**Software Requirements**

**Specification**

**for**

**PetroPulse**

**Petrol Station**

**Management System**

**Prepared by**

***Anas Bin Rashid 22I-0907***

***Adan Malik 22I-1000***

***Hasnain Akhtar 22I-1241***

**March 21, 2025**

**Table of Contents**

[**Introduction** 3](#_Toc194770380)

[**Purpose** 3](#_Toc194770381)

[**Document Conventions** 3](#_Toc194770382)

[**Intended Audience and Reading Suggestions** 3](#_Toc194770383)

[**Product Scope** 3](#_Toc194770384)

[**References** 4](#_Toc194770385)

[**Overall Description** 4](#_Toc194770386)

[**Product Perspective** 4](#_Toc194770387)

[**Product Functions** 4](#_Toc194770388)

[**User Classes and Characteristics** 4](#_Toc194770389)

[**Operating Environment** 5](#_Toc194770390)

[**Design and Implementation Constraints** 5](#_Toc194770391)

[**User Documentation** 5](#_Toc194770392)

[**Assumptions and Dependencies** 5](#_Toc194770393)

[**External Interface Requirements** 6](#_Toc194770394)

[**User Interfaces** 6](#_Toc194770395)

[**Hardware Interfaces** 6](#_Toc194770396)

[**Software Interfaces** 6](#_Toc194770397)

[**Communications Interfaces** 6](#_Toc194770398)

[**System Features** 6](#_Toc194770399)

[**Other Non-Functional Requirements** 22](#_Toc194770400)

[**Design** 24](#_Toc194770401)

[**Activity Diagrams** 24](#_Toc194770402)

[**Use Case Diagram** 27](#_Toc194770403)

[**Sequence Diagrams** 28](#_Toc194770404)

[**Class Diagram** 35](#_Toc194770405)

[**Current Sprint Backlog** 35](#_Toc194770406)

[**Next Sprint Backlog** 44](#_Toc194770407)

[**Supporting Information** 47](#_Toc194770408)

[**GitHub Repository** 47](#_Toc194770409)

[**Trello Snapshots** 49](#_Toc194770410)

# **Introduction**

## **Purpose**

PetroPulse is a web-based Petrol Station Management System, designed to streamline fuel sales, station operations, and transaction processing. The system enhances efficiency by digitizing workflows, reducing manual errors, and ensuring real-time reporting.

## **Document Conventions**

* **Bold** text represents key terms.
* *Italicized text* is used for emphasis.
* Monospace font is used for code snippets and database schema elements.
* Hierarchical numbering is used for structured organization.
* Functional Requirements (FR) and Non-Functional Requirements (NFR) are prefixed accordingly.

## **Intended Audience and Reading Suggestions**

This document is intended for:

* **Developers**: To understand system architecture and requirements.
* **Project Managers**: To track progress and feature development.
* **QA Team**: For testing and validation of functionalities.
* **End Users** (Petrol Station Owners & Staff): To understand the capabilities and benefits of the system.
* **Investors and Stakeholders**: To evaluate the feasibility and impact of the system.

## **Product Scope**

PetroPulse will provide the following core functionalities:

* **Fuel Purchase & Payments**: Customers can buy fuel and pay via multiple methods.
* **User Management**: Roles include station owners, workers, and customers.
* **Inventory Management**: Real-time fuel stock tracking and alerts.
* **Transaction Logging & Reporting**: Comprehensive data analysis for decision-making.
* **Cloud-Based Storage**: Data persistence using MongoDB and Firebase.

By transitioning to a web-based solution using MERN Stack (MongoDB, Express.js, React, Node.js), PetroPulse will ensure seamless operations and accessibility across multiple devices.

## **References**

* IEEE Standard 830-1998 for Software Requirements Specification.
* MongoDB Documentation for NoSQL Data Modeling.
* ReactJS and Node.js Official Documentation.

# **Overall Description**

## **Product Perspective**

PetroPulse is a web-based SaaS (Software as a Service) application. It consists of a frontend (React.js), a backend (Node.js/Express.js), and a NoSQL database (MongoDB) for data persistence. The system will also integrate with cloud services for scalability.

## **Product Functions**

The primary functions include:

* **Fuel Management**: Sales tracking, pricing updates, and inventory monitoring.
* **User Roles & Permissions**: Owner, worker, and customer access levels.
* **Payment Processing**: Secure transactions via card, cash, or online payments.
* **Data** **Analytics**: Reports on sales, transactions, and fuel stock levels.
* **Authentication & Security**: JWT-based authentication and role-based access control.

## **User Classes and Characteristics**

1. **Station Owner**: Monitors financial reports and manages operations.
2. **Station Workers**: Handles fuel sales and transactions.
3. **Customers**: Purchases fuel and views loyalty points.

## **Operating Environment**

* **Frontend**: React.js with Material UI for responsive design.
* **Backend**: Node.js with Express.js for REST API handling.
* **Database**: MongoDB for NoSQL storage.
* **Cloud** **Hosting**: Vercel for server deployment.
* **Security**: HTTPS encryption, JWT authentication.

## **Design and Implementation Constraints**

* Must support MongoDB for data storage.
* User authentication should be implemented using hashed passwords.
* System should operate with minimal latency (< 2s response time for transactions).

## **User Documentation**

* User manual for system operations.
* Installation and setup guide.
* API documentation for third-party integrations.

## **Assumptions and Dependencies**

* The station will have a stable internet connection for cloud-based reporting.
* All users will have basic familiarity with computers.
* Payment gateway integration requires compliance with PCI-DSS standards.
* System performance will be optimized for high transaction loads.

# **External Interface Requirements**

## **User Interfaces**

* **Web Dashboard** (React.js): Intuitive UI for fuel sales, inventory, and analytics.
* **Mobile Optimization**: Responsive design for seamless mobile access.

## **Hardware Interfaces**

* **POS System Integration**: Direct API connectivity with POS terminals.
* **IoT Support**: Future capability for fuel dispenser automation.

## **Software Interfaces**

* **MongoDB Atlas**: Cloud-based database service for high availability.
* **Firebase Authentication**: Optional Google Sign-In integration.

## **Communications Interfaces**

* **RESTful API (Express.js)**: Communication between frontend and backend.
* **WebSockets**: For real-time notifications and transaction updates.

# **System Features**

**USER STORY 1: User Registration**

**Description and Priority**  
The User Registration feature enables new users to create an account on the system by submitting required personal information. This allows access to personalized services, purchase tracking, and loyalty features.  
**Priority:** High  
• Benefit: 9 (Facilitates user onboarding and personalized service)  
• Penalty: 8 (Users cannot use the system without registration)  
• Cost: 4 (Standard input forms and validation logic)  
• Risk: 3 (Low risk; basic feature in most applications)

**Stimulus/Response Sequences**

1. **User Action:** A new user clicks the "Register" button and enters their details.  
   o **System Response:** Validates and stores user data in the database.
2. **User Action:** A user attempts to register with an existing email.  
   o **System Response:** Displays an error and prevents duplication.
3. **User Action:** A user completes registration successfully.  
   o **System Response:** Sends a welcome or confirmation message.

**Functional Requirements**  
REQ-1: The system shall allow new users to register by entering name, email, phone number, and password.  
REQ-2: The system shall validate user input before submission.  
REQ-3: The system shall prevent duplicate registration using the same email.  
REQ-4: Upon successful registration, the system shall store user data and send a confirmation message.

**USER STORY 2: User Login**

**Description and Priority**  
The User Login feature allows registered users to securely access the system using their credentials. It provides authenticated access to dashboards and personalized data.  
**Priority:** High  
• Benefit: 9 (Enables secure, personalized access)  
• Penalty: 7 (Without login, users can't use the platform)  
• Cost: 3 (Basic credential check and session management)  
• Risk: 4 (Moderate risk due to potential security concerns)

**Stimulus/Response Sequences**

1. **User Action:** A registered user enters valid login credentials.  
   o **System Response:** Authenticates and redirects to the user dashboard.
2. **User Action:** A user enters incorrect credentials.  
   o **System Response:** Displays an error message and denies access.
3. **User Action:** An unregistered user attempts to log in.  
   o **System Response:** Informs the user to register first.

**Functional Requirements**  
REQ-5: The system shall allow users to log in using email and password.  
REQ-6: The system shall authenticate credentials against the user database.  
REQ-7: The system shall redirect authenticated users to the appropriate dashboard.  
REQ-8: The system shall show an error message if login credentials are invalid.

**USER STORY 3: Password Reset**

**Description and Priority**  
The Password Reset feature allows users who have forgotten their passwords to recover their accounts by verifying their email and setting a new password.  
**Priority:** Medium  
• Benefit: 8 (Improves user accessibility and reduces account loss)  
• Penalty: 6 (Users may abandon service if unable to access accounts)  
• Cost: 4 (Email verification and form setup)  
• Risk: 5 (Security must be maintained during the reset process)

**Stimulus/Response Sequences**

1. **User Action:** A user clicks on "Forgot Password."  
   o **System Response:** Prompts for email and sends a reset link.
2. **User Action:** A user clicks the reset link and enters a new password.  
   o **System Response:** Updates the password and confirms the change.
3. **User Action:** A user enters an unregistered email.  
   o **System Response:** Displays an error and denies reset request.

**Functional Requirements**  
REQ-9: The system shall provide a "Forgot Password" option on the login page.  
REQ-10: The system shall send a reset link to the registered email.  
REQ-11: The system shall allow users to create a new password using the link.  
REQ-12: The system shall validate and securely update the password.

