**Software Design & Analysis**

**Software Requirements & Design Document**

**PetroPulse**

**HAArtBeat Solutions**

Anas Bin Rashid i220907

Adan Malik i221000

Hasnain Akhtar i221241

**Table of Contents**

[**INTRODUCTION** 2](#_Toc183545204)

[**Purpose** 2](#_Toc183545205)

[**Product Scope** 2](#_Toc183545206)

[**Title** 3](#_Toc183545207)

[**Objectives** 3](#_Toc183545208)

[**Problem Statement** 3](#_Toc183545209)

[**OVERALL DESCRIPTION** 4](#_Toc183545210)

[**Product Perspective** 4](#_Toc183545211)

[**Product Functions** 5](#_Toc183545212)

[**List of Use Cases** 5](#_Toc183545213)

[**Extended Use Cases** 6](#_Toc183545214)

[**Use Case Diagram** 21](#_Toc183545215)

[**OTHER NONFUNCTIONAL REQUIREMENTS** 22](#_Toc183545216)

[**Performance Requirements** 22](#_Toc183545217)

[**Safety Requirements** 22](#_Toc183545218)

[**Security Requirements** 22](#_Toc183545219)

[**Software Quality Attributes** 22](#_Toc183545220)

[**Business Rules** 23](#_Toc183545221)

[**Operating Environment** 23](#_Toc183545222)

[**User Interfaces** 23](#_Toc183545223)

[**DOMAIN MODEL** 36](#_Toc183545224)

[**SYSTEM SEQUENCE DIAGRAM** 37](#_Toc183545225)

[**SEQUENCE DIAGRAM** 44](#_Toc183545226)

[**CLASS DIAGRAM** 60](#_Toc183545227)

[**COMPONENT DIAGRAM** 61](#_Toc183545228)

[**PACKAGE DIAGRAM** 61](#_Toc183545229)

[**DEPLOYMENT DIAGRAM** 61](#_Toc183545230)

[**DESIGN PATTERNS** 62](#_Toc183545231)

[**GRASP Patterns** 62](#_Toc183545232)

[**Gang of Four Patterns** 64](#_Toc183545233)

# **INTRODUCTION**

## **Purpose**

The purpose of this document is to define the software requirements for **PetroPulse**, a petrol station management software that allows station owners to manage stations, handle sales transactions, fuel orders, payments, customer loyalty, and various administrative tasks. This document details the system's functional and non-functional requirements, providing a clear understanding of the system's goals and its implementation.

## **Product Scope**

**PetroPulse** provides a comprehensive solution for managing petrol stations. It integrates with payment processing, fuel supply management, and customer interaction. The software helps owners, workers, and customers interact seamlessly with smooth fuel purchase, inventory management, and customer loyalty system. It supports multiple users, each with specific roles, like owner, worker, and customer.

The software supports:

* Fuel station management
* Inventory tracking
* Payment and transaction processing
* Customer management
* Worker scheduling

## **Title**

**PetroPulse - Petrol Station Management System**

## **Objectives**

1. **Automated** **Operations**: Streamline critical tasks such as employee management, fuel dispensing, and transactions, reducing errors and manual workload.

2. **Financial** **Management**: Ensure accurate tracking of sales, expenses, and profits, supporting effective financial oversight.

3. **Inventory** **Monitoring**: Provide an efficient system to track fuel stocks and station inventory.

4. **Employee** **&** **Shift** **Management**: Simplify shift scheduling, minimizing administrative tasks.

5. **Enhanced** **Customer** **Service**: Speed up transactions and offer features like loyalty programs, improving the customer experience.

6. **Data** **Analytics** **&** **Reporting**: Deliver detailed analytics on fuel usage, sales, and employee wages, aiding in informed decision-making.

## **Problem Statement**

In the fast-paced petroleum industry, petrol stations face numerous operational challenges, from managing fuel inventory and handling transactions to overseeing employee shifts. Many stations still rely on outdated or manual processes, leading to inefficiencies, higher error rates, and administrative burdens. These issues, compounded by the need for real-time data, make it difficult for petrol stations to operate smoothly and meet the increasing demands of customers.

PetroPulse aims to address these problems by targeting petrol station owners, operators, and workers, aiming to modernize and automate daily operations. The system addresses the inefficiencies and errors in manual operations by providing a comprehensive management solution. The growing demand for automation in the petroleum industry proves the Problem Solution fit, and the platform's unique value proposition lies in its integrated, customizable features that streamline operations and enhance customer service. Product specifications include automation of tasks such as fuel tracking, sales, inventory, and employee management, all within a single, scalable system.

The product-market fit is evidenced by industry’s demand for efficiency and accuracy, with distribution channels involving direct sales and partnerships with fuel brands. Revenue streams include software licensing and maintenance contracts, while competitive positioning focuses on adaptability across different stations. Resources needed include developers and cloud infrastructure, with customer retention supported through continuous feedback and training. Business model validation will come from pilot testing, ensuring the solution is viable. The implementation plan involves a phased rollout, beginning with key stations and expanding as the system is optimized.

# **OVERALL DESCRIPTION**

## **Product Perspective**

**PetroPulse** is a self-contained product designed to automate the key operations of a petrol station. It will connect with external systems, including fuel price systems, fuel suppliers, and customer data systems, to ensure seamless operations.

## **Product Functions**

* **Fuel Management**: Tracks fuel supply, sales, and inventory.
* **Customer Management**: Manages customer accounts, loyalty points, and membership.
* **Worker Management**: Schedules shifts, and calculates salaries.
* **Payment Processing**: Handles cash, card, and other payment methods.
* **Reporting**: Generates reports for worker wages, customer management and sales monitoring.

## **List of Use Cases**

1. Apply for Membership
2. Purchase Fuel
3. Make Payment
4. View Loyalty Points
5. Dispense Fuel
6. View Schedule
7. Perform Station Operations (Worker)
8. Set Employee Schedule
9. Manage Customers
10. Manage Workers
11. Order Fuel
12. Order Maintenance
13. View Reports
14. Add Fuel Station
15. Perform Station Operations (Owner)

## **Extended Use Cases**

**1. Use Case Name: Apply for Membership**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Customer  
**Stakeholders and Interests:**

* **Customer**: Wants to apply for a membership to receive benefits like discounts or loyalty points.
* **Owner**: Wants to manage memberships effectively and maintain customer records.
* **Database**: Must store the customer’s membership details securely.

**Preconditions:**

* Customer must have an account in the system.
* Customer details (name, contact info, etc.) must be valid and up to date.

**Postconditions:**

* Customer is registered as a member in the system.
* Membership details are stored in the database.

**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Customer selects the "Apply for Membership" option. | 2. System displays the details for confirmation. |
| 3. Customer confirms the required details (e.g., name, email). | 4. System validates the details. |
| 5. Customer confirms the application. | 6. System registers the customer as a member and membership fee is charged. |
|  | 7. System generates a membership confirmation. |

**Extensions:**

* **3a.** Customer has invalid or incomplete details:
  + System prompts the customer to correct the errors.
* **6a.** System fails to process the registration due to server issues:
  + System notifies the customer and suggests retrying later.

**2. Use Case Name: Purchase Fuel**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Customer  
**Stakeholders and Interests:**

* **Customer**: Wants to purchase the desired type and amount of fuel.
* **Owner**: Wants accurate records of all fuel purchases.
* **Supplier**: Needs fuel usage data to manage restocking.
* **Database**: Must log the transaction and update inventory.

**Preconditions:**

* Fuel is available in the inventory.
* Payment gateway is operational.

**Postconditions:**

* Fuel is delivered to the customer.
* The transaction is recorded in the database.

**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Customer selects the fuel type and quantity. | 2. System displays the total cost. |
| 3. Customer confirms the purchase. | 4. System checks fuel availability and authorizes the transaction. |
| 5. Fuel is dispensed to the customer. | 6. System updates the inventory. |

**Extensions:**

* **1a.** Customer selects an unavailable fuel type or quantity:
  + System notifies the customer of unavailability.
* **5a.** Fuel Unavailable:
  + System alerts the customer.

**3. Use Case Name: View Loyalty Points**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Customer  
**Stakeholders and Interests:**

* **Customer**: Wants to view accumulated loyalty points.
* **Owner**: Wants customers to be incentivized through loyalty programs.
* **Database**: Must store and retrieve loyalty points data accurately.

**Preconditions:**

* Customer must have a registered account.
* Customer must have accumulated loyalty points.

**Postconditions:**

* Loyalty points are displayed to the customer.

**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Customer selects "View Loyalty Points" option. | 2. System retrieves loyalty points from the database. |
| 3. Customer views the displayed loyalty points. | 4. System logs the action for record-keeping. |

**Extensions:**

* **2a.** System fails to retrieve loyalty points due to server issues:
  + System notifies the customer and suggests trying again later.

**4. Use Case Name: Make Payment**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Customer  
**Stakeholders and Interests:**

* **Customer**: Wants to make payments quickly and securely.
* **Payment Gateway**: Ensures the payment is processed securely.
* **Owner**: Wants to track all transactions accurately.
* **Database**: Must record the transaction details.

**Preconditions:**

* Customer has selected a service or product.
* Payment gateway is operational.

**Postconditions:**

* Payment is successfully processed.
* The transaction is logged in the system.

**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Customer selects a payment option (e.g., cash, card). | 2. System prompts for payment details. |
| 3. Customer provides the payment details. | 4. System validates and processes the payment. |
| 5. Customer receives confirmation of payment. | 6. System generates a receipt and logs the transaction. |

**Extensions:**

* **3a.** Customer provides invalid payment details:
  + System notifies the customer and prompts for correct details.
* **4a.** Payment gateway is unavailable:
  + System notifies the customer and suggests retrying later.

**5. Use Case Name: Dispense Fuel**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Worker  
**Stakeholders and Interests:**

* **Worker**: Wants to dispense fuel efficiently and avoid errors.
* **Customer**: Wants the requested amount of fuel.
* **Owner**: Wants accurate fuel dispensing records and inventory updates.
* **Database**: Must log the transaction and adjust inventory accordingly.

**Preconditions:**

* Fuel is available in inventory.
* Worker is logged into the system.

**Postconditions:**

* Fuel is dispensed to the customer.
* The transaction is logged and inventory updated.

**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Worker selects the "Dispense Fuel" option. | 2. System displays fuel options and quantities. |
| 3. Worker selects fuel type and quantity. | 4. System validates the selection and authorizes the pump. |
| 5. Worker initiates the dispensing process. | 6. System tracks the fuel dispensing and updates inventory. |
| 7. Worker completes the process and confirms it. | 8. System logs the transaction. |

**Extensions:**

* **3a.** Fuel type/quantity is unavailable:
  + System notifies the worker of the shortage.
* **5a.** Pump malfunction occurs:
  + System alerts the worker.

**6. Use Case Name: View Schedule**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Worker  
**Stakeholders and Interests:**

* **Worker**: Wants to view their assigned schedule.
* **Owner**: Wants workers to follow their schedules.

**Preconditions:**

* Worker must be logged into the system.
* Schedule must already be defined.

**Postconditions:**

* Worker views their schedule.

**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Worker selects the "View Schedule" option. | 2. System retrieves the worker’s schedule. |
| 3. Worker views their assigned schedule. | 4. System logs the action. |

**Extensions:**

* **2a.** Schedule is not defined for the worker:
  + System notifies the worker that no schedule is available.

**7. Use Case Name: Perform Station Operations**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Worker  
**Stakeholders and Interests:**

* **Worker**: Wants to ensure smooth operations by managing equipment, inventory, and tasks.
* **Owner**: Wants the station to run efficiently and issues resolved promptly.

**Preconditions:**

* Worker must be logged into the system.
* System must be operational.

**Postconditions:**

* Station operations are recorded, and any inventory changes are logged.

**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Worker selects the "Perform Station Operations" option. | 2. System displays available operations (e.g., check inventory, report issue). |
| 3. Worker selects to change the fuel price and enters a new price. | 4. System validates and changes the fuel price for the fuelstand. |
| 5. Worker can audit performed operations such as cleaning. | 6. System logs the operation details. |

**Extensions:**

* **3a.** Worker enters wrong information:
  + System notifies the worker.
* **4a.** System encounters an error while performing the operation:
  + System notifies the worker and suggests retrying or contacting support.

**8. Use Case Name: Set Employee Schedule**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Owner  
**Stakeholders and Interests:**

* **Owner**: Wants to assign schedules to employees efficiently.
* **Worker**: Wants to know their assigned working hours.
* **Database**: Must store and retrieve employee schedules.

**Preconditions:**

* Owner must be logged into the system.
* Employee records must exist in the system.

**Postconditions:**

* Employee schedule is updated in the system.

**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Owner selects the "Set Employee Schedule" option. | 2. System retrieves the list of employees. |
| 3. Owner selects an employee and specifies a schedule. | 4. System validates the schedule and updates it for the selected employee. |
| 5. Owner confirms the schedule update. | 6. System logs the updated schedule and notifies the employee. |

**Extensions:**

* **3a.** Selected employee does not exist in the system:
  + System notifies the owner and displays valid options.
* **4a.** Entered schedule conflicts with an existing schedule:
  + System prompts the owner to resolve the conflict.

**9. Use Case Name: Manage Customers**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Owner  
**Stakeholders and Interests:**

* **Owner**: Wants to manage customer details for memberships, loyalty programs, and transactions.
* **Database**: Must store and retrieve customer records.

**Preconditions:**

* Owner must be logged into the system.

**Postconditions:**

* Customer details are updated in the system.

**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Owner selects the "Manage Customers" option. | 2. System displays the list of customers. |
| 3. Owner selects a customer to update details. | 4. System retrieves and displays the customer’s current details. |
| 5. Owner updates the necessary information. | 6. System validates and updates the customer record. |
| 7. Owner confirms the update. | 8. System logs the updated information. |

**Extensions:**

* **3a.** Customer is not found in the system:
  + System notifies the owner and suggests adding a new customer.
* **5a.** Invalid or incomplete data is entered:
  + System prompts the owner to correct the input.

**10. Use Case Name: Manage Workers**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Owner  
**Stakeholders and Interests:**

* **Owner**: Wants to manage worker details for assigning roles and schedules.
* **Worker**: Wants their records to be accurate and up to date.
* **Database**: Must store and retrieve worker records.

**Preconditions:**

* Owner must be logged into the system.

**Postconditions:**

* Worker details are updated in the system.

**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Owner selects the "Manage Workers" option. | 2. System displays the list of workers. |
| 3. Owner selects a worker to update details. | 4. System retrieves and displays the worker’s current details. |
| 5. Owner updates the necessary information. | 6. System validates and updates the worker record. |
| 7. Owner confirms the update. | 8. System logs the updated information. |

**Extensions:**

* **3a.** Worker is not found in the system:
  + System notifies the owner and suggests adding a new worker.
* **5a.** Invalid or incomplete data is entered:
  + System prompts the owner to correct the input.

**11. Use Case Name: Order Fuel**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Owner  
**Stakeholders and Interests:**

* **Owner**: Wants to ensure the station has sufficient fuel inventory.
* **Supplier**: Wants accurate and timely fuel orders.
* **Database**: Must log fuel orders for tracking and reconciliation.

**Preconditions:**

* Owner must be logged into the system.
* Supplier details must exist in the system.

**Postconditions:**

* Fuel order is placed and logged.

**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Owner selects the "Order Fuel" option. | 2. System displays available suppliers and fuel types. |
| 3. Owner selects a supplier and specifies the fuel order. | 4. System validates the order details and sends it to the supplier. |
| 5. Owner confirms the order placement. | 6. System logs the order and notifies the supplier. |

**Extensions:**

* **3a.** Supplier is unavailable:
  + System notifies the owner and suggests an alternative supplier.
* **4a.** Entered fuel order exceeds budget or capacity:
  + System notifies the owner and prompts for adjustment.

**12. Use Case Name: Order Maintenance**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Owner  
**Stakeholders and Interests:**

* **Owner**: Wants to ensure station equipment and infrastructure are well-maintained.
* **Supplier**: Wants accurate and timely maintenance requests.
* **Database**: Must log maintenance requests for tracking and reconciliation.

**Preconditions:**

* Owner must be logged into the system.
* Maintenance providers must exist in the system.

**Postconditions:**

* Maintenance request is placed and logged.

**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Owner selects the "Order Maintenance" option. | 2. System displays available maintenance providers. |
| 3. Owner selects a provider and specifies the maintenance request. | 4. System validates the request details and sends it to the provider. |
| 5. Owner confirms the request placement. | 6. System logs the request and notifies the provider. |

**Extensions:**

* **3a.** Maintenance provider is unavailable:
  + System notifies the owner and suggests alternatives.
* **4a.** Maintenance request exceeds allocated budget:
  + System notifies the owner and prompts for adjustment.

**13. Use Case Name: View Reports**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Owner  
**Stakeholders and Interests:**

* **Owner**: Wants to access detailed reports to analyze performance and make data-driven decisions.
* **Database**: Must store and retrieve all relevant data for generating reports.

**Preconditions:**

* Owner must be logged into the system.
* Reports must be configured and up-to-date in the system.

**Postconditions:**

* The requested report is generated and displayed to the owner.

**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Owner selects the "View Reports" option. | 2. System displays a list of available report types (e.g., sales, inventory, worker reports). |
| 3. Owner selects the desired report type. | 4. System generates the report based on selected criteria. |
| 5. Owner reviews the report and saves or downloads it if needed. | 6. System logs the report access activity. |

**Extensions:**

* **3a.** Report type is not available:
  + System notifies the owner and suggests similar reports.
* **4a.** No data is available for the selected report:
  + System informs the owner and suggests a different type.

**14. Use Case Name: Add Fuel Station**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Owner  
**Stakeholders and Interests:**

* **Owner**: Wants to expand operations by adding new stations to the system.
* **Database**: Must store and update fuel station records.

**Preconditions:**

* Owner must be logged into the system.

**Postconditions:**

* The new fuel station is added to the system and is available for operations.

**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Owner selects the "Add Fuel Station" option. | 2. System displays a form to input new station details (e.g., location, capacity). |
| 3. Owner fills in the station details and submits the form. | 4. System validates the information and adds the new station. |
| 5. Owner confirms the addition. | 6. System logs the new station and updates the database. |

**Extensions:**

* **3a.** Entered details are invalid or incomplete:
  + System prompts the owner to correct the input.
* **4a.** The station already exists in the system:
  + System notifies the owner and suggests updating the existing station instead.

**15. Use Case Name: Perform Station Operations**

**Scope:** PETROPULSE (Petrol Station Management System)  
**Level:** User-goal level  
**Primary Actor:** Owner  
**Stakeholders and Interests:**

* **Owner**: Wants smooth operations and accurate inventory and staff performance.
* **Worker**: Wants clear instructions and tools for efficient operations.
* **Database**: Must store and update operational data.

**Preconditions:**

* Worker or owner must be logged into the system.

**Postconditions:**

* The station’s operational data is updated in the system.

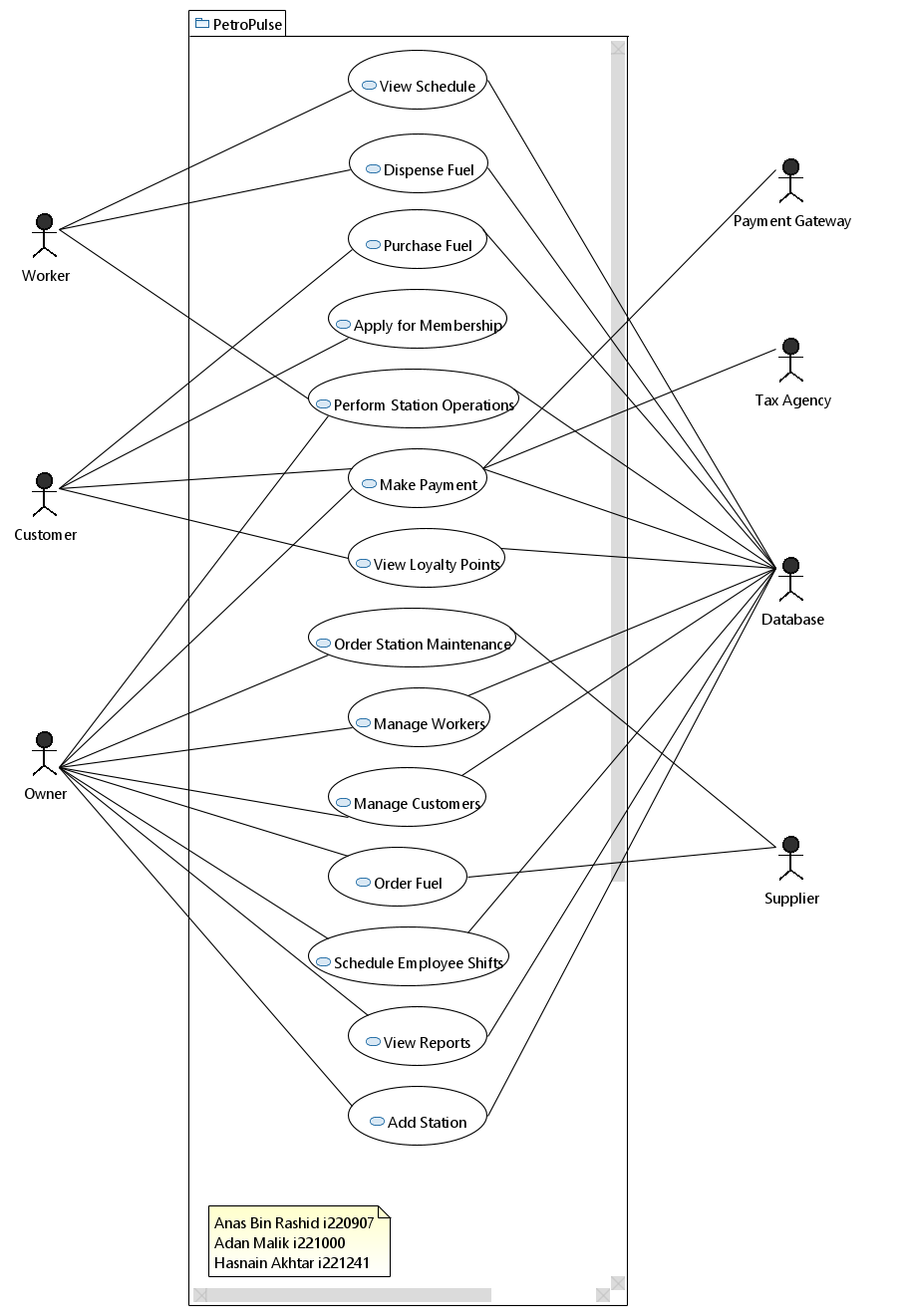
**Main Success Scenario:**

| **Actor’s Actions** | **System’s Responses** |
| --- | --- |
| 1. Worker or owner selects "Perform Station Operations." | 2. System displays a list of operational tasks (e.g., change fuel price, add fuelstand). |
| 3. Actor selects a specific task to perform. | 4. System provides tools or instructions for completing the task. |
| 5. Actor completes the task and submits the update. | 6. System logs the task completion and updates the station records. |

**Extensions:**

* **3a.** Task is unavailable due to system maintenance:
  + System notifies the actor and suggests alternative tasks.
* **5a.** Task completion data is invalid or incomplete:
  + System prompts the actor to re-enter the details.

## **Use Case Diagram**



# **OTHER NONFUNCTIONAL REQUIREMENTS**

## **Performance Requirements**

* The system should be able to handle concurrent usage.
* Transaction processing should take very small time.

## **Safety Requirements**

* Ensure data privacy and protection of sensitive customer data.
* Secure transaction handling, with encryption for sensitive payment information.

## **Security Requirements**

* User authentication and role-based access control for owners, workers, and customers.
* Encrypted communication for all transactions and customer interactions.

## **Software Quality Attributes**

* **Maintainability**: The system should be modular and easy to maintain.
* **Scalability**: The system should be able to scale to support multiple stations.
* **Reliability**: System must be operational 99% of the time.
* **Usability**: The user interface should be intuitive and easy to navigate.

## **Business Rules**

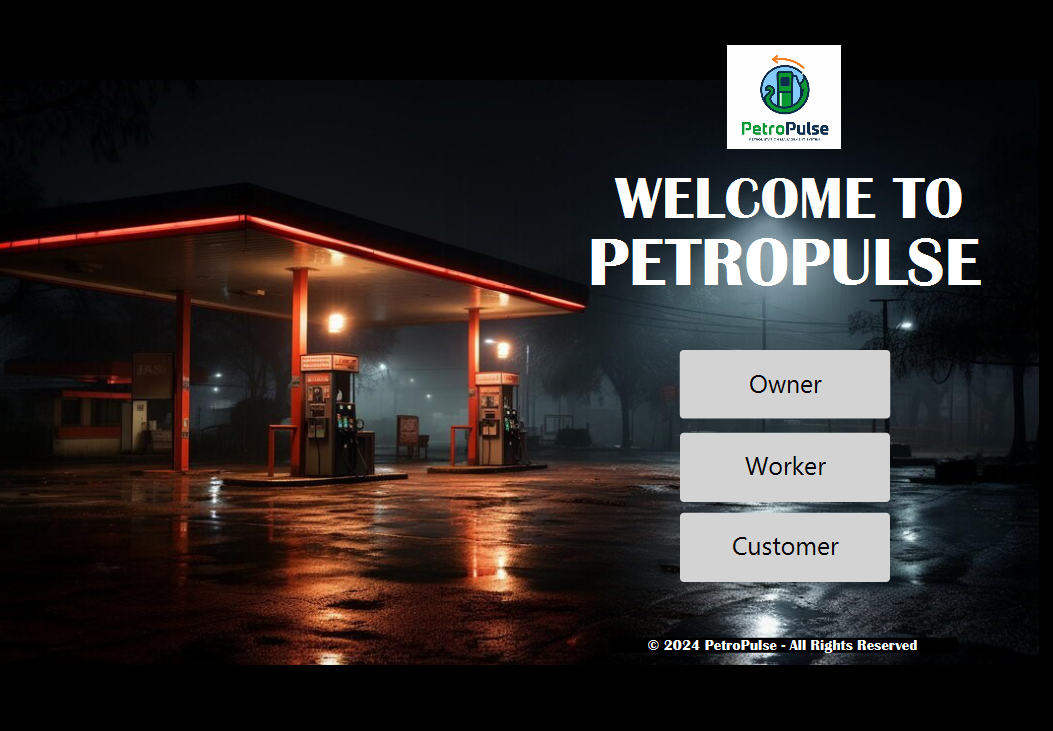
* Only the station owner can add/remove workers.
* Workers can only access their schedules and sales data.
* Customers must be authenticated to make a payment.

## **Operating Environment**

* The system will run on web browsers, supported by major operating systems (Windows, macOS, Linux).
* The backend will run on a MySQL database.

## **User Interfaces**

* Landing Page

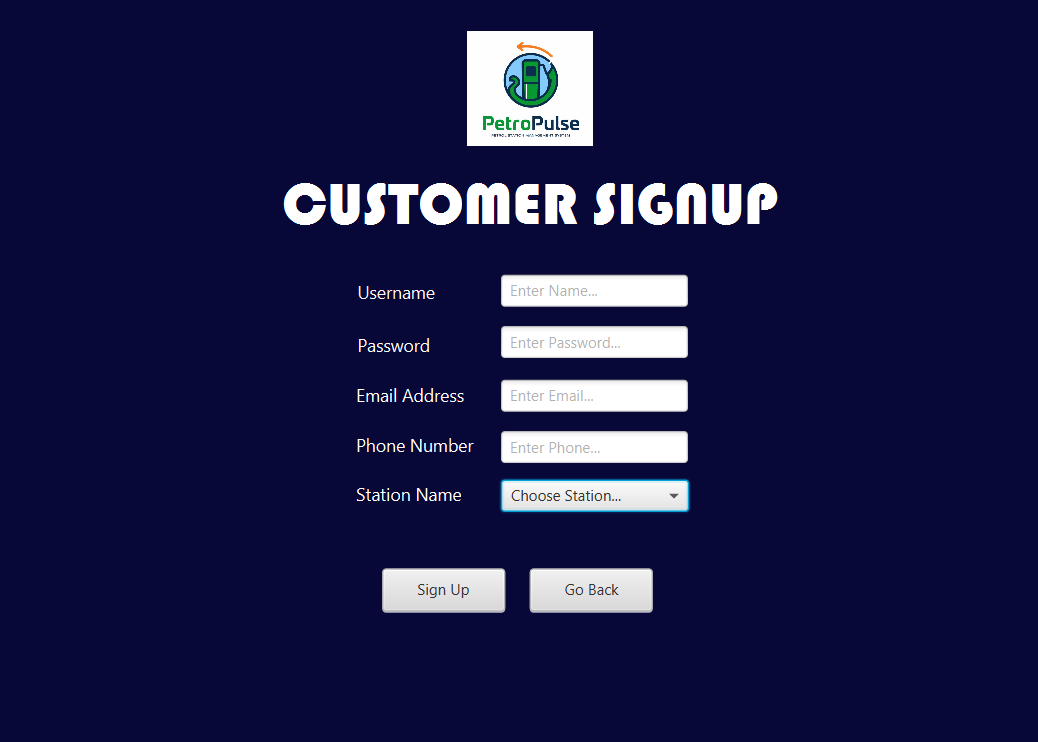


* Login Pages (similar login pages for Worker and Owner)

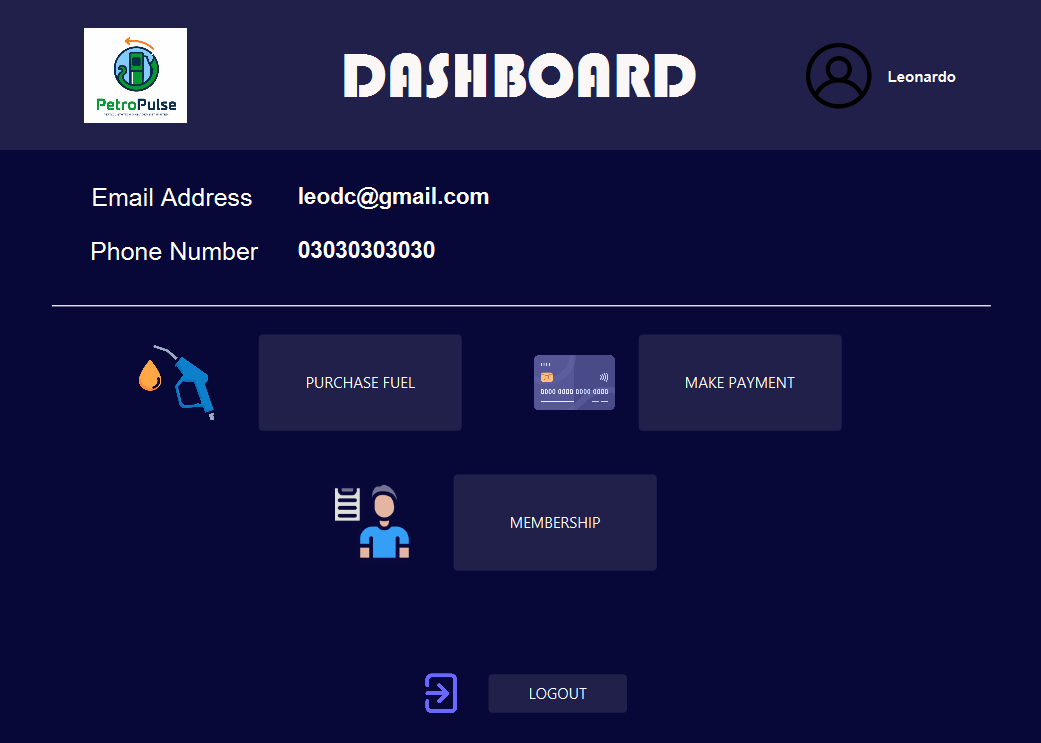
A screenshot of a login screen

Description automatically generated

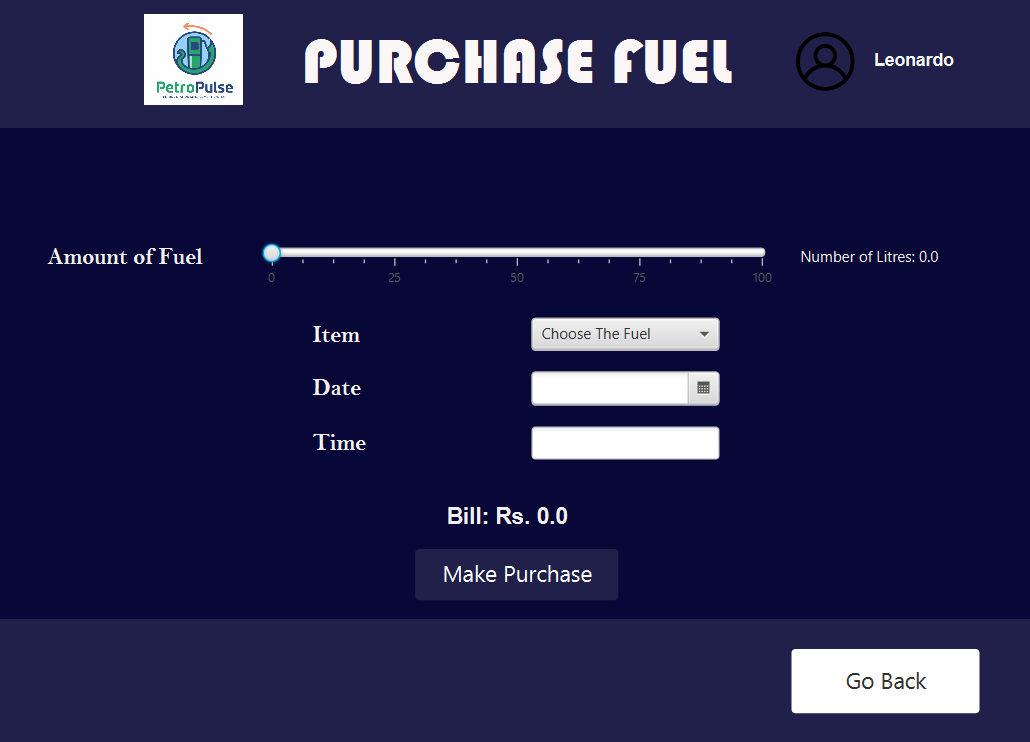
* Signup Pages



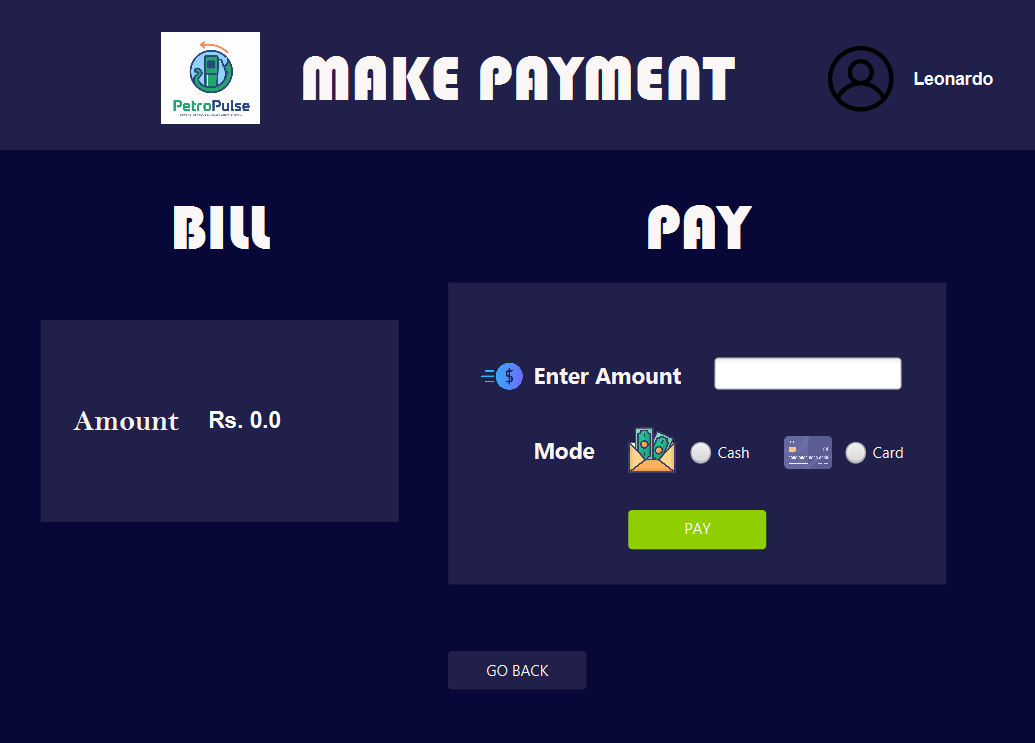
* Customer Home Page



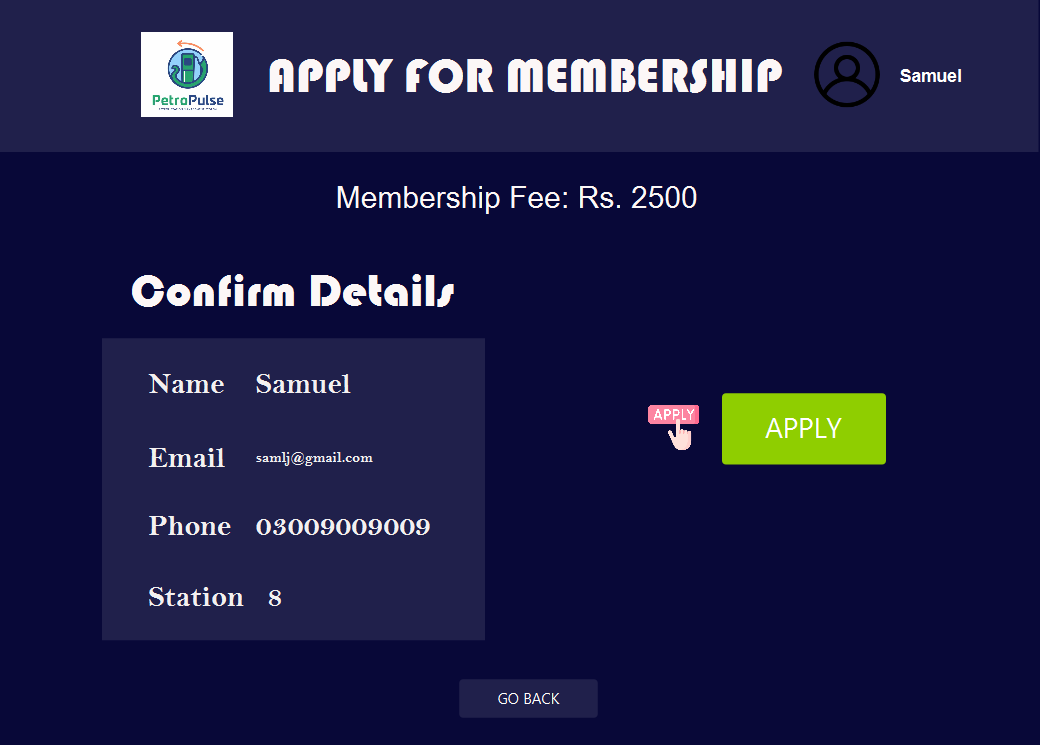
* Customer Purchase Fuel



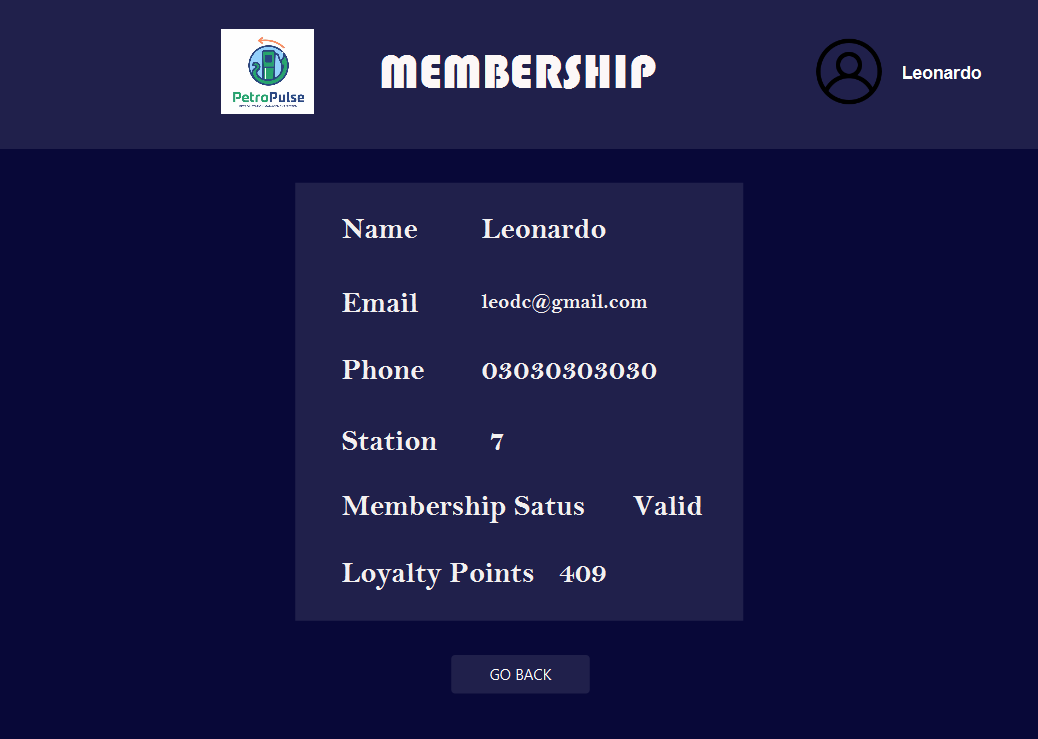
* Customer Make Payment



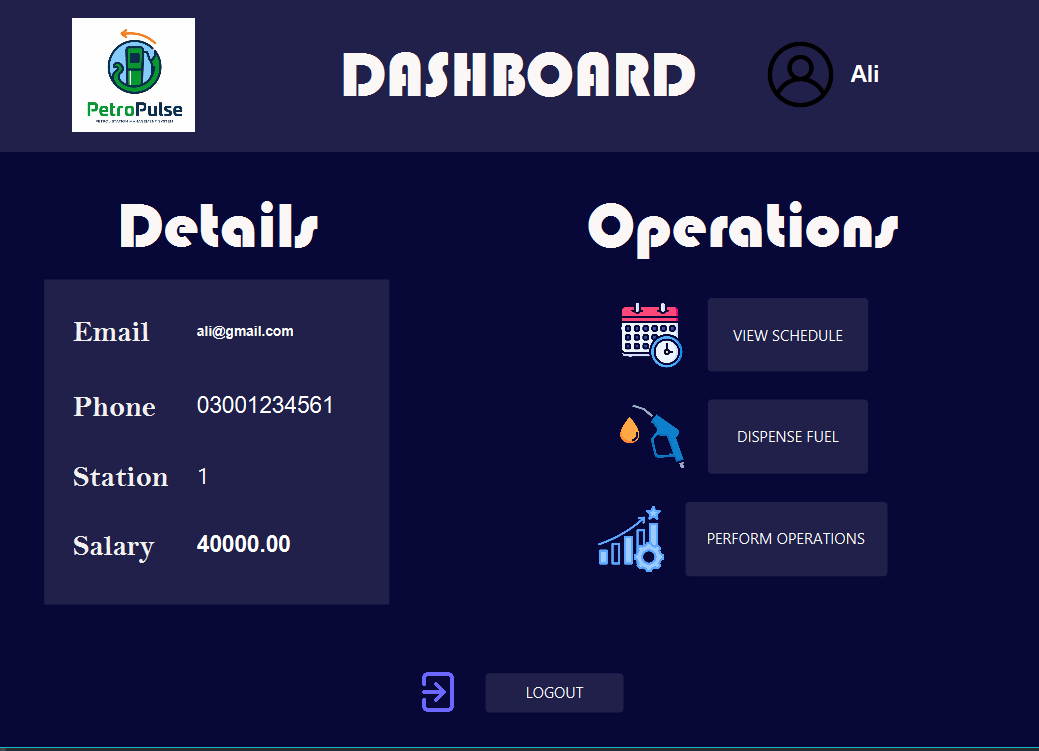
* Customer Apply for Membership



* Customer View Loyalty Points



* Worker Home Page



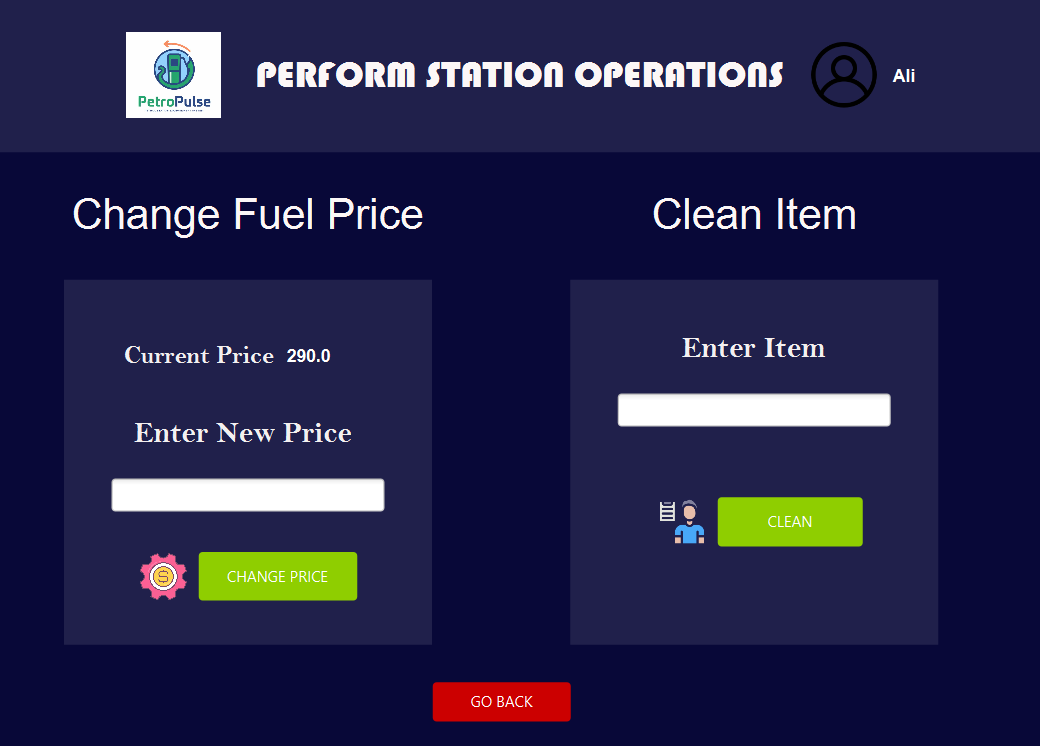
* Worker Dispense Fuel



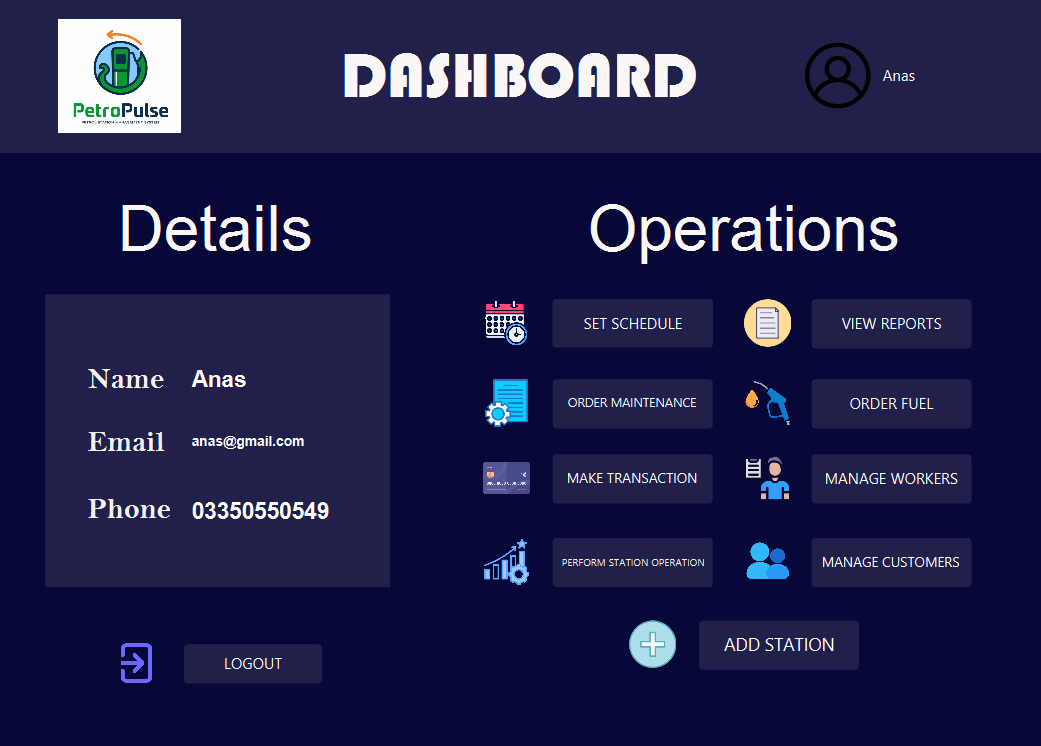
* Worker View Schedule



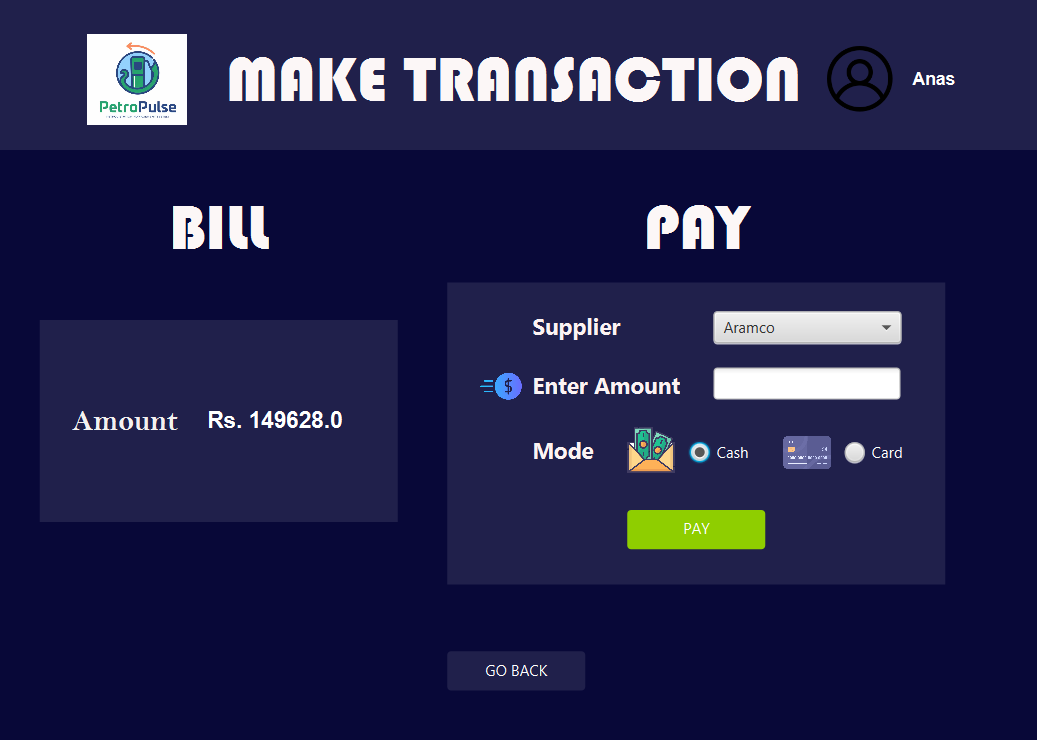
* Worker Perform Station Operations



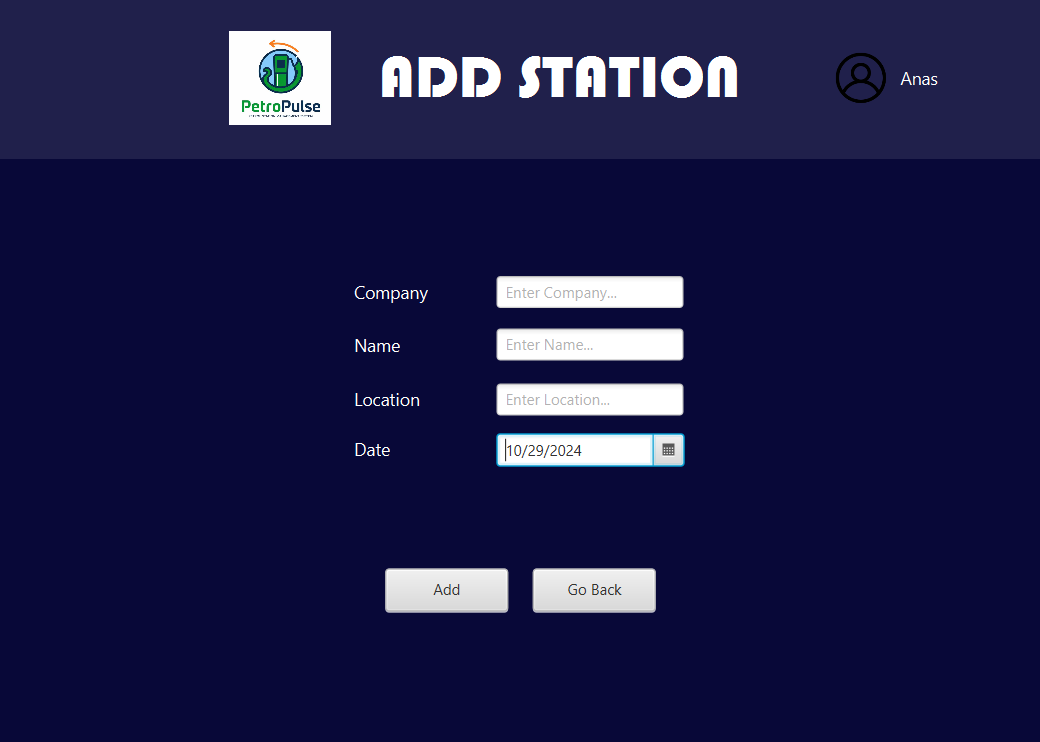
* Owner Home Page



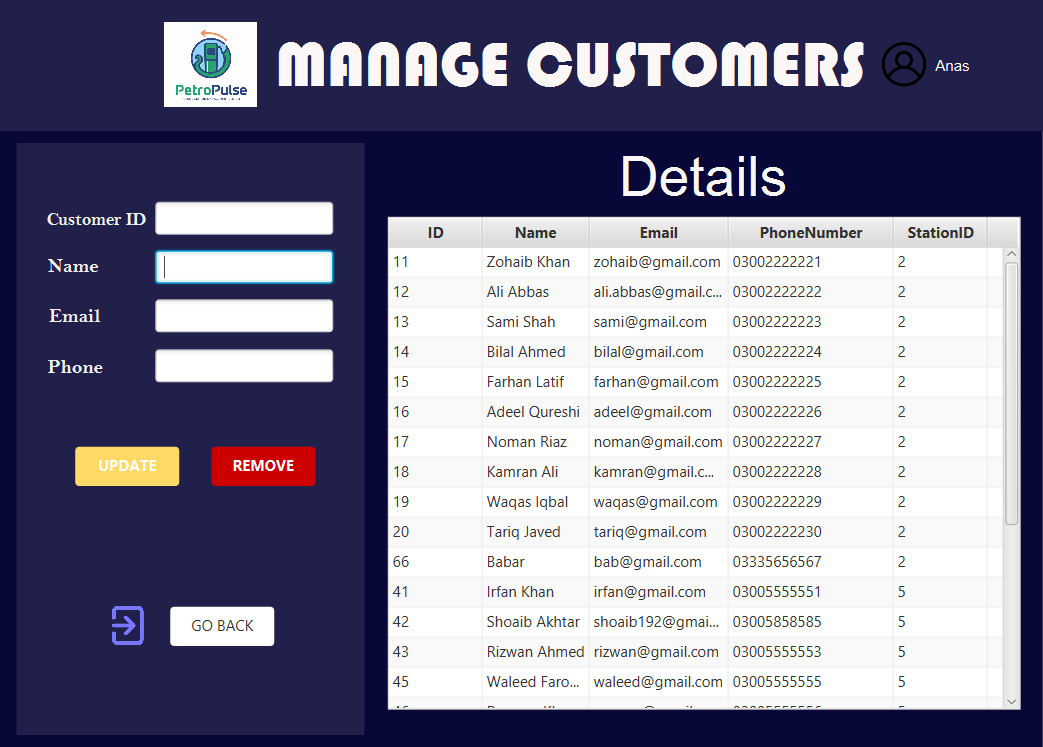
* Owner Make Transaction



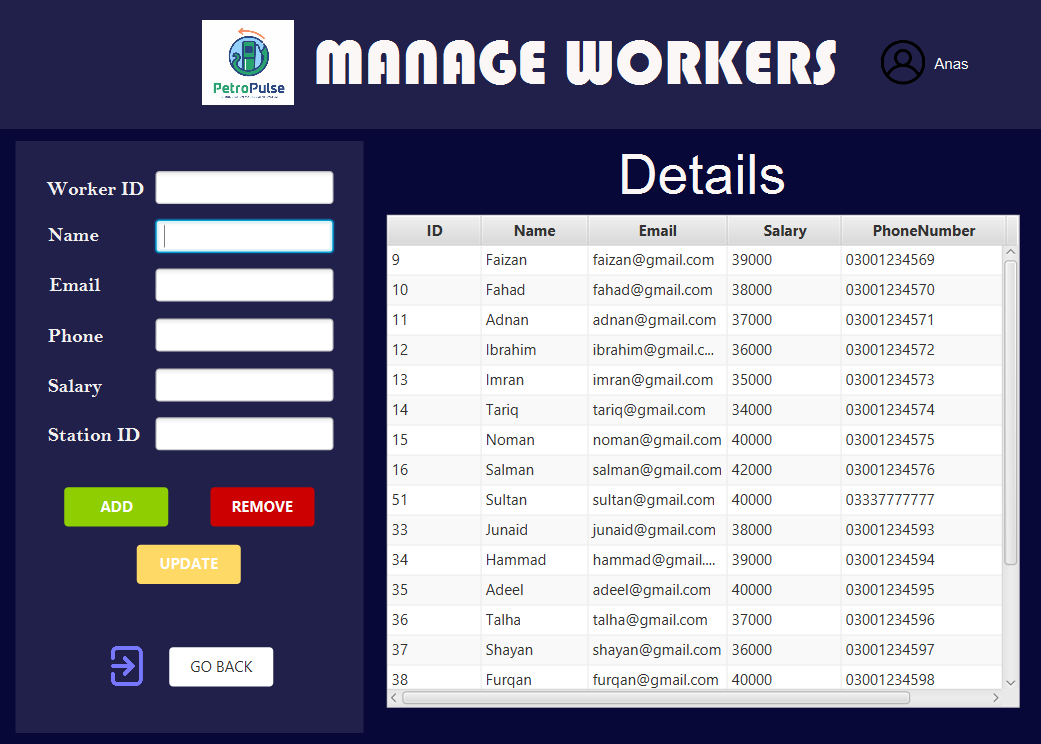
* Owner Add Station



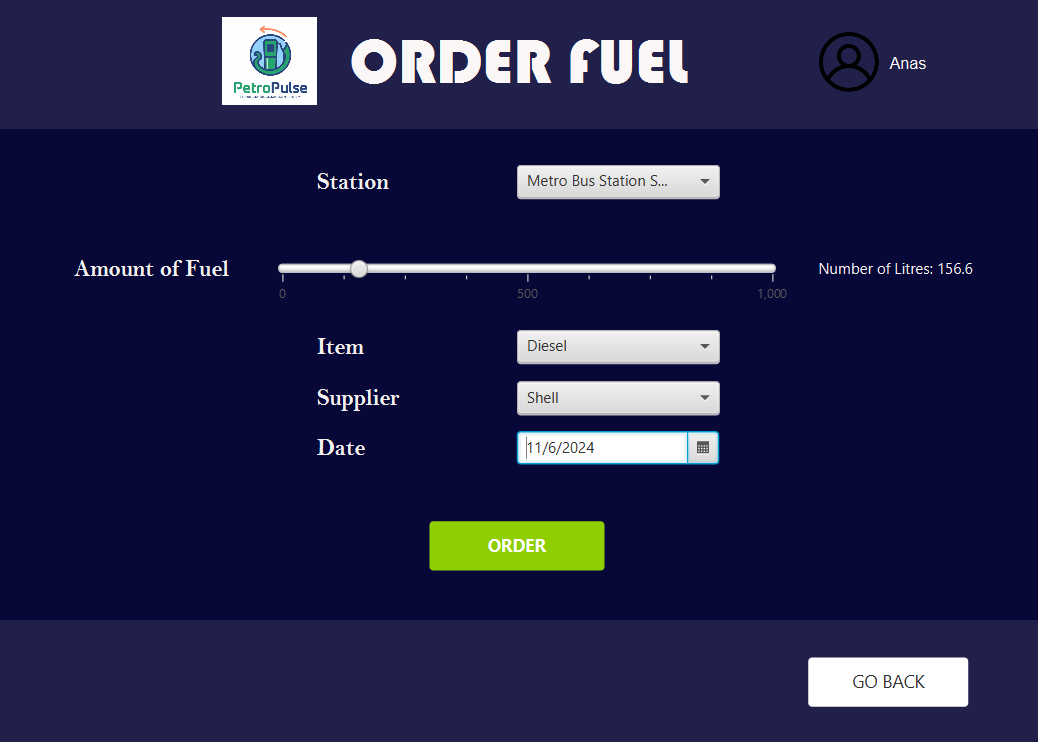
* Owner Manage Customers



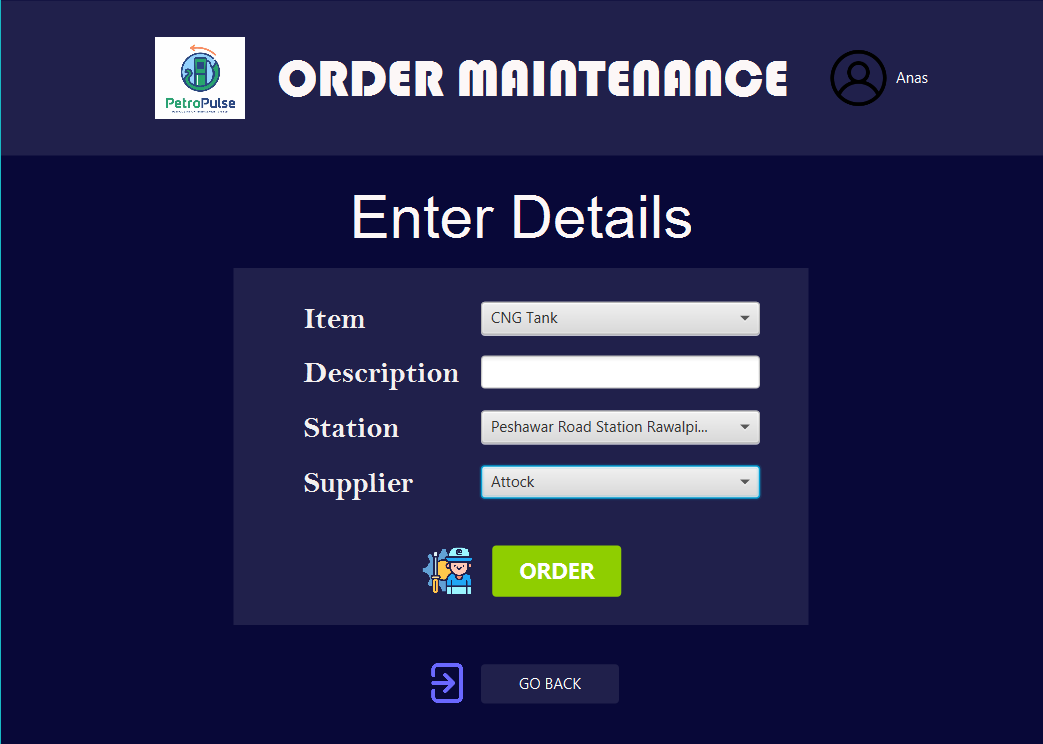
* Owner Manage Workers



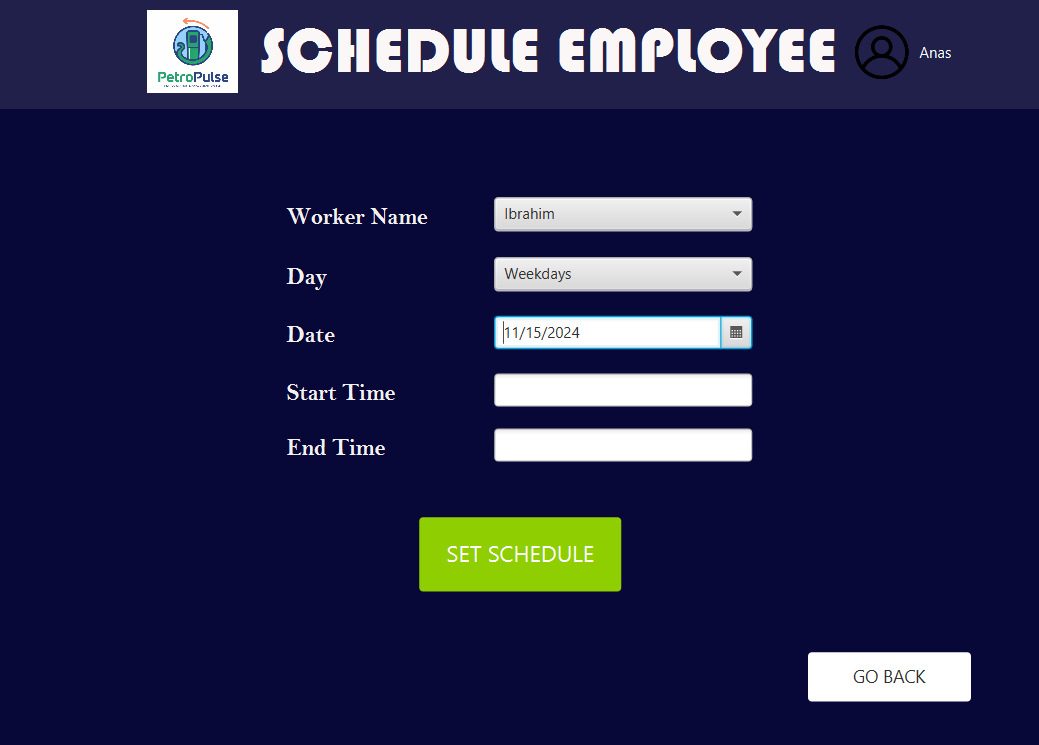
* Owner Order Fuel



* Owner Order Maintenance

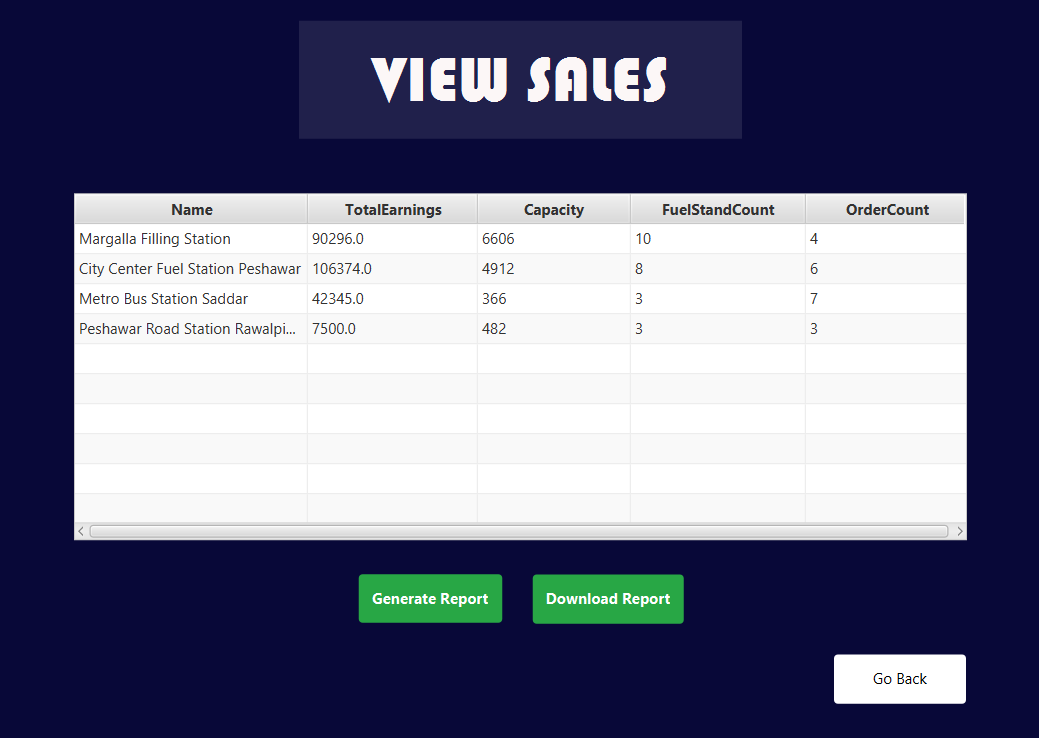


* Owner Set Schedule

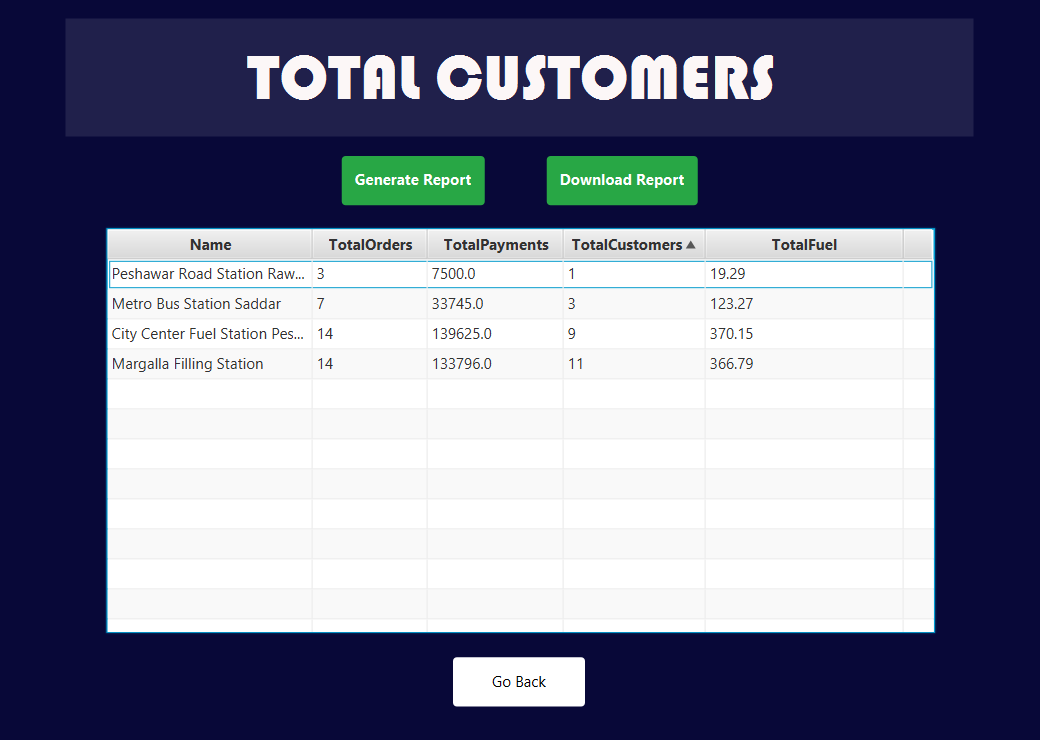


* Owner View Reports

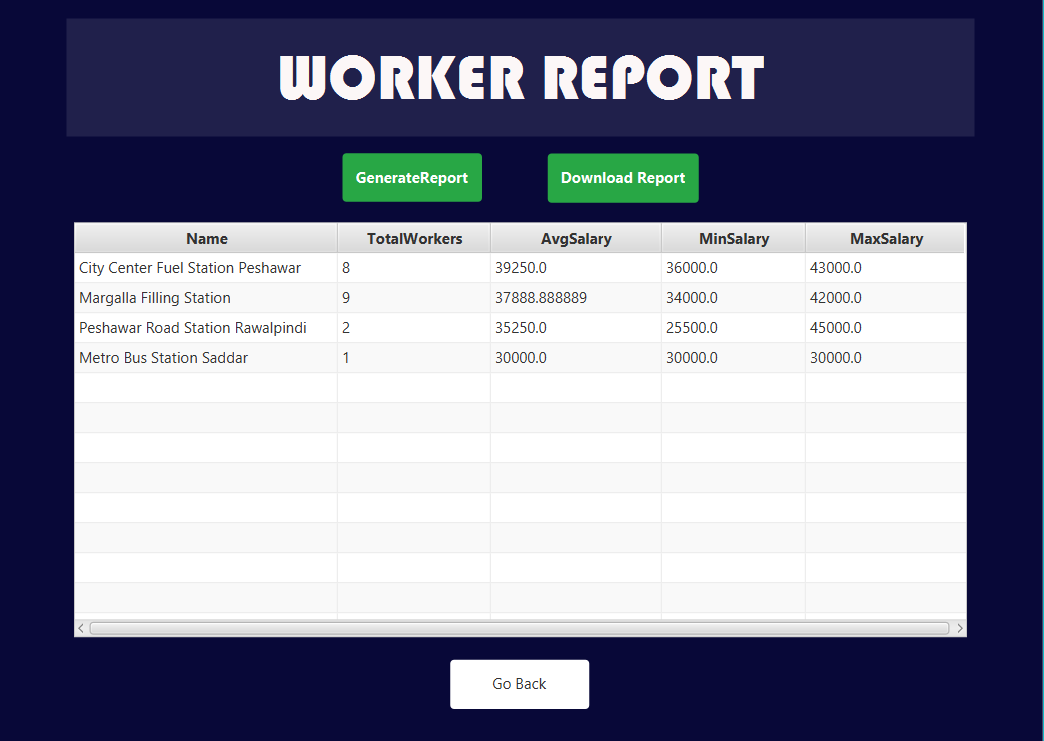
1. View Sales Reports



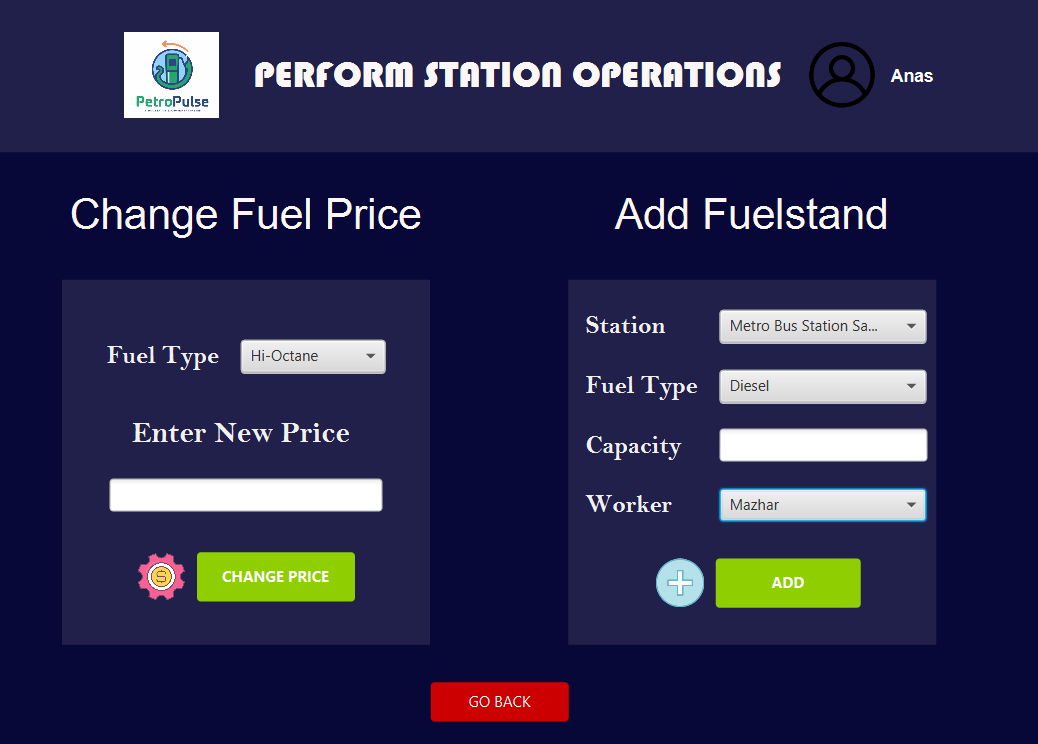
1. View Customer Reports



1. View Worker Reports



* Owner Perform Station Operations



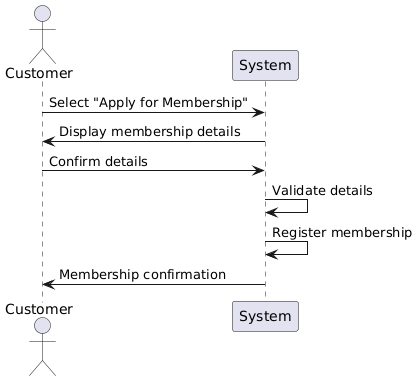
# **DOMAIN MODEL**

A diagram of a company

Description automatically generated with medium confidence

# **SYSTEM SEQUENCE DIAGRAM**

1. Apply for Membership



1. Purchase Fuel

A diagram of a customer

Description automatically generated

1. Make Payment

A diagram of a system

Description automatically generated

1. View Loyalty Points

A diagram of a system

Description automatically generated

1. Dispense Fuel

A diagram of a system

Description automatically generated

1. View Schedule

A diagram of a system

Description automatically generated

1. Perform Station Operations (Worker)

A diagram of a system

Description automatically generated

1. Set Employee Schedule

A diagram of a system

Description automatically generated

1. Manage Customers

A diagram of a system

Description automatically generated

1. Manage Workers

A diagram of a system

Description automatically generated

1. Order Fuel

A diagram of a system

Description automatically generated

1. Order Maintenance

A diagram of a system

Description automatically generated

1. View Reports

A diagram of a system

Description automatically generated

1. Add Fuel Station

A diagram of a system

Description automatically generated

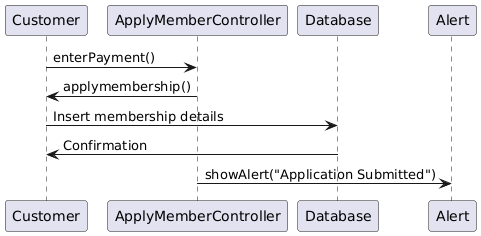
1. Perform Station Operations (Owner)

