



*Project on*

**“Railway Reservation System”**

*Submitted in partial fulfillment of the requirements for the award of degree of*

**Bachelor of Technology  
in  
Computer Science & Engineering**

**UE21CS352B – Object Oriented Analysis & Design using Java**

*Submitted by:*

<b>Srikrishna Sripati Nayak</b>	<b>PES1UG21CS620</b>
<b>Srikrishna B</b>	<b>PES1UG21CS618</b>
<b>Sudeep Dhotre</b>	<b>PES1UG21CS631</b>
<b>Srijan Badhya</b>	<b>PES1UG21CS616</b>

*Under the guidance of*

**Prof. Bhargavi Mokashi**  
Professor

**January - May 2024**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
FACULTY OF ENGINEERING  
PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)

## Problem Statement

This is a simple Railway Reservation System built using Java Spring Boot Framework and MySQL database. The system allows users to book train tickets and view their bookings.

Features:

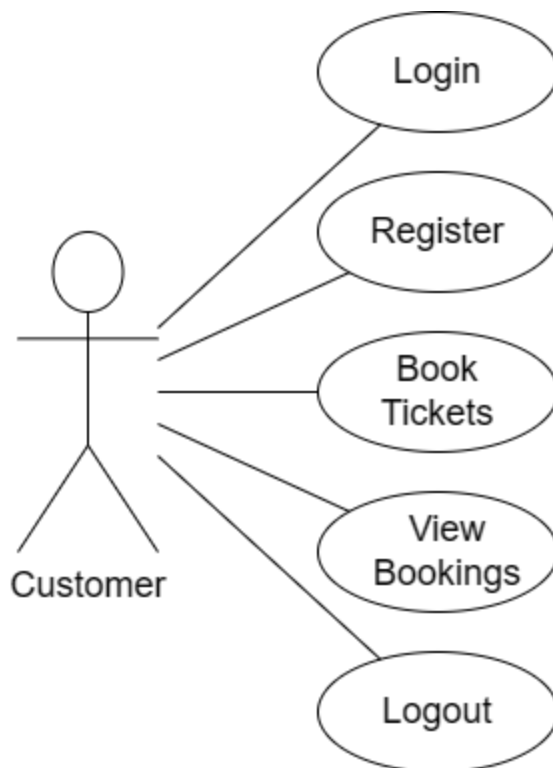
User registration and authentication

Book train tickets

View booked tickets

## Analysis and Design Models

### 1. Use Case Diagram



2. Class Diagram
3. Activity Diagram
4. State Diagram

## Software Design Principles Followed with screenshots of code

Single Responsibility Principle (SRP):

Each class and component in our system has a single responsibility. For example, controllers handle the user's requests, services contain business logic, repositories handle database operations, etc. This principle helps in making the codebase more modular, maintainable, and testable.

All of our classes are in charge of only one job.

```
@Table(name = "roles")
public class Role {
    @Id no usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Getter 2 usages
    @Enumerated(EnumType.STRING)
    @Column(length = 20)
    private ERole name;

    > public Role() { this.name = ERole.ROLE_USER; }

    > public Role(ERole name) { this.name = name; }
}
```

Interface Segregation (IS):

Each interface involves only the function that it needs to. The client isn't exposed to extra functions which could cause errors and make the code messy. So our three different interfaces keep the code clean.