**USER STORY 4: Add Fuel Station**

**Description and Priority**  
The Add Fuel Station feature allows owners to add new stations to the system. This enables multi-station management and scalability of operations across various locations.  
**Priority:** High  
• Benefit: 9 (Allows owners to scale their business and manage stations efficiently)  
• Penalty: 7 (Lack of this feature restricts expansion)  
• Cost: 5 (Requires form handling and station database integration)  
• Risk: 4 (Low risk if validation and access control are handled correctly)

**Stimulus/Response Sequences**

1. **User Action:** The owner fills out the station details form and clicks "Submit."  
   o **System Response:** Registers the new station and updates the station list.
2. **User Action:** The owner provides incomplete or invalid data.  
   o **System Response:** Displays validation errors and prevents submission.
3. **User Action:** The owner successfully adds a station.  
   o **System Response:** Confirms station creation and refreshes dashboard.

**Functional Requirements**  
REQ-13: The system shall allow logged-in owners to add new fuel stations.  
REQ-14: The system shall validate station data before saving.  
REQ-15: The system shall store new station data in the database.  
REQ-16: The system shall update the owner’s station list upon addition.

**USER STORY 5: View Fuel Stations**

**Description and Priority**  
The View Fuel Stations feature enables owners to view all stations they manage in one centralized list. This supports better oversight and operational monitoring.  
**Priority:** High  
• Benefit: 8 (Improves visibility and ease of station management)  
• Penalty: 6 (Owners may struggle to track operations without access)  
• Cost: 3 (Read operation with basic UI)  
• Risk: 3 (Minimal risk)

**Stimulus/Response Sequences**

1. **User Action:** The owner navigates to the "Manage Stations" section.  
   o **System Response:** Displays a list of all owned fuel stations.
2. **User Action:** Owner refreshes the page.  
   o **System Response:** Ensures updated station information is shown.
3. **User Action:** Owner views station list after adding or deleting a station.  
   o **System Response:** Reflects the updated list dynamically.

**Functional Requirements**  
REQ-17: The system shall display a list of stations owned by the logged-in user.  
REQ-18: The system shall fetch station data from the database.  
REQ-19: The system shall dynamically reflect changes in the station list.

**USER STORY 6: Update Fuel Station Details**

**Description and Priority**  
This feature allows owners to update their fuel station’s details, ensuring that operational and contact information remains accurate and up to date.  
**Priority:** Medium  
• Benefit: 7 (Helps maintain data accuracy and consistency)  
• Penalty: 5 (Inaccurate data could mislead staff or customers)  
• Cost: 4 (Form editing and update logic)  
• Risk: 4 (Moderate—requires proper access and validation)

**Stimulus/Response Sequences**

1. **User Action:** The owner edits station details and clicks "Save."  
   o **System Response:** Updates the record and displays confirmation.
2. **User Action:** The owner submits invalid or incomplete data.  
   o **System Response:** Rejects the update and prompts for corrections.
3. **User Action:** The owner updates station details multiple times.  
   o **System Response:** Reflects the latest saved values.

**Functional Requirements**  
REQ-20: The system shall allow owners to edit and update station details.  
REQ-21: The system shall validate input before saving updates.  
REQ-22: The system shall reflect updated station data in the system.

**USER STORY 7: Remove Fuel Station**

**Description and Priority**  
This feature allows owners to delete fuel stations they no longer manage due to closure or sale. It helps keep the station list relevant and up to date.  
**Priority:** Medium  
• Benefit: 6 (Keeps data clean and relevant)  
• Penalty: 5 (Irrelevant data may confuse the owner or clutter the UI)  
• Cost: 3 (Delete logic and confirmation modal)  
• Risk: 4 (Accidental deletion risk)

**Stimulus/Response Sequences**

1. **User Action:** The owner clicks the "Delete" button and confirms the action.  
   o **System Response:** Removes the station from the database and updates the list.
2. **User Action:** The owner cancels the delete confirmation.  
   o **System Response:** No action is taken; the station remains.
3. **User Action:** The owner deletes multiple stations.  
   o **System Response:** Updates the list after each deletion.

**Functional Requirements**  
REQ-23: The system shall allow owners to delete fuel stations they own.  
REQ-24: The system shall prompt a confirmation before deletion.  
REQ-25: The system shall remove the station from the database upon confirmation.  
REQ-26: The system shall refresh the station list after deletion.

**USER STORY 8: Register Workers**

**Description and Priority**  
The Register Workers feature allows station owners to add staff members responsible for daily fuel station operations, improving workflow delegation and access control.  
**Priority:** High  
• Benefit: 8 (Improves staffing and task allocation)  
• Penalty: 7 (Operations cannot proceed without staff accounts)  
• Cost: 4 (Form handling and association with station)  
• Risk: 3 (Low—basic functionality)

**Stimulus/Response Sequences**

1. **User Action:** The owner fills in worker details and clicks "Register."  
   o **System Response:** Validates and saves the worker’s data.
2. **User Action:** The owner submits a form with missing fields.  
   o **System Response:** Displays error messages and prevents submission.
3. **User Action:** A worker is successfully added.  
   o **System Response:** Confirms registration and displays worker in the list.

**Functional Requirements**  
REQ-27: The system shall allow station owners to register workers by providing name, contact info, and credentials.  
REQ-28: The system shall validate input before submission.  
REQ-29: The system shall store the new worker in the database and link them to the station.  
REQ-30: The system shall display a confirmation message after successful registration.

**USER STORY 9: Assign Workers to Stations**

**Description and Priority**  
The Assign Workers to Stations feature enables owners to allocate workers to specific fuel stations, ensuring clear task delegation and accountability.  
**Priority:** Medium  
• Benefit: 7 (Improves operational clarity and workforce organization)  
• Penalty: 5 (Workers may be unassigned or misallocated)  
• Cost: 4 (Requires user-station linking logic)  
• Risk: 4 (Moderate risk of incorrect assignment if not validated)

**Stimulus/Response Sequences**

1. **User Action:** The owner selects a worker and assigns them to a station.  
   o **System Response:** Updates the worker's assigned station in the system.
2. **User Action:** The owner changes a worker’s assignment.  
   o **System Response:** Reflects the new station assignment immediately.
3. **User Action:** The owner tries to assign a worker without selecting a station.  
   o **System Response:** Displays an error and prevents submission.

**Functional Requirements**  
REQ-31: The system shall allow owners to assign workers to specific fuel stations.  
REQ-32: The system shall validate that a valid station is selected.  
REQ-33: The system shall update the worker’s record with the assigned station.  
REQ-34: The system shall reflect station-worker relationships in real time.

**USER STORY 10: Worker Login**

**Description and Priority**  
The Worker Login feature allows registered workers to securely access the system to manage daily station activities.  
**Priority:** High  
• Benefit: 8 (Enables secure access to station operations)  
• Penalty: 7 (Workers cannot operate without access)  
• Cost: 3 (Standard login functionality)  
• Risk: 3 (Low—well-supported functionality)

**Stimulus/Response Sequences**

1. **User Action:** The worker enters valid credentials and clicks "Login."  
   o **System Response:** Authenticates and redirects to the worker dashboard.
2. **User Action:** The worker enters incorrect credentials.  
   o **System Response:** Displays an error message and denies access.
3. **User Action:** A session times out due to inactivity.  
   o **System Response:** Logs out the worker for security.

**Functional Requirements**  
REQ-35: The system shall allow registered workers to log in using credentials.  
REQ-36: The system shall validate login information securely.  
REQ-37: The system shall redirect authenticated users to their respective dashboard.  
REQ-38: The system shall log failed login attempts for auditing.

**USER STORY 11: Record Fuel Sales**

**Description and Priority**  
The Record Fuel Sales feature allows workers to log transactions, ensuring accurate sales tracking and real-time revenue updates.  
**Priority:** High  
• Benefit: 9 (Crucial for financial tracking and reporting)  
• Penalty: 8 (Lack of records leads to financial ambiguity)  
• Cost: 5 (Involves database transactions and fuel deduction)  
• Risk: 5 (Moderate—requires accurate input and consistency)

**Stimulus/Response Sequences**

1. **User Action:** The worker enters sale details and submits the form.  
   o **System Response:** Saves the sale and deducts fuel from inventory.
2. **User Action:** The worker submits an invalid sale entry.  
   o **System Response:** Displays validation errors.
3. **User Action:** Sales are recorded in quick succession.  
   o **System Response:** Processes each transaction and updates inventory in real time.

**Functional Requirements**  
REQ-39: The system shall allow workers to input and submit fuel sale transactions.  
REQ-40: The system shall validate the sale data before processing.  
REQ-41: The system shall deduct sold fuel from the station’s inventory.  
REQ-42: The system shall timestamp and store transaction records.

**USER STORY 12: Generate Sales Reports**

**Description and Priority**  
The Generate Sales Reports feature enables owners to analyze transaction data over time, supporting business decisions and financial planning.  
**Priority:** High  
• Benefit: 9 (Helps owners make informed business decisions)  
• Penalty: 7 (No insight into performance metrics without reports)  
• Cost: 6 (Requires aggregation, filtering, and formatting logic)  
• Risk: 4 (Data integrity and filtering must be reliable)

**Stimulus/Response Sequences**

1. **User Action:** The owner selects a date range and clicks "Generate Report."  
   o **System Response:** Displays sales figures and metrics for that period.
2. **User Action:** The owner views reports for different stations.  
   o **System Response:** Filters and shows station-specific sales data.
3. **User Action:** The owner exports the report.  
   o **System Response:** Provides a downloadable file (e.g., PDF/CSV).

**Functional Requirements**  
REQ-43: The system shall allow owners to generate sales reports by date range.  
REQ-44: The system shall support filtering by station and fuel type.  
REQ-45: The system shall display revenue, quantity sold, and transactions.  
REQ-46: The system shall allow report export in standard formats.