A diagram of a system

Description automatically generated

# **SEQUENCE DIAGRAM**

1. Apply for Membership
   1. *enterPayment()*



1. Purchase Fuel
   1. *showbill()*

A screenshot of a computer

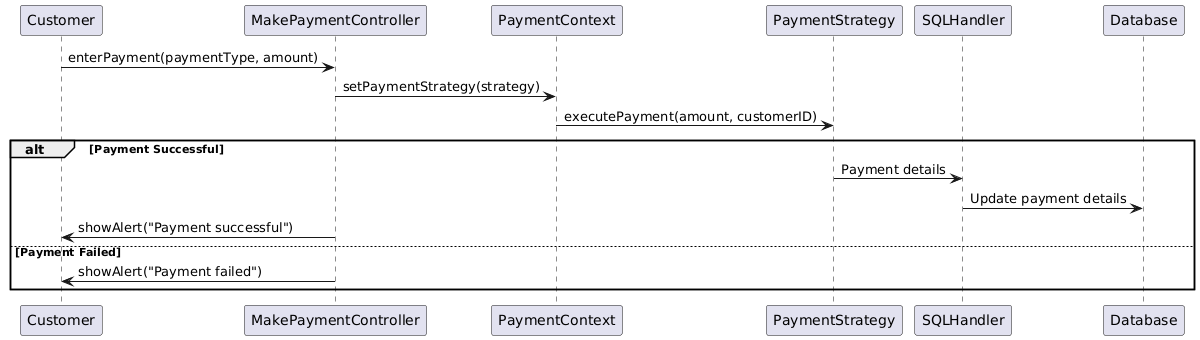
Description automatically generated

* 1. *saleDetected()*

A screenshot of a computer

Description automatically generated

1. Make Payment
   1. *enterPayment()*



1. View Loyalty Points
   1. *setval()*

A diagram of a customer

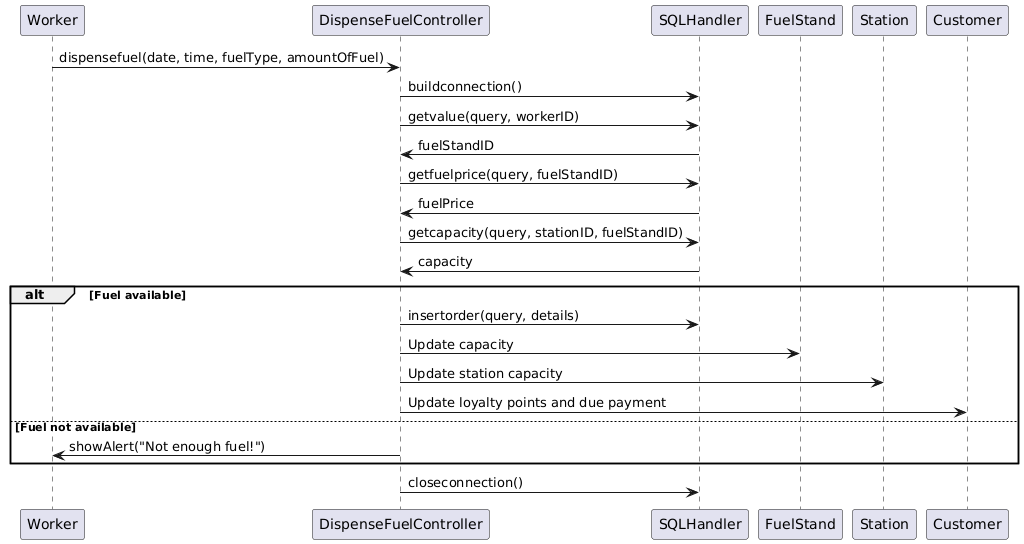
Description automatically generated

1. Dispense Fuel
   1. *showbill()*

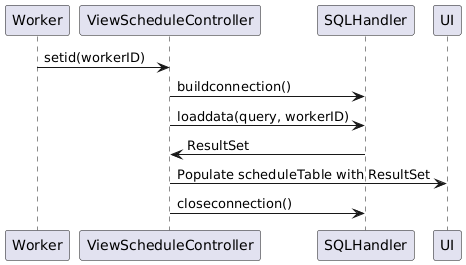
A diagram of a machine

Description automatically generated

* 1. *dispensefuel()*



1. View Schedule
   1. *loadschedule()*



1. Perform Station Operations (Worker)
   1. *changeprice()*

A screenshot of a computer program

Description automatically generated

* 1. *clean()*

A screenshot of a computer program

Description automatically generated

1. Set Employee Schedule
   1. *setschedule()*

A screenshot of a computer

Description automatically generated

1. Manage Customers
2. *initialize()*

A diagram of a software developer

Description automatically generated

1. *removecustomer()*

A diagram of a program

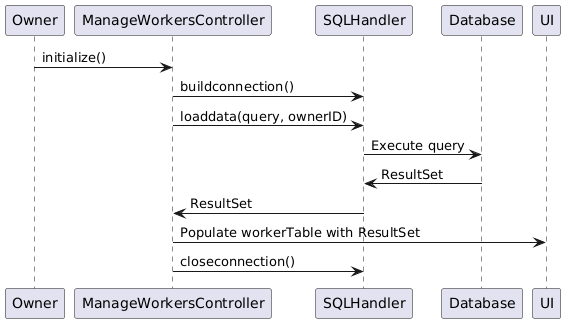
Description automatically generated

1. *updatecustomer()*

A screenshot of a computer program

Description automatically generated

1. Manage Workers
   1. *initialize()*



* 1. *addworker()*

A screenshot of a computer program

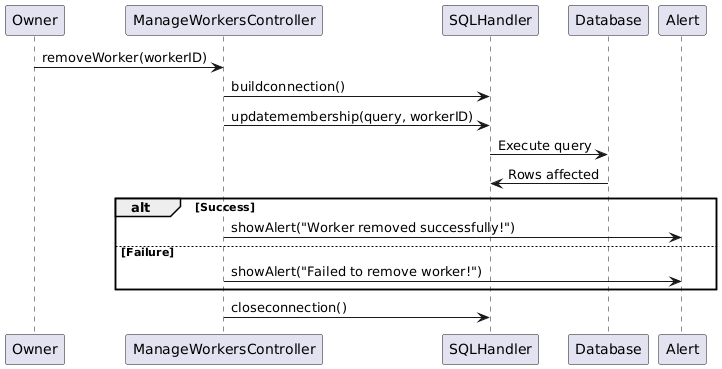
Description automatically generated

* 1. *updateworker()*

A screenshot of a computer

Description automatically generated

* 1. *removeworker()*



1. Order Fuel
   1. *showbill()*

A diagram of a function

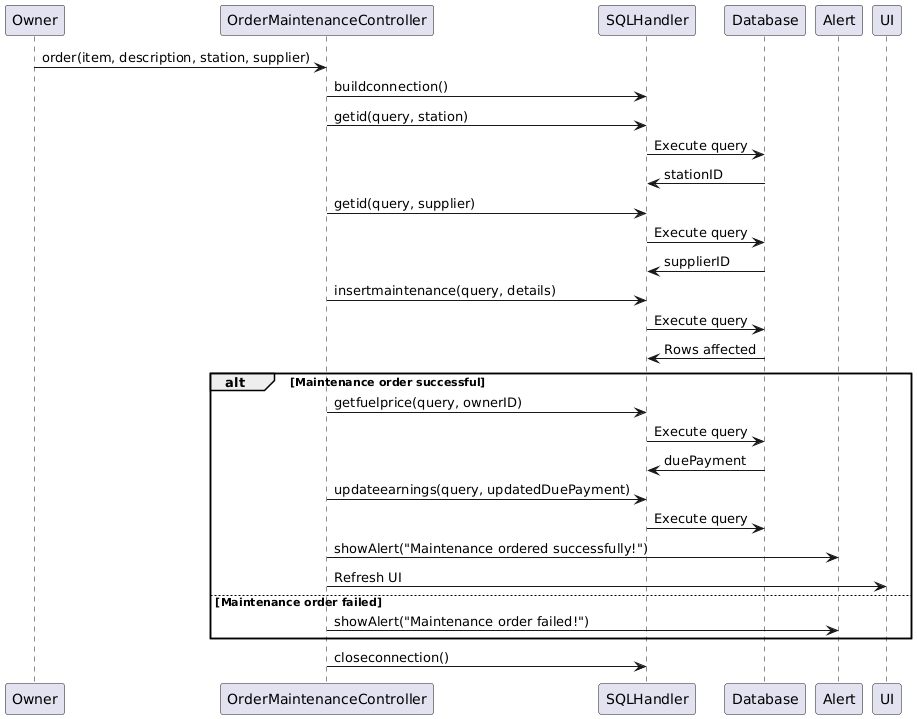
Description automatically generated

* 1. *orderfuel()*

A screenshot of a computer

Description automatically generated

1. Order Maintenance
   1. *order()*



1. View Reports
   1. *viewworker()*

A diagram of a program

Description automatically generated

* 1. *viewcustomer()*

A diagram of a program

Description automatically generated

* 1. *viewsales()*

A diagram of a program

Description automatically generated

1. Add Fuel Station
   1. *addstation()*

A screenshot of a computer

Description automatically generated

1. Perform Station Operations (Owner)
   1. *changeprice()*

A diagram of a program

Description automatically generated with medium confidence

* 1. *chooseworker()*

A diagram of a server

Description automatically generated

* 1. *addfuelstand()*

A screenshot of a computer

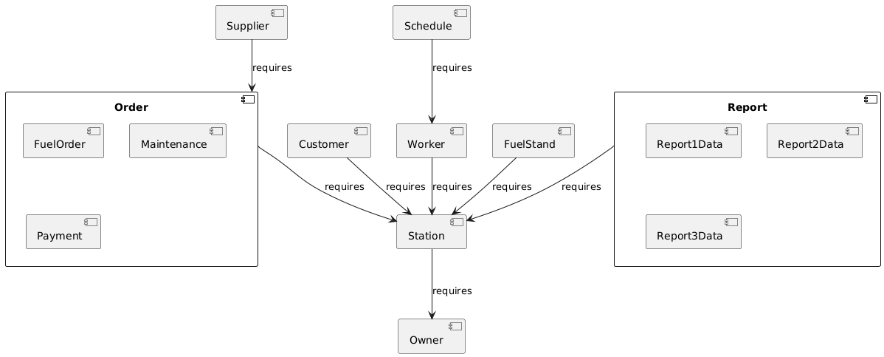
Description automatically generated

# **CLASS DIAGRAM**

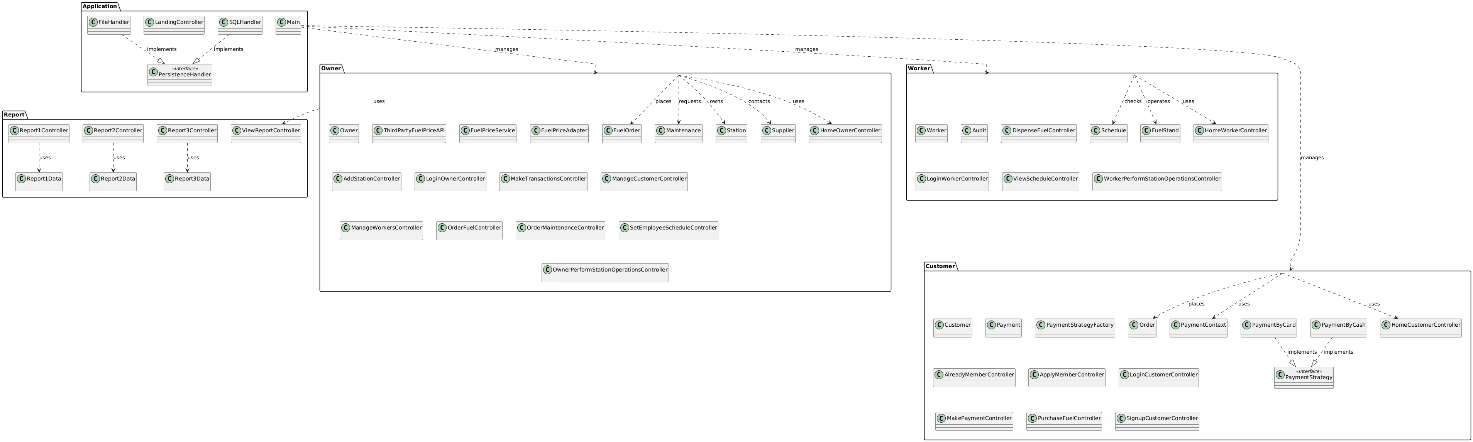
A diagram of a computer

Description automatically generated with medium confidence

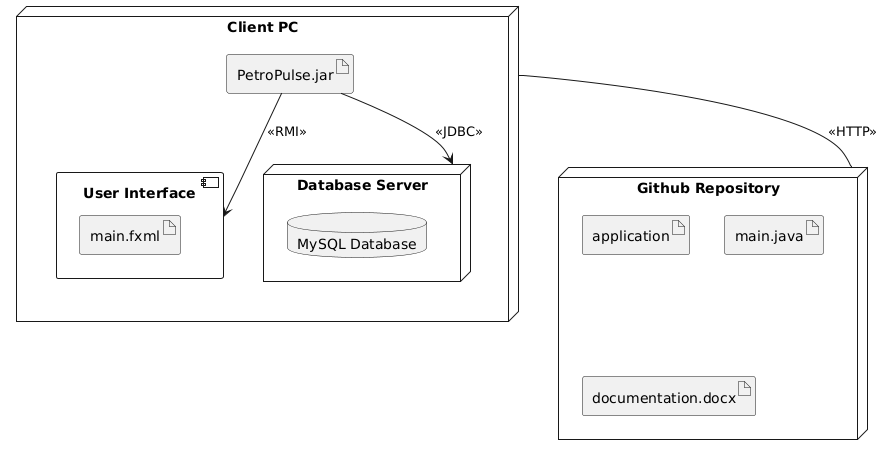
# **COMPONENT DIAGRAM**



# **PACKAGE DIAGRAM**



# **DEPLOYMENT DIAGRAM**



# **DESIGN PATTERNS**

## **GRASP Patterns**

The PetroPulse project leverages GRASP (General Responsibility Assignment Software Patterns) principles to ensure clear responsibility distribution and high cohesion across classes. Below is an analysis of the GRASP patterns implemented in the system:

**1. Information Expert**

This pattern is utilized to assign responsibilities to the class with the most information needed to fulfill them.

* **Implementation in PetroPulse:**
  + **Station Class:** Handles inventory, earnings, and workers for the station, as it encapsulates all relevant station information.
  + **Customer Class:** Manages customer-specific data like loyalty points and payment methods, centralizing responsibility for customer operations.

**2. Creator**

This pattern assigns the responsibility of creating instances to the class that aggregates or closely uses them.

* **Implementation in PetroPulse:**
  + **Station Class:** Creates and maintains instances of FuelStand, Worker, and Schedule because they are intrinsic components of a station.

**3. Controller**

This pattern assigns the responsibility of handling system events to a controller class.

* **Implementation Examples in PetroPulse:**
  + **HomeOwnerController:** Handles requests from station owners for redirection to different scenes for different use cases.
  + **HomeCustomerController:** Handles requests from station customers for redirection to different scenes for different use cases.

**4. High Cohesion**

This pattern ensures that classes maintain high functional coherence and do not assume excessive responsibilities.

* **Implementation in PetroPulse:**
  + **Schedule Class:** Exclusively manages schedules, ensuring focused responsibility.
  + **Transaction Class:** Handles payment-related functionality, maintaining clear boundaries.

**5. Low Coupling**

This pattern minimizes dependencies between classes to increase reusability and scalability.

* **Implementation in PetroPulse:**
  + **PersistenceHandler:** Abstracts database operations, reducing direct coupling between business logic and database access.
  + **PaymentStrategy Subclasses:** Decouples PaymentByCard and PaymentByCash, ensuring flexibility and low interdependence.

**6. Polymorphism**

This pattern assigns responsibility to the classes through polymorphic behavior.

* **Implementation in PetroPulse:**
  + **PaymentStrategy Class and Subclasses:** Uses polymorphism to handle different payment methods (PaymentByCard, PaymentByCash) uniformly.

## **Gang of Four Patterns**

**1. Singleton**

Ensures that only one instance of a class exists and provides a global access point.

* **Implementation in PetroPulse:**
  + **PersistenceHandler Subclasses:**
    - Ensures a single instance of the SQLHandler and FileHandler is shared across the application to avoid redundant connections and ensure resource efficiency.
    - Example: A getInstance() method ensures only one active database connection exists.

**2. Factory Method**

Defines an interface for creating objects but allows subclasses to decide which class to instantiate.

* **Implementation in PetroPulse:**
  + **PaymentMethod Creation:**
    - A PaymentStrategyFactory could generate instances of one of the Payment Method based on user input or system configurations.
    - Helps decouple the creation logic from the client code that uses the orders.

**3. Strategy**

Encapsulates a family of algorithms, allowing them to be selected at runtime.

* **Implementation in PetroPulse:**
  + **Payment Strategy:**
    - The system provides multiple payment strategies:
      * PaymentByCard and PaymentByCash subclasses encapsulate different payment processing logic.
      * These can be dynamically chosen at runtime depending on the customer's preference.
    - The strategy pattern is evident in the abstraction of payment handling and the delegation to specific payment classes.

**4. Adapter**

Allows incompatible interfaces to work together by providing a bridge.

* **Implementation in PetroPulse:**
  + **Fuel Price Adapter:**
    - The Fuel Price Adapter class implements Fuel Price Service and composes the Third Party Fuel Price API hence connecting both classes and letting the functionality of API be used by Fuel Price Service.