Petrol Pump Management System Project

REPORT

Team Members

Gagan Gireesh Krishna (AM.AIU4AID23008)
S. Sreyas (AM.AI.U4AID23019)

1. INTRODUCTION

1.1 Project Overview

The Petrol Pump Management System is designed to streamline the operations of petrol pumps, enhancing efficiency, security, and compliance with regulatory requirements. The system features user authentication, inventory management, reporting, and notifications, tailored specifically for the Indian market. The aim is to reduce manual errors, improve operational efficiency, and provide real time insights into the business operations.

1.2 Objectives

- To develop a robust and user friendly system for managing petrol pump operations.
- To ensure secure and efficient user authentication using Aadhaar Based verification.
- To provide real time inventory management and alerts for low stock levels.
- To generate comprehensive reports for sales and inventory to assist in decision making.
- To integrate a notification system for timely updates and promotions.

1.3 Scope

The system covers the following aspects of petrol pump management:

- User registration and authentication.
- Management of fuel purchases and sales transactions.
- Inventory tracking and management.
- Generation of sales and inventory reports.
- Notification and alert management.

2. Requirements

2.1 Customer Requirements

The primary requirements gathered from potential users and stakeholders include:

- **User Registration and Authentication**: Users Should be able to register and log in using a secure and straightforward process, ideally integrating with existing Aadhaar infrastructure.

- **RealTime Inventory Management** The system should provide real time updates on fuel inventory levels and generate alerts when stocks fall below a predefined threshold.

- **Comprehensive Reporting:** The ability to generate detailed reports on sales and inventory to facilitate business analysis and decision making.

- **Notification System:** An integrated system to send notifications to customers about updates, promotions, and important alerts.

2.2 System Requirements

From a technical standpoint, the system should meet the following requirements:

- **AadhaarBased User Authentication:** To leverage existing national infrastructure for secure user authentication.

- **Modular Design**: The system should be designed in a modular fashion to allow for easy maintenance, scalability, and future enhancements.

- **Integration with Accounting Software**: To streamline financial management and ensure accurate accounting and reporting.

2.3 NonFunctional Requirements

The system must also meet several nonfunctional requirements to ensure its reliability and usability:

- **Performance:** The system should be able to handle concurrent users and transactions efficiently.

- **Security:** Data security and user privacy must be a top priority, especially given the use of Aadhaar data.

- **Usability:** The interface should be intuitive and user friendly to facilitate easy adoption by staff and customers.

- **Scalability:** The system should be able to scale to accommodate additional features and increased user load.

3. Core Use Cases

3.1 Use Case 1: User Registration

Actors: Users
Description: Users register by providing their name and phone number. They receive a unique customer ID after registration. The system also integrates password verification for added security.
Steps:

1. User provides a name and phone number and a password.
2. Userenters password to complete registration.
3. System generates and assigns a unique UserID.

3.2 Use Case 2: Fuel Purchase

Actors: Customer, SalesManager
Description: Customers log in, select fuel type and quantity, and complete the purchase. SalesManager records the transaction in the system.
Steps:

1. Customer logs in using their credentials.(mobile no)

2. Customer selects the fuel type and enters the desired quantity.
3. System calculates the total cost and displays it to the customer.
4. Customer confirms the purchase.
5. SalesManager processes the transaction and updates inventory.

3.3 Use Case 3: Inventory Management

Actors: InventoryManager
Description: Manages fuel stock, updates inventory levels, and generates alerts when stock is low.
Steps:

1. InventoryManager checks current stock levels.
2. System generates alerts if stock is below the threshold.
3. InventoryManager updates stock levels as needed.

3.4 Use Case 4: Report Generation

Actors: ReportManager
Description: Generates daily, weekly, and monthly reports on sales and inventory.
Steps:

1. ReportManager selects the type of report (daily, weekly, monthly).
2. System retrieves relevant data and generates the report.
3. ReportManager reviews and exports the report as needed.

4. Basic Design

The design of the Petrol Pump Management System involves several key classes and their interactions. The major classes include:

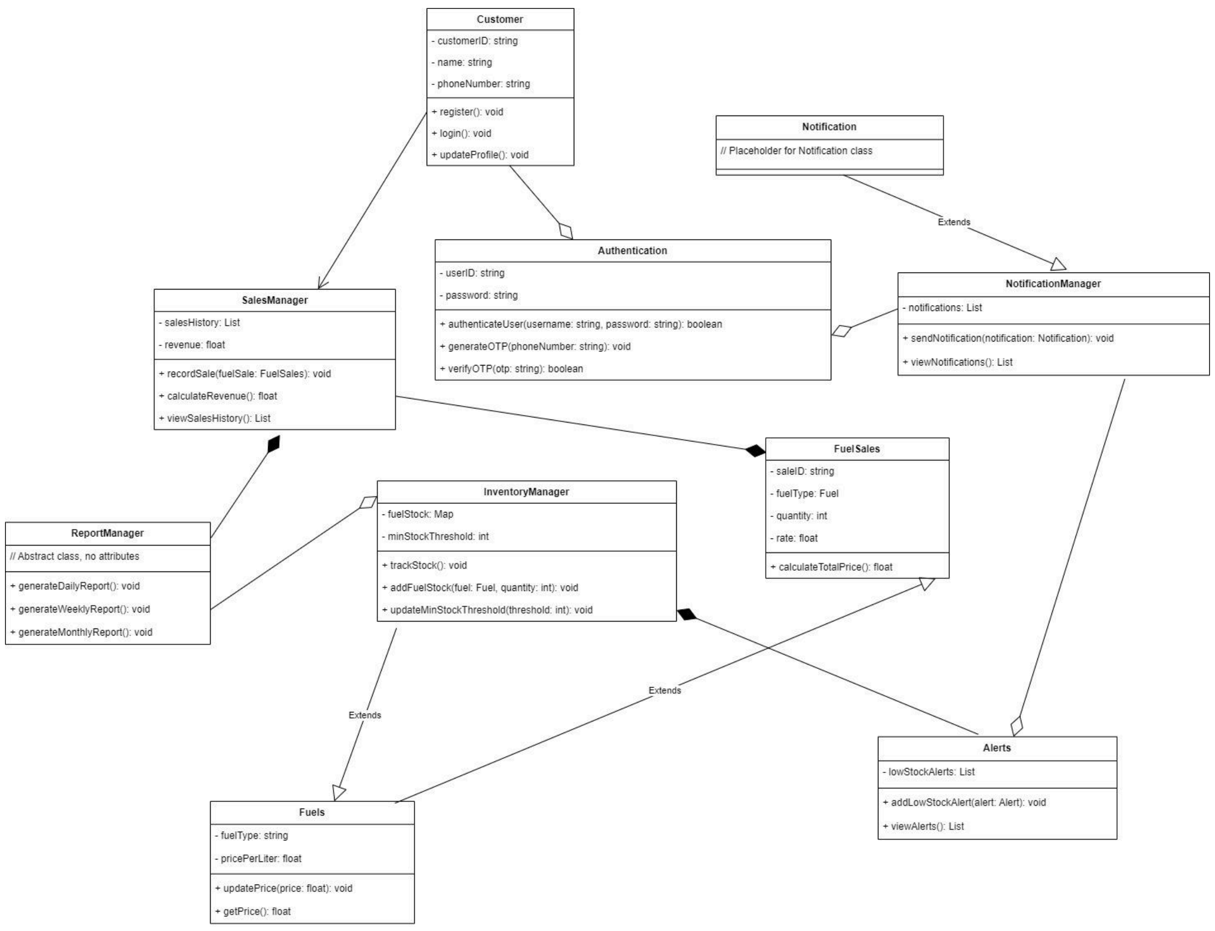
4.1 Class Descriptions

- **Customer:** Handles customer registration, login, and profile updates.
- **Authentication**: Manages user authentication, including OTP verification.
- **InventoryManager**: Manages fuel stock levels and generates alerts for low stock.
- **ReportManager**: Generates and manages various reports on sales and inventory.
- **SalesManager**: Manages sales transactions and updates inventory.
- **FuelSales**: Represents individual fuel sales transactions.

- **Fuels**: Represents different types of fuels available at the petrol pump.
- **Alerts**: Manages system alerts and notifications.
- **NotificationManager**: Sends notifications to customers about updates and promotions.

4.2 Class Diagram

The class diagram provides a visual representation of the system's structure, showing the classes and their relationships.


Customer
- customerID: string
- name: string
- phoneNumber: string
+ register(): void
+ login(): void
+ updateProfile(): void
Notification
// Placeholder for Notification class
Extends
Authentication
- userID: string
- password: string
+ authenticateUser(username: string, password: string): boolean
+ generateOTP(phoneNumber: string): void
+ verifyOTP(otp: string): boolean
NotificationManager
- notifications: List
+ sendNotification(notification: Notification): void
+ viewNotifications(): List
SalesManager
- salesHistory: List
- revenue: float
+ recordSale(fuelSale: FuelSales): void
+ calculateRevenue(): float
+ viewSalesHistory(): List
FuelSales
- saleID: string
- fuelType: Fuel
- quantity: int
- rate: float
+ calculateTotalPrice(): float
InventoryManager
- fuelStock: Map
- minStockThreshold: int
+ trackStock(): void
+ addFuelStock(fuel: Fuel, quantity: int): void
+ updateMinStockThreshold(threshold: int): void
ReportManager
// Abstract class, no attributes
+ generateDailyReport(): void
+ generateWeeklyReport(): void
+ generateMonthlyReport(): void
Extends
Extends
Alerts
- lowStockAlerts: List
+ addLowStockAlert(alert: Alert): void
+ viewAlerts(): List
Fuels
- fuelType: string
- pricePerLiter: float
+ updatePrice(price: float): void
+ getPrice(): float