**USER STORY 13: Track Fuel Inventory**

**Description and Priority**  
The Track Fuel Inventory feature gives owners visibility into the current fuel levels at each station, helping ensure continuous supply.  
**Priority:** High  
• Benefit: 8 (Enables proactive management of fuel supply)  
• Penalty: 7 (Unnoticed shortages can disrupt sales)  
• Cost: 5 (Requires integration with sales and refills)  
• Risk: 4 (Risk if levels are not updated in real time)

**Stimulus/Response Sequences**

1. **User Action:** The owner checks the fuel inventory from the dashboard.  
   o **System Response:** Displays real-time fuel levels for each station.
2. **User Action:** A worker records a sale.  
   o **System Response:** Deducts the fuel from the current inventory.
3. **User Action:** Fuel is restocked.  
   o **System Response:** Updates inventory with the refilled amount.

**Functional Requirements**  
REQ-47: The system shall display fuel levels for each station in real time.  
REQ-48: The system shall update inventory upon sales and restocking.  
REQ-49: The system shall alert users when data is stale or unavailable.  
REQ-50: The system shall log inventory changes for future reference.

**USER STORY 14: Alert for Low Fuel Levels**

**Description and Priority**  
This feature notifies owners when fuel levels drop below a predefined threshold, helping them avoid stockouts and schedule refills in time.  
**Priority:** High  
• Benefit: 9 (Prevents fuel shortages and customer dissatisfaction)  
• Penalty: 8 (Fuel outages can cause loss of revenue and reputation)  
• Cost: 4 (Trigger logic and notification setup)  
• Risk: 3 (Low risk if thresholds are properly configured)

**Stimulus/Response Sequences**

1. **User Action:** Fuel level drops below threshold.  
   o **System Response:** Sends a low fuel alert to the station owner.
2. **User Action:** Owner responds to alert.  
   o **System Response:** Provides fuel ordering options.
3. **User Action:** Fuel level returns to normal.  
   o **System Response:** Cancels or resets alert status.

**Functional Requirements**  
REQ-51: The system shall monitor fuel levels against predefined thresholds.  
REQ-52: The system shall trigger alerts when levels fall below the threshold.  
REQ-53: The system shall notify the respective station owner.  
REQ-54: The system shall allow owners to configure alert thresholds.

**USER STORY 15: Order Fuel Supply**

**Description and Priority**  
The Order Fuel Supply feature allows station owners to place refill orders when fuel levels are low, ensuring uninterrupted availability at the station.  
**Priority:** High  
• Benefit: 9 (Maintains fuel availability and supports smooth operations)  
• Penalty: 8 (Delays in refueling lead to sales loss)  
• Cost: 6 (Requires supplier integration and order management logic)  
• Risk: 5 (Incorrect orders or delays can disrupt refueling)

**Stimulus/Response Sequences**

1. **User Action:** The owner navigates to the fuel supply page and places an order.  
   o **System Response:** Processes and confirms the order.
2. **User Action:** Owner specifies the fuel type and quantity.  
   o **System Response:** Validates and routes the order to the supplier.
3. **User Action:** Owner checks order status.  
   o **System Response:** Displays the status and estimated delivery.

**Functional Requirements**  
REQ-55: The system shall allow owners to place fuel refill orders.  
REQ-56: The system shall validate the fuel type and quantity before submission.  
REQ-57: The system shall log the order and notify the supplier.  
REQ-58: The system shall track and display the status of each order.

**USER STORY 16: View Loyalty Points**

**Description and Priority**  
The View Loyalty Points feature allows customers to check their reward balance for tracking and potential redemption.  
**Priority: Medium**  
• Benefit: 6 (Enhances customer engagement)  
• Penalty: 4 (Customers may feel disconnected from the reward program)  
• Cost: 3 (Simple query from database)  
• Risk: 2 (Low risk feature)

**Stimulus/Response Sequences**  
**User Action:** The customer logs in and navigates to the Loyalty Points page.  
o **System Response:** Displays the current loyalty point balance.

**Functional Requirements**  
REQ-59: The system shall allow customers to access their loyalty point balance.  
REQ-60: The system shall fetch and display the correct point total from the database.  
REQ-61: The system shall restrict loyalty point visibility to the logged-in customer.

**USER STORY 17: Redeem Loyalty Points**

**Description and Priority**  
The Redeem Loyalty Points feature enables customers to use earned points for discounts during fuel purchases.  
**Priority: Medium**  
• Benefit: 7 (Incentivizes frequent usage)  
• Penalty: 5 (Customers may feel unrewarded)  
• Cost: 4 (Requires integration with payment and loyalty system)  
• Risk: 3 (Misuse or incorrect point deduction)

**Stimulus/Response Sequences**  
**User Action:** The customer redeems loyalty points at checkout.  
o **System Response:** Applies the discount and deducts points from their account.

**Functional Requirements**  
REQ-62: The system shall allow redemption of loyalty points for purchases.  
REQ-63: The system shall validate sufficient point balance before redemption.  
REQ-64: The system shall update the loyalty point balance after redemption.  
REQ-65: The system shall apply the correct discount based on redeemed points.

**USER STORY 18: Customer Fuel Purchase**

**Description and Priority**  
The Customer Fuel Purchase feature enables users to refuel by selecting options and paying at the station.  
**Priority: High**  
• Benefit: 9 (Core revenue-generating function)  
• Penalty: 8 (Lost sales if unavailable)  
• Cost: 5 (Payment and inventory logic)  
• Risk: 4 (Inventory mismatch or failed transactions)

**Stimulus/Response Sequences**  
**User Action:** Customer selects fuel type, quantity, and payment method.  
o **System Response:** Processes the payment and updates inventory accordingly.

**Functional Requirements**  
REQ-66: The system shall allow customers to purchase fuel.  
REQ-67: The system shall deduct the purchased quantity from station inventory.  
REQ-68: The system shall handle different payment methods.  
REQ-69: The system shall generate a transaction record for each purchase.

**USER STORY 19: Manage Pricing for Fuel Types**

**Description and Priority**  
The Manage Pricing feature enables station owners to update fuel prices based on market trends.  
**Priority: High**  
• Benefit: 8 (Supports profitability)  
• Penalty: 6 (Leads to outdated or incorrect pricing)  
• Cost: 4 (Database update and UI refresh)  
• Risk: 3 (Incorrect input may impact sales)

**Stimulus/Response Sequences**  
**User Action:** Owner updates fuel prices on the pricing page.  
o **System Response:** Reflects updated prices in real time.

**Functional Requirements**  
REQ-70: The system shall allow fuel price updates by the owner.  
REQ-71: The system shall validate new price inputs.  
REQ-72: The system shall update all relevant records and displays.  
REQ-73: The system shall maintain a history of price changes for auditing.

**USER STORY 20: Manage Maintenance Requests**

**Description and Priority**  
The Manage Maintenance feature allows owners to log and monitor station issues for timely resolution.  
**Priority: Medium**  
• Benefit: 7 (Ensures operational efficiency)  
• Penalty: 5 (Neglected maintenance could halt services)  
• Cost: 4 (Form submission and database entry)  
• Risk: 3 (Delayed responses or ignored issues)

**Stimulus/Response Sequences**  
**User Action:** Owner submits a maintenance request.  
o **System Response:** Logs and tracks the maintenance issue.

**Functional Requirements**  
REQ-74: The system shall allow submission of maintenance requests.  
REQ-75: The system shall record the request details with a timestamp.  
REQ-76: The system shall display the request status to the owner.  
REQ-77: The system shall enable filtering and tracking of open/closed requests.

**USER STORY 21: View Transaction History**

**Description and Priority**  
The View Transaction History feature lets customers review their fuel purchase records.  
**Priority: Medium**  
• Benefit: 6 (Improves transparency and trust)  
• Penalty: 3 (Minor if unavailable)  
• Cost: 3 (Simple retrieval from database)  
• Risk: 2 (Data exposure if not filtered correctly)

**Stimulus/Response Sequences**  
**User Action:** Customer accesses the transactions section after login.  
o **System Response:** Displays all past fuel purchases.

**Functional Requirements**  
REQ-78: The system shall display the transaction history for customers.  
REQ-79: The system shall show the date, amount, station, and payment type.  
REQ-80: The system shall restrict data to the logged-in customer.  
REQ-81: The system shall allow filtering by date or station.

**USER STORY 22: Payment via Cash**

**Description and Priority**  
The Payment via Cash feature enables customers to complete fuel purchases using cash.  
**Priority: High**  
• Benefit: 7 (Supports offline customers)  
• Penalty: 6 (Limits customer options if unavailable)  
• Cost: 3 (Simple record update)  
• Risk: 2 (Minimal, as no online processing is involved)

**Stimulus/Response Sequences**  
**User Action:** Customer selects cash during payment.  
o **System Response:** Registers the cash payment and updates records.

**Functional Requirements**  
REQ-82: The system shall allow cash as a valid payment option.  
REQ-83: The system shall update transaction records accordingly.  
REQ-84: The system shall reflect inventory changes post-payment.  
REQ-85: The system shall issue a receipt for the transaction.

**USER STORY 23: Payment via Card**

**Description and Priority**  
The Payment via Card feature enables smooth and quick payments for customers using credit/debit cards.  
**Priority: High**  
• Benefit: 8 (Faster, modern payment method)  
• Penalty: 6 (Could drive away tech-savvy users)  
• Cost: 5 (Requires payment gateway integration)  
• Risk: 4 (Transaction failure, card rejection)

**Stimulus/Response Sequences**  
**User Action:** Customer selects card payment at checkout.  
o **System Response:** Processes the card and updates payment records.

**Functional Requirements**  
REQ-86: The system shall accept card payments for purchases.  
REQ-87: The system shall validate card information.  
REQ-88: The system shall confirm successful transaction before updating records.  
REQ-89: The system shall maintain payment security and encrypt data.

**USER STORY 24: Manage Supplier Information**

**Description and Priority**  
The Manage Supplier Information feature allows owners to add, update, or remove fuel suppliers as needed.  
**Priority: Medium**  
• Benefit: 7 (Maintains supplier control)  
• Penalty: 4 (Mismanaged supplier data can cause delays)  
• Cost: 4 (Simple CRUD operations)  
• Risk: 3 (Incorrect data could impact orders)

**Stimulus/Response Sequences**  
**User Action:** Owner adds, edits, or deletes a supplier entry.  
o **System Response:** Updates the supplier database accordingly.

