LA-2

Designing real world app and developing test suite

for

Railway Management system

by

Ankith S Menon 1NT21CS034

Nagaraj A 1NT21CS002

Akash P1NT21CS023

Aditya Ijari 1NT21CS019

28/07/2023

Table of Contents:

1 Introduction:

1.1 Overview of the Software Project and Its Purpose:

The Railway Management System is a human-friendly software solution designed to streamline and optimize various operations of the railway operations. Its main purpose is to improve efficiency, reliability, and safety of the railway management processes.

1.2 High-Level Description of the Problem and It Aims to Solve:

The current railway management systems often involve manual and paper based processes, leading to huge inefficiencies, delays, and other potential errors. The Railway Management System aims to address these challenges by automating tasks, centralizing data, and providing real-time information to stakeholders.

1.3 Scope and Objectives of the Software:

The scope of the Railway Management System includes train scheduling, ticketing, passenger and cargo management, maintenance and repair tracking, resource allocation, and reporting. The main purpose of this software are to optimize train schedules, enhance ticketing services, improve passenger experience, and ensure smooth railway operations.

2 Architectural Design:

2.1 Architectural Design Pattern:

The railway management system software can adopt a suitable architectural design pattern to structure and organize its components and interactions. One common architectural design pattern for such systems is the Model-View-Controller (MVC) pattern. The MVC pattern separates the application into three interconnected components:

- Model: Represents the data and business logic of the railway management system. It handles data manipulation and interacts with the database or other data sources.

- View: Handles the presentation layer, allowing users to interact with the system. It displays information and receives user input.

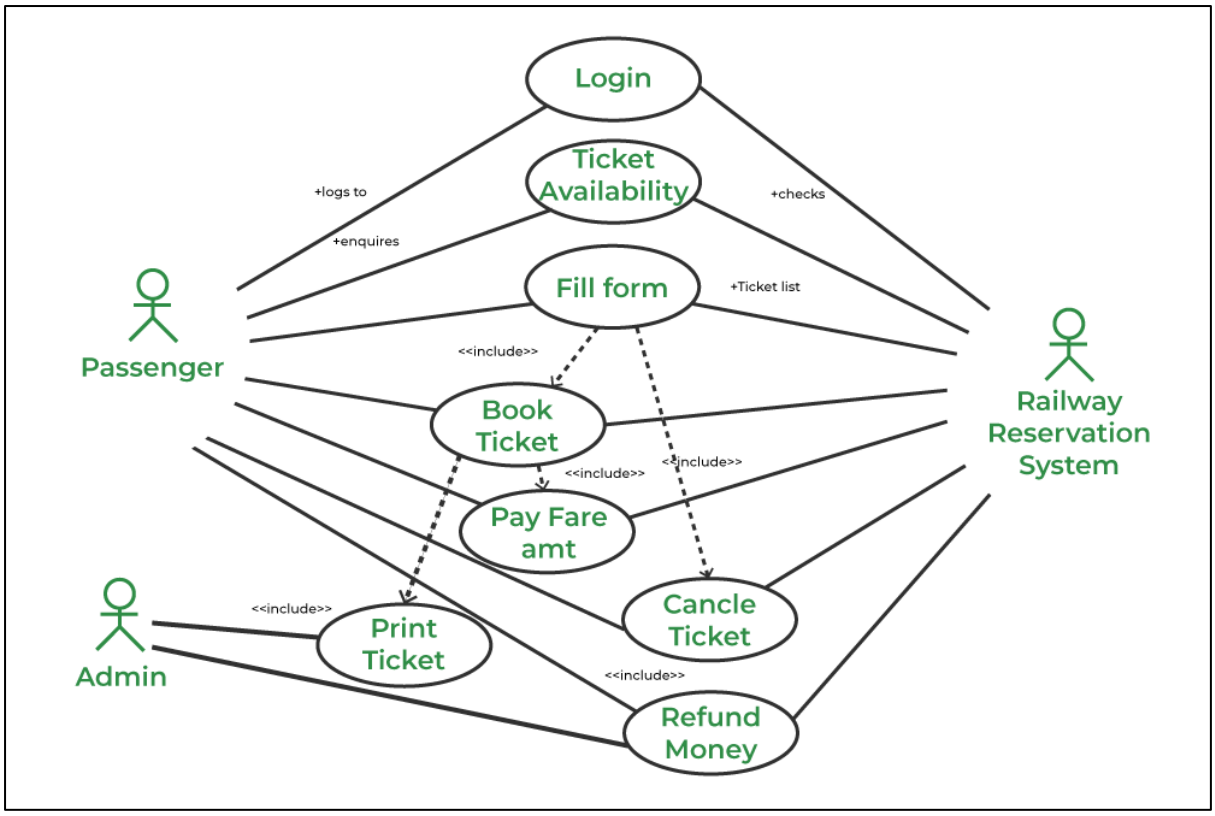
- Controller: Acts as an intermediary between the Model and View components. It receives user input from the View, processes it, and updates the Model and View accordingly.

2.2 Context Diagrams:

A context diagram provides the overview of the railway management software and its interactions with external entities and systems. It portrays the boundaries of the system and its high-level interactions with users, other systems, or databases. It shows what the system interacts as an abstraction without going into detail about the internal workings of the system.

