CROSS-PLATFORM ASSIGNMENT

SUBMITTED BY -

Aryan Saxena – 2021HS70016
Deev Pal – 2021HS70025
Shravya Shreya – 2021HS70011
Yash Kanoria – 2021HS70002

SAAS-Based Transport Management Application

This repository contains the SAAS-Based Transport Management Application, a cloud-based solution designed to streamline operations for transportation companies. The following sections provide an overview of the application's functionalities, technical architecture, development process, and testing strategies.

1. Introduction

Transportation companies often face challenges in managing Bilty, Hire Challen, and Invoices. This application aims to automate these tasks and enhance operational efficiency.

2. System Functionalities

Bilty Management

- **Create and manage Bilty documents**: Includes shipment details like origin, destination, goods description, consignee, and consignor information.
- **E-way bill integration**: Ensures seamless compliance with regulatory requirements.

Hire Challen Management

- **Issue Hire Challens**: Specifies vehicle details, duration, and cost for vehicle hiring.

Invoice Management

- **Automatic invoice generation**: Based on Bilty and Hire Challen data.
- **Tax calculations and customization**: Allows for different pricing structures.

Additional Features (Optional)

- **Driver Management**: Tracks driver details, licenses, and job assignments.
- **Fleet Management**: Monitors vehicle location, maintenance schedules, and fuel consumption.
- **Customer Management**: Manages customer information, tracks shipment history, and offers self-service options.

3. Technical Architecture

Frontend Development

- **Framework**: Built using Flutter for a single codebase across Android and iOS platforms.
- **Benefits**: Reduced development time and native-like performance.

Backend Development

- **Framework**: Developed using Spring Boot for its robust architecture.
- **Features**: Utilizes MVC pattern, dependency injection, and RESTful API development.

Server Infrastructure

- **Deployment**: Hosted on AWS EC2 instances for scalability and elasticity.
- **Load Balancer**: Distributes traffic efficiently across multiple EC2 instances.

Database

- **Technology**: Uses MySQL or PostgreSQL stored on AWS RDS.
- **Benefits**: Effective structured data management.

4. Entity-Relationship Diagram (ERD)

An ERD visualizes relationships between entities like Customer, Driver, Vehicle, Bilty, Hire Challen, and Invoice. Relationships include:

- One-to-Many between Customer and Bilty/Hire Challen.
- Many-to-One between Bilty and Customer (Consignor and Consignee).
- One-to-One or Optional between Bilty and Hire Challen.
- One-to-One between Invoice and Bilty/Hire Challen.

5. Class Diagram

The class diagram showcases backend logic classes with attributes and methods, adhering to object-oriented design principles.

Customer

- **Attributes**: customerID, name, address, phone, email.
- **Methods**: getCustomerDetails(), updateCustomer().

Driver (Optional)

- **Attributes**: driverID, name, licenseNumber, contactInfo.
- **Methods**: getDriverDetails(), assignDriverToJob().

Vehicle (Optional)

- **Attributes**: vehicleID, registrationNumber, model, capacity.
- **Methods**: getVehicleDetails(), trackVehicleLocation().

Bilty

- **Attributes**: biltyID, customerID (FK), consigneeID (FK), origin, destination, goodsDescription, ewayBillNumber (Optional).
- **Methods**: createBilty(), generateEwayBill() (Optional).

Hire Challen

- **Attributes**: challenID, customerID (FK), vehicleID (FK - Optional), startDate, endDate, cost.
- **Methods**: createHireChallen(), assignVehicle() (Optional).

Invoice

- **Attributes**: invoiceID, biltyID (FK), challenID (FK - Optional), amount, taxDetails.
- **Methods**: generateInvoice().

6. Testing Strategies

Unit Testing

- **Importance**: Ensures code quality by testing individual classes and functions.
- **Framework**: JUnit for Java.

Integration Testing

- **Description**: Tests interaction and functionality of different modules.

System Testing

- **Outline**: Ensures the application meets all functional and non-functional requirements.

User Acceptance Testing (UAT)

- **Importance**: Gathers feedback from potential users to refine the application before deployment.

7. Software Development Life Cycle (SDLC) Model

Model

- **Chosen Model**: Agile.

Phases

- **Planning, Requirement Analysis, Design, Development, Testing, Deployment, Maintenance**: Each phase will be applied to the development of the SAAS-Based Application.

8. Deployment and Maintenance

Deployment Process

- **Infrastructure**: AWS with configuration management tools like Chef or Ansible.

Maintenance Strategy

- **Bugs, Features, Security**: Regular updates to address bugs, implement new features, and ensure security.

9. Security Considerations

Measures

- **Authentication and Access Control**: Implement user authentication and access control mechanisms.
- **Data Encryption**: Encrypt sensitive information.
- **Secure Coding Practices**: Regular vulnerability scanning and compliance with data privacy regulations.