**Functional Requirements**  
REQ-90: The system shall allow management of supplier details.  
REQ-91: The system shall validate supplier inputs on addition/editing.  
REQ-92: The system shall prevent deletion if supplier is linked to active orders.  
REQ-93: The system shall log all supplier modifications for auditing.

**USER STORY 25: Generate Maintenance Reports**

**Description and Priority**  
The Generate Maintenance Reports feature enables owners to review past and upcoming station repairs.  
**Priority: Medium**  
• Benefit: 6 (Improves maintenance scheduling)  
• Penalty: 4 (Overlooked issues may worsen)  
• Cost: 3 (Report generation logic)  
• Risk: 2 (Low)

**Stimulus/Response Sequences**  
**User Action:** Owner selects maintenance reports on the reports page.  
o **System Response:** Displays a report of all maintenance tasks.

**Functional Requirements**  
REQ-94: The system shall generate maintenance history reports.  
REQ-95: The system shall include task description, date, and status in the report.  
REQ-96: The system shall allow filtering by station and time period.  
REQ-97: The system shall export maintenance reports as PDF or CSV.

**USER STORY 26: Employee Attendance Tracking**

**Description and Priority**  
The Employee Attendance Tracking feature enables station owners to monitor workers’ daily attendance and shift hours, ensuring accountability and effective workforce management.  
**Priority:** Medium

* **Benefit:** 7 (Improves employee accountability and payroll accuracy)
* **Penalty:** 6 (Lack of tracking may lead to time misuse and wage disputes)
* **Cost:** 5 (Requires UI support and backend for time tracking)
* **Risk:** 4 (Incorrect time logs can affect payroll)

**Stimulus/Response Sequences**

* **User Action:** A worker logs into the system and marks attendance for their shift.
  + **System Response:** The system records the timestamp and shift hours.
* **User Action:** Owner checks attendance logs.
  + **System Response:** Displays detailed records for each worker.

**Functional Requirements**

* **REQ-77:** The system shall allow workers to mark attendance for their shifts.
* **REQ-78:** The system shall record login time and calculate working hours.
* **REQ-79:** The system shall provide attendance history to the owner.
* **REQ-80:** The system shall restrict attendance marking to valid user sessions.

**USER STORY 27: Secure Access**

**Description and Priority**  
The Secure Access feature ensures that only authorized users can access system settings and configurations, protecting sensitive controls from misuse.  
**Priority:** High

* **Benefit:** 9 (Ensures system integrity and security)
* **Penalty:** 9 (Unauthorized access can compromise operations)
* **Cost:** 4 (Basic credential management logic needed)
* **Risk:** 6 (Security breaches can disrupt services)

**Stimulus/Response Sequences**

* **User Action:** A user enters admin credentials on the login page.
  + **System Response:** Validates credentials and grants access to the admin panel.
* **User Action:** Unauthorized user attempts access.
  + **System Response:** Denies access and displays an error message.

**Functional Requirements**

* **REQ-81:** The system shall require admin authentication to access the configuration panel.
* **REQ-82:** The system shall validate credentials securely before granting access.
* **REQ-83:** The system shall deny access for invalid or non-admin users.
* **REQ-84:** The system shall log access attempts for security auditing.

**USER STORY 28: View Worker Profiles**

**Description and Priority**  
The View Worker Profiles feature enables station owners to review employee details and roles for effective management and operational planning.  
**Priority:** Medium

* **Benefit:** 7 (Facilitates staff supervision and role allocation)
* **Penalty:** 5 (Incomplete data visibility may hinder management)
* **Cost:** 4 (Requires data retrieval and display logic)
* **Risk:** 3 (Minimal, mostly UI-related)

**Stimulus/Response Sequences**

* **User Action:** Owner navigates to the "Manage Workers" page and selects a worker.
  + **System Response:** Displays the worker’s name, contact, assigned role, and station details.
* **User Action:** Owner switches between profiles.
  + **System Response:** Loads corresponding worker details instantly.

**Functional Requirements**

* **REQ-85:** The system shall allow owners to view individual worker profiles.
* **REQ-86:** The system shall display worker details including name, role, contact, and station.
* **REQ-87:** The system shall fetch data dynamically when a worker is selected.
* **REQ-88:** The system shall ensure only authorized users can access worker profiles.

# **Other Non-Functional Requirements**

**Performance Requirements**

The system must ensure high performance to handle peak loads efficiently. It should meet the following requirements:

* NFR-01: The system shall support up to 5,000 concurrent users without degradation in response time.
* NFR-02: The backend shall process at least 100 transactions per second to maintain operational efficiency.
* NFR-05: API response time shall be less than 1 second for critical operations such as payment processing and fuel dispensing.

**Safety Requirements**

The system should ensure that failures or unexpected conditions do not lead to significant financial or operational losses. While no direct physical hazards are associated with this system, it must provide:

* Automatic rollback mechanisms in case of failed transactions.
* Alerts for system administrators in case of unexpected downtime.
* Logging of all critical operations to facilitate incident investigation.

**Security Requirements**

Security is a top priority for handling sensitive customer and business data. The system shall adhere to industry security best practices, including:

* NFR-04: The system shall encrypt sensitive data using AES-256 encryption to protect user credentials, payment information, and transaction logs.
* Secure authentication mechanisms, including multi-factor authentication (MFA) for administrative access.
* Compliance with GDPR and PCI-DSS regulations for data protection and payment security.

**Software Quality Attributes**

To ensure a high-quality and maintainable system, the following attributes shall be prioritized:

* Availability: The system should be accessible 99.9% of the time (NFR-03) to ensure business continuity.
* Scalability: The system must be designed to scale horizontally to support more concurrent users as demand increases.
* Reliability: Automated error recovery mechanisms should ensure minimal disruption to users.
* Maintainability: Code should be modular and well-documented to allow future enhancements with minimal effort.

**Business Rules**

The system shall enforce the following business rules to maintain operational integrity:

* Only authorized administrators can add, update, or delete user accounts.
* Payment transactions must be validated and confirmed before fuel is dispensed.
* Users must have a valid account and authentication credentials before accessing purchase or loyalty features.
* System logs should be retained for a minimum of one year for auditing and compliance purposes.

# **Design**

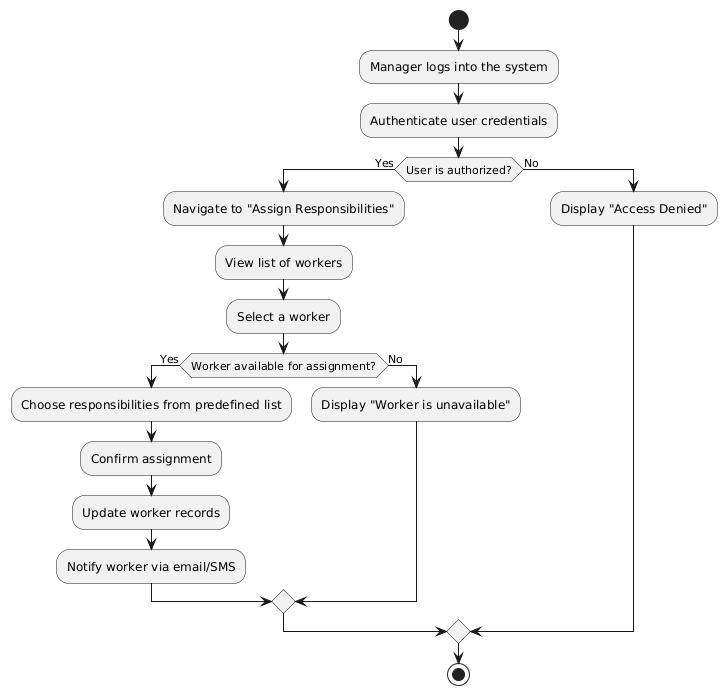
## **Activity Diagrams**

1. **Record Sales**

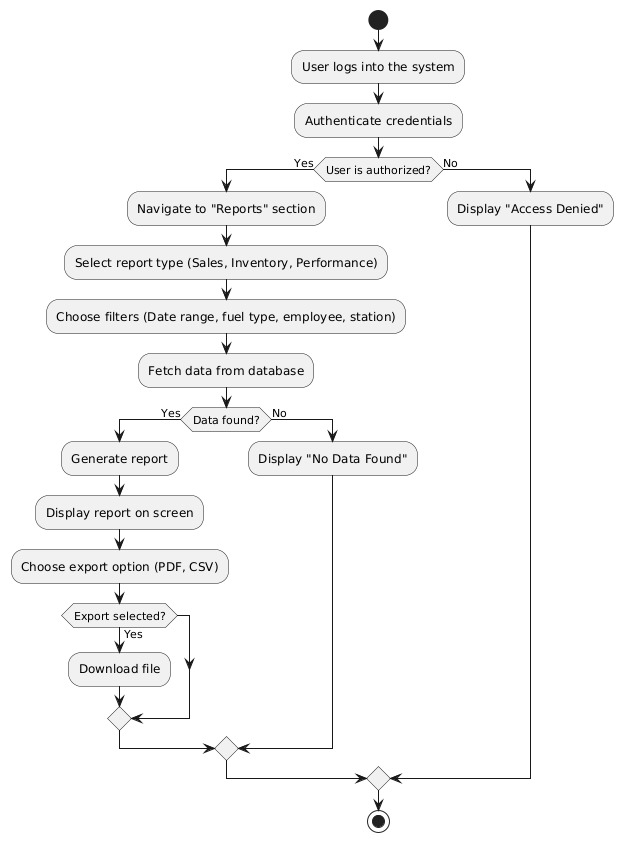
**A flowchart of a payment method

AI-generated content may be incorrect.**

1. **Assign Worker Schedule**

****

1. **View Report**

****

## **Use Case Diagram**

**A diagram of a network

AI-generated content may be incorrect.**

## **Sequence Diagrams**

1. **Purchase Fuel**

**System Sequence Diagram**

A diagram of a customer

Description automatically generated

**Sequence Diagrams**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

1. **View Reports**

**System Sequence Diagram**

A diagram of a system

Description automatically generated

**Sequence Diagrams**

**A diagram of a program

Description automatically generated**

**A diagram of a program

Description automatically generated**

**A diagram of a program

Description automatically generated**

1. **Manage Customers**

**System Sequence Diagram**

A diagram of a system

Description automatically generated

**Sequence Diagrams**

**A diagram of a software developer

Description automatically generated**

**A diagram of a program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

## **Class Diagram**

**A diagram of a computer

Description automatically generated with medium confidence**

# **Current Sprint Backlog**

**Objective**

The goal of this sprint is to develop the entire frontend of the PetroPulse system. The frontend will be built using React.js with a focus on providing an intuitive, responsive, and user-friendly experience for all stakeholders, including station owners, workers, and customers. This sprint aims to implement core UI components, integrate design principles, and ensure smooth navigation.