2.3 Use Case Diagrams:

Use case diagrams



2.4 Data-Flow Diagrams:

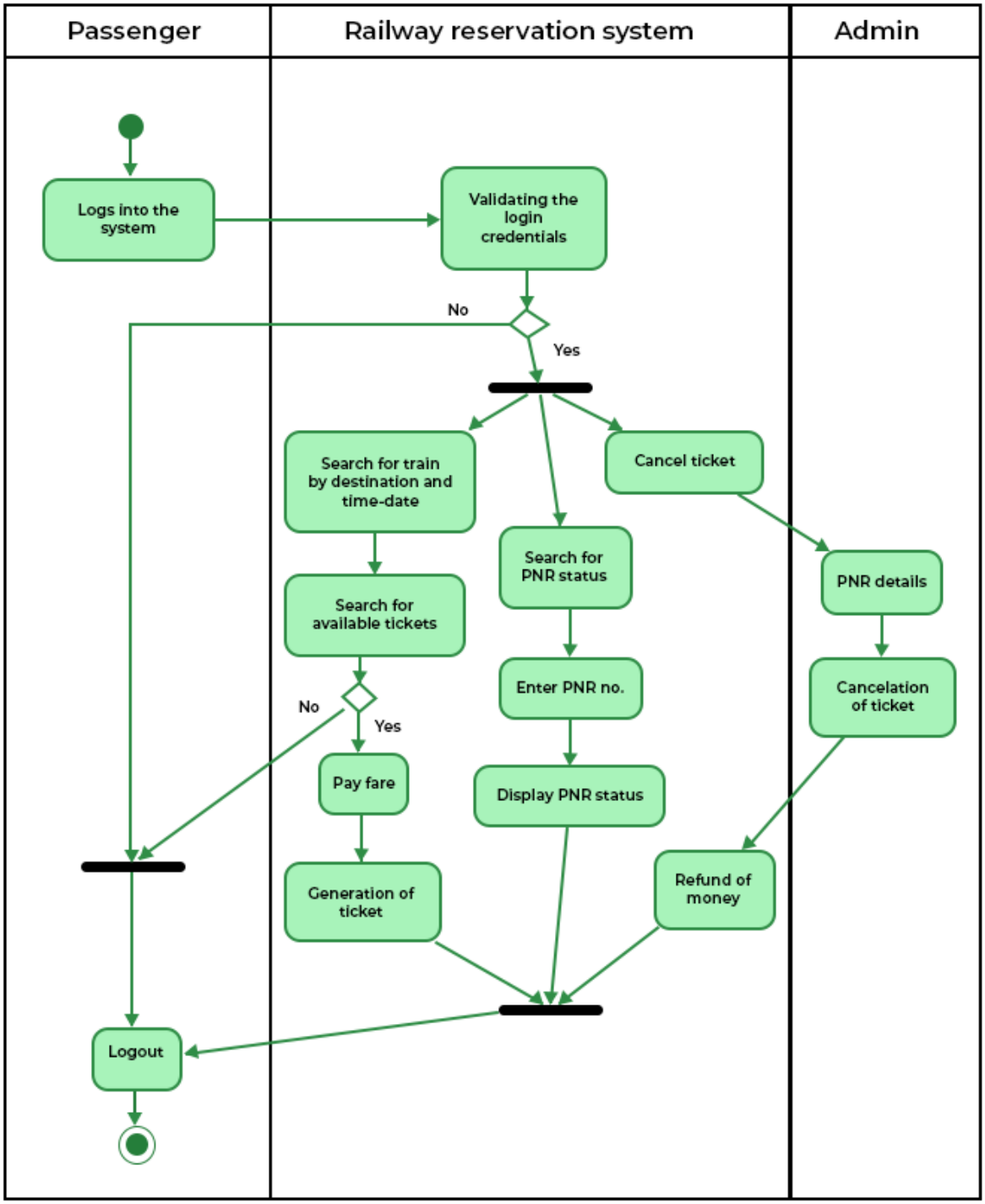
Data-flow diagrams (DFDs) represent the flow of data within the railway management system software. It visually depicts how data moves from one process to another and how it is stored or processed along the way. DFDs are useful for understanding the data flow and identifying potential bottlenecks or inefficiencies in the system.

2.5 Sequential Diagrams:

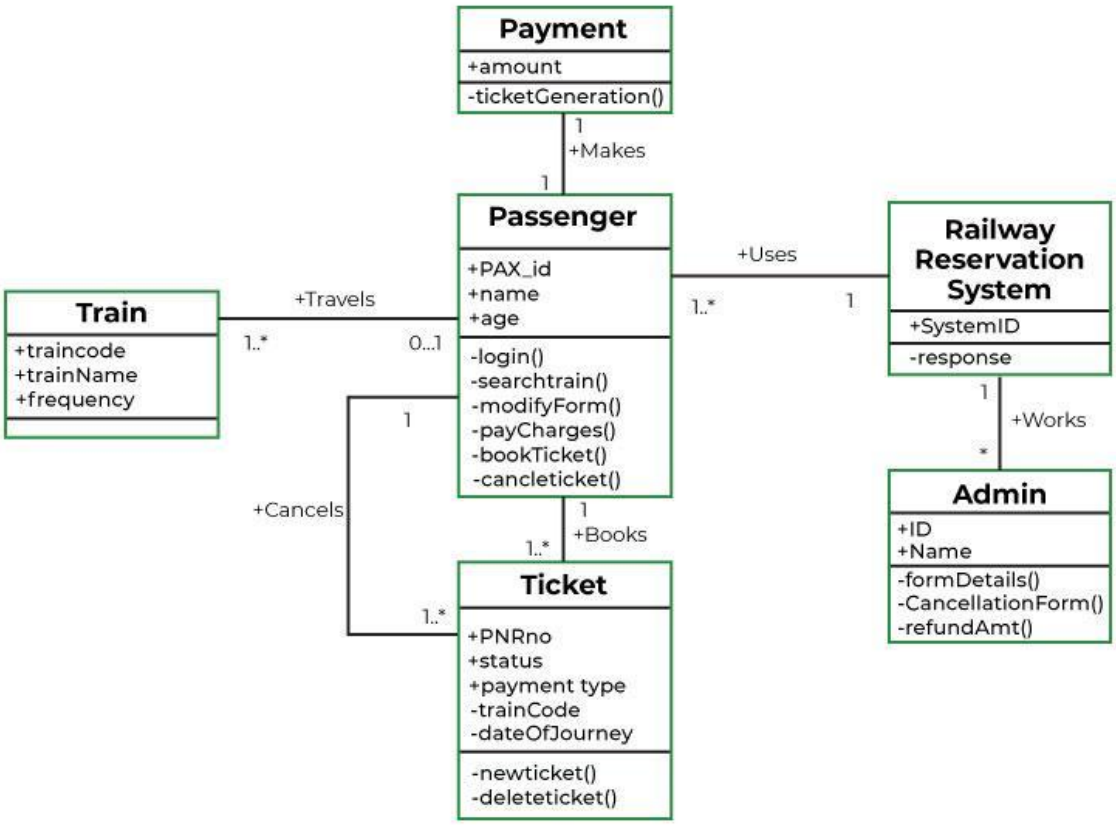
Sequential diagrams, such as Sequence Diagrams, depict the interactions between various components or objects of the railway management system over a specific time sequence. They showcase the chronological order of messages or method calls between objects, helping to understand the system's behavior during particular scenarios or use cases.