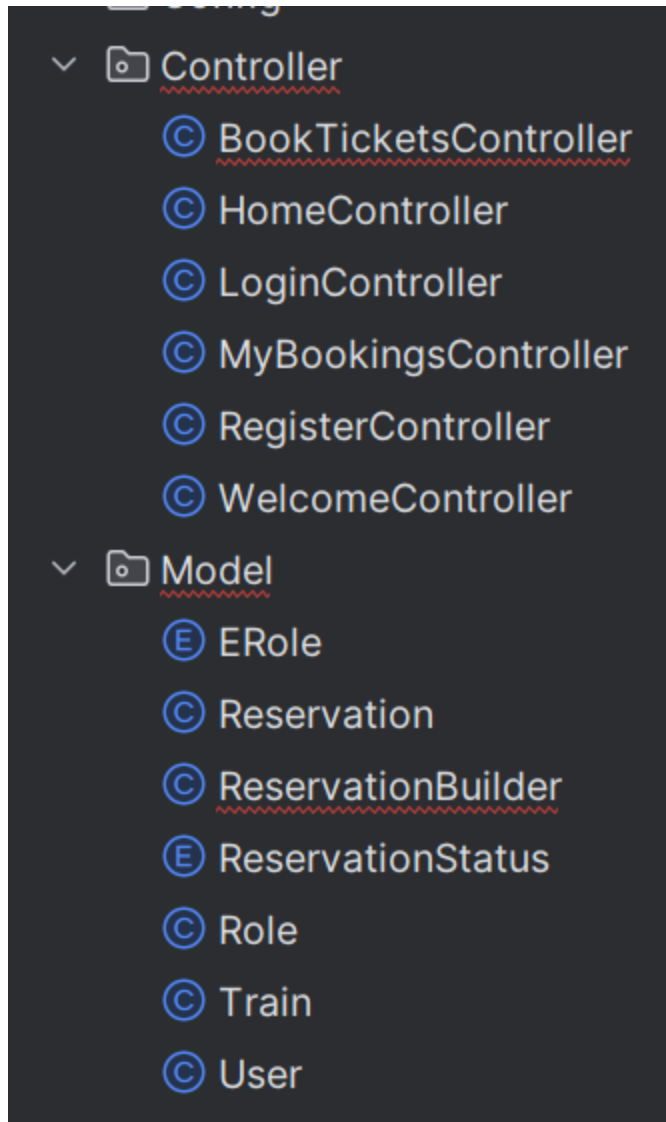
```
package com.ooadclass.railway_reservation_new.Repository;

> import ...

@Repository 2 usages Srikrishna S Nayak
public interface ReservationRepository extends JpaRepository<Reservation, Long> {
    List<Reservation> findByUser(User user); 1 usage Srikrishna S Nayak
}
```

Model-View-Controller (MVC) Architecture:

Our project follows the MVC architectural pattern, where models represent data, controllers handle user input and interaction, and views display the user interface. This separation of concerns makes the codebase more organized and easier to maintain.



And the different views come into play on the My bookings page, where the viewer can also view their section of the database which consists of their tickets only.

Open/Closed Principle (OCP):

Our codebase is designed to be open for extension but closed for modification. This means that we can add new features or make changes to existing ones without modifying the existing codebase extensively.

Hence you can make children out of our classes and extend, but we don't have too much code within the class itself which modifies a lot.

```

@Service 2 usages Srikrishna S Nayak
public class TrainService {
    @Autowired 8 usages
    private TrainRepository trainRepository;

    public List<Train> getAllTrains() { return trainRepository.findAll(); }

    public Train getTrainById(Long id) { return trainRepository.findById(id).orElse(oth

    public Train saveTrain(Train train) { return trainRepository.save(train); }

    public void deleteTrain(Long id) { trainRepository.deleteById(id); }

    public Set<String> getAllStations() { no usages Srikrishna S Nayak
        List<Train> trains = trainRepository.findAll();
        return trains.stream() Stream<Train>
            .flatMap(train -> Stream.of(train.getSource(), train.getDestination()))
            .distinct()
            .collect(Collectors.toSet());
    }
}

```

## Software Design Patterns Used with screenshots of code

Builder Pattern:

The Builder pattern is implemented with the ReservationBuilder class. This pattern is used to construct a complex object step by step. In our railway ticket booking system, the ReservationBuilder is used to create reservation objects by setting various attributes of the reservation in a step-by-step manner.

```
private LocalDate reservationDate; 2 usages
private ReservationStatus status; 2 usages
private Long bookedTrainID; 2 usages

public ReservationBuilder setUser(User user) { 1 Srikrishna S Nayak
    this.user = user;
    return this;
}

public ReservationBuilder setTrain(Train train) { 1 Srikrishna S Nayak
    this.train = train;
    return this;
}

public ReservationBuilder setNumberOfSeats(int numberOfSeats) { 1 usage 1 Srikrishna S Nayak
    this.numberOfSeats = numberOfSeats;
    return this;
}

public ReservationBuilder setTotalAmount(double totalAmount) { 1 usage 1 Srikrishna S Nayak
    this.totalAmount = totalAmount;
    return this;
}

public ReservationBuilder setBookingTime(LocalDateTime bookingTime) { 1 usage 1 Srikrish
    this.bookingTime = bookingTime;
}
```