**Scope of Work**

The sprint will cover the following aspects of frontend development:

1. **Setup & Configuration**

* Initialize the React.js project with a clean folder structure.
* Install necessary dependencies such as React Router, Redux (if needed), Material UI, Axios, and other required libraries.

1. **Authentication & Authorization UI**

* Create the login and registration pages for different user roles (Station Owner, Worker, Customer).
* Implement form validation for input fields.
* Design a password reset feature UI (functionality to be implemented in the backend later).

1. **Dashboard UI for Different Users**
   * **Owner Dashboard:** 
     + Overview of earnings, inventory levels, and worker activity.
     + Navigation to detailed reports and analytics.
   * **Worker Dashboard:** 
     + Interface for handling transactions (fuel purchases).
     + Quick view of fuel availability and customer requests.
   * **Customer Interface:** 
     + View fuel prices, purchase fuel, and check loyalty points.
     + History of previous fuel purchases.
2. **Fuel Management UI**

* Display available fuel types and their pricing.
* User input for fuel quantity and payment method selection.
* Interactive elements for transaction processing.

1. **Inventory Management UI**

* Real-time stock tracking display.
* Alerts for low fuel levels.

1. **Transactions & Payment UI**

* Integration of UI elements for selecting payment methods (cash, card, online).
* Order confirmation and receipt generation UI.

1. **Reporting & Analytics UI**

* Interactive graphs and tables for sales, stock levels, and earnings.
* Exporting reports in PDF/Excel format (if feasible in frontend).