2.6 Activity Diagrams:

Activity diagrams describe the workflow or process flow within the railway management system. They show the sequence of activities, decisions, and parallel or concurrent flows that occur in the system. Activity diagrams are beneficial in understanding complex processes and ensuring efficient execution of tasks.

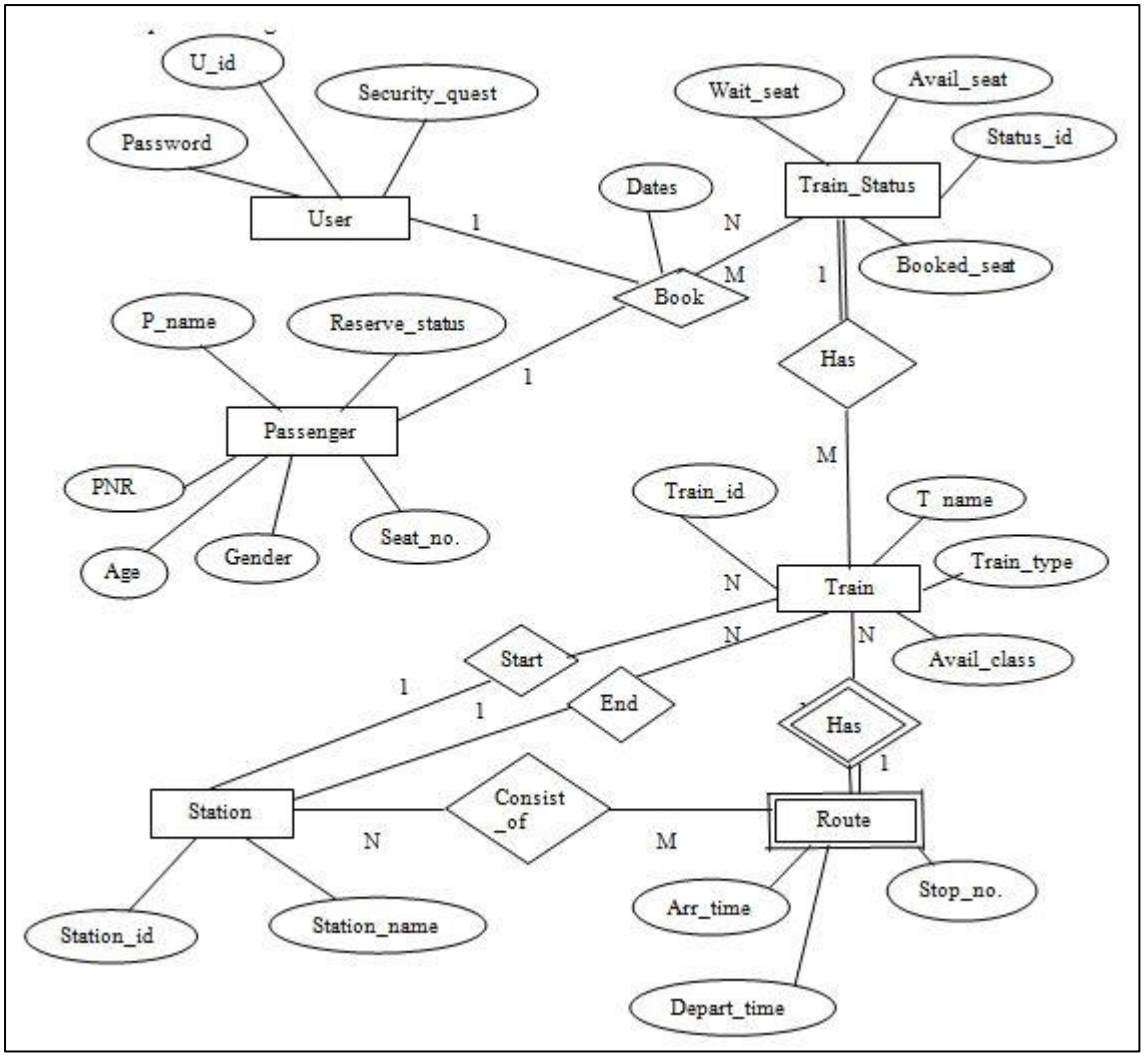


2.7 Class Diagrams:



2.8 ER Diagram:

An Entity-Relationship (ER) diagram is useful for representing the database schema of the railway management system. It showcases the entities (such as trains, stations, passengers) and their relationships with each other. ER diagrams are essential for database design and ensuring data integrity within the system.



These architectural design elements provide a comprehensive understanding of the railway management system software, ensuring efficient development, and effective communication among the development team.

3.Description of the User Interface:

The user interface (UI) of the railway management system software plays a crucial role in facilitating user interactions and providing a seamless experience for managing railway operations. The UI design should be intuitive, user-friendly, and efficient to cater to different types of users, including administrators, operators, and passengers. Below are some key aspects of the user interface:

3.1 Dashboard:

The main dashboard serves as the central hub where users can access various functionalities and information related to railway management. It should present a clear and necessary overview of critical system aspects, such as train schedules, station status, ongoing operations, and any alerts or notifications. The dashboard may use charts, graphs, and other visual toolsto present data in a visually appealing manner for quick comprehension.

3.2 Navigation and Menus:

The UI should have a well the organized navigation system with intuitive menus and buttons to allow users to move between different sections and features seamlessly. Logical grouping of functionalities and a consistent layout across pages will enhance user familiarity and reduce the learning curve.

3.3 Ticket Booking Interface:

For passengers, the ticket booking interface should be straightforward and easy to use. Users should be able to search for available trains, view seat availability, select travel dates, and make secure online payments. A step-by-step booking process with clear instructions will help users complete the booking without confusion.

3.4 Train and Schedule Management:

Administrators and operators should have access to an interface that allows them to manage trains, routes, and schedules efficiently. They should be able to add new train services, modify existing schedules, and handle cancellations or delays. A well-designed interface can streamline these operations and reduce the chances of errors.

3.5 Passenger Information:

The UI should display only necessary passenger information, such as reservation details, ticket status, and journey history, in a user-friendly manner. Passengers should be able to access their booking info and be easily be able to update their profile.

3.6 Real-time Updates and Alerts:

To ensure smooth railway operations, the UI should provide real-time updates on train statuses, delays, cancellations, and other critical information. Timely alerts through notifications or pop-ups will keep both users and administrators informed about any changes or issues.

3.7 Accessibility Considerations:

The UI design should take into account accessibility standards to accommodate users with disabilities. Providing options for font size adjustments, contrast settings, and keyboard navigation will make the software accessible to a broader range of users.

3.8 Error Handling and Help:

The UI should incorporate clear error messages and guidance to assist users in case they encounter any issues during their interactions with the system. Offering contextual help and FAQs can also reduce user frustration and enhance their overall experience.

In conclusion, the user interface of the railway management system should prioritize user experience and efficient task completion. A well-designed and user-friendly UI will enhance user satisfaction, increase system adoption, and contribute to the overall success of the software.

4 Security Handling:

4.1 Description of Security Measures Implemented:

User Authentication: Strong and secure authentication methods, including multi-factor authentication (MFA) and biometric authentication.

Role-Based Access Control (RBAC): Limiting user access based on assigned roles (e.g., administrator, operator, passenger).

Encryption: Sensitive data encrypted in transit and at rest using secure encryption algorithms (e.g., AES).

Secure Communication: HTTPS protocol for encrypted client-server communication.

Input Validation: Thorough validation of user inputs to prevent injection attacks (e.g., SQL injection, XSS).

Session Management: Secure session tracking and expiration after inactivity to prevent unauthorized access.

Audit Trail: Logging and monitoring of user activities for detecting suspicious behavior.

Regular Updates and Patches: Frequent updates and security patches to address vulnerabilities.

Secure Coding Practices: Following secure coding guidelines to mitigate security risks.

Backup and Disaster Recovery: Regular data backups and a robust disaster recovery plan.

5 Error Handling:

5.1 Description of How Errors and Exceptions are Handled:

In the railway management system software, a comprehensive error handling mechanism is implemented to detect, manage, and handle errors and exceptions that may occur during its operation. The goal is to ensure smooth system functioning, prevent crashes, and provide meaningful feedback to users when errors occur. Here's how errors and exceptions are handled in the system:

5.1.1 Exception Handling:

The software is designed to catch and handle exceptions gracefully. When unexpected situations or errors occur, the system identifies the type of exception and executes appropriate error-handling routines. This prevents the application from terminating abruptly and allows for controlled recovery or graceful degradation.

5.1.2 Error Messages:

When errors are encountered, user-friendly error messages are displayed to inform users about the issue in a clear and understandable manner. The error messages provide relevant details about the nature of the problem and, whenever possible, suggestions on how to resolve it or seek further assistance.

5.1.3 Logging:

The system incorporates logging functionality to record information about errors, exceptions, and system events. Detailed logs are generated to help developers and administrators diagnose issues and analyze the root cause of errors. Logging also aids in monitoring system health and performance.

5.1.4 Validation and Error Prevention:

To minimize the occurrence of errors, the software enforces robust input validation mechanisms. User inputs are validated at various stages to ensure they conform to the expected format and range. By preventing invalid data from entering the system, the risk of errors is significantly reduced.

5.1.5 Graceful Degradation:

In situations where critical components or services are unavailable, the system gracefully degrades its functionality rather than crashing or becoming entirely unusable. This allows users to access essential features of the application, even if some non-essential features are temporarily unavailable.

5.1.6 Error Recovery:

The system implements error recovery strategies to handle non-fatal errors and resume normal operation when possible. This may involve attempting automatic recovery procedures, providing alternative options, or notifying users about potential workarounds.

5.1.7 Error Reporting and Monitoring:

To proactively address issues, error reporting and monitoring tools are integrated into the system. These tools help detect recurring errors, patterns, or performance bottlenecks, enabling the development team to prioritize and resolve critical issues promptly.

5.1.8 User Support and Assistance:

The system offers user support and assistance through help documentation, FAQs, and customer support channels. Users facing errors can refer to these resources for guidance and solutions, reducing frustration and improving the overall user experience.

By incorporating a robust error handling approach, the railway management system software can provide a reliable and user-friendly experience, even when unforeseen errors or exceptions occur. The goal is to minimize disruptions, enhance user satisfaction, and ensure the system operates efficiently under various conditions.

6 Testing Strategy:

6.1 Description of the Testing Approach and Methodologies Used for Unit Testing:

The testing strategy for the railway management system software relies primarily on Unit Testing using JUnit, a widely used unit testing framework for Java. The focus of unit testing is to validate the individual units or components of the code in isolation, ensuring that each unit performs as expected and meets its design specifications. The following approach is followed for unit testing:

6.1.1 Test-Driven Development (TDD):

Test-Driven Development (TDD) is adopted as a development practice. Before writing the implementation code, developers create unit test cases based on the expected behavior and requirements. These test cases act as a specification for the code's functionality.

6.1.2 Test Coverage:

Aim for high test coverage by creating comprehensive test cases that cover different code paths, edge cases, and potential exceptions. The goal is to validate as much of the codebase as possible through unit tests.

6.1.3 Isolation of Units:

Each unit is tested in isolation from other units and external dependencies. Mock objects and stubs are used to simulate the behavior of external components, ensuring that the tests are independent and repeatable.

6.1.4 Continuous Integration (CI):

Unit tests are integrated into the Continuous Integration (CI) process. Whenever code changes are committed to the version control system, the CI system automatically triggers the execution of unit tests. This ensures that any code changes do not break existing functionality.

6.1.5 Test Refactoring:

As the code evolves, unit tests are also refactored to accommodate changes in requirements and improvements in code design. Keeping the unit tests up-to-date with the codebase is essential for maintaining their effectiveness.

6.1.6 Test Automation:

JUnit provides an automated way to execute unit tests, making it efficient and repeatable. By automating unit tests, developers can run them frequently and easily identify regressions or issues early in the development cycle.

6.1.7 Continuous Improvement:

Feedback from unit testing helps identify areas for improvement and helps in enhancing the overall software quality. Developers learn from failed tests, identify patterns of errors, and refine their coding practices accordingly.

By employing unit testing with JUnit, the railway management system software can achieve greater code reliability, maintainability, and faster development cycles. Unit tests act as safety nets, catching potential defects early and providing the necessary confidence to developers when making changes to the code.

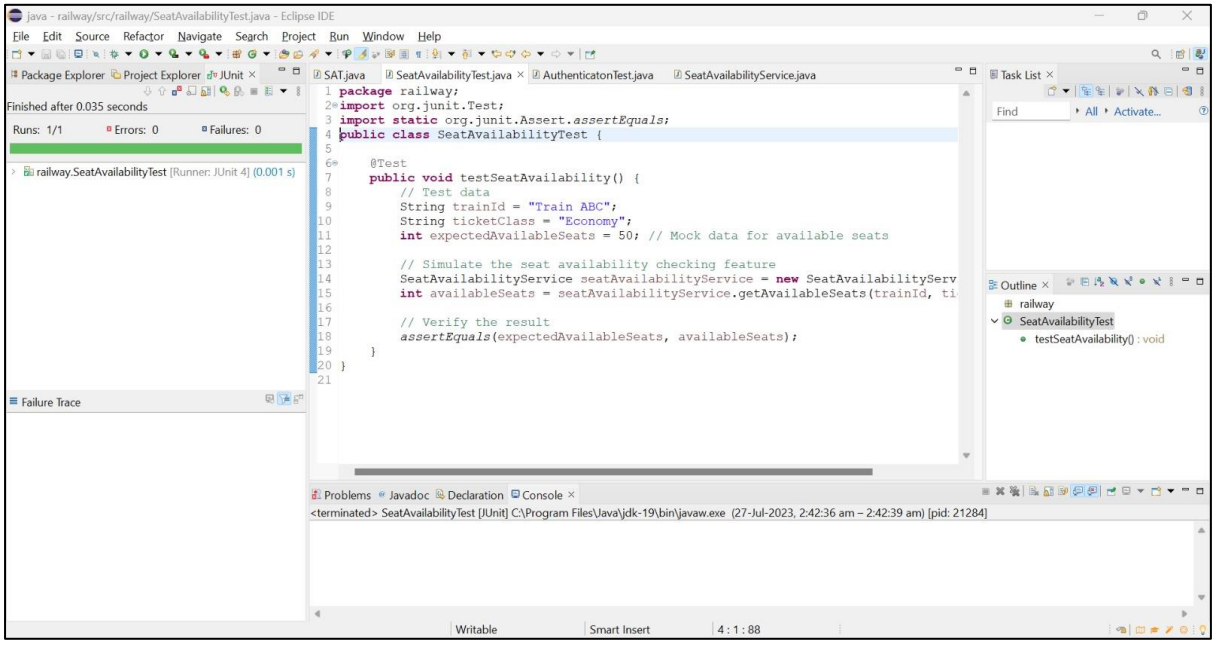
Let’s now look at the test cases we used for

```java
package railway;

public class SeatAvailabilityService {
    public int getAvailableSeats(String trainId, String ticketClass) {
        // In a real application, this method would fetch data from a database or external service.
        // For the test, we are providing mock data.
        if (trainId.equals("Train ABC") && ticketClass.equals("Economy")) {
            return 50;
        } else {
            return 0;
        }
    }
}
```

a.1.Code for seat availability

```java
package railway;
import org.junit.Test;
import static org.junit.Assert.assertEquals;
public class SeatAvailabilityTest {

    @Test
    public void testSeatAvailability() {
        // Test data
        String trainId = "Train ABC";
        String ticketClass = "Economy";
        int expectedAvailableSeats = 50; // Mock data for available seats

        // Simulate the seat availability checking feature
        SeatAvailabilityService seatAvailabilityService = new SeatAvailabilityService();
        int availableSeats = seatAvailabilityService.getAvailableSeats(trainId, ticketClass);

        // Verify the result
        assertEquals(expectedAvailableSeats, availableSeats);
    }
}
```