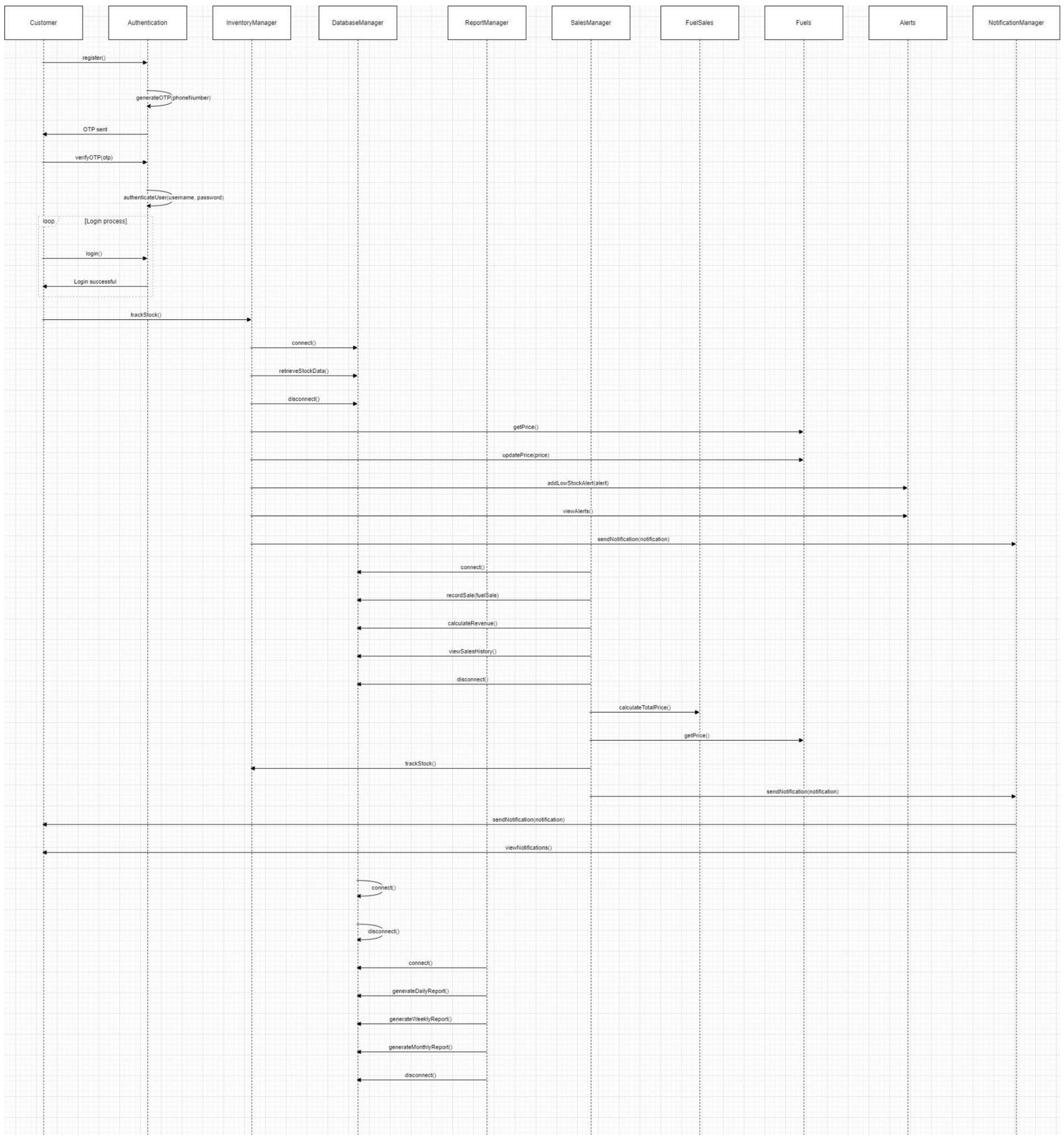
5. Design Verification

5.1 Verification of Core Use Case: UserRegistration

To verify the design, we simulate the core use case of customer registration. The steps involved are:
1. User registers with name and phone number.
2. Authentication class generates and sends OTP.
3. Customer verifies OTP and completes registration.

5.2 Sequence Diagram

The sequence diagram illustrates the interactions between objects over time, providing a clear view of the flow of messages.



6. Parallel Coding

6.1 Module Division

To enhance development efficiency, the project is divided into the following modules:

- **Customer Module:** Handles user registration and authentication.

- **Sales Module:** Manages fuel sales transactions.
- **Inventory Module**: Tracks and updates fuel stock.
- **Report Module:** Generates various sales and inventory reports.
- **Notification Module**: Manages system notifications.

6.2 Independent Development and Testing

Each module is developed and tested independently, ensuring that interfaces are clearly defined. Independent testing includes creating test cases for each module and verifying their functionality before integration.

6.3 Sample Code

This is just a sample of the code structure that we have used. The actual code has a few improvements, completions and bug fixes.

Customer Class

```
public class Customer {
    private String customerID;
    private String name;
    private String phoneNumber;

    public void register(String name, String phoneNumber) {
        this.name = name;
        this.phoneNumber = phoneNumber;
        this.customerID = generateUniqueID();
    }

    public boolean login(String username, String password) {
        // Authentication logic
        return true;
    }

    public void updateProfile(String name, String phoneNumber) {
        this.name = name;
        this.phoneNumber = phoneNumber;
    }

    private String generateUniqueID() {
        // Unique ID generation logic
        return "CUST" + System.currentTimeMillis();
    }
}
```

InventoryManager Class

```
import java.util.Map;

public class InventoryManager {
    private Map<String, Integer> fuelStock;
    private int minStockThreshold;

    public void trackStock() {
        // Code to track fuel stock levels
    }

    public void addFuelStock(String fuelType, int quantity) {
        // Code to add new fuel stock to the inventory
    }

    public void updateMinStockThreshold(int threshold) {
        this.minStockThreshold = threshold;
    }

    public boolean isStockLow(String fuelType) {
        return fuelStock.get(fuelType) < minStockThreshold;
    }
}
```

7. Initial Integration and Replanning

7.1 Integration Challenges

During initial integration, several challenges were identified, including interface mismatches and data inconsistencies. These issues were addressed through debugging and refining the interfaces between modules.

7.2 Replanning

Based on the outcomes of initial integration, the project plan was updated. New tasks were identified, and resources were reallocated to ensure that the project stayed on track. The replanning phase involved:

- Adjusting timelines for module completion.
- Prioritising critical issues identified during integration.

7.3 Enhanced Communication

Enhanced communication among team members was established to ensure that integration issues were promptly addressed. Regular meetings and status updates were conducted to keep everyone aligned.

8. Final Integration

8.1 Integration of Modules

The final integration phase involved combining all modules into a cohesive system. This phase focused on ensuring that all modules worked seamlessly together and that the overall system met the requirements.

8.2 System Testing

Extensive testing was conducted to verify the functionality and performance of the integrated system. This included:

- **Functional Testing**: Ensuring that all features worked as expected.
- **Performance Testing:** Checking the system's response time and ability to handle concurrent users.
- **Security Testing:** Verifying data security and user authentication mechanisms.

8.3 Bug Fixing and Optimization

Any bugs identified during testing were fixed, and the system was optimized for better performance. This included:

- Refining algorithms for faster processing.
- Enhancing database queries for efficient data retrieval.
- Improving user interface for better usability.

9. Conclusion

9.1 Project Summary

The Petrol Pump Management System offers a comprehensive solution for managing petrol pump operations. Key features include secure user authentication, real time inventory tracking, detailed reporting, and an effective notification system. This system enhances operational efficiency, security, and compliance with regulatory requirements, empowering petrol pump owners to optimize their business processes.

9.2 Achievements

The project successfully achieved the following:

- Developed a robust and scalable system for petrol pump management.
- Provided real time inventory management with low stock alerts.
- Enabled comprehensive report generation for informed decisionmaking.

9.3 Future Enhancements

Several future enhancements are planned to further improve the system:

- **Mobile Application**: Developing a mobile app for better accessibility and convenience.
- **Advanced Analytics:** Integrating advanced analytics to provide deeper insights into business operations.
- **IoT Integration:** Leveraging IoT devices for realtime monitoring and automation of petrol pump operations.
- **Customer Loyalty Program:** Implementing a loyalty program to reward frequent customers and boost customer retention.

9.4 Lessons Learned

The project provided valuable insights into the challenges and complexities of developing a comprehensive management system. Key lessons learned include:

- The importance of modular design for scalability and maintainability.
- The need for thorough testing at each stage of development.
- The benefits of regular communication and collaboration among team members.