#### Facade Pattern:

Our services (ReservationService, TrainService, UserService) act as facades. The Facade pattern provides a simplified interface to a complex system of classes, making it easier to interact with. Each service encapsulates a part of the functionality related to reservations, trains, and users respectively, providing a simplified interface for the controllers to interact with the underlying business logic.

```

@Override 5 usages Srikrishna S Nayak
@Transactional
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    User user = userRepository.findByUsername(username)
        .orElseThrow(() -> new UsernameNotFoundException("User Not Found with username: " + username));

    return UserPrincipal.build(user);
}

public User getUserByUsername(String username) { 2 usages Srikrishna S Nayak
    return userRepository.findByUsername(username).orElseThrow(() -> new UsernameNotFoundException("User Not Found with username: " + username));
}

public User saveUser(User user) { 1 usage Srikrishna S Nayak
    String encodedPassword = passwordEncoder.encode(user.getPassword());
    user.setPassword(encodedPassword);
    user.setEnabled(true); // Set the user as enabled by default
    return userRepository.save(user);
}

```

### Chain of Responsibility Pattern:

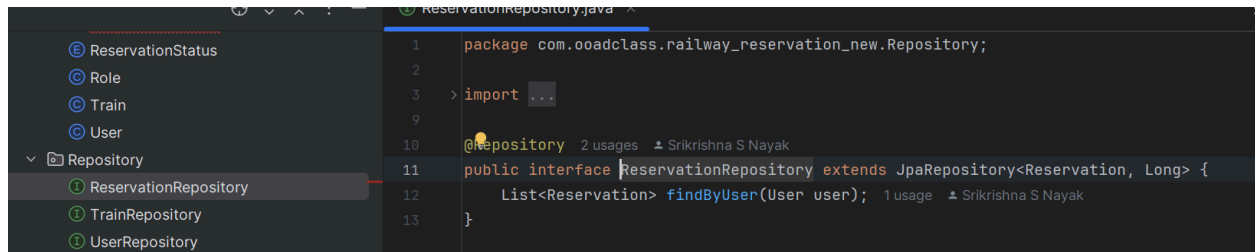
Spring Security is configured to enforce authentication for accessing all pages, which resembles the Chain of Responsibility pattern. In this pattern, a request is passed through a chain of handlers until it is handled by the appropriate handler. Similarly, in our application, each page request goes through Spring Security's filter chain, and if the user is not authenticated, they are redirected to the login page. Once authenticated, they gain access to the requested page.



### Repository Pattern:

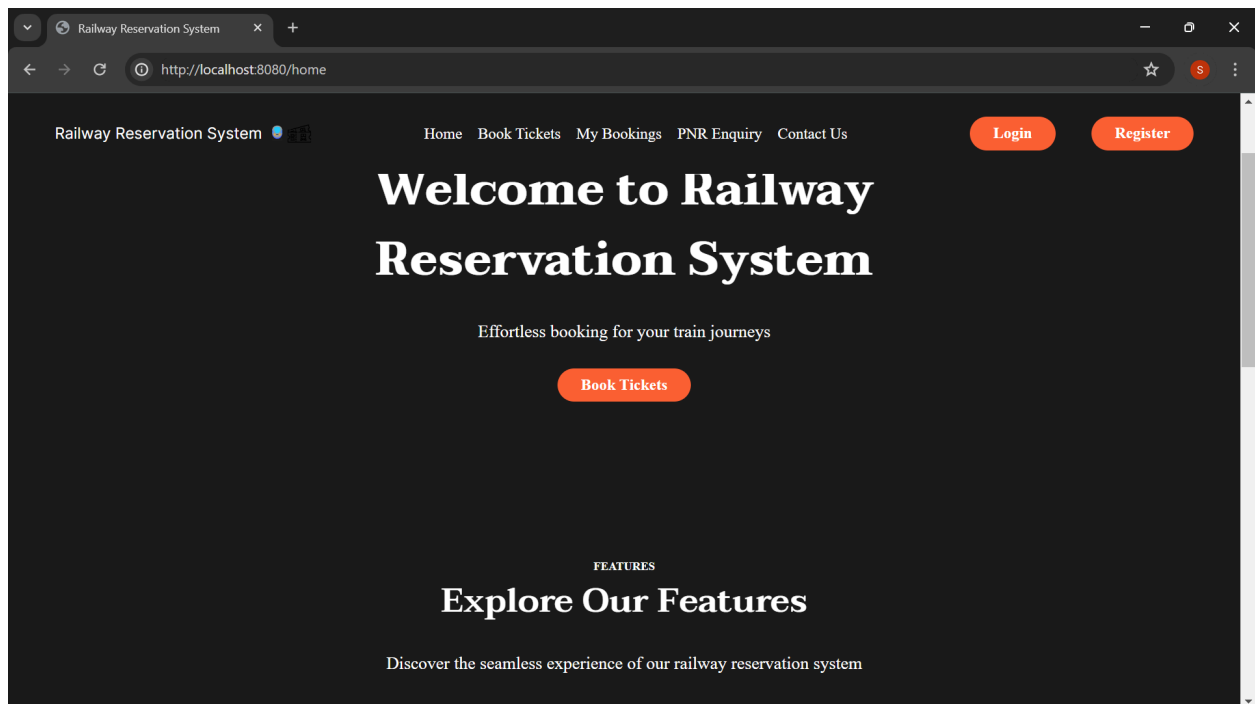
The Repository pattern is used with the repository classes (ReservationRepository, TrainRepository, UserRepository). This pattern provides a way to encapsulate data access

logic, allowing the rest of the application to interact with the database without exposing the underlying database implementation details. Each repository is responsible for CRUD (Create, Read, Update, Delete) operations related to its corresponding entity (reservation, train, user).



## User Interface Screenshots

Home Page, loaded when you access localhost8080



After clicking the register button, you are taken to the register page. The screenshot is an example of the register page not proceeding when you enter invalid information. The password entered by the user is hashed and stored in the database.



Railway Reservation System - R x +

http://localhost:8080/register

Railway Reservation System Home Book Tickets My Bookings PNR Enquiry Contact Us Login

# Register

Email: haha

! Please include an '@' in the email address. 'haha' is missing an '@'.

Last Name: Baunya

Username: Srijan

Password: .....

Confirm Password: .....

Register

When you go to the login page and you enter information not in the database, you remain at the login page.

Railway Reservation System - L x +

http://localhost:8080/login?error=true

Railway Reservation System Home Book Tickets My Bookings PNR Enquiry Contact Us Register

# Login

Username: Srija

Password: .....

Login

Invalid username or password.

Once you successfully log in, your username appears in the top right corner of the page.

Home Book Tickets My Bookings PNR Enquiry Contact Us

Srijan

Logout

These are the options when you visit the booking page after logging in.

Railway Reservation System

Home Book Tickets My Bookings PNR Enquiry Contact Us

## Book Tickets

Source: MANGALURU CNTL

Destination: KSR BENGALURU

Travel Date: 09-05-2024

Train: Select a train

Number of Seats: Select a train

Book

- UDYAN EXPRESS (11301)
- UDYAN EXPRESS (11302)
- YPR MAQ EXP (16565)
- MAQ YPR EXP (16566)
- PANCHAGANGA EXP (16595)
- PANCHAGANGA EXP (16596)

After you are done booking a ticket, you are taken to a page which displays the details of all your bookings.

Railway Reservation System

Home Book Tickets My Bookings PNR Enquiry Contact Us

Login Register

## My Bookings

Booking ID	Train Name	Source → Destination	Departure Time	Arrival Time	Number of Seats	Total Amount	Booking Time	Status
6	YPR MAQ EXP	KSR BENGALURU → MANGALURU CNTL	23:55	16:15	4	332.0	2024-05-03T21:32:43.540016	CONFIRMED
7	MAQ YPR EXP	MANGALURU CNTL → KSR BENGALURU	20:05	15:00	1	332.0	2024-05-03T21:33:07.268650	CONFIRMED
8	MAQ YPR EXP	MANGALURU CNTL → KSR BENGALURU	20:05	15:00	1	332.0	2024-05-03T21:33:21.771555	CONFIRMED