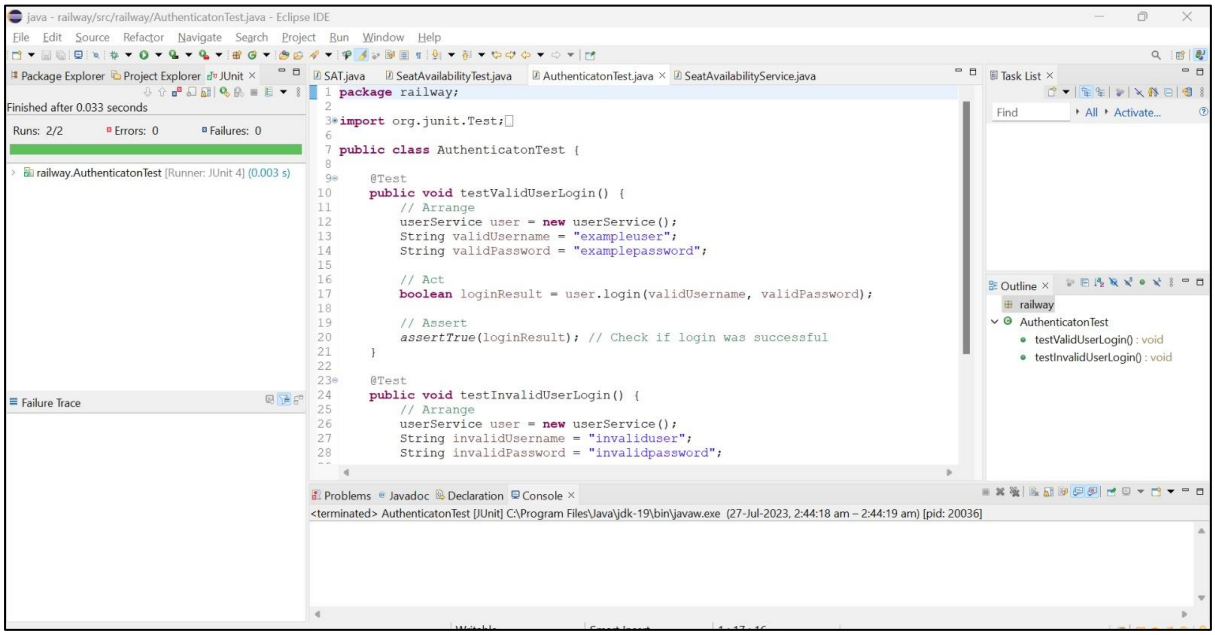
a.2.Code for running testcases on seat availability



a.3 code displaying successful test run of the seat availabiltiy service.

```java
package railway;

import org.junit.Test;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;

public class AuthenticatonTest {

    @Test
    public void testValidUserLogin() {
        // Arrange
        userService user = new userService();
        String validUsername = "exampleuser";
        String validPassword = "examplepassword";

        // Act
        boolean loginResult = user.login(validUsername, validPassword);

        // Assert
        assertTrue(loginResult); // Check if login was successful
    }

    @Test
    public void testInvalidUserLogin() {
        // Arrange
        userService user = new userService();
        String invalidUsername = "invaliduser";
        String invalidPassword = "invalidpassword";

        // Act
        boolean loginResult = user.login(invalidUsername, invalidPassword);

        // Assert
        assertFalse(loginResult); // Check if login was unsuccessful
    }
}
```

b.1. Code testing the user login fucntion.



b.2 The successful test run of the user login function

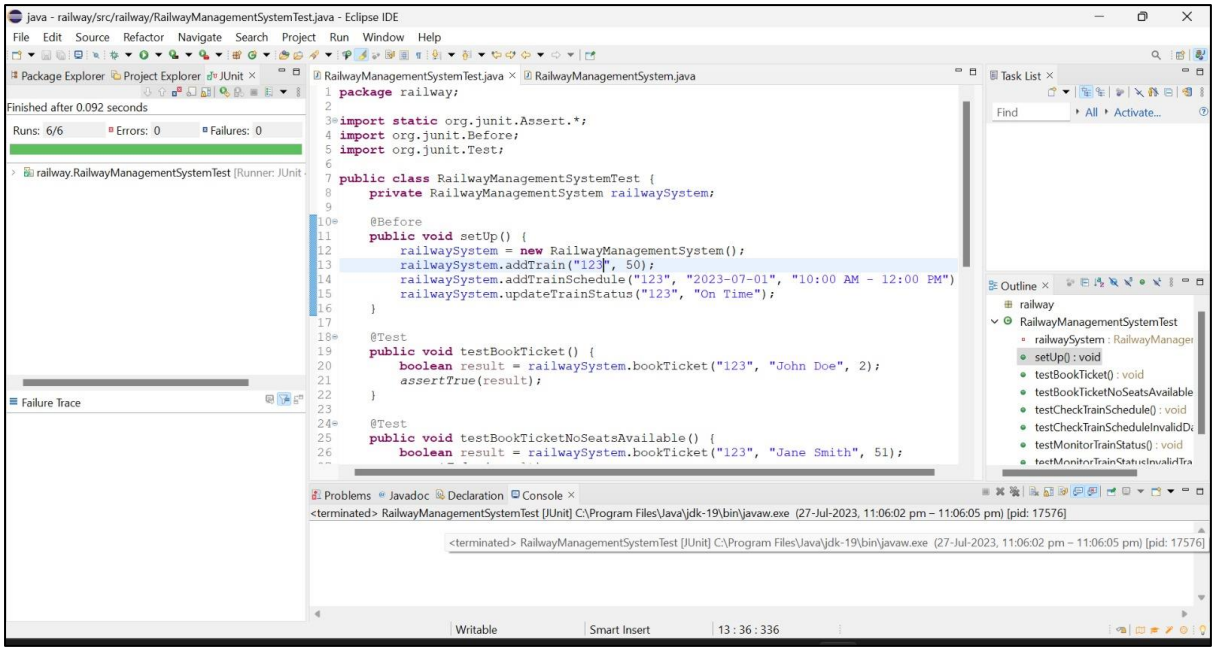
```java
package railway;
import java.util.HashMap;
import java.util.Map;

public class RailwayManagementSystem {
    private Map<String, Integer> trainSeats;
    private Map<String, String> trainSchedules;
    private Map<String, String> trainStatus;

    public RailwayManagementSystem() {
        trainSeats = new HashMap<>();
        trainSchedules = new HashMap<>();
        trainStatus = new HashMap<>();
    }

    public boolean bookTicket(String trainId, String passengerName, int numberOfSeats) {
        if (!trainSeats.containsKey(trainId)) {
            return false;
        }

        int availableSeats = trainSeats.get(trainId);
        if (numberOfSeats > availableSeats) {
            return false;
        }

        trainSeats.put(trainId, availableSeats - numberOfSeats);
        return true;
    }
```

```java
        if (numberOfSeats > availableSeats) {
            return false;
        }

        trainSeats.put(trainId, availableSeats - numberOfSeats);
        return true;
    }

    public String checkTrainSchedule(String trainId, String date) {
        String schedule = trainSchedules.get(trainId + "_" + date);
        return schedule;
    }

    public String monitorTrainStatus(String trainId) {
        return trainStatus.get(trainId);
    }

    public void addTrain(String trainId, int totalSeats) {
        trainSeats.put(trainId, totalSeats);
    }

    public void addTrainSchedule(String trainId, String date, String schedule) {
        trainSchedules.put(trainId + "_" + date, schedule);
    }

    public void updateTrainStatus(String trainId, String status) {
        trainStatus.put(trainId, status);
    }
}
```

c.1. Has the implementation of the functions for checking train schedule, to monitor train status, to add trains, to add train schedule and to update the train status.

```java
package railway;

import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

public class RailwayManagementSystemTest {
    private RailwayManagementSystem railwaySystem;

    @Before
    public void setUp() {
        railwaySystem = new RailwayManagementSystem();
        railwaySystem.addTrain("123", 50);
        railwaySystem.addTrainSchedule("123", "2023-07-01", "10:00 AM - 12:00 PM");
        railwaySystem.updateTrainStatus("123", "On Time");
    }

    @Test
    public void testBookTicket() {
        boolean result = railwaySystem.bookTicket("123", "John Doe", 2);
        assertTrue(result);
    }

    @Test
    public void testBookTicketNoSeatsAvailable() {
        boolean result = railwaySystem.bookTicket("123", "Jane Smith", 51);
        assertFalse(result);
    }

    @Test
    public void testCheckTrainSchedule() {
        String schedule = railwaySystem.checkTrainSchedule("123", "2023-07-01");
        assertNotNull(schedule);
        assertEquals("10:00 AM - 12:00 PM", schedule);
    }

    @Test
    public void testCheckTrainScheduleInvalidDate() {
        String schedule = railwaySystem.checkTrainSchedule("123", "2023-07-10");
        assertNull(schedule);
    }

    @Test
    public void testMonitorTrainStatus() {
        String status = railwaySystem.monitorTrainStatus("123");
        assertNotNull(status);
        assertEquals("On Time", status);
    }

    @Test
    public void testMonitorTrainStatusInvalidTrain() {
        String status = railwaySystem.monitorTrainStatus("999");
        assertNull(status);
    }
}
```

c.2. Has the test run codes of the functions for checking train schedule, to monitor train status, to add trains, to add train schedule and to update the train status.



c.3. displays the successful execution of the codes of the functions for checking train schedule, to monitor train status, to add trains, to add train schedule and to update the train status.

7 Deployment:

7.1 Explanation of how the software will be deployed and installed.

The deployment of the railway management system is a critical phase to ensure its successful implementation and efficient functioning. The following steps outline the process of deploying and installing the software:

7.1.1 System Requirements Analysis:

Before we go ahead with the deployment of the railway management system, we will carefully analyze the railway's current setup. This includes looking at the existing infrastructure, like computers and networks, as well as the software they use. This analysis is crucial because it helps us figure out if the new system will work well with what they already have.

If there are any differences or gaps between what they need and what they currently have, we will identify them during this process. For instance, we might find that some components need to be upgraded, or they might need additional hardware or software to make everything compatible and work smoothly.

By doing this thorough analysis, we can make sure that when we deploy the railway management system, it fits right in with their existing environment, and everything functions as expected. It helps us be prepared and ensures a successful deployment that meets their specific needs.

7.1.2 Pre-Deployment Testing:

Before we officially put the software into use, we will test it thoroughly in a safe and controlled environment. During this testing phase, we will carefully check for any problems or bugs in the software and fix them. This is done to make sure that when the software is deployed for real use, it is stable and works smoothly without any issues. The testing process helps ensure that the software is ready and reliable for live operation.

7.1.3 Data Migration:

If the railway already has data from a previous system, we will create a clear and detailed plan for moving that data to the new railway management system. This process is known as data migration.

During data migration, we will be very cautious and meticulous to ensure that all the information from the old system is transferred accurately and without any errors. This means we will pay close attention to maintain data integrity, meaning that the data remains complete and consistent throughout the migration process.

By following this well-defined data migration plan, we can make sure that the new railway management system starts with the correct and up-to-date information, avoiding any disruptions or loss of data during the transition. This helps in keeping the operations smooth and uninterrupted.

7.1.4 Deployment Strategy:

The way we deploy the railway management system will be thoughtfully designed to minimize any disruptions to the normal railway operations. One approach we might use is a phased deployment, where we implement specific modules or functions of the system in stages. This means we won't change everything all at once.

Instead, we will gradually introduce the new system, allowing people to adapt to it step by step. By doing it this way, we can provide training and support for each phase, making sure everyone is comfortable and well-prepared before moving on to the next stage.

This approach helps in reducing the risk of major disruptions and allows for a smoother transition to the new system. It ensures that railway personnel can continue their work without significant interruptions and gives them time to get familiar with the new system gradually.

7.1.5 User Training:

Comprehensive training sessions will be conducted for railway staff and personnel who will be using the system. This will ensure that they understand how to operate the software efficiently and maximize its benefits.

7.1.6 Pilot Deployment:

A pilot deployment may be conducted in a controlled railway segment or a limited number of stations to validate the system's performance and gather feedback from users.

7.1.7 Full Deployment:

Once the pilot phase is successful and any necessary adjustments are made, the software will be fully deployed across the entire railway network. The implementation team will closely monitor the system during this phase to address any unforeseen issues promptly.

7.1.8 Post-Deployment Review:

After the full deployment, a thorough review will be conducted to assess the overall success of the deployment process and identify areas for improvement.

8 Maintenance and Support:

8.1 Guidelines for ongoing maintenance and support of the software.

A robust maintenance and support strategy is vital for the continuous functionality and reliability of the railway management system. The following guidelines outline the approach to ongoing maintenance and support:

8.1.1 Regular Software Updates:

The software will undergo regular updates to address any bugs, security vulnerabilities, and to introduce new features or enhancements. Before implementing these updates to the live system, they will be thoroughly tested in a staging environment. This ensures that any potential issues are identified and resolved before the changes are made available to users in the production environment.

8.1.2 24/7 Technical Support:

A specialized technical support team will be accessible round-the-clock to assist railway staff in the event of any problems, outages, or emergencies. Users can seek help by using a support ticketing system, sending an email, or calling a dedicated helpline. The support team will be ready to provide prompt assistance and resolve any issues that may arise during system operation.

8.1.3 Performance Monitoring:

The system's performance will be continuously monitored to proactively identify any potential bottlenecks, resource limitations, or unusual activities. This ongoing monitoring is aimed at ensuring the system operates at its best, minimizing any possible downtime and ensuring optimal performance. By promptly detecting and addressing any issues, we can maintain the system's reliability and efficiency.

8.1.4 Data Backup and Disaster Recovery:

Regular backups of critical data to protect against data loss. In addition, we will have a comprehensive disaster recovery plan in place to ensure a swift recovery in case of any catastrophic events. By taking these measures, we aim to safeguard our data assets and minimize any potential disruptions to our operations in the face of unexpected incidents or disasters.

8.1.5 User Training and Documentation Updates:

As the railway management system evolves with updates and new features, we will conduct regular user training sessions to keep the railway staff well-informed about the system's capabilities. These training sessions will ensure that users are up-to-date with the latest functionalities and can make the most of the system's potential. Additionally, the system's documentation will be updated to accurately reflect these changes, providing comprehensive and current information for users to refer to when needed. This approach will enable a smooth transition with each update and help users stay proficient in utilizing the system effectively.

8.1.6 Feedback and Continuous Improvement:

We will set up a feedback mechanism to collect input from users concerning the system's performance and usability. This feedback will play a crucial role in driving continuous improvements and enhancements to the system. By actively listening to user experiences and suggestions, we can identify areas that need attention and make the necessary adjustments to enhance the overall user experience and optimize the system's functionality. Your feedback will be invaluable in shaping the system to better meet your needs and expectations.

8.1.7 Compliance and Security Audits:

The software will be designed to adhere to relevant industry standards, and we will conduct periodic security audits to ensure that the system consistently meets these standards and remains secure. These security audits will help us identify and address any potential vulnerabilities, ensuring that the system's data and operations are safeguarded against potential threats. By adhering to industry best practices and conducting regular security assessments, we can maintain the system's compliance and uphold a high level of security to protect both user data and system integrity.

By adhering to these deployment, maintenance, and support guidelines, the railway management system will be well-prepared to meet the operational requirements of the railway and deliver efficient and dependable services. The comprehensive training, continuous monitoring, and regular updates, along with the feedback mechanism and user training, will ensure that the system remains optimized, secure, and up-to-date with the latest features and improvements. With a dedicated technical support team available 24/7 and a robust disaster recovery plan, any issues that arise can be promptly addressed, minimizing downtime and ensuring smooth operations. By following these best practices, the railway management system will be a reliable and effective tool to support and enhance railway operations, benefitting both the staff and passengers alike.

9.Conclusion:

The successful deployment and continuous maintenance of the railway management system are vital to ensuring the efficient and reliable operation of the entire railway network. To achieve this, we will follow a meticulous approach, starting with careful planning, rigorous testing, and thorough training. This will guarantee a seamless transition to the new system, minimizing any disruptions to ongoing railway operations.

The implementation of the software will be carried out in a well-structured, step-by-step manner, with pilot deployments serving as real-world tests. These pilot deployments will provide opportunities for validation in real operational scenarios and enable us to gather valuable feedback from users. This iterative approach allows us to make necessary adjustments and improvements before rolling out the system fully.

By adhering to these practices, we aim to deliver a highly effective and reliable railway management system, providing enhanced efficiency and service quality throughout the entire railway network.

To ensure the railway management system's long-term success, we have a strong maintenance and support strategy. Regular software updates, available technical support 24/7, performance monitoring, and data backups with disaster recovery procedures are all in place to ensure the system runs smoothly and protect against data loss.

To empower railway staff, we provide user training and keep the documentation updated, enabling them to fully utilize the system's capabilities. Furthermore, a feedback mechanism is established, encouraging users to provide input for continuous improvement and enhancement of the system. With these measures, we aim to deliver an efficient and reliable railway management system for years to come.

By following industry standards and conducting security audits, the railway management system will stay compliant and secure against potential threats. This will lead to better efficiency, increased safety, and smoother operations for the railway.

In conclusion, the deployment and maintenance of the railway management system are carefully planned and based on best practices, guaranteeing its successful integration into the railway network. With ongoing support and continuous improvement, the system will be a valuable asset in managing railway operations, contributing to the overall growth and success of the railway industry.