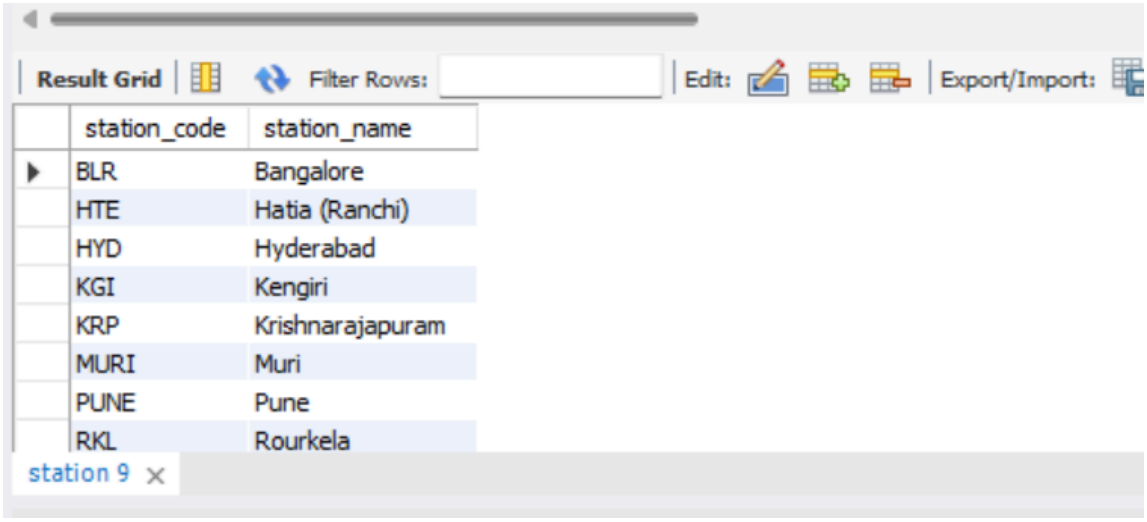


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	available_seats	base_price	cancellation_charge	classes	date	fare_perkm	last_lower_seat	last_unbooked_lower_seat	last_
11	30	100	50	Sleeper	2024-04-19	20	29	1	2
12	30	100	50	AC3Tier	2024-04-19	20	29	1	2
13	0	100	50	AC2Tier	2024-04-19	20	9	11	12
14	28	100	50	AC3Tier	2024-04-23	20	29	3	4
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

seat_availability 8 | Apply | Revert

Station:

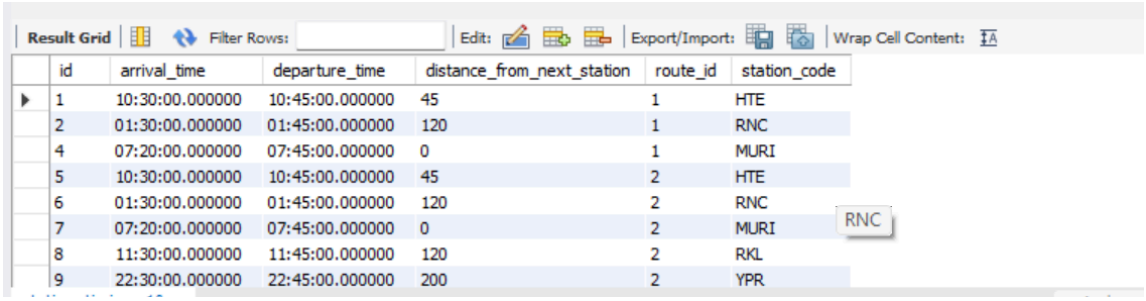


Result Grid | Filter Rows: | Edit: | Export/Import:

station_code	station_name
BLR	Bangalore
HTE	Hatia (Ranchi)
HYD	Hyderabad
KGI	Kengiri
KRP	Krishnarajapuram
MURI	Muri
PUNE	Pune
RKL	Rourkela

station 9

Station Timings:

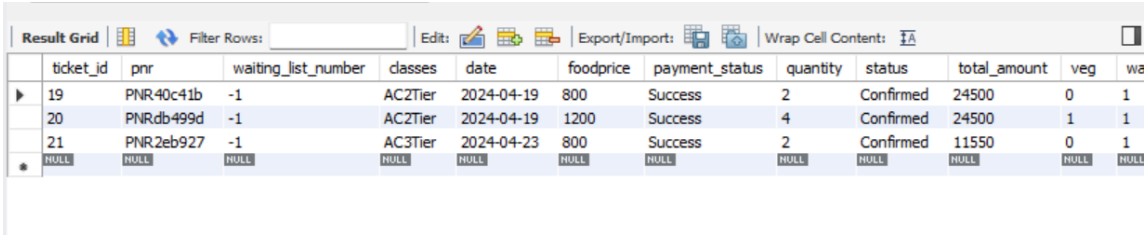


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	arrival_time	departure_time	distance_from_next_station	route_id	station_code
1	10:30:00.000000	10:45:00.000000	45	1	HTE
2	01:30:00.000000	01:45:00.000000	120	1	RNC
4	07:20:00.000000	07:45:00.000000	0	1	MURI
5	10:30:00.000000	10:45:00.000000	45	2	HTE
6	01:30:00.000000	01:45:00.000000	120	2	RNC
7	07:20:00.000000	07:45:00.000000	0	2	MURI
8	11:30:00.000000	11:45:00.000000	120	2	RKL
9	22:30:00.000000	22:45:00.000000	200	2	YPR

RNC

Ticket:



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

ticket_id	pnr	waiting_list_number	classes	date	foodprice	payment_status	quantity	status	total_amount	veg	wa
19	PNR40c41b	-1	AC2Tier	2024-04-19	800	Success	2	Confirmed	24500	0	1
20	PNRdb499d	-1	AC2Tier	2024-04-19	1200	Success	4	Confirmed	24500	1	1
21	PNR2eb927	-1	AC3Tier	2024-04-23	800	Success	2	Confirmed	11550	0	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ticket_passenger:

```
13 •  select * from ticket_passengers;
```

Result Grid | Filter Rows: | Edit:

ticket_ticket_id	passengers_id
19	32
19	33
19	34
19	35
19	36
20	37
20	38
20	39

Passenger:

```
14 •  select * from passenger;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	age	birthpreference	gender	is_senior_citizen	is_waiting_list	name	seat_no	ticket_id	user_id	coach_no
37	11	Lower	Female	0	0	P1	9	20	2	10
38	34	Lower	Male	0	0	p2	11	20	2	12
39	34	Lower	Male	0	0	p3	11	20	2	12
40	32	Lower	Female	0	0	p4	11	20	2	12
41	5	Lower	Female	0	0	p5	11	20	2	12
42	12	Lower	Female	0	0	Shre...	1	21	2	2
43	21	Upper	Female	0	0	adrija	2	21	2	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Payment:

Result Grid | Filter Rows:

id	status	ticket_id
12	Success	19
13	Success	20
14	Success	21
NULL	NULL	NULL

payment 14 ×

Feedback:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	comment	rating	ticket_id	user_id
1	good	4	20	2
2	okayish	2	19	2
NULL	NULL	NULL	NULL	NULL

cancelled_seats:

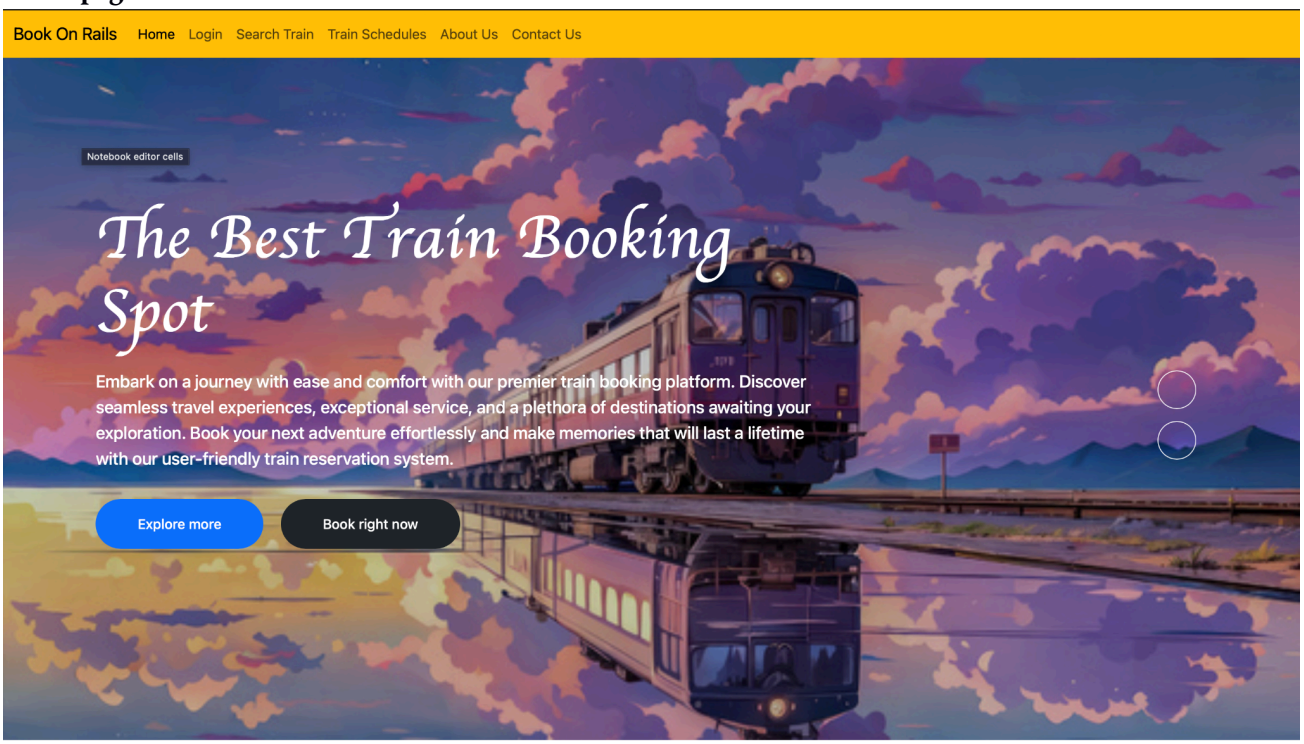
```
17 •  select * from cancelled_seats;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

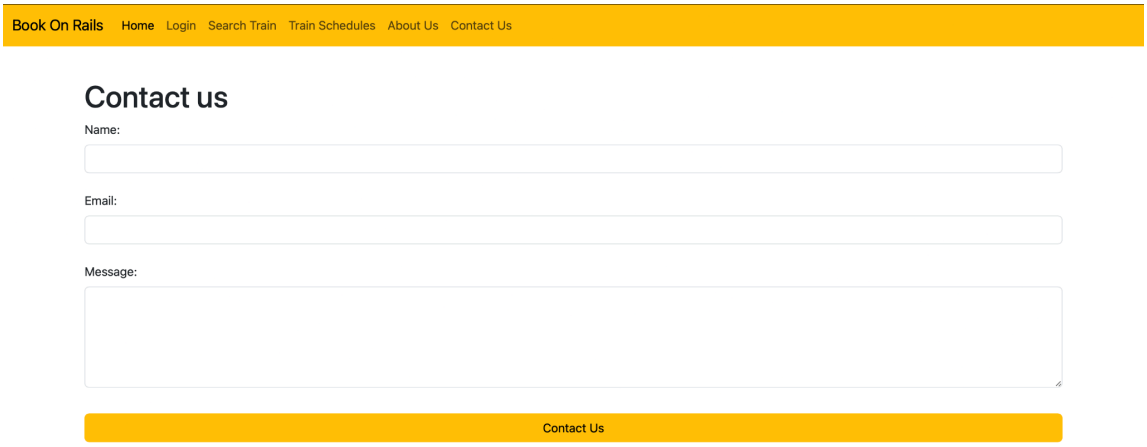
seat_availability_id	seat_number
5	2

Frontend

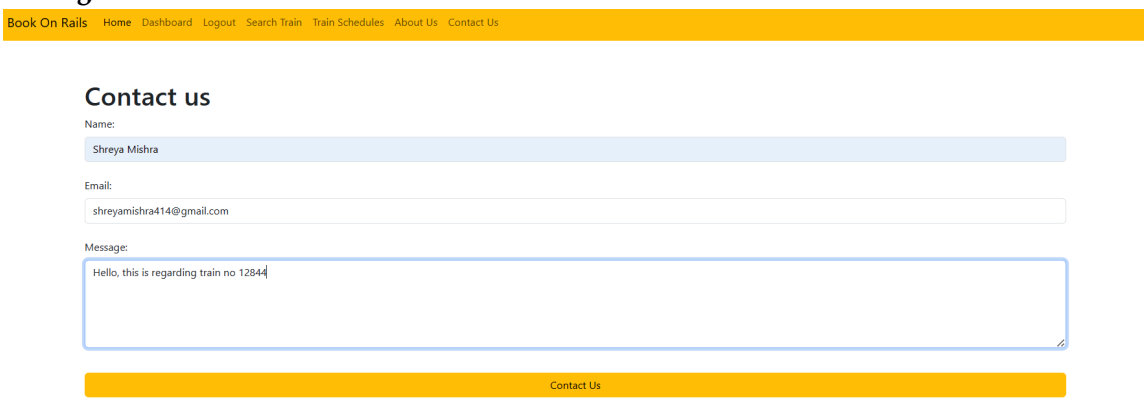
Home page :

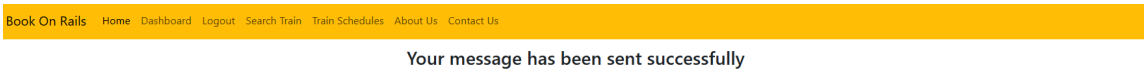


Contact Us :



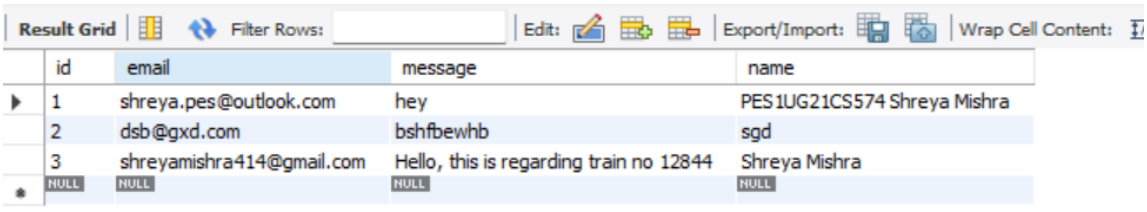
Adding Data :





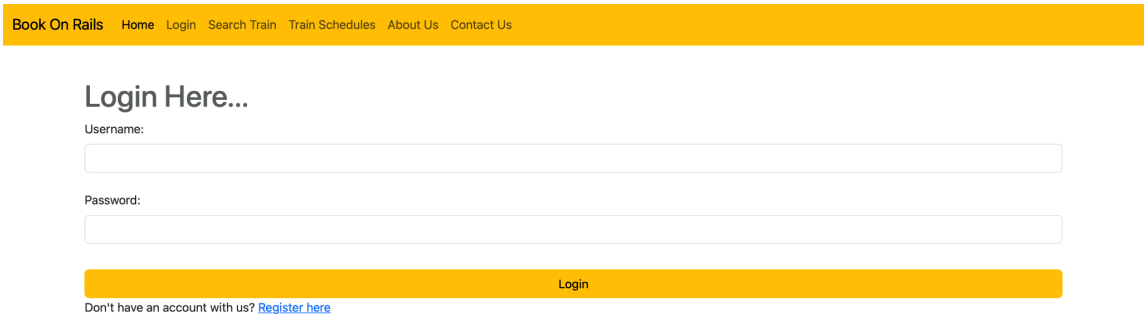
Your message has been sent successfully

Added to Database :

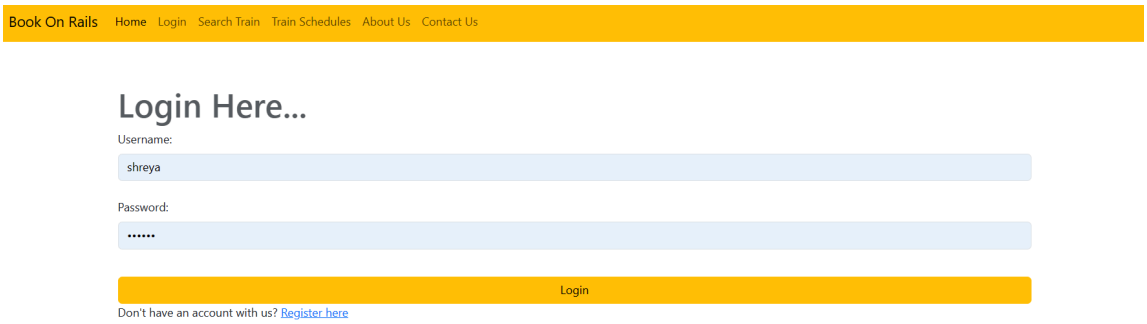


id	email	message	name
1	shreya.pes@outlook.com	hey	PES1UG21CS574 Shreya Mishra
2	dsb@gxd.com	bshfbewhb	sgd
3	shreyamishra414@gmail.com	Hello, this is regarding train no 12844	Shreya Mishra
NULL	NULL	NULL	NULL

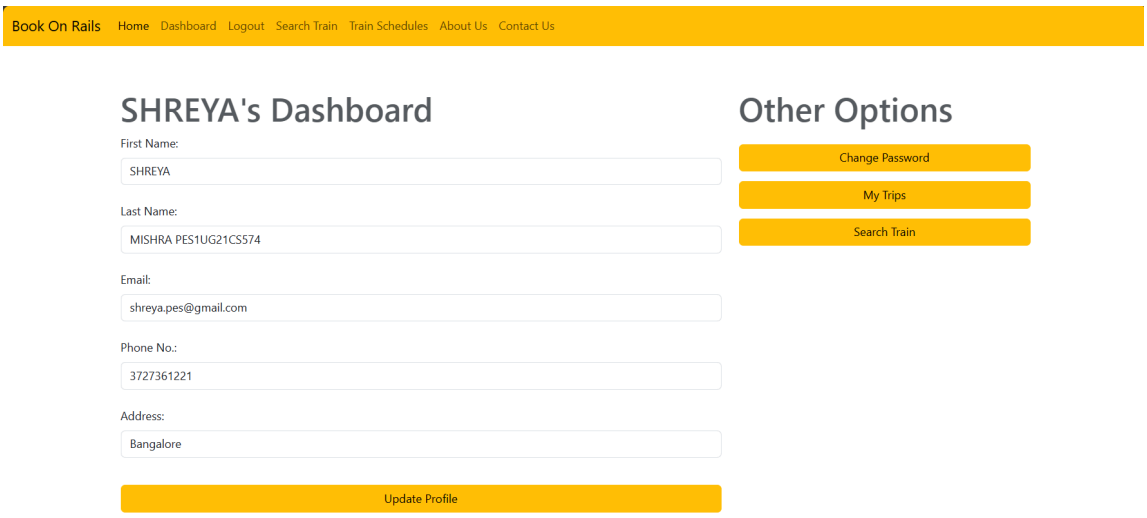
Login Page :



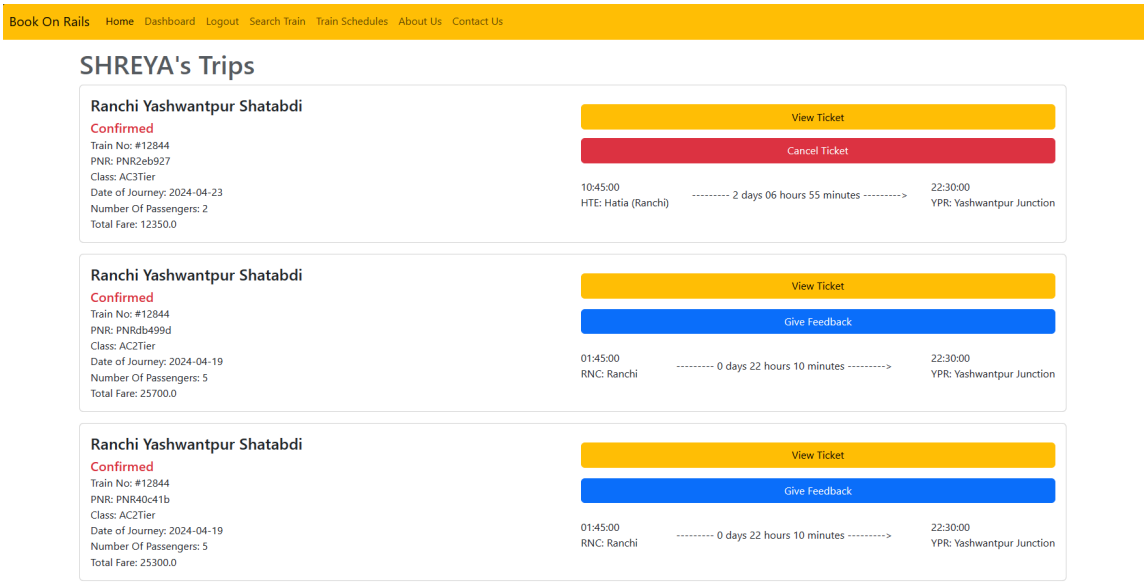
Logging in using shreya username:



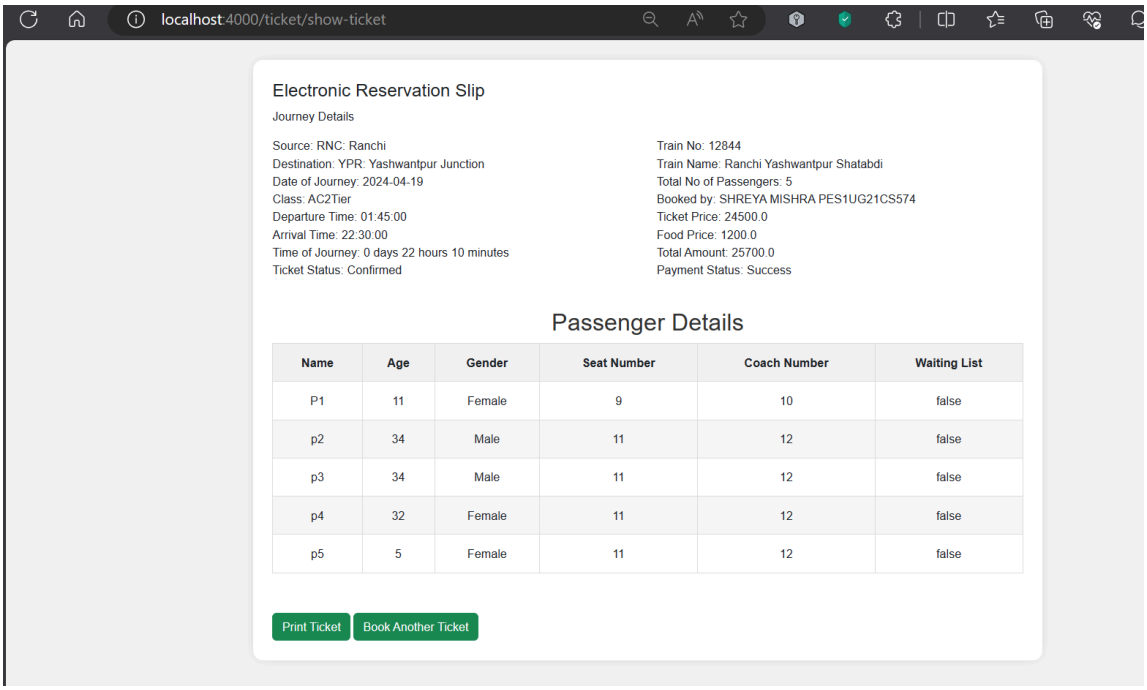
User Dashboard:



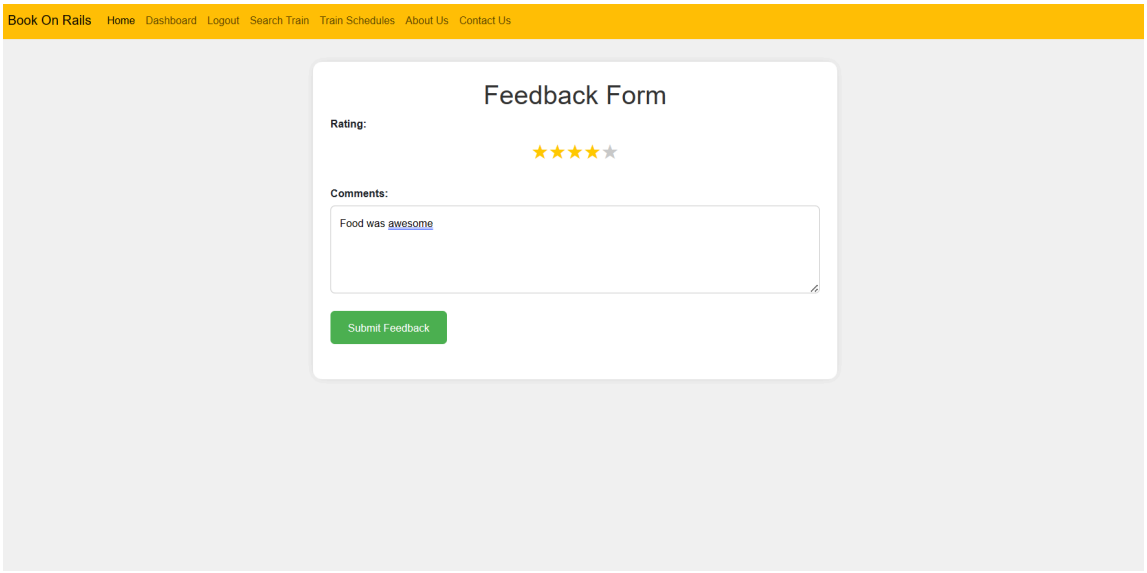
User's Booked Trips:



View Ticket page:

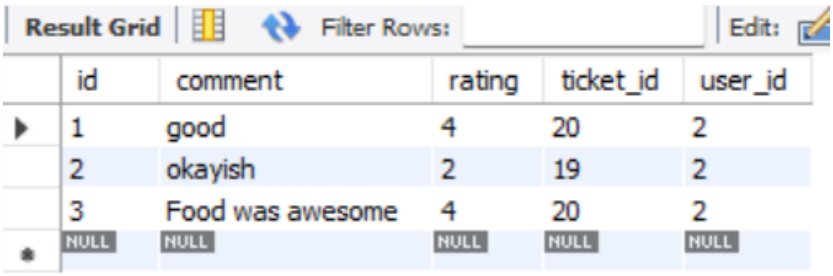


Feedback :



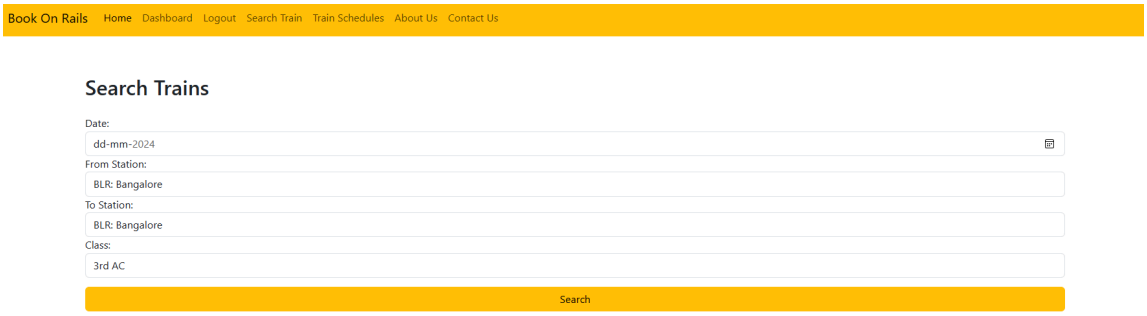
Thank you for your valuable feedback

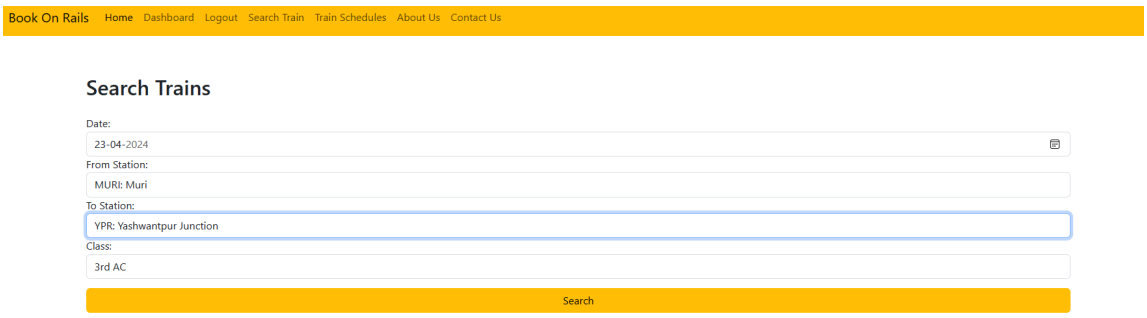
Feedback added:



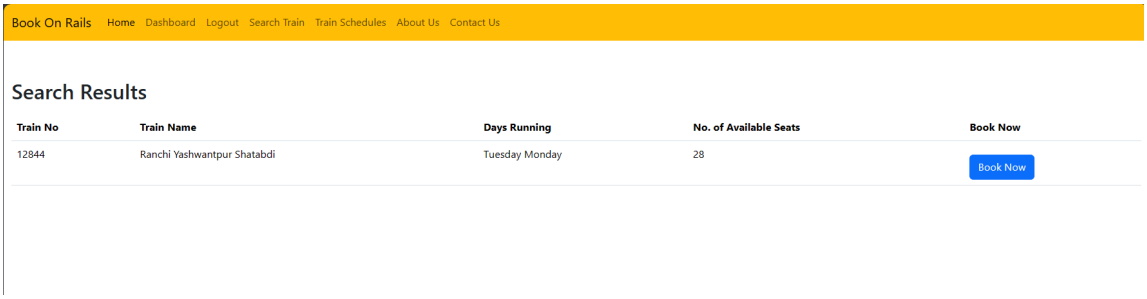
id	comment	rating	ticket_id	user_id
1	good	4	20	2
2	okayish	2	19	2
3	Food was awesome	4	20	2
NULL	NULL	NULL	NULL	NULL

Search Train:

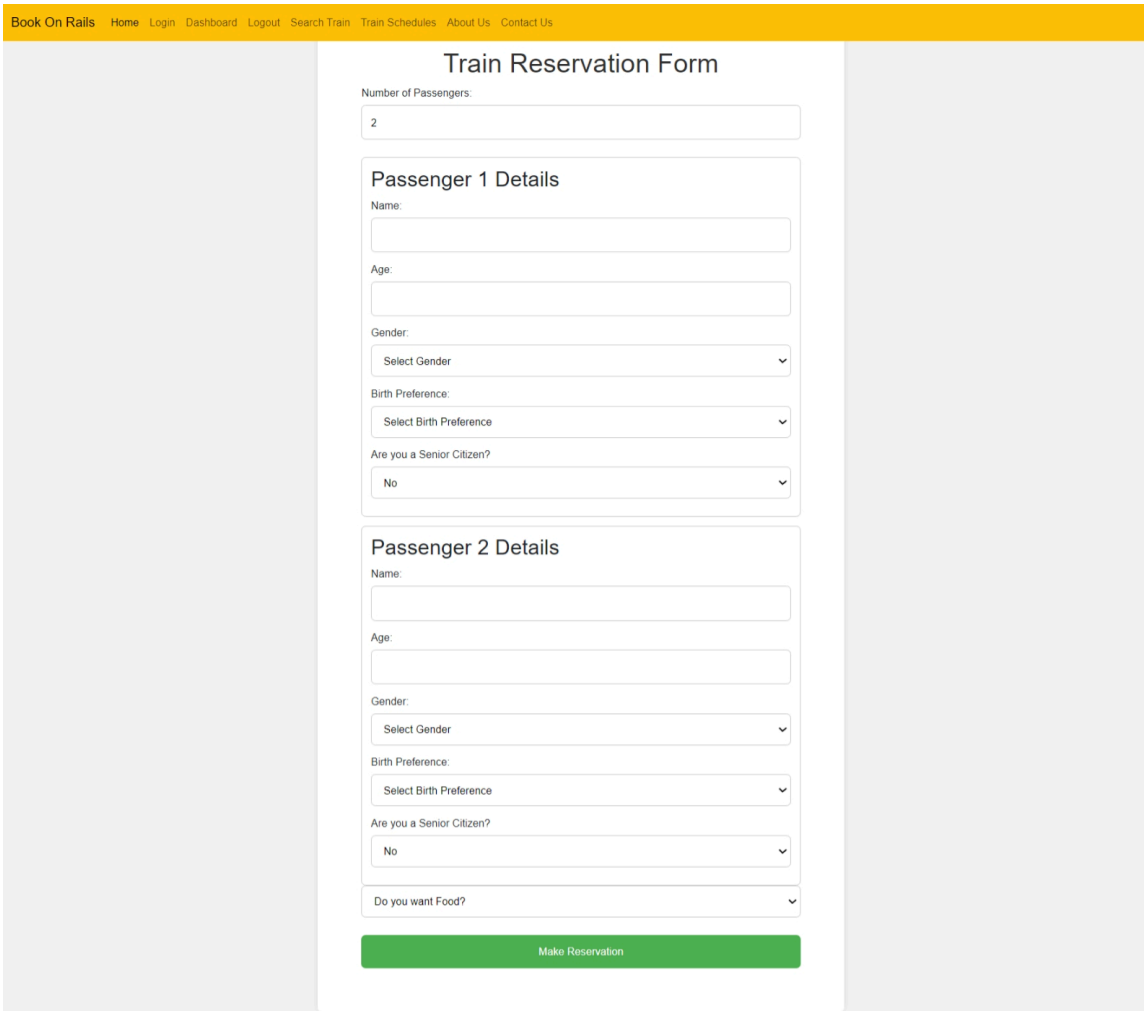


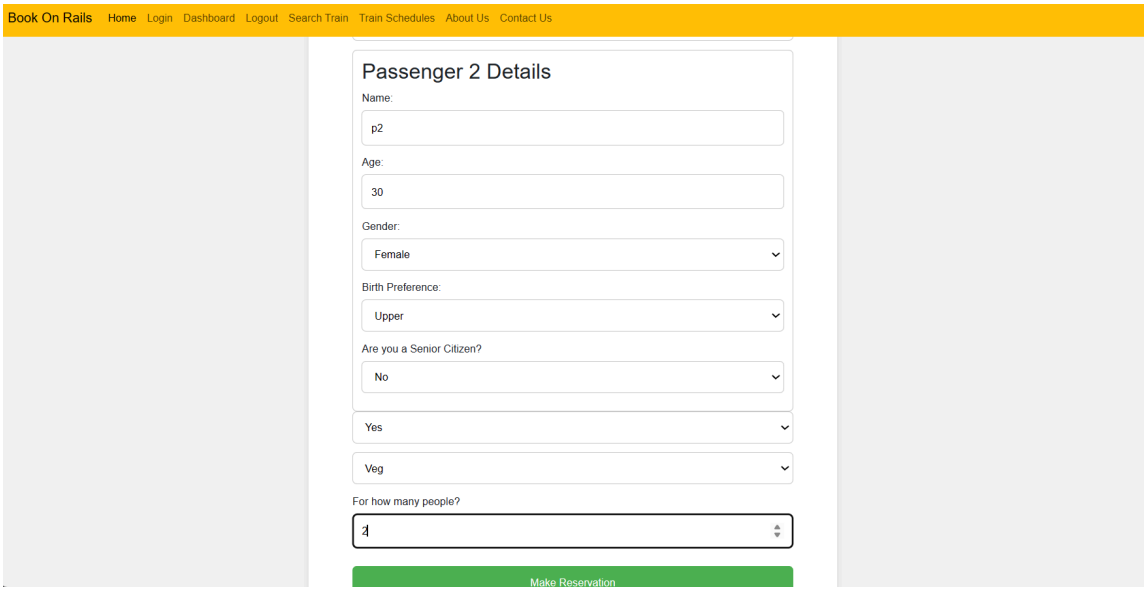


Rendering trains available for the query:

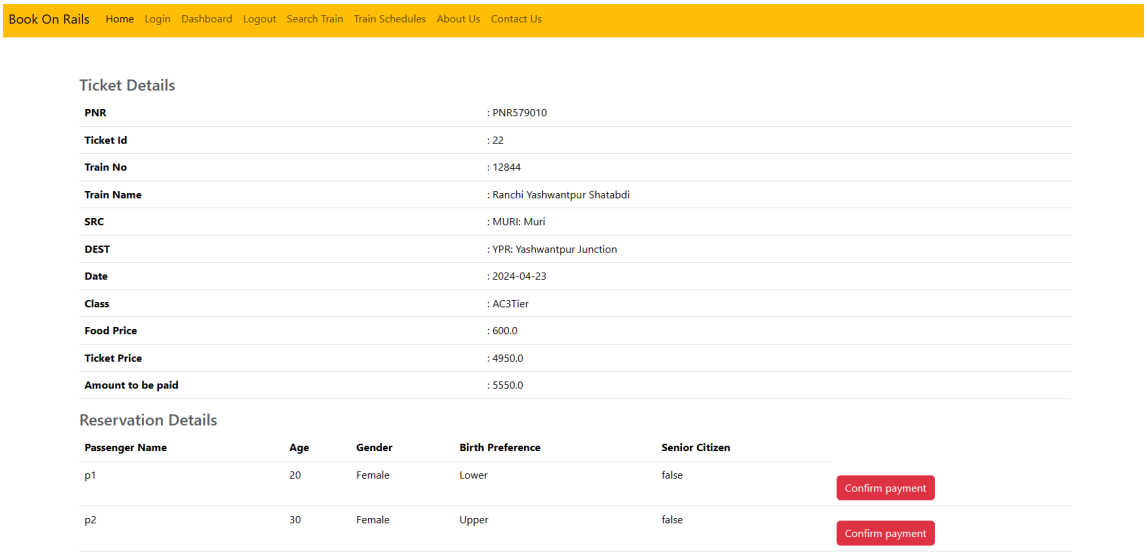


Clicking on Book now, Train Reservation form opens up

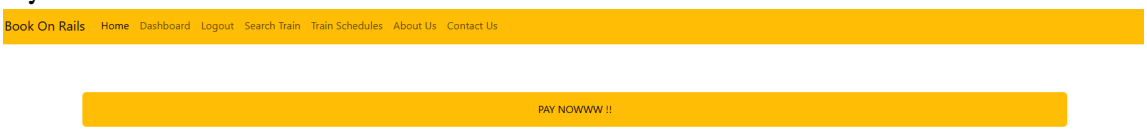




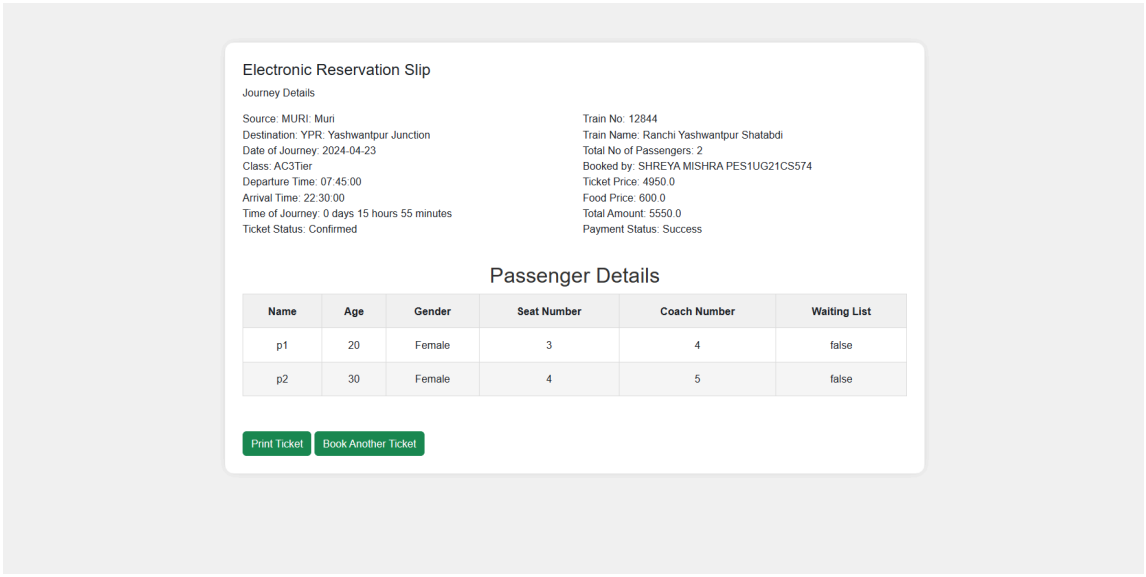
Ticket Details before Payment

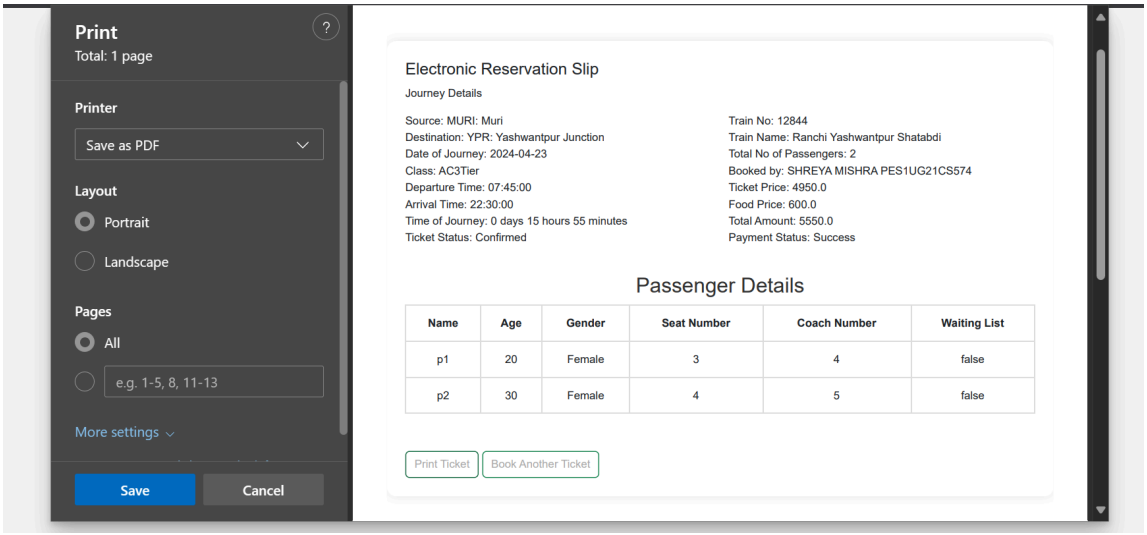


Pay

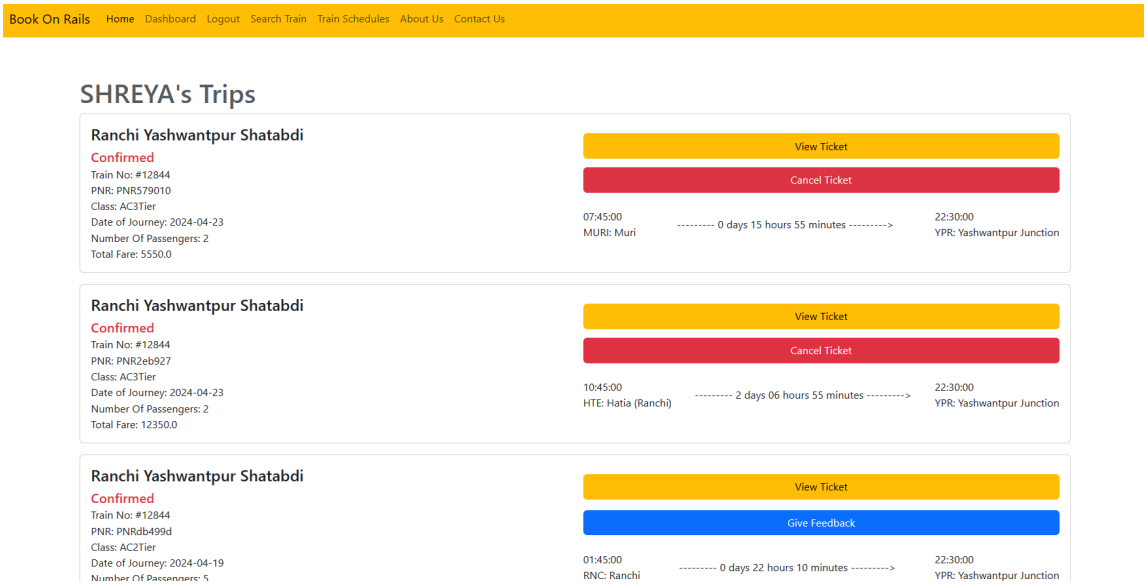


Ticket booked

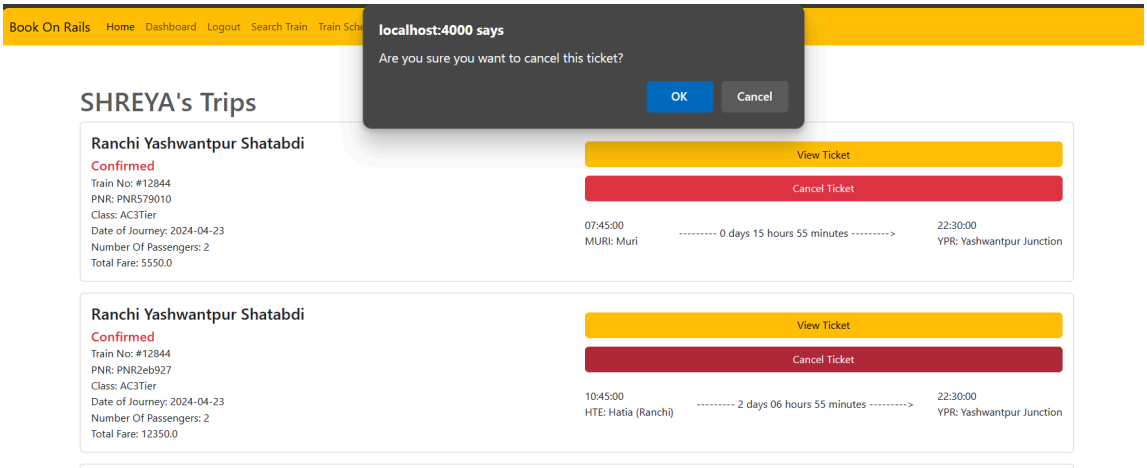




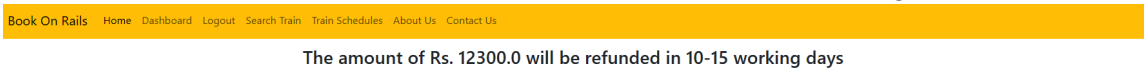
Ticket reflected in my trips



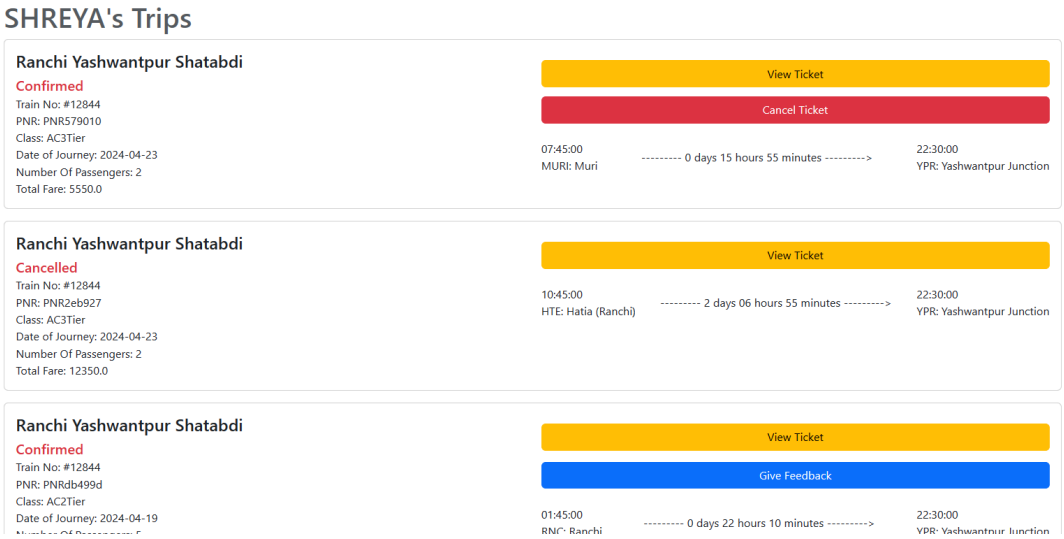
Cancel 2nd Ticket



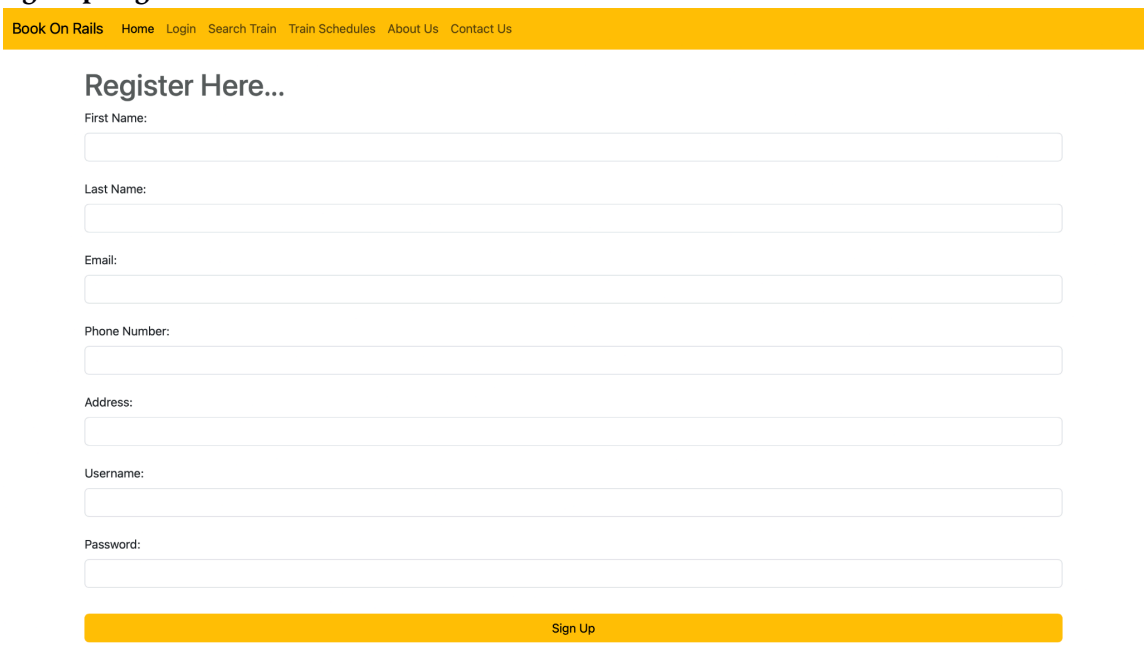
Ticket Cancelled (Amount that will be refunded is after cancellation charge is subtracted)



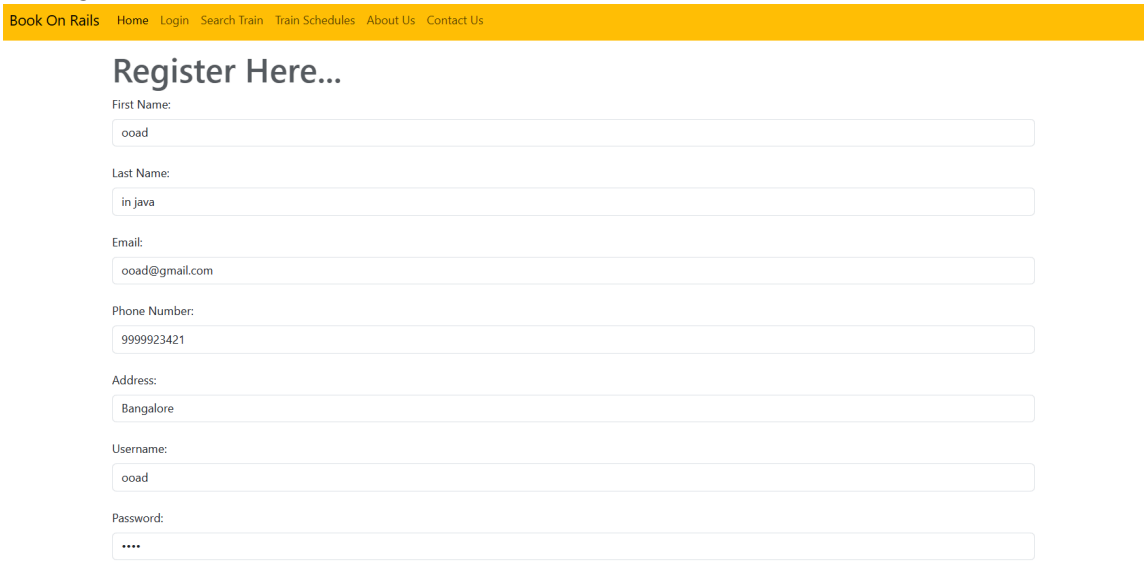
Ticket Cancelled



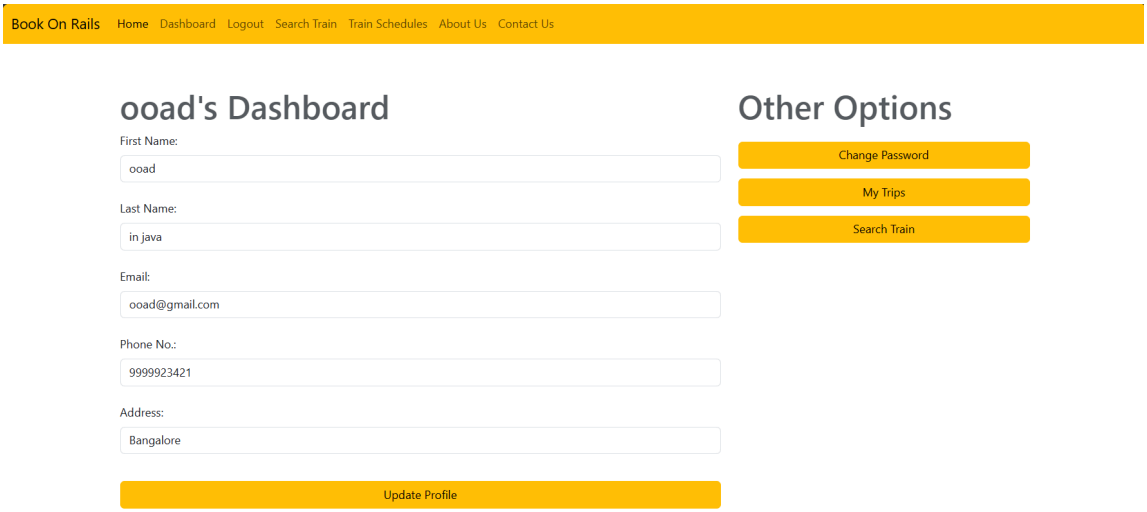
Sign Up Page



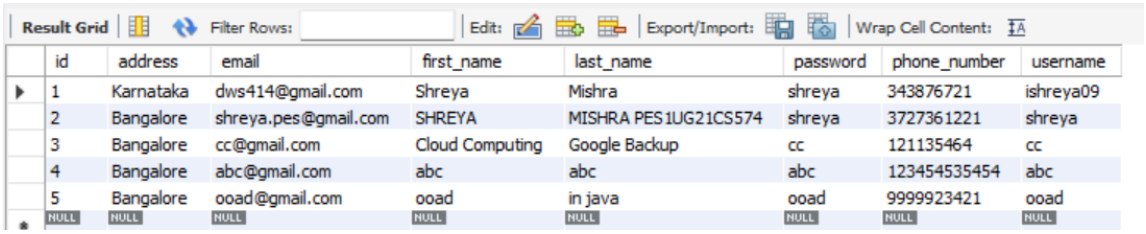
Adding User



User Dashboard for ooad user



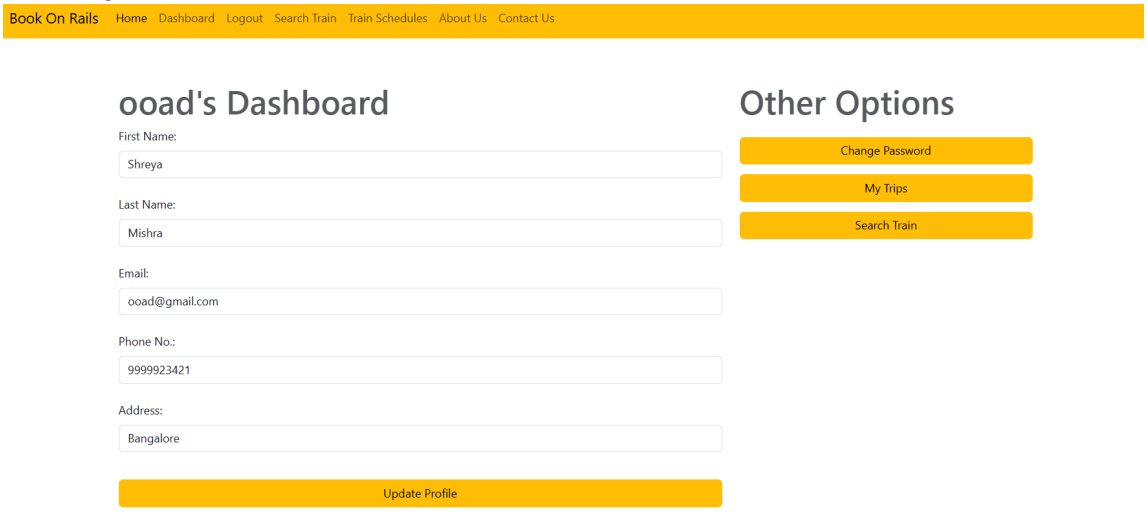
User added in database



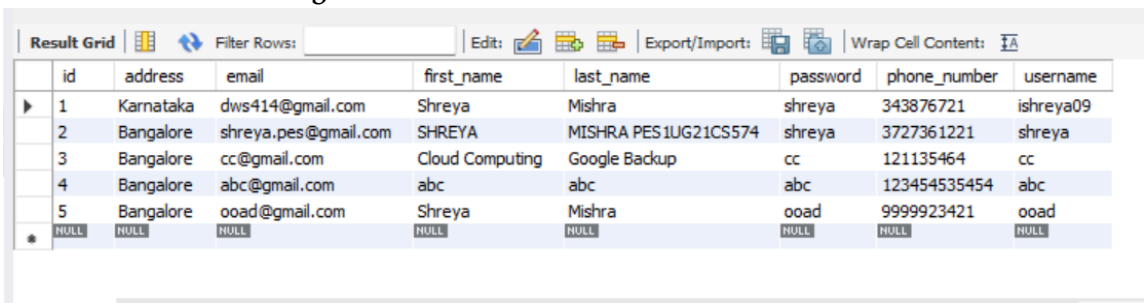
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	address	email	first_name	last_name	password	phone_number	username
1	Karnataka	dws414@gmail.com	Shreya	Mishra	shreya	343876721	ishreya09
2	Bangalore	shreya.pes@gmail.com	SHREYA	MISHRA PES1UG21CS574	shreya	3727361221	shreya
3	Bangalore	cc@gmail.com	Cloud Computing	Google Backup	cc	121135464	cc
4	Bangalore	abc@gmail.com	abc	abc	abc	123454535454	abc
5	Bangalore	ooad@gmail.com	ooad	in java	ooad	9999923421	ooad
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Updating user



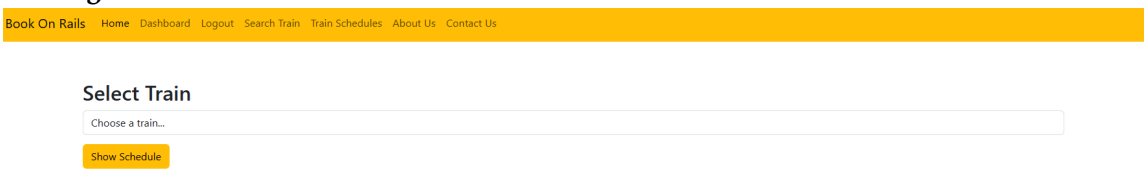
First and Last Name changed in database



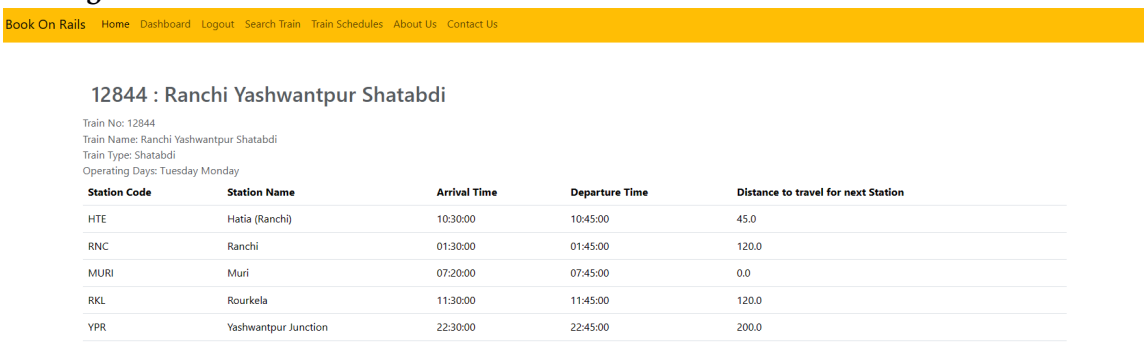
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	address	email	first_name	last_name	password	phone_number	username
1	Karnataka	dws414@gmail.com	Shreya	Mishra	shreya	343876721	ishreya09
2	Bangalore	shreya.pes@gmail.com	SHREYA	MISHRA PES1UG21CS574	shreya	3727361221	shreya
3	Bangalore	cc@gmail.com	Cloud Computing	Google Backup	cc	121135464	cc
4	Bangalore	abc@gmail.com	abc	abc	abc	123454535454	abc
5	Bangalore	ooad@gmail.com	Shreya	Mishra	ooad	9999923421	ooad
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Viewing Train Schedules

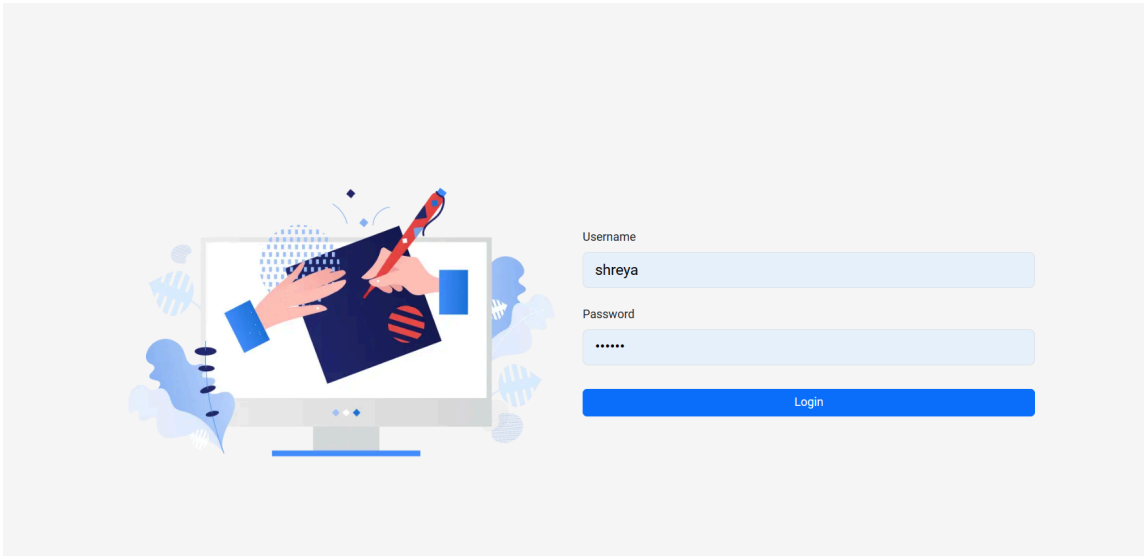


Viewing Schedule for a selected train

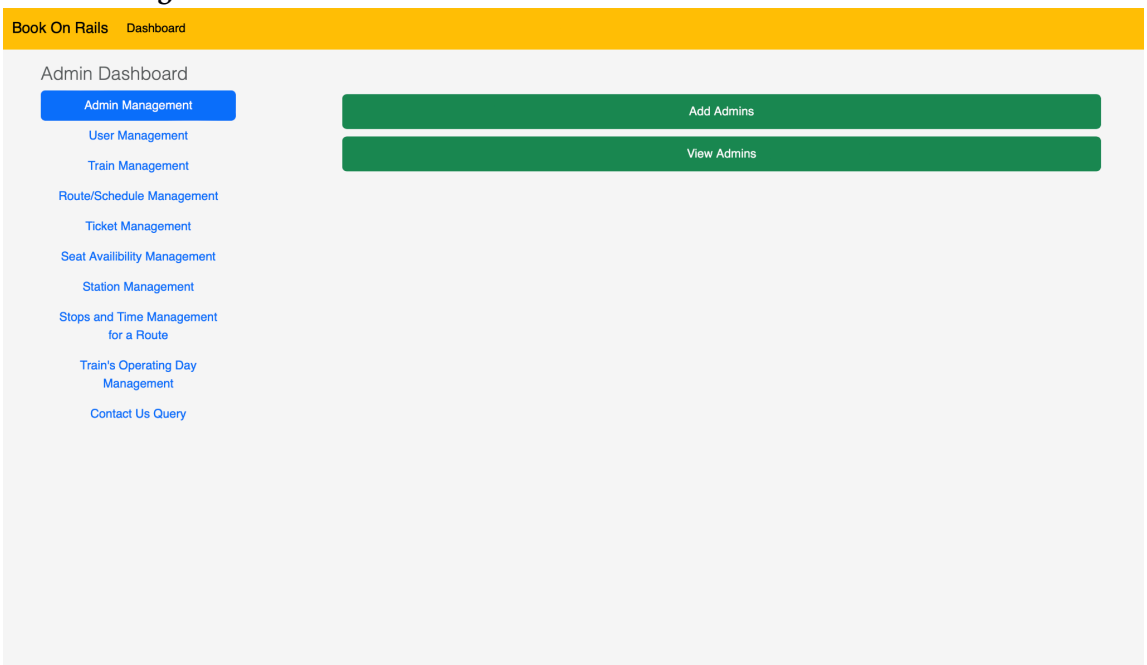


Admin side

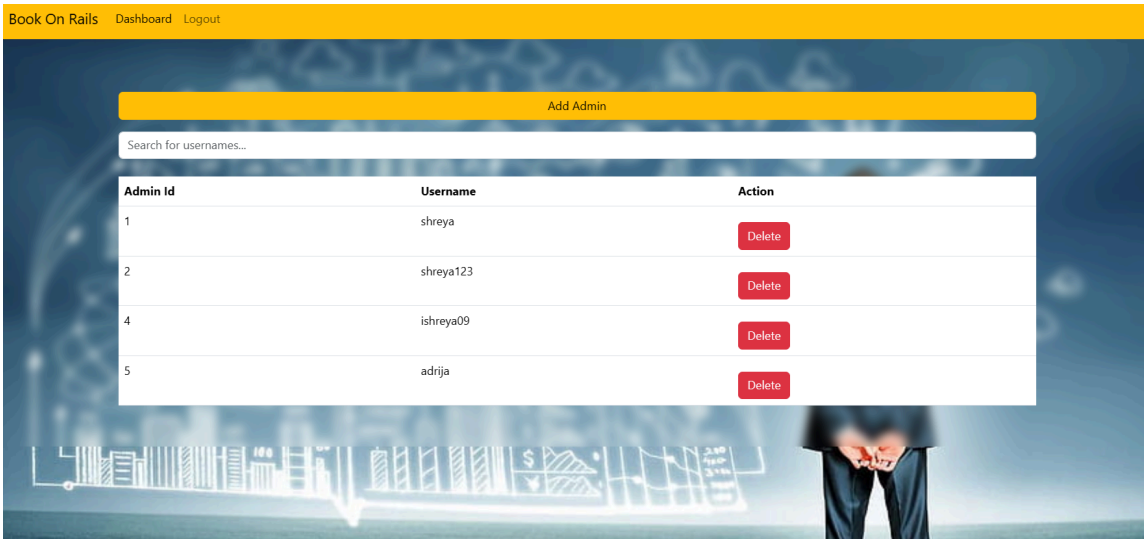
Admin Login



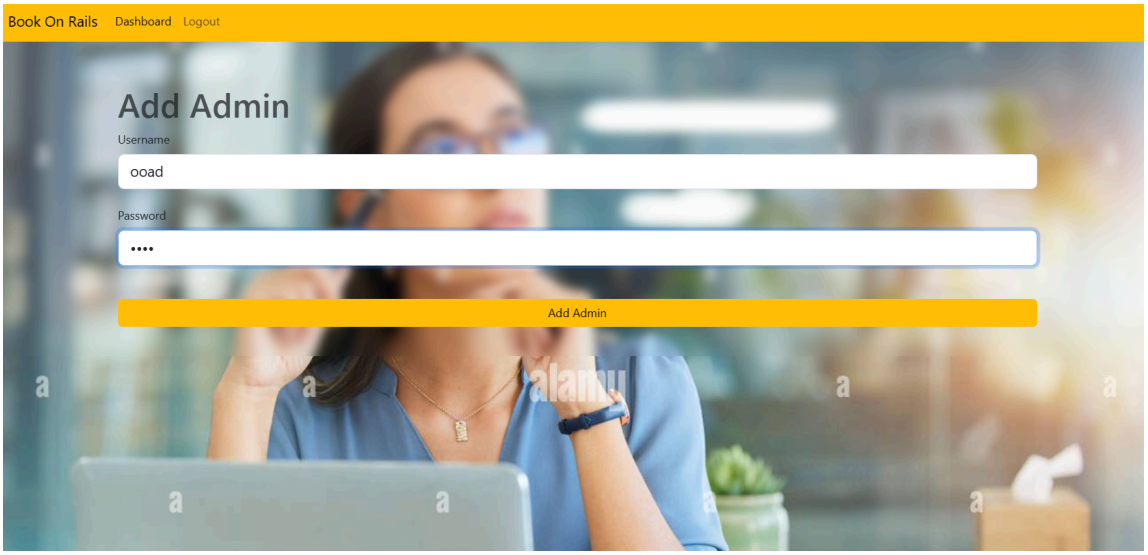
Admin management



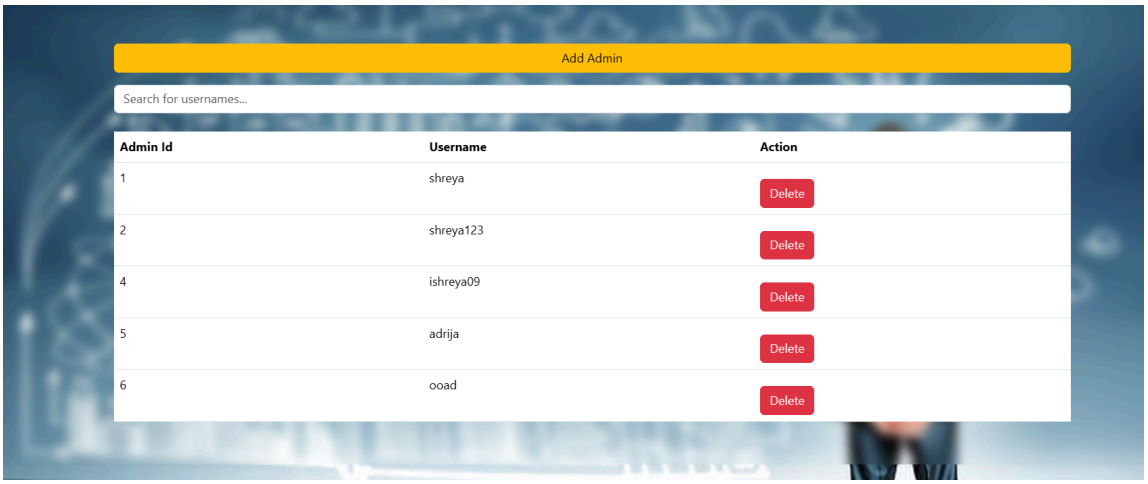
View Admin



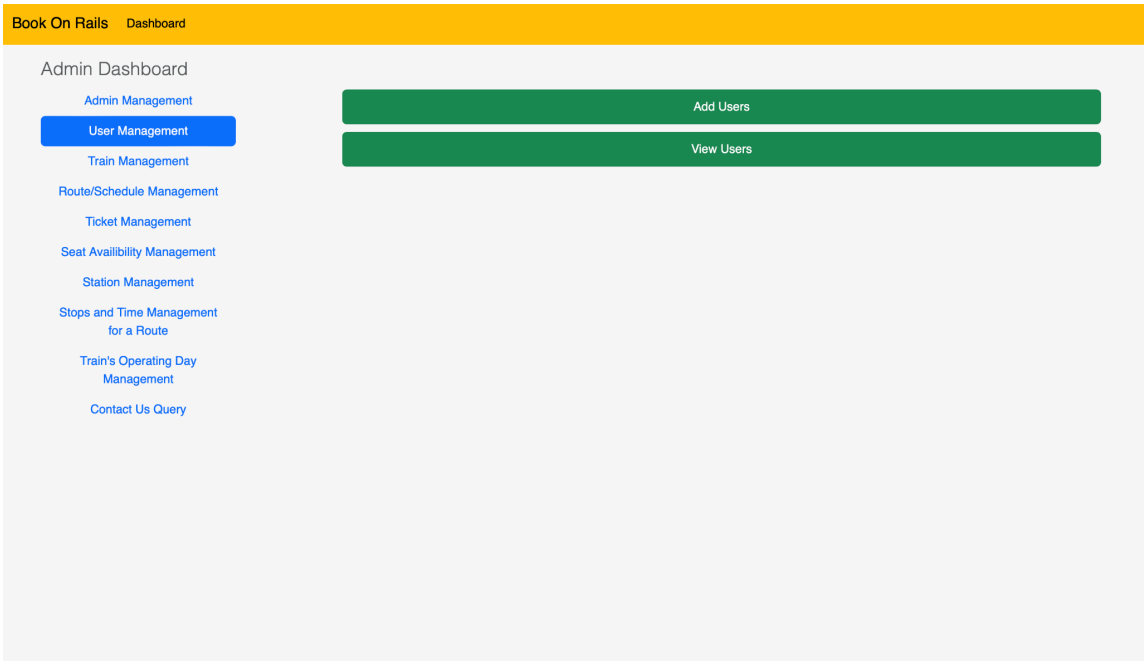
Add Admin



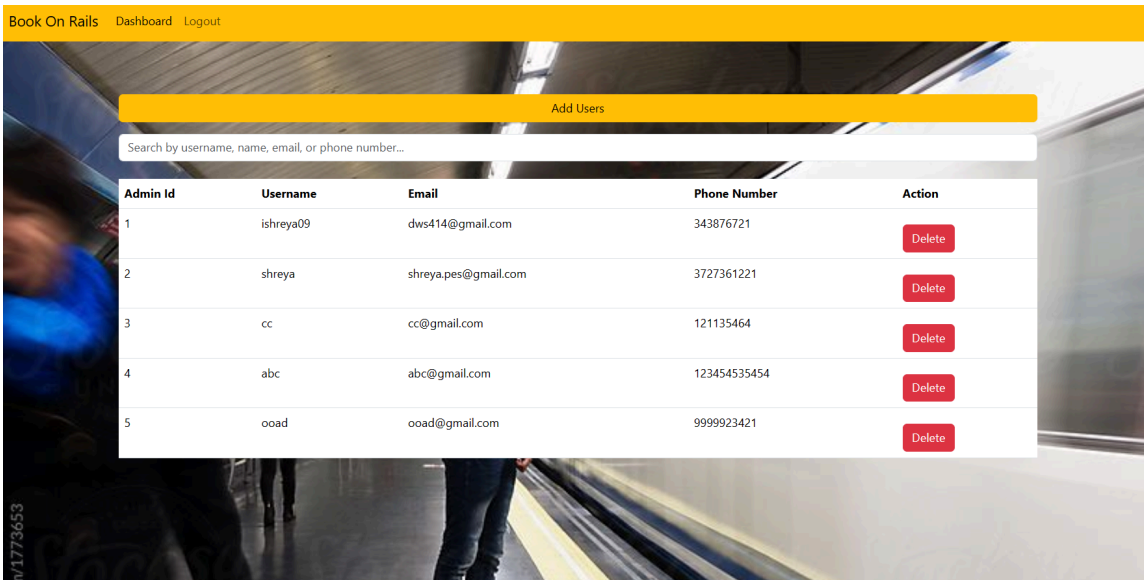
ooad admin user added



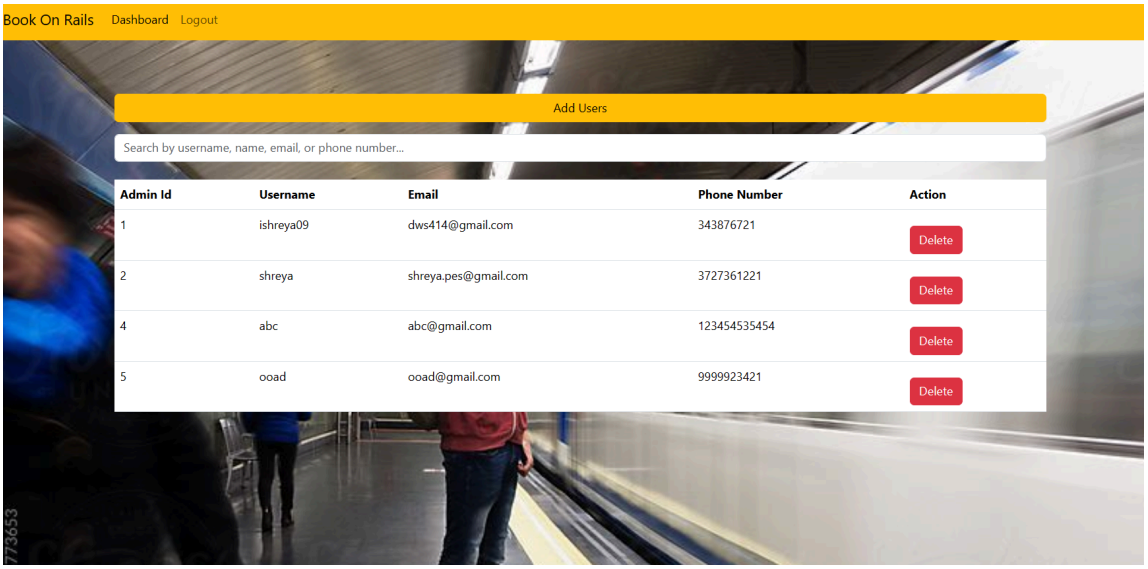
User management



View User

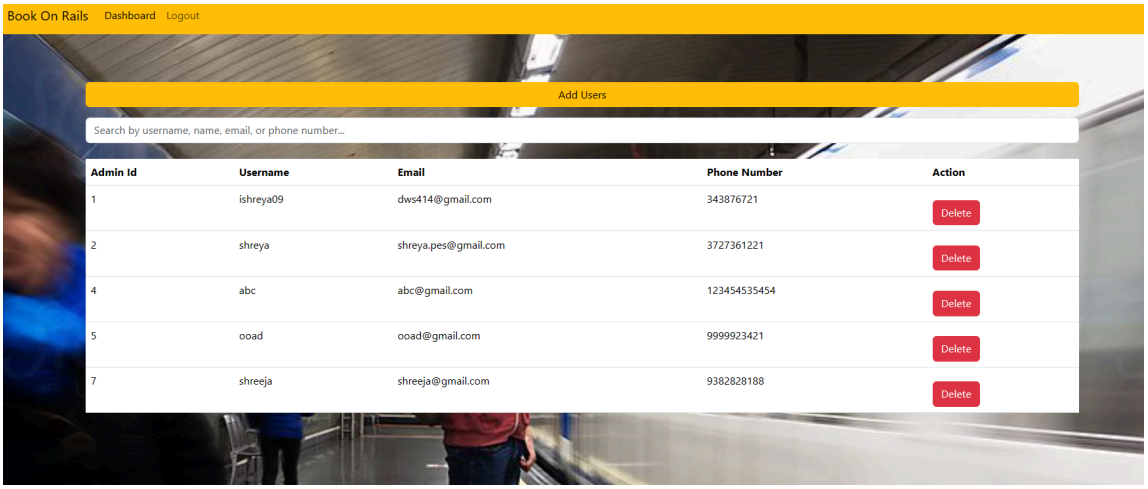


Admin can delete a user in case of suspicious activity (We will delete cc)

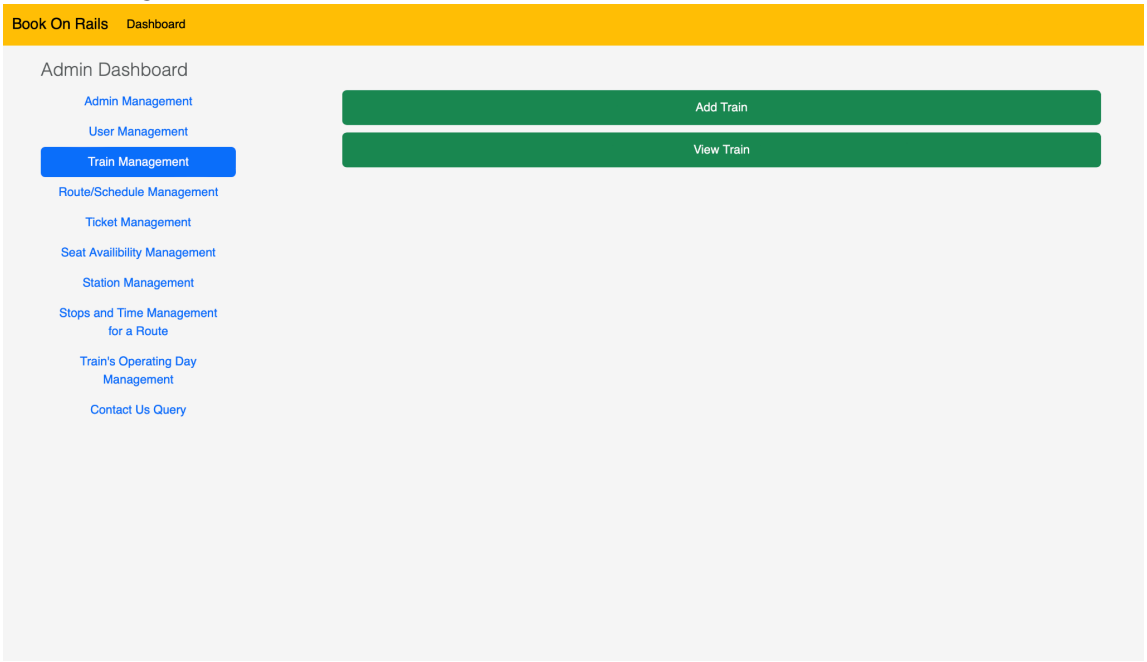


Add user

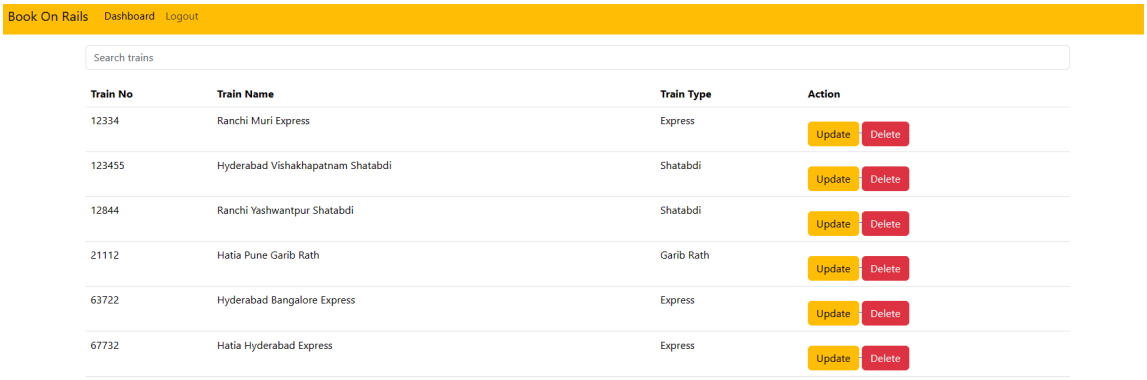
User added



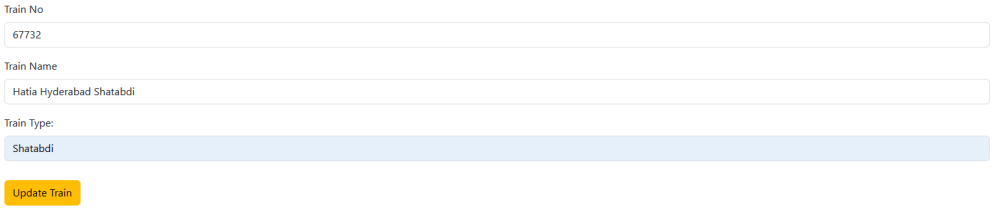
Train management



View Train



Update a train

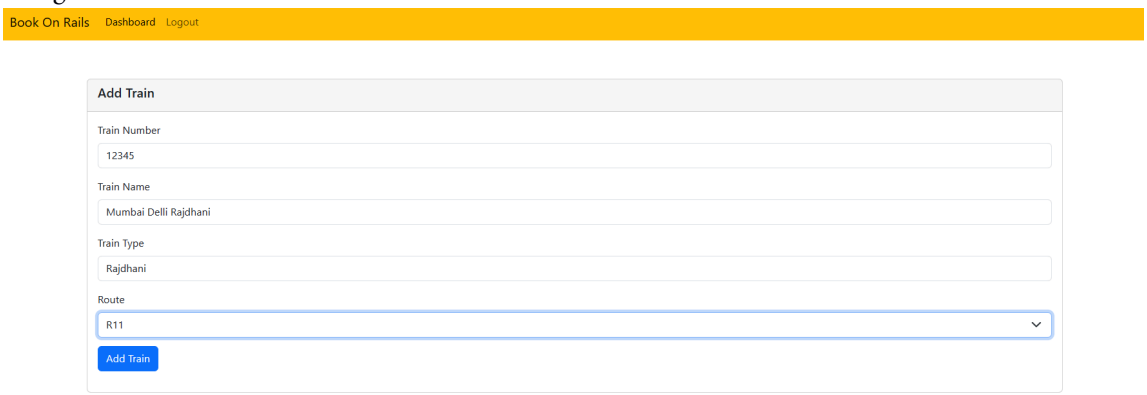


Last Train updated

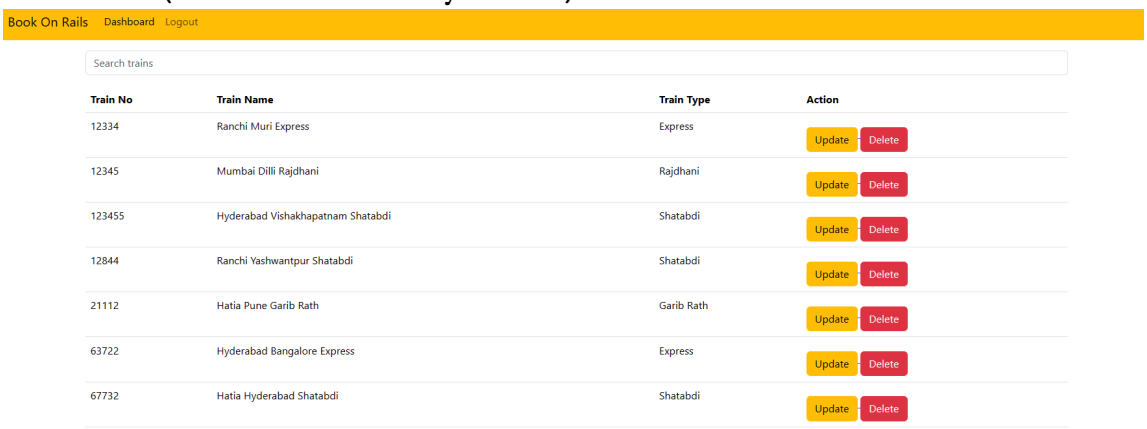
Train No	Train Name	Train Type	Action
12334	Ranchi Muri Express	Express	Update Delete
123455	Hyderabad Vishakhapatnam Shatabdi	Shatabdi	Update Delete
12844	Ranchi Yashwantpur Shatabdi	Shatabdi	Update Delete
21112	Hatia Pune Garib Rath	Garib Rath	Update Delete
63722	Hyderabad Bangalore Express	Express	Update Delete
67732	Hatia Hyderabad Shatabdi	Shatabdi	Update Delete

Add Train

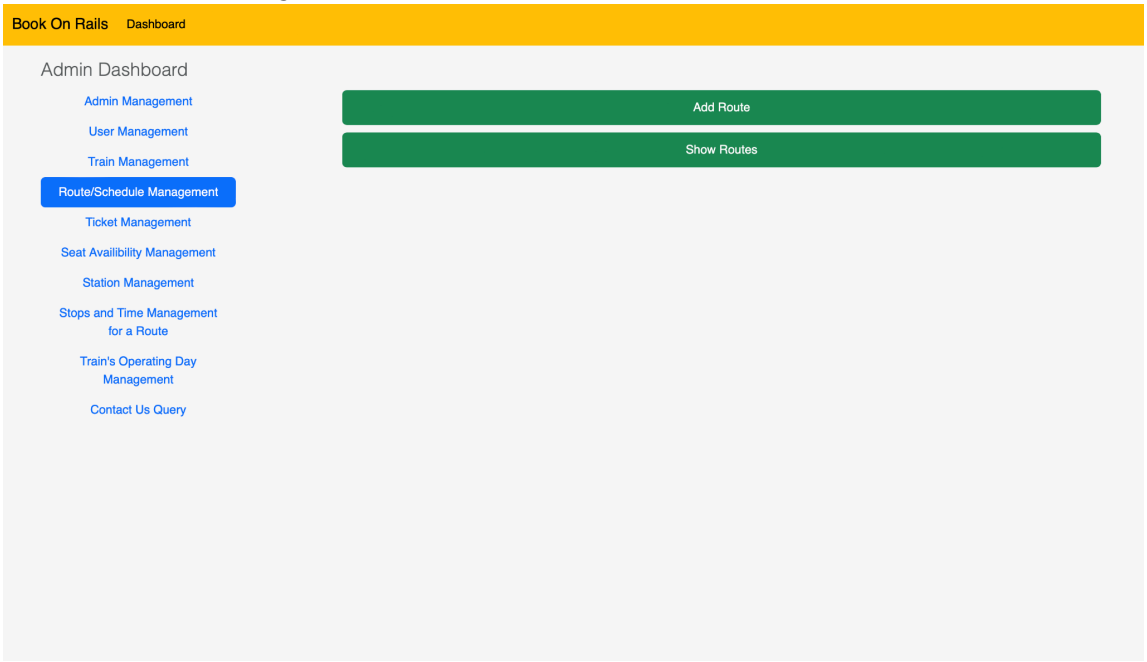
Using R11 route to add train now



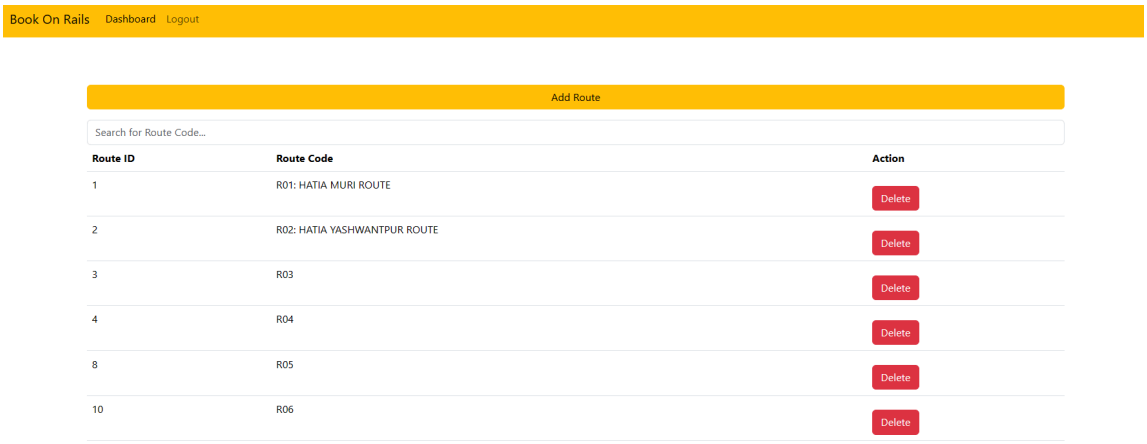
Train Added (2nd row- List sorted by train no)



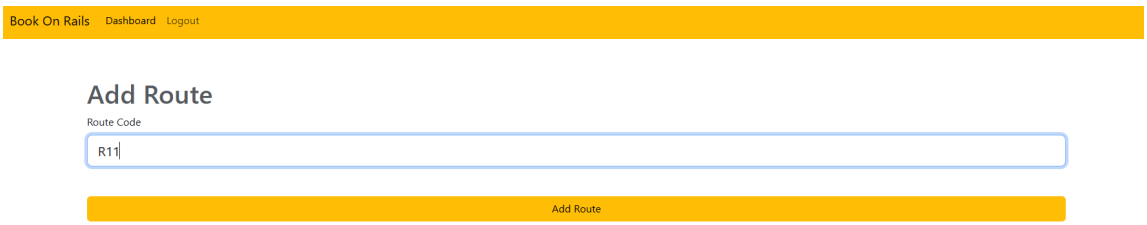
Route/ Schedule Management



View Route

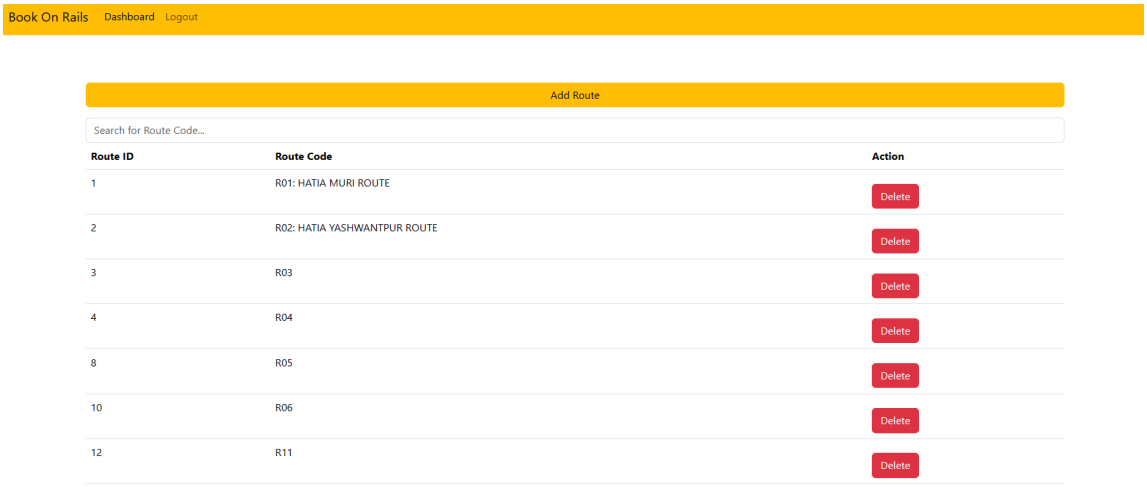


Add Route

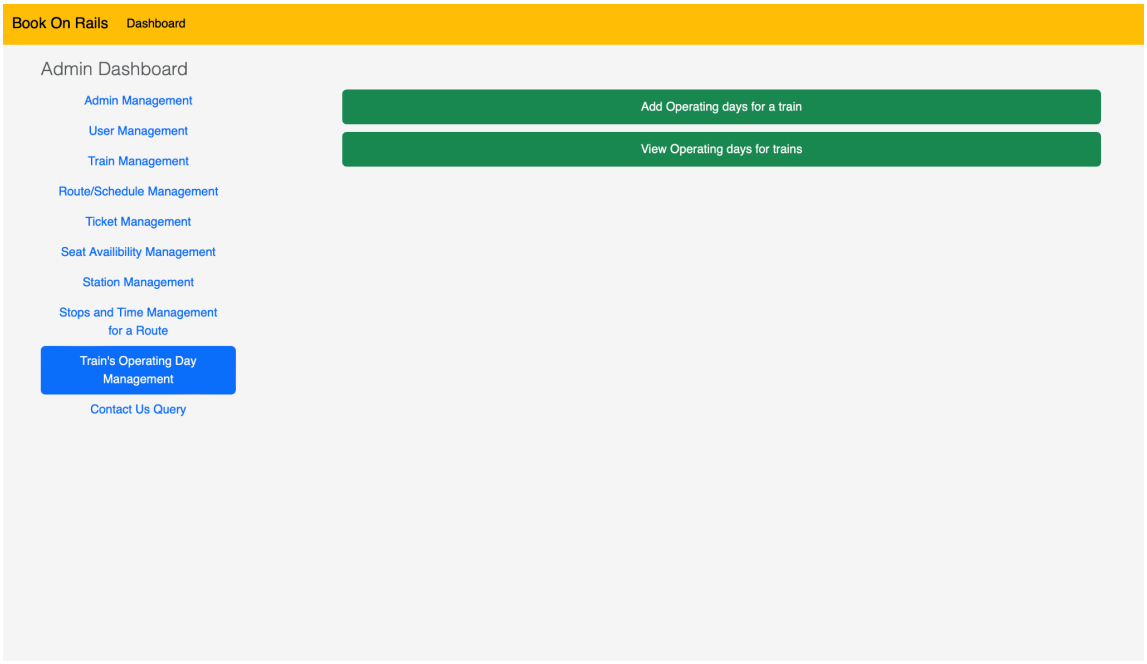


Route Added

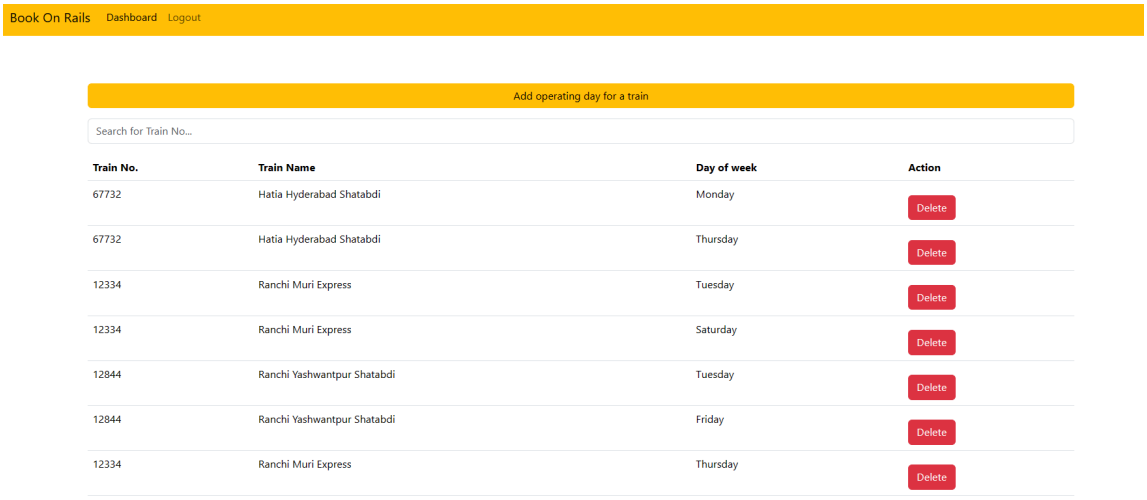
We will use the R11 route now to create a train.



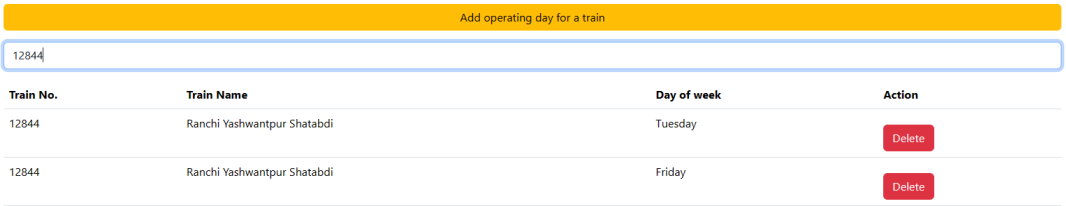
Train's Operating Day Management :



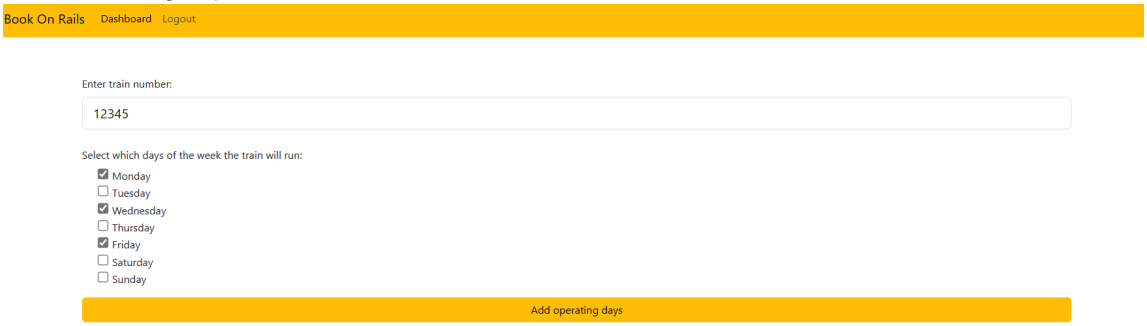
View Operating Days



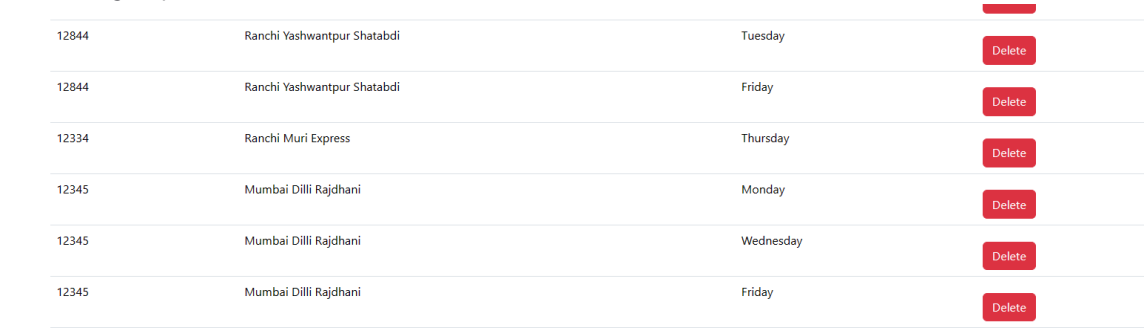
Can search by train no



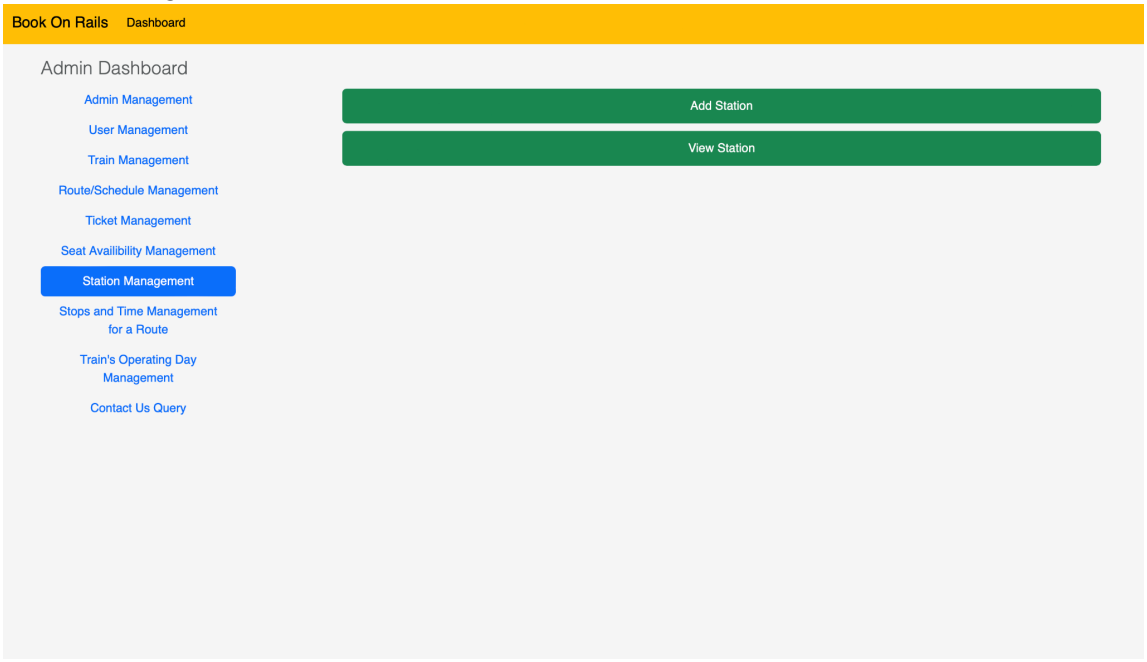
Add operating day



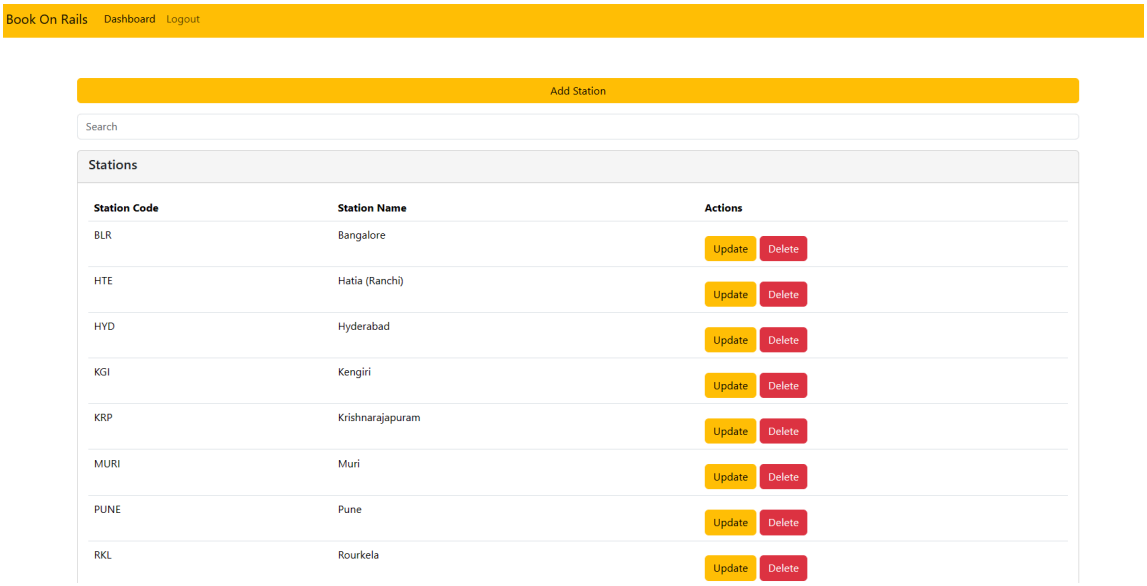
Operating days added



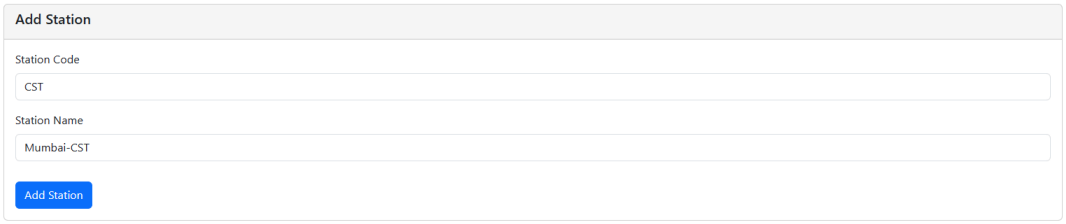
Station Management

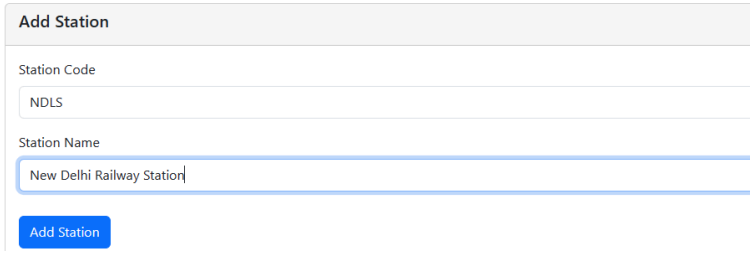


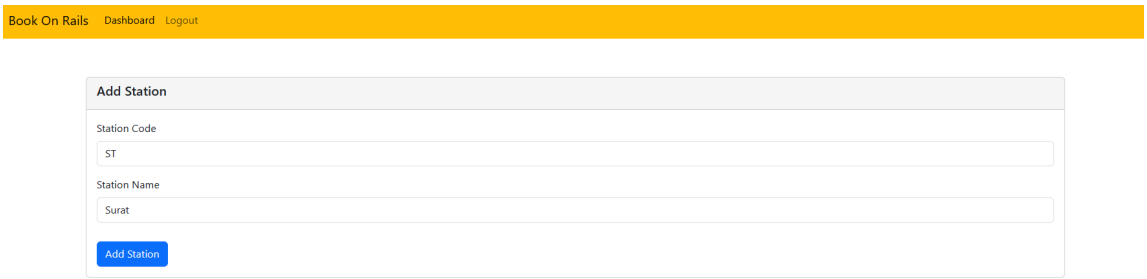
View Stations

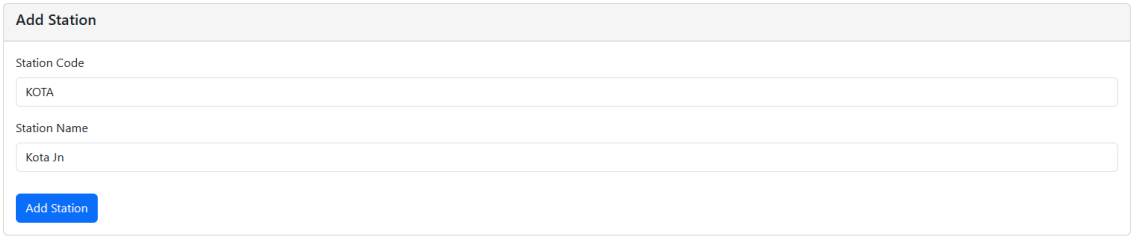


Add Station

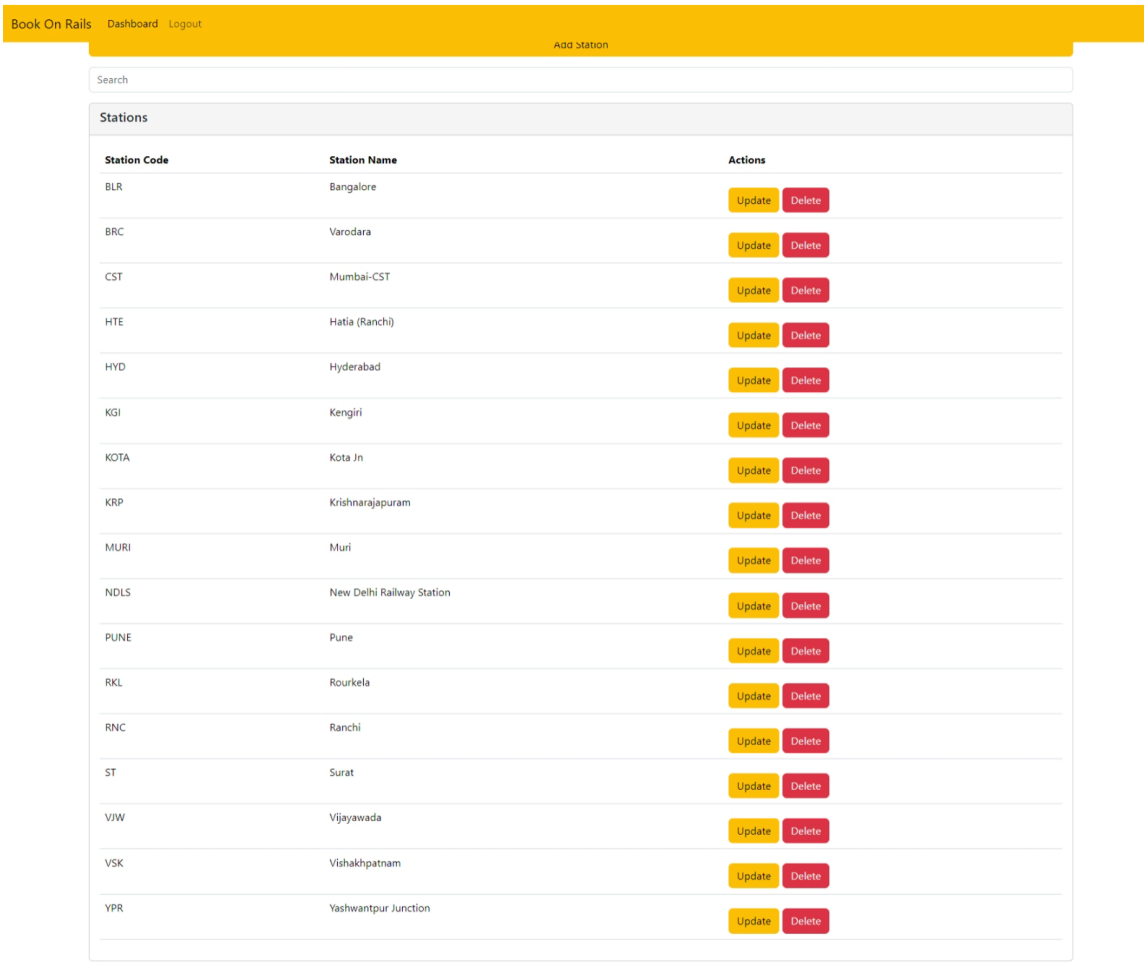




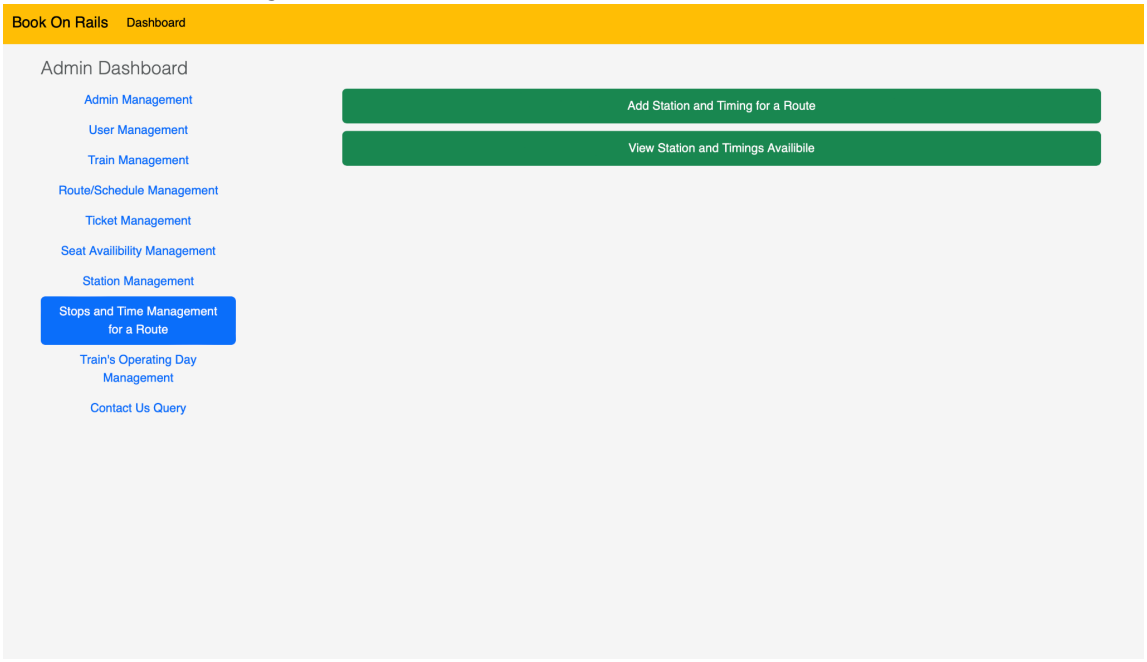




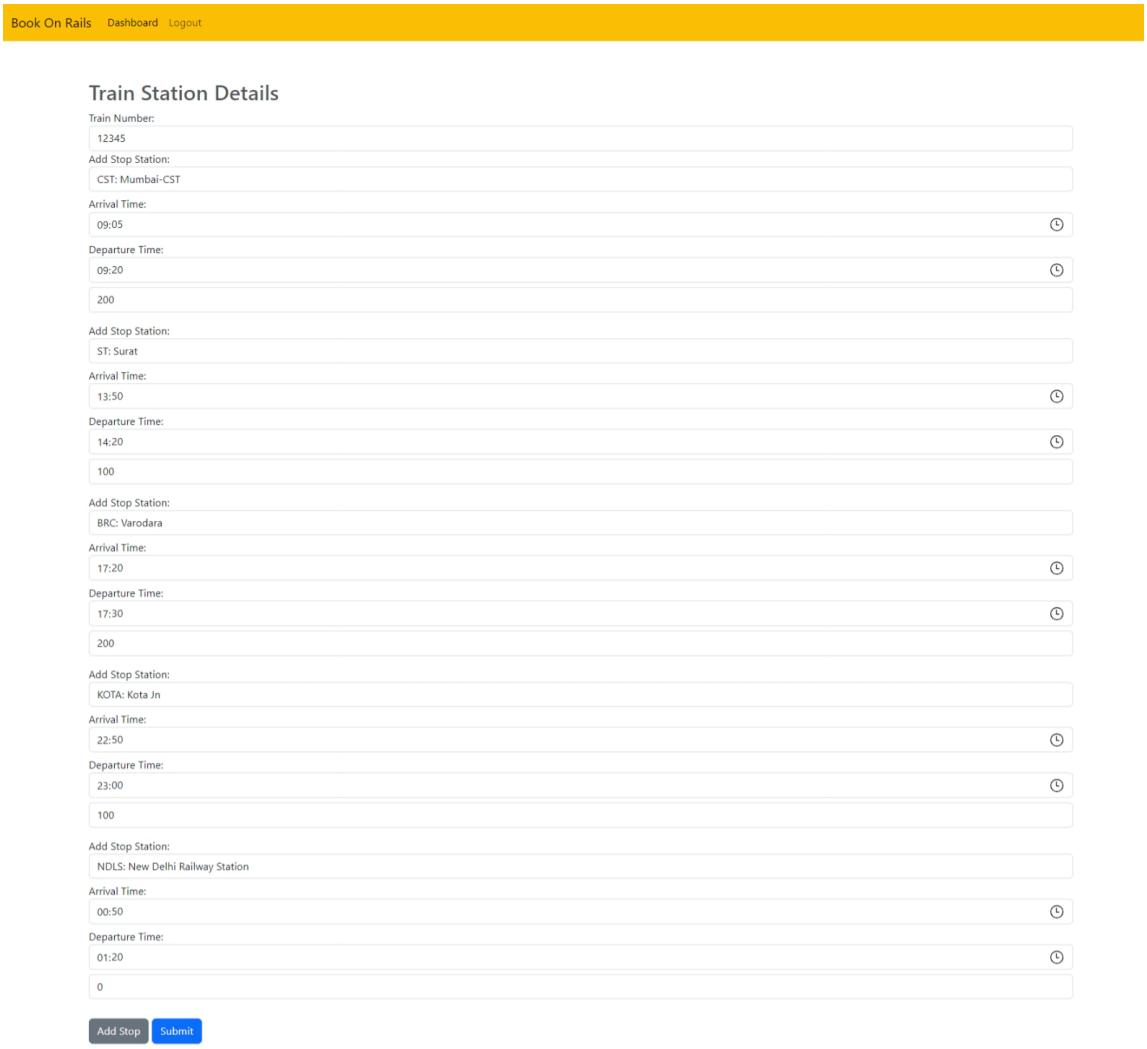
All stations Added



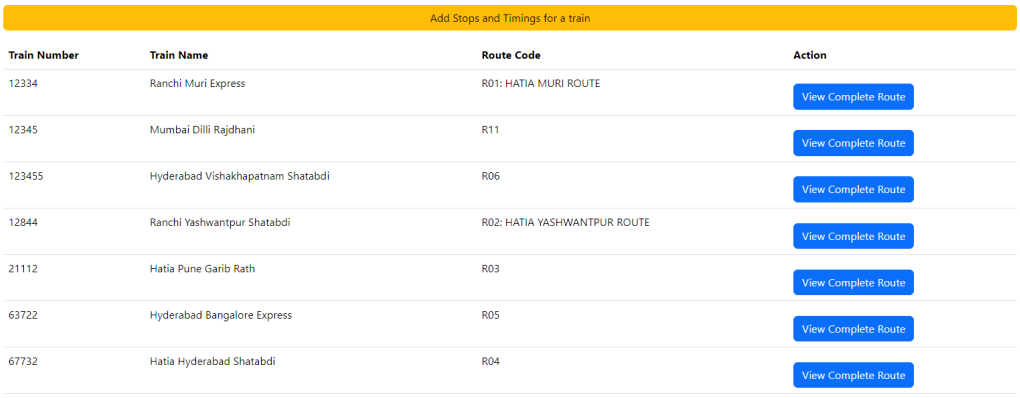
Stops and Time Management for a Route



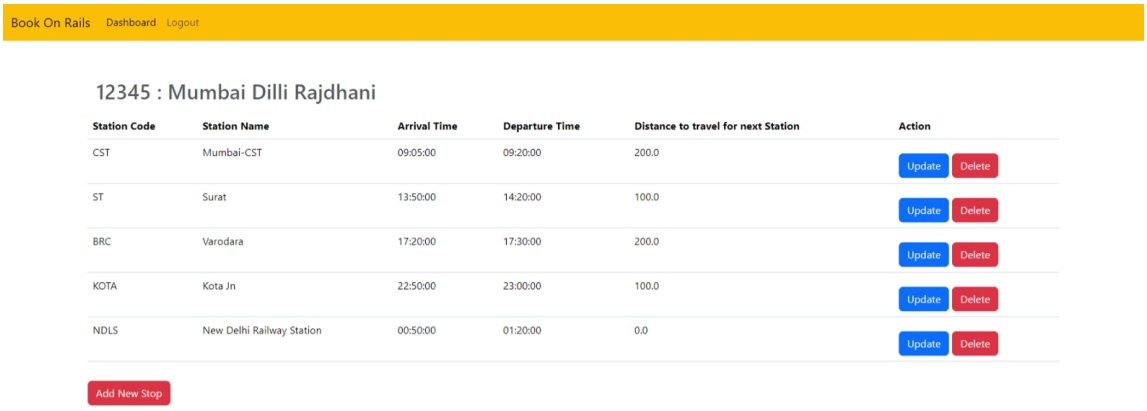
Add Stops for a train



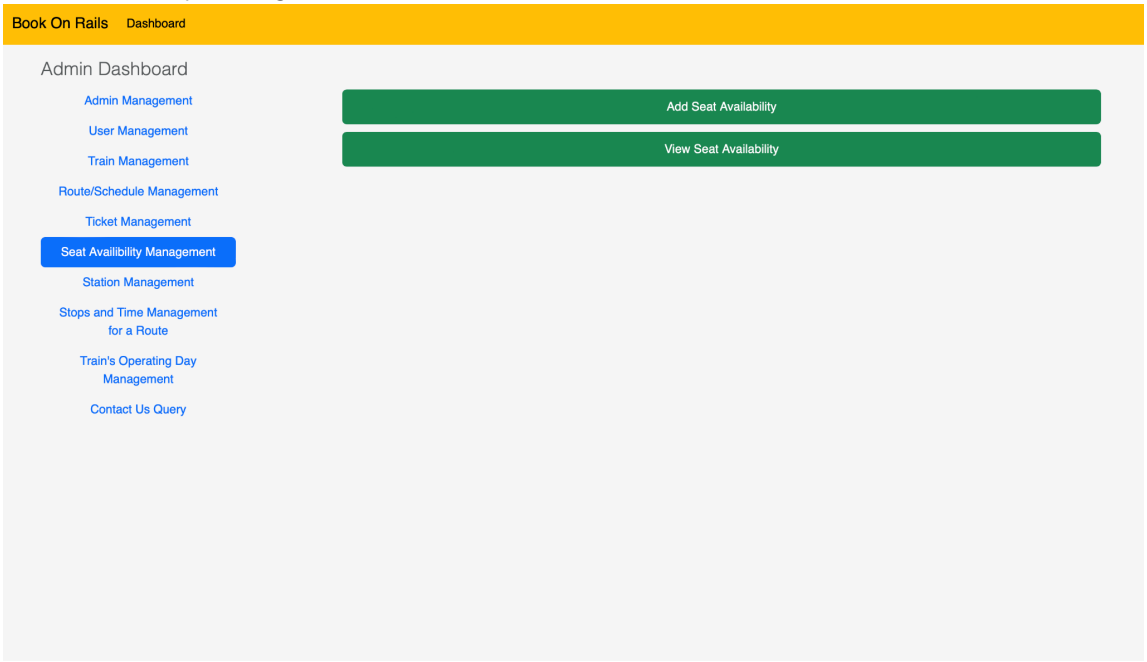
Stops for our train added



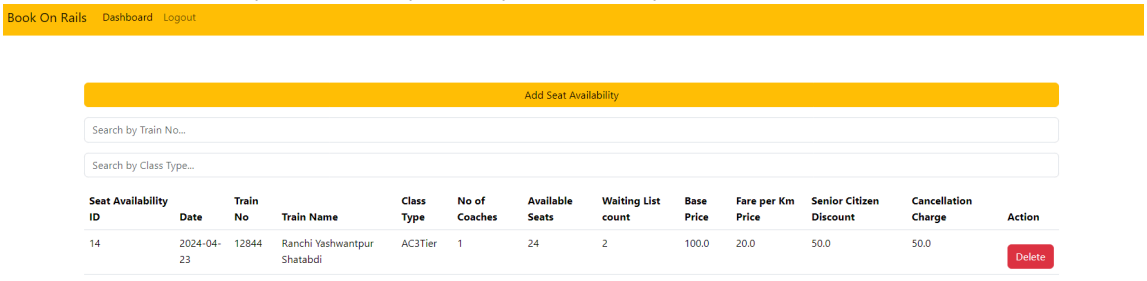
View Route for the added train



Seat Availability Management

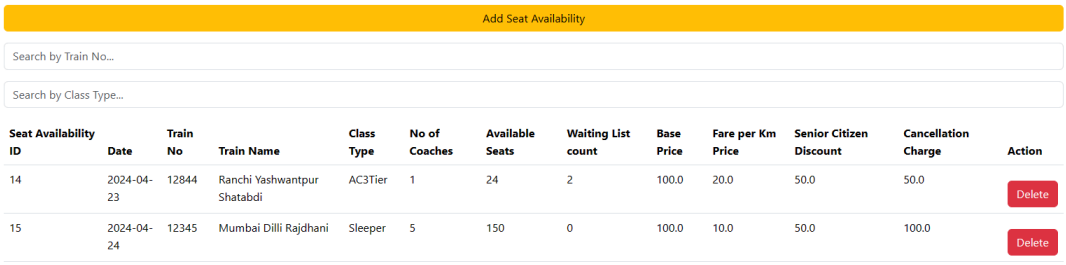


View Seat Availability (Shows only for days after today)

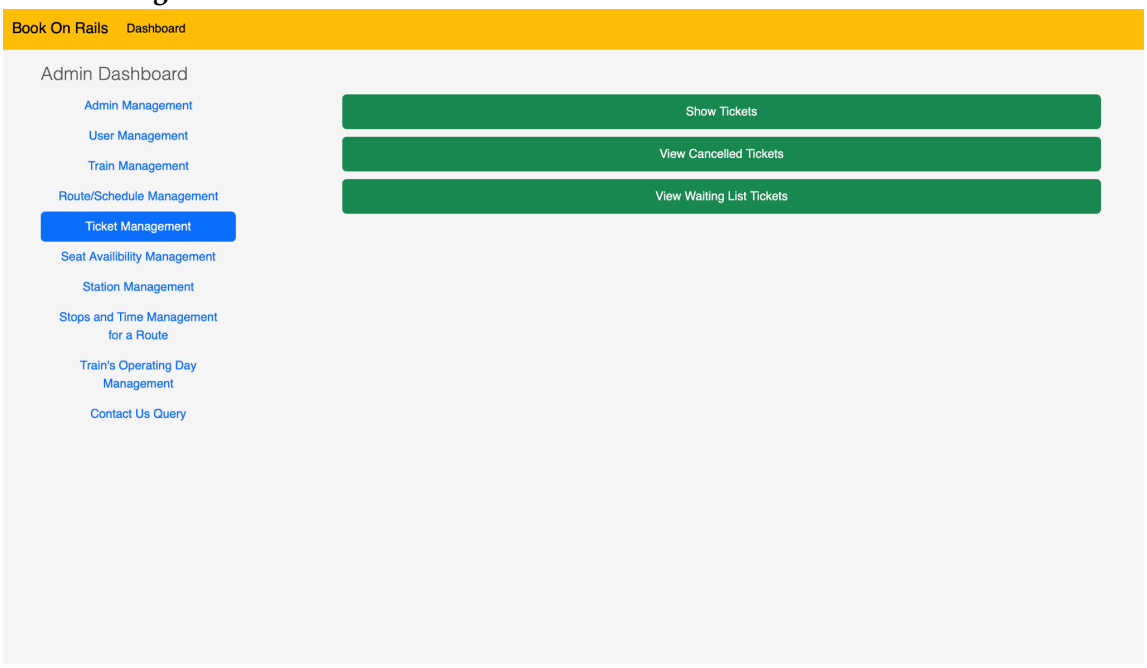


Add Seat Availability

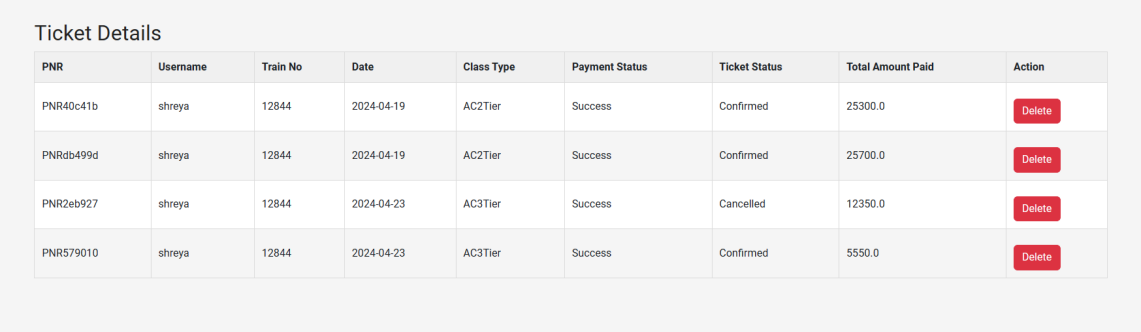
Seat Availability added



Ticket Management



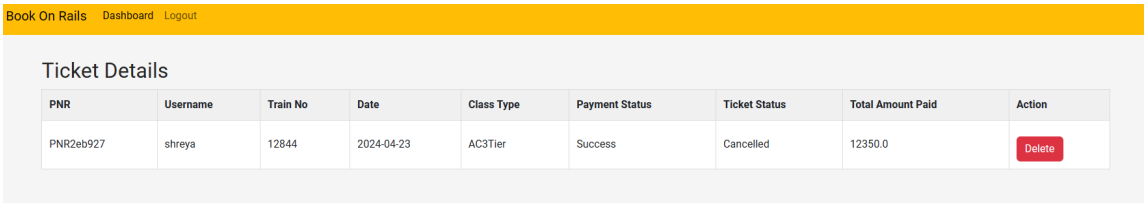
Show Tickets



Ticket Details

PNR	Username	Train No	Date	Class Type	Payment Status	Ticket Status	Total Amount Paid	Action
PNR40c41b	shreya	12844	2024-04-19	AC2Tier	Success	Confirmed	25300.0	Delete
PNRdb499d	shreya	12844	2024-04-19	AC2Tier	Success	Confirmed	25700.0	Delete
PNR2eb927	shreya	12844	2024-04-23	AC3Tier	Success	Cancelled	12350.0	Delete
PNR579010	shreya	12844	2024-04-23	AC3Tier	Success	Confirmed	5550.0	Delete

Show Cancelled Tickets

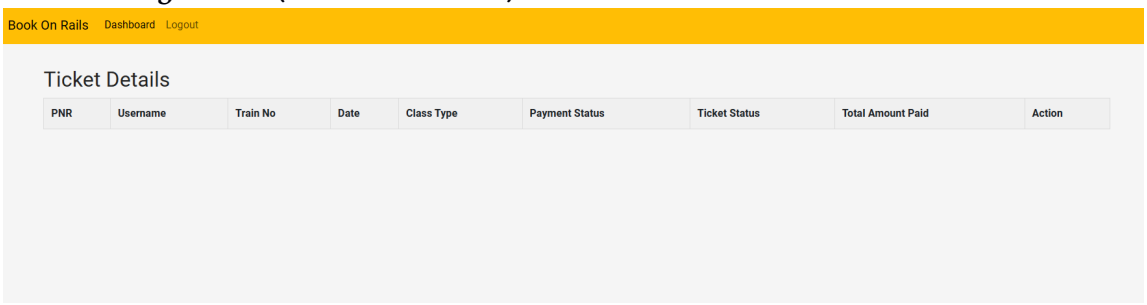


Book On Rails Dashboard Logout

Ticket Details

PNR	Username	Train No	Date	Class Type	Payment Status	Ticket Status	Total Amount Paid	Action
PNR2eb927	shreya	12844	2024-04-23	AC3Tier	Success	Cancelled	12350.0	Delete

Show Waiting Tickets (in this case : None)

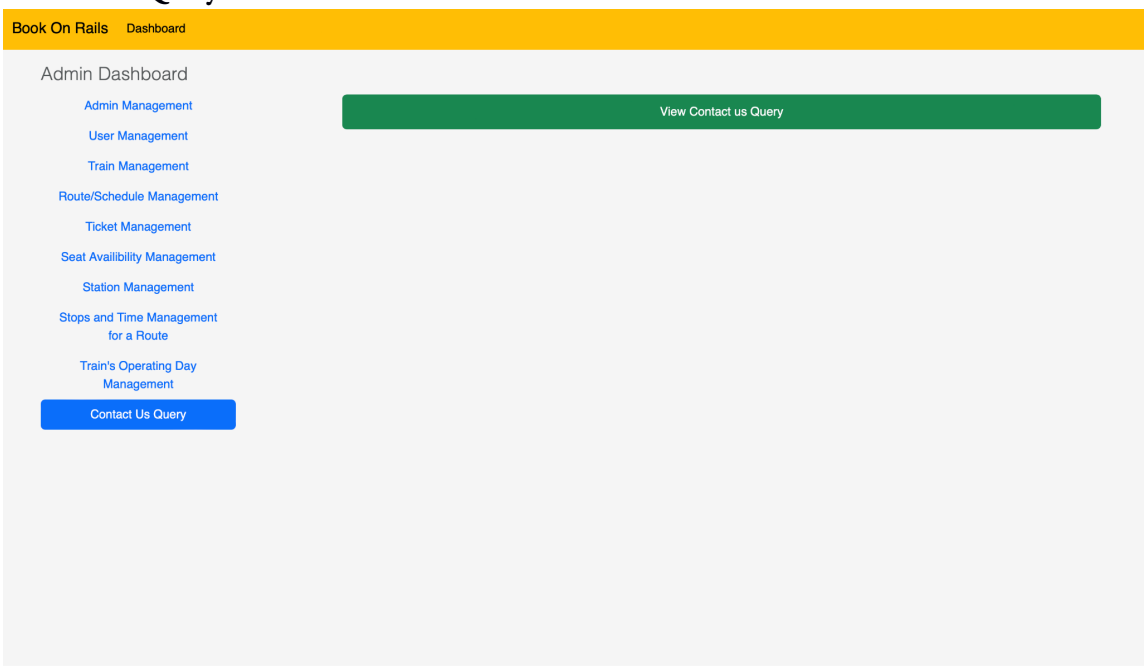


Book On Rails Dashboard Logout

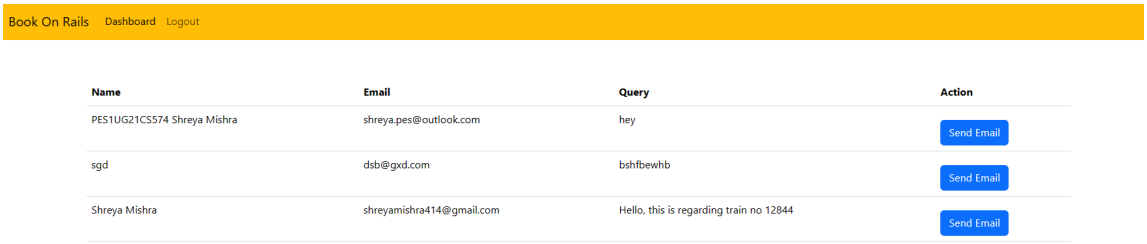
Ticket Details

PNR	Username	Train No	Date	Class Type	Payment Status	Ticket Status	Total Amount Paid	Action

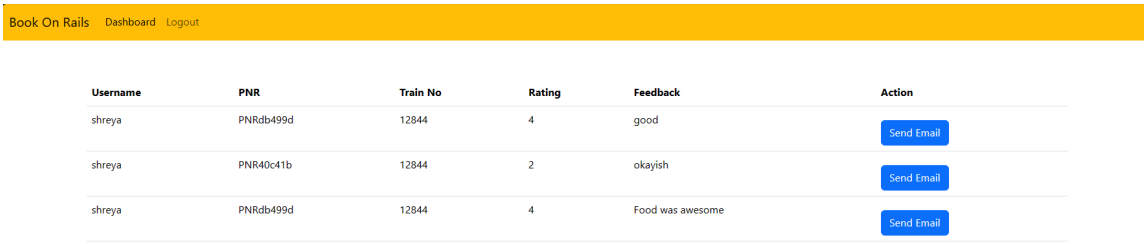
Contact Us Query :



View Contact us

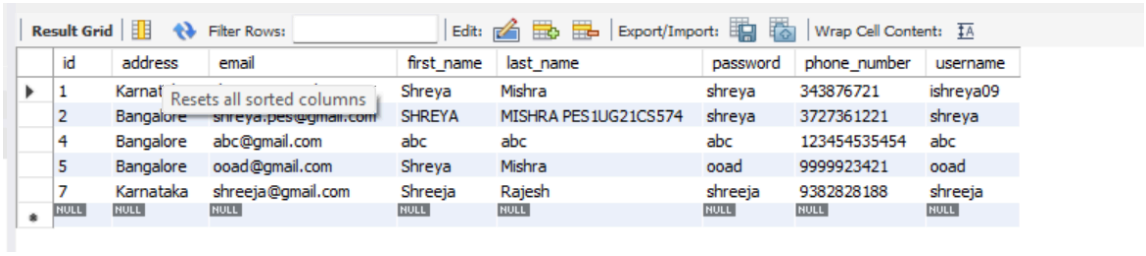


View Feedbacks

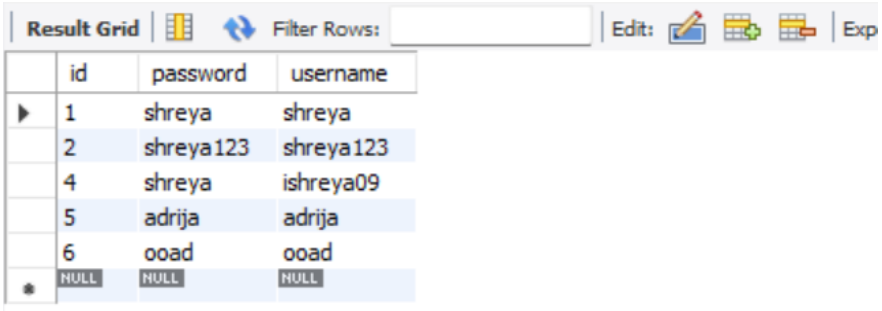


Database After Changes

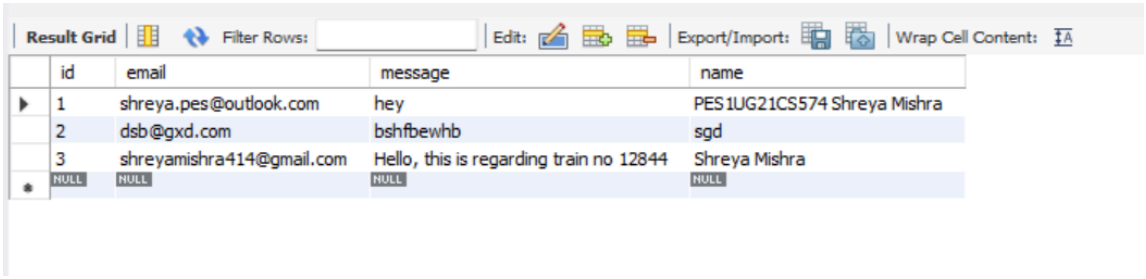
User



Admin



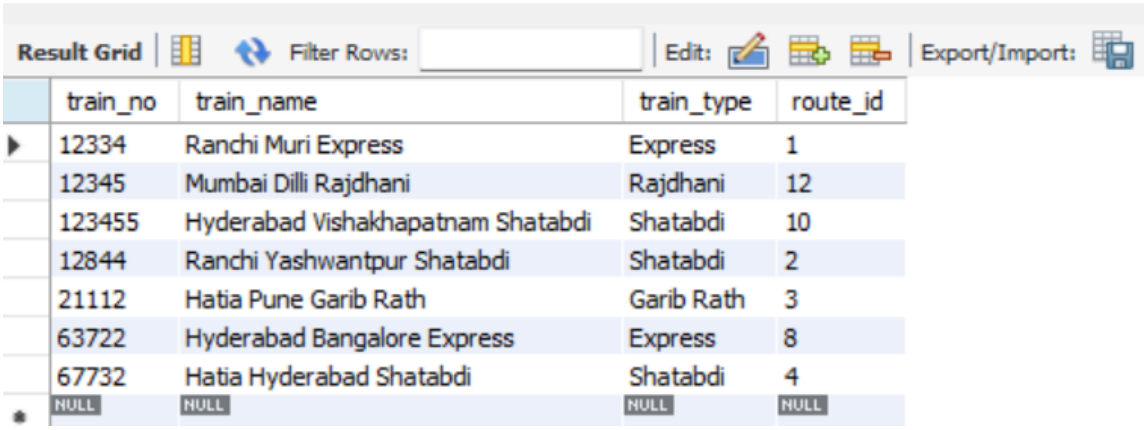
Contact



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	email	message	name
1	shreya.pes@outlook.com	hey	PES1UG21CS574 Shreya Mishra
2	dsb@gxd.com	bshfbewhb	sgd
3	shreyamishra414@gmail.com	Hello, this is regarding train no 12844	Shreya Mishra
NULL	NULL	NULL	NULL

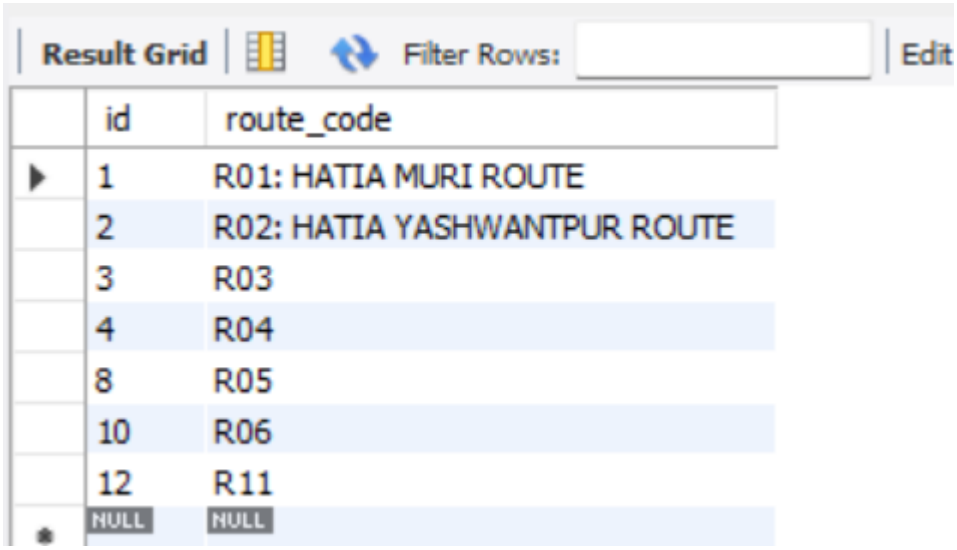
Train



Result Grid | Filter Rows: | Edit: | Export/Import:

train_no	train_name	train_type	route_id
12334	Ranchi Muri Express	Express	1
12345	Mumbai Dilli Rajdhani	Rajdhani	12
123455	Hyderabad Vishakhapatnam Shatabdi	Shatabdi	10
12844	Ranchi Yashwantpur Shatabdi	Shatabdi	2
21112	Hatia Pune Garib Rath	Garib Rath	3
63722	Hyderabad Bangalore Express	Express	8
67732	Hatia Hyderabad Shatabdi	Shatabdi	4
NULL	NULL	NULL	NULL

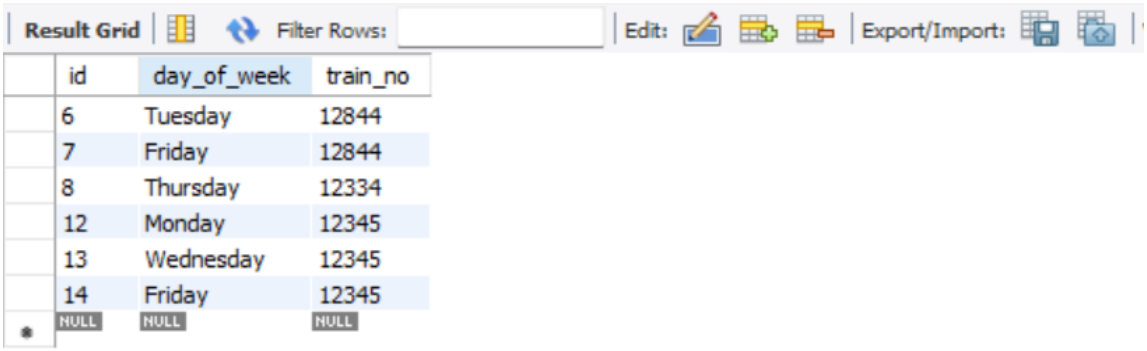
Route



Result Grid | Filter Rows: | Edit

id	route_code
1	R01: HATIA MURI ROUTE
2	R02: HATIA YASHWANTPUR ROUTE
3	R03
4	R04
8	R05
10	R06
12	R11
NULL	NULL

Operating Day



Result Grid | Filter Rows: | Edit: | Export/Import:

id	day_of_week	train_no
6	Tuesday	12844
7	Friday	12844
8	Thursday	12334
12	Monday	12345
13	Wednesday	12345
14	Friday	12345
NULL	NULL	NULL

Station Timings

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	arrival_time	departure_time	distance_from_next_station	route_id	station_code
23	10:00:00.000000	10:30:00.000000	0	8	BLR
25	09:05:00.000000	09:20:00.000000	200	12	CST
26	13:50:00.000000	14:20:00.000000	100	12	ST
27	17:20:00.000000	17:30:00.000000	200	12	BRC
28	22:50:00.000000	23:00:00.000000	100	12	KOTA
29	00:50:00.000000	01:20:00.000000	0	12	NDLS
NULL	NULL	NULL	NULL	NULL	NULL

Station

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

station_code	station_name
BRC	Varodara
CST	Mumbai-CST
HTE	Hatia (Ranchi)
HYD	Hyderabad
KGI	Kengiri
KOTA	Kota Jn
KRP	Krishnarajapuram

Seat Availability

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	available_seats	base_price	cancellation_charge	classes	date	fare_perkm	last_lower_seat	last_unbooked_lower_seat	last_
11	30	100	50	Sleeper	2024-04-19	20	29	1	2
12	30	100	50	AC3Tier	2024-04-19	20	29	1	2
13	0	100	50	AC2Tier	2024-04-19	20	9	11	12
14	22	100	50	AC3Tier	2024-04-23	20	29	9	10
15	150	100	100	Sleeper	2024-04-24	10	149	1	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

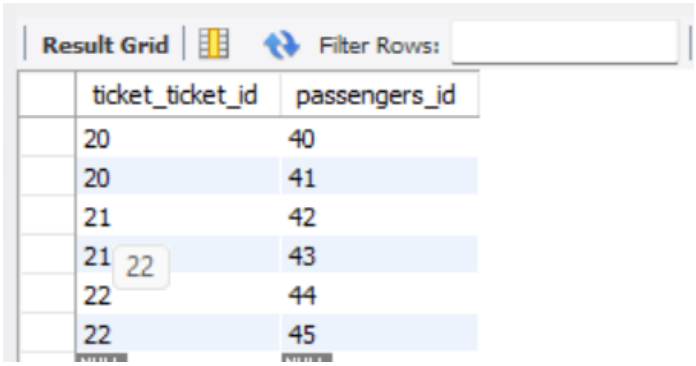
cm	last_lower_seat	last_unbooked_lower_seat	last_unbooked_upper_seat	last_upper_seat	no_of_coaches	senior_citizen_discount	train_no
	29	1	2	30	1	50	12844
	29	1	2	30	1	50	12844
	9	11	12	10	1	50	12844
	29	9	10	30	1	50	12844
	149	1	2	150	5	50	12345
	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ticket

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

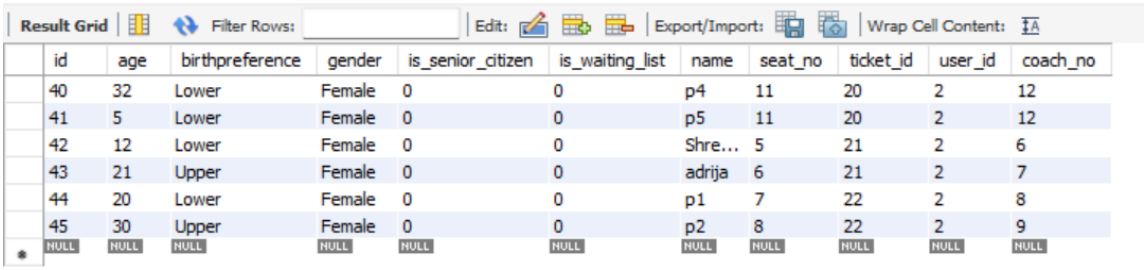
ticket_id	pnr	waiting_list_number	classes	date	foodprice	payment_status	quantity	status	total_amount	veg	war
19	PNR40c41b	-1	AC2Tier	2024-04-19	800	Success	2	Confirmed	24500	0	1
20	PNRdb499d	-1	AC2Tier	2024-04-19	1200	Success	4	Confirmed	24500	1	1
21	PNR2eb927	-1	AC3Tier	2024-04-23	800	Success	2	Cancelled	11550	0	1
22	PNR579010	-1	AC3Tier	2024-04-23	600	Success	2	Confirmed	4950	1	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ticket_Passengers



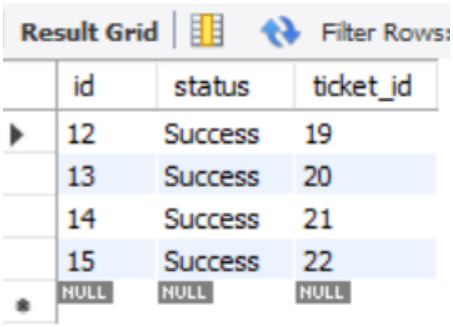
ticket_ticket_id	passengers_id
20	40
20	41
21	42
21	43
22	44
22	45

Passenger



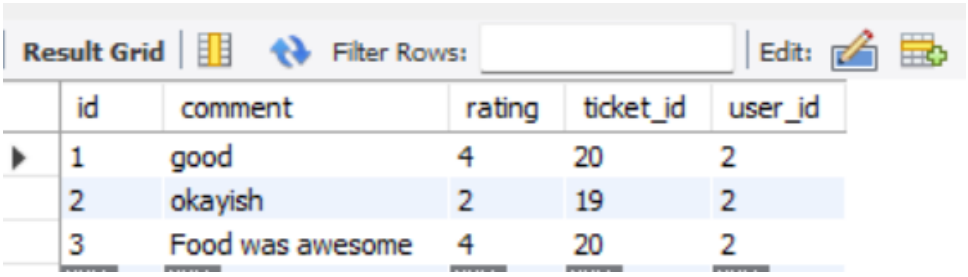
id	age	birthpreference	gender	is_senior_citizen	is_waiting_list	name	seat_no	ticket_id	user_id	coach_no
40	32	Lower	Female	0	0	p4	11	20	2	12
41	5	Lower	Female	0	0	p5	11	20	2	12
42	12	Lower	Female	0	0	Shre...	5	21	2	6
43	21	Upper	Female	0	0	adrija	6	21	2	7
44	20	Lower	Female	0	0	p1	7	22	2	8
45	30	Upper	Female	0	0	p2	8	22	2	9

Payment



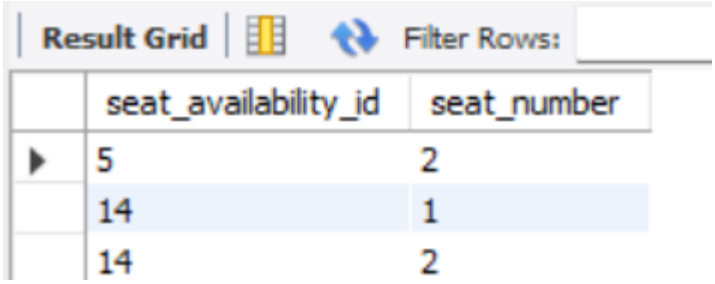
id	status	ticket_id
12	Success	19
13	Success	20
14	Success	21
15	Success	22

Feedback



id	comment	rating	ticket_id	user_id
1	good	4	20	2
2	okayish	2	19	2
3	Food was awesome	4	20	2

Cancelled_seats



seat_availability_id	seat_number
5	2
14	1
14	2