**Completed Work**

**User Stories Completed**

User Story 1 – User Registration

User Story 2 – User Login

User Story 3 – Password Reset

User Story 5 – View Fuel Stations

User Story 10 – Worker Login

User Story 16 – View Loyalty Points

User Story 28 – View Worker Profiles

**Screenshots of Website**

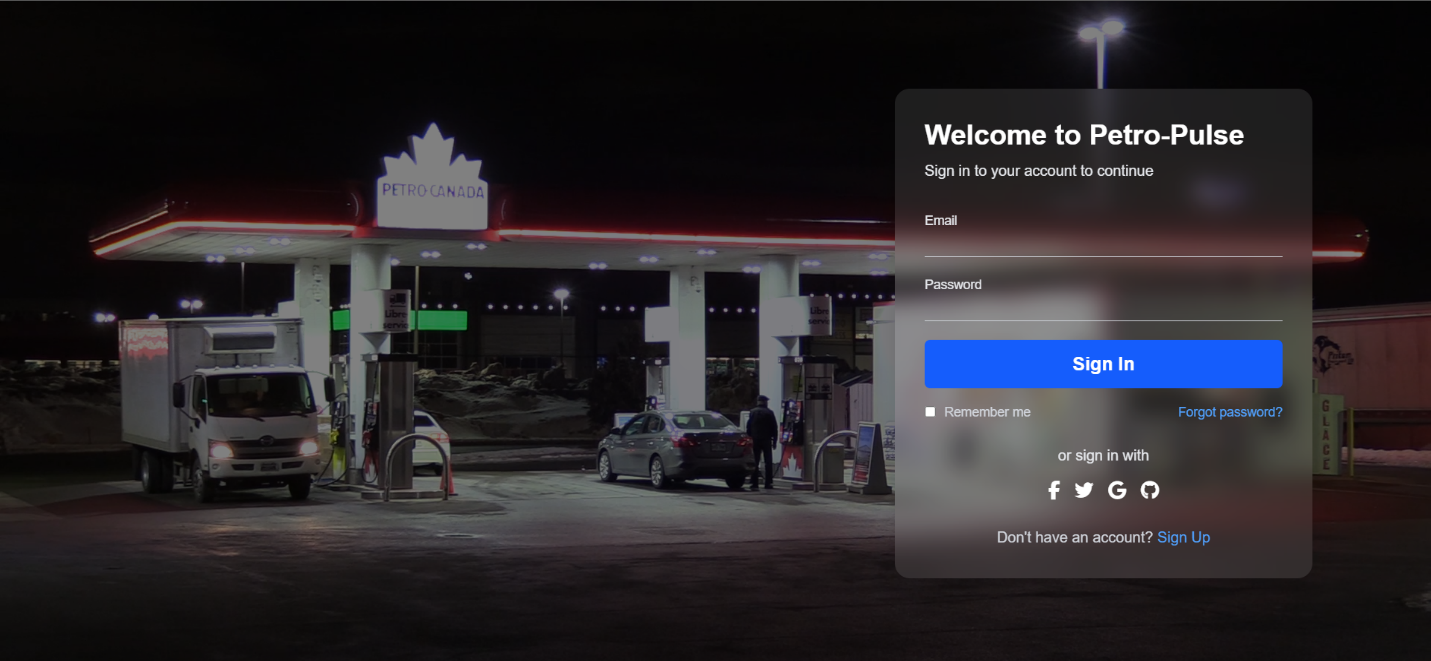
****

Figure 1 Sign in

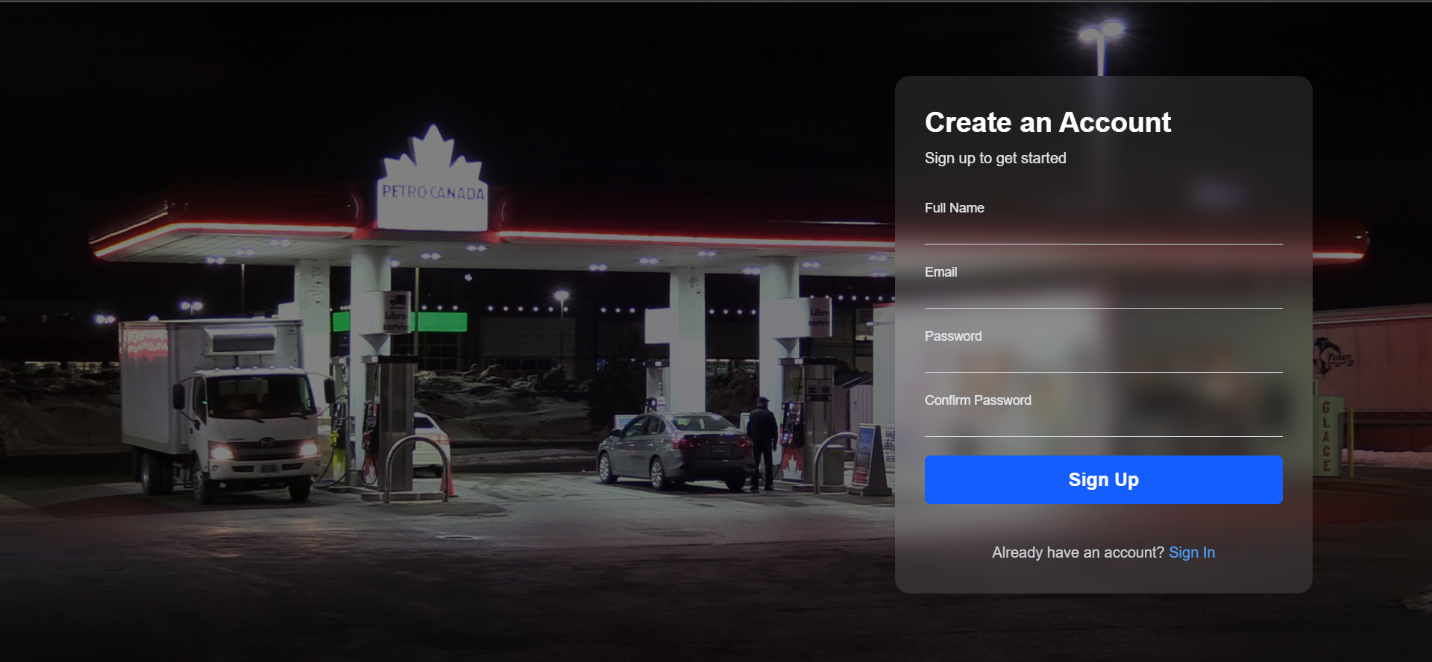
****

Figure 2 Signup

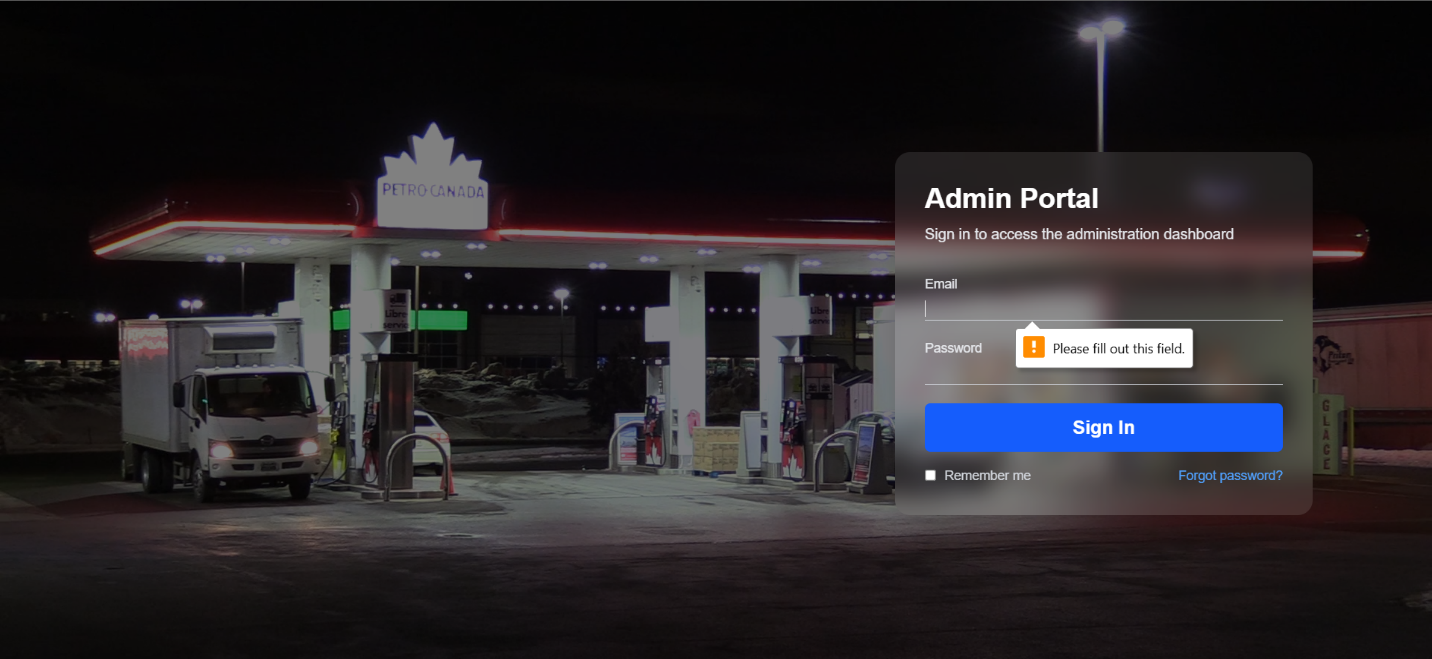
****

Figure 3 Admin Portal

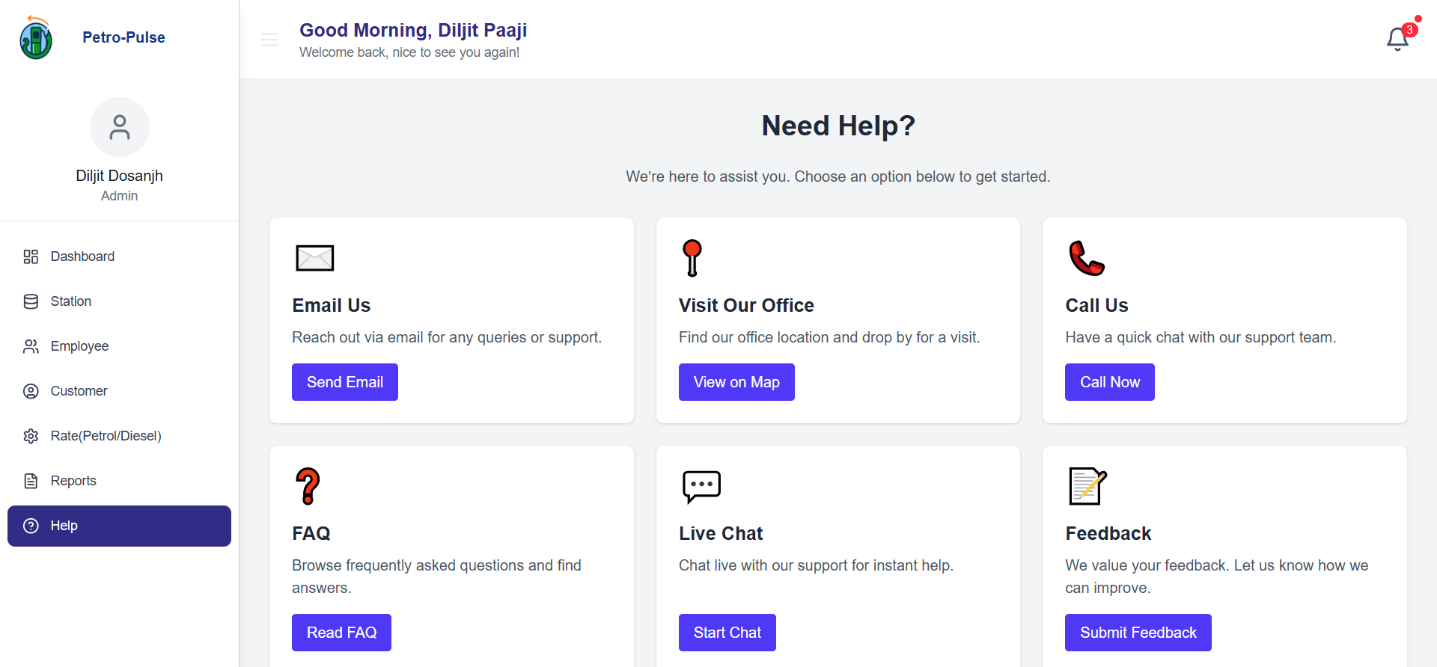
****

Figure 4 Help Centre

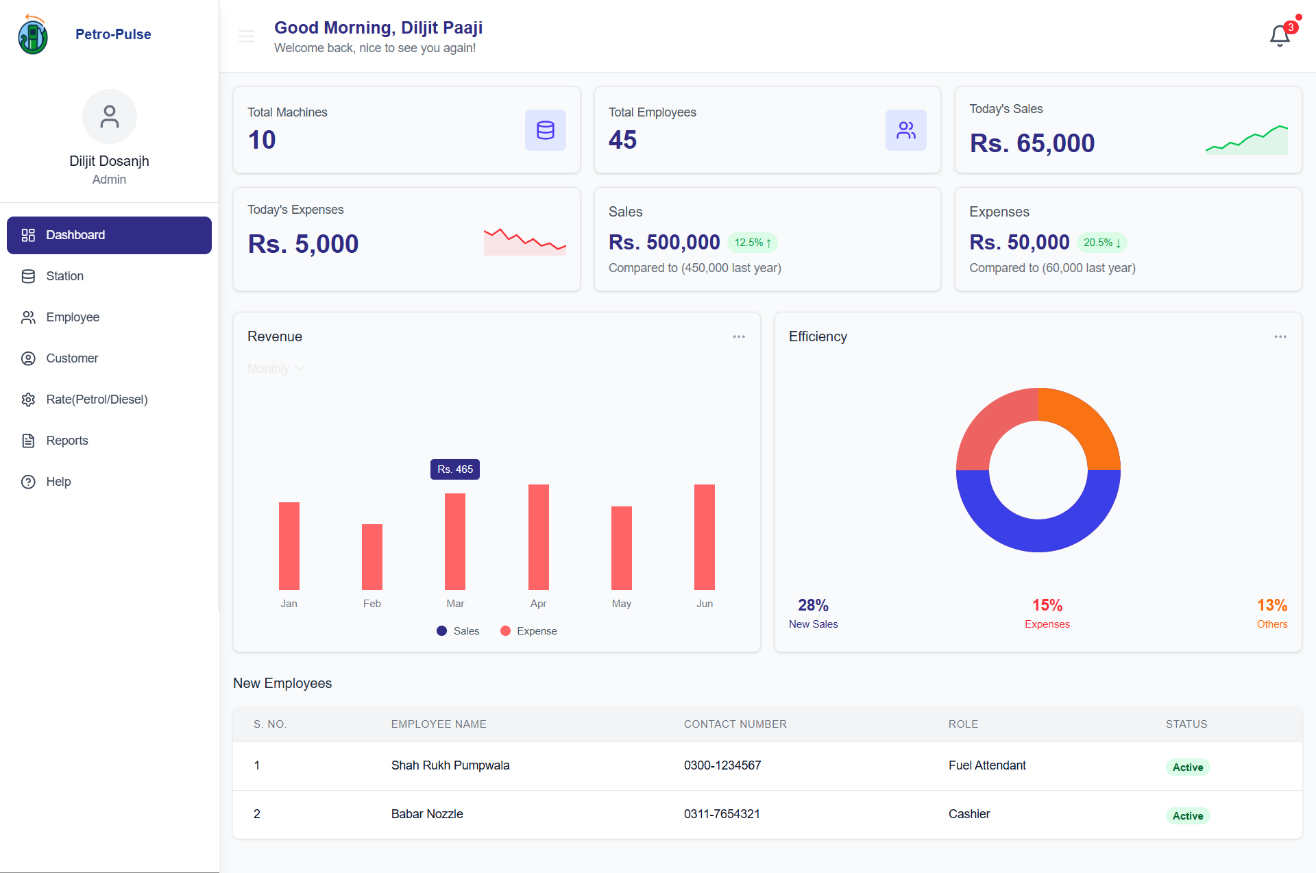
****

Figure 5 Admin Home

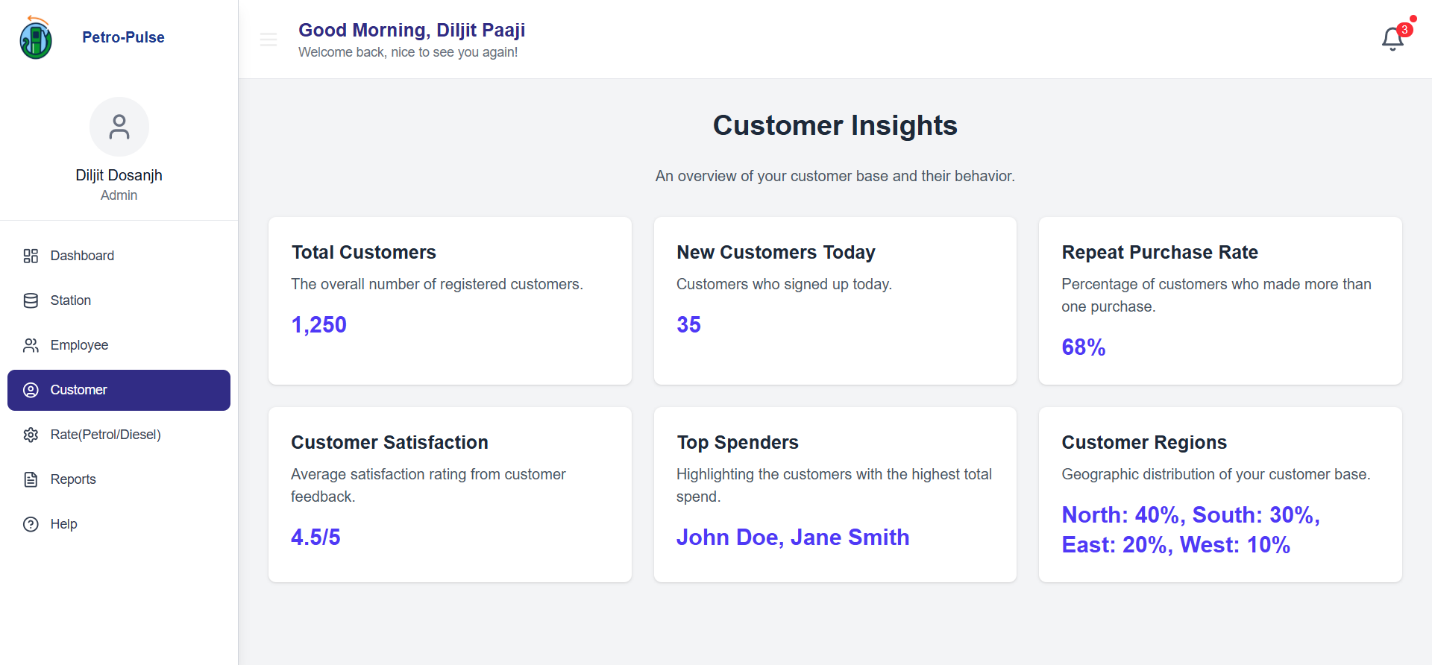
****

Figure 6 Manage Customers

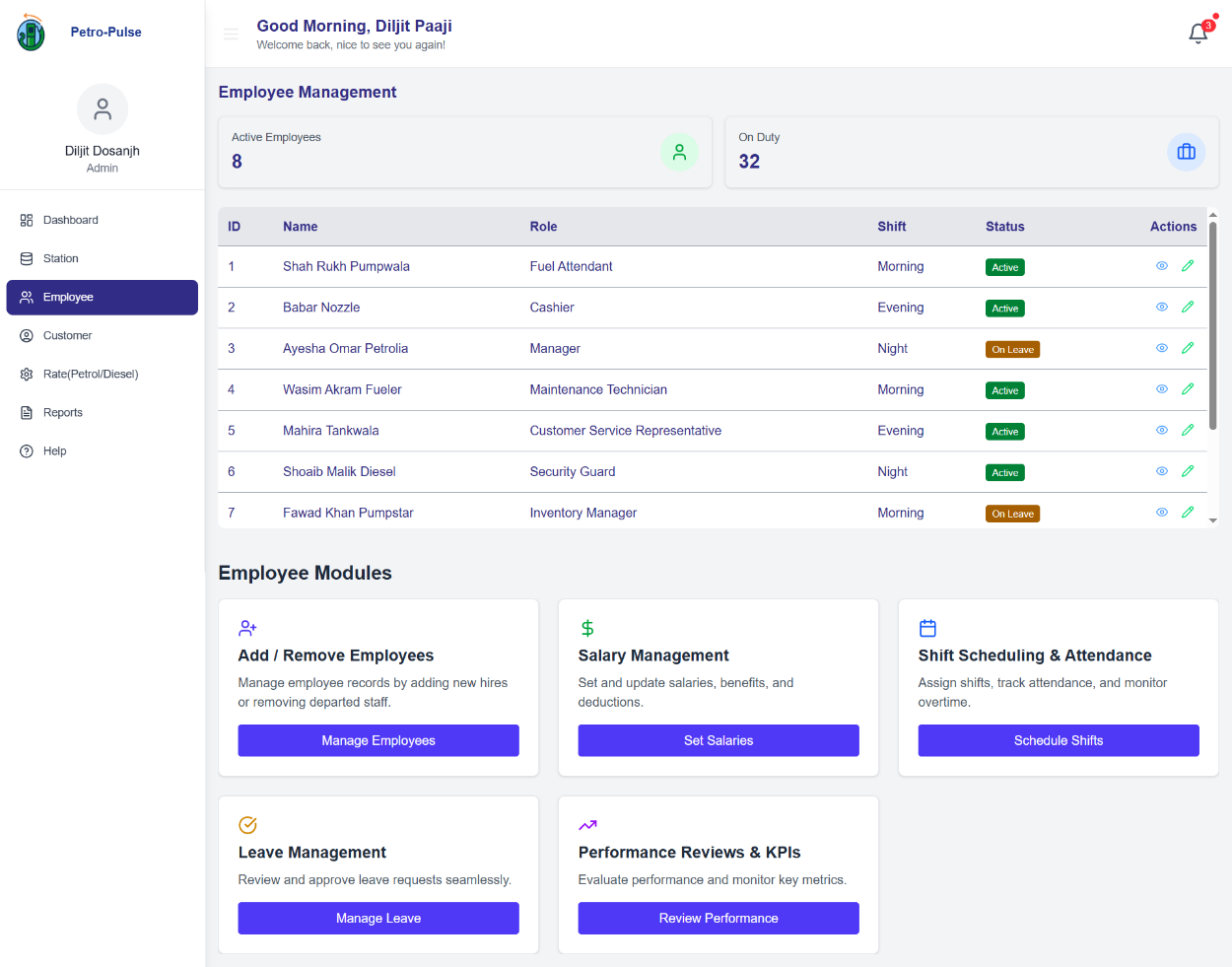
****

Figure 7 Manage Workers

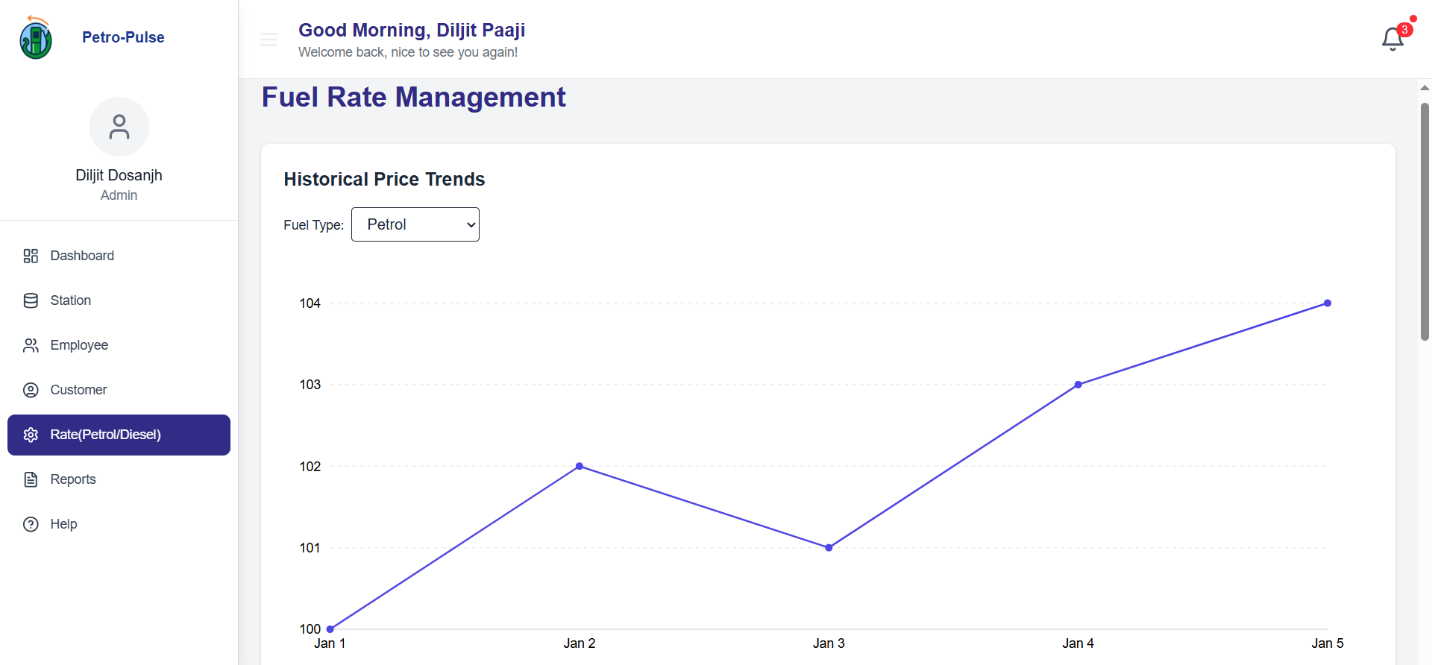
****

Figure 8 Analyze Fuel Rates

**A screenshot of a computer

AI-generated content may be incorrect.**

Figure 9 Analyze Fuel Rates

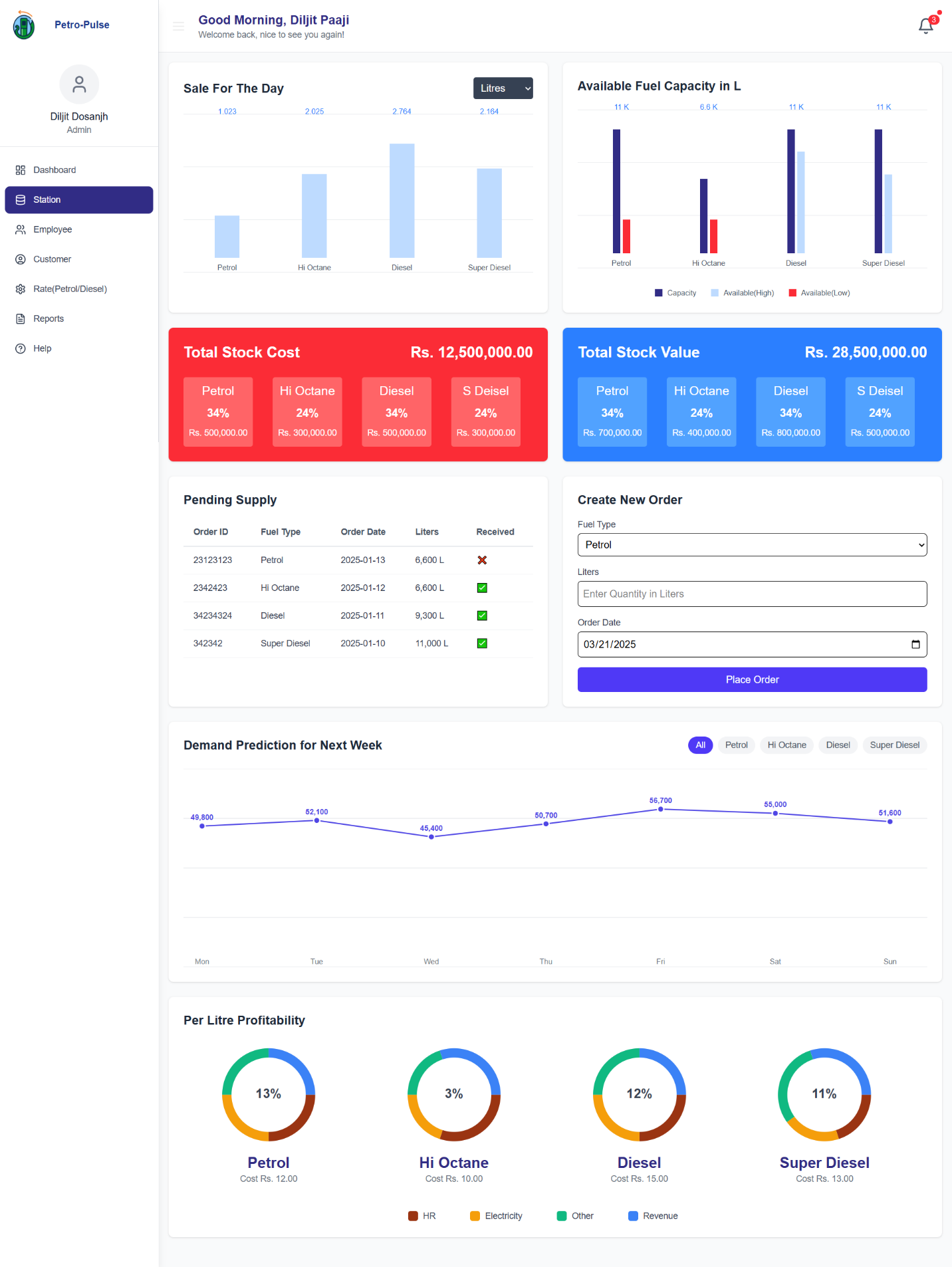
****

Figure 10 Manage Station

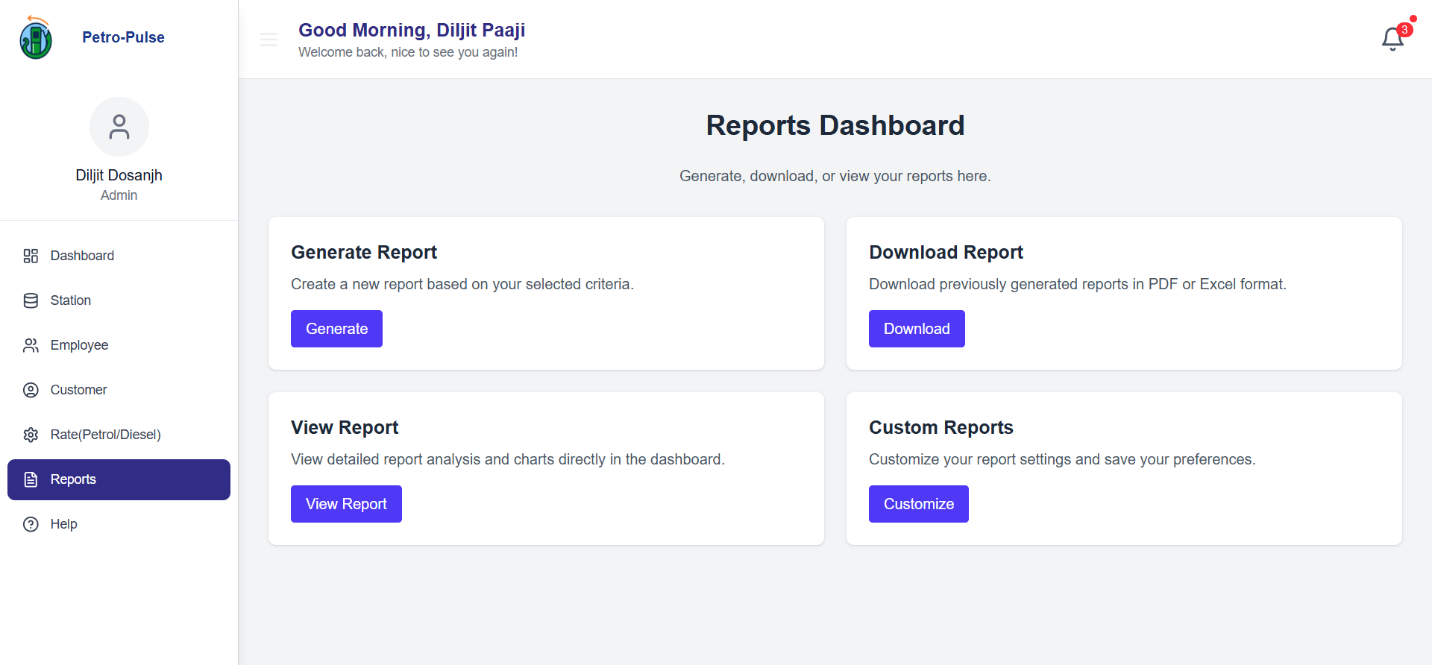
****

Figure 11 View Reports

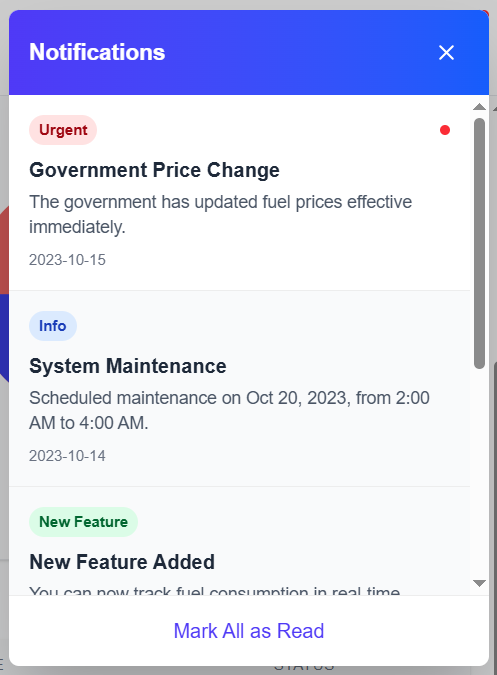
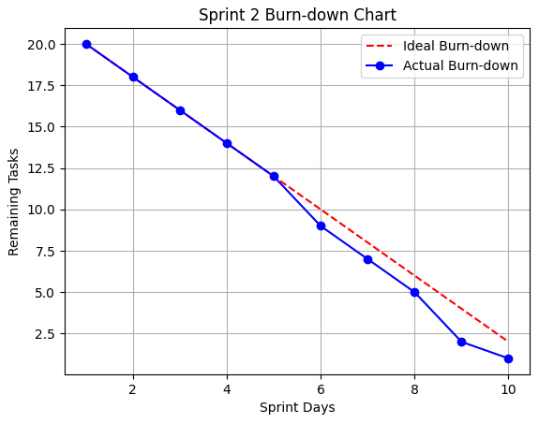
****

Figure 12 Notifications

**Leftover Work**

Customer Dashboard is the Leftover Work for this sprint. It will be completed in the next sprint.



# **Next Sprint Backlog**

**Objective**

The next sprint will focus on backend development using Node.js and Express.js, integrating the MongoDB database, and implementing APIs for communication between frontend and backend. Security measures, authentication, and data persistence will be key priorities in this sprint.

**Scope of Work**

1. **Backend Setup & Configuration**

* Initialize Node.js and Express.js project.
* Install necessary packages.
* Set up MongoDB Atlas for cloud-based storage.

1. **User Authentication & Authorization**

* Implement JWT-based authentication for login and registration.
* Hash passwords using bcrypt for security.
* Role-based access control (RBAC) for owners, workers, and customers.

1. **Fuel Purchase & Transaction Management**

* Implement API endpoints for:
  + Fetching available fuel types and prices.
  + Processing fuel purchases (updating inventory and logging transactions).
  + Validating payments and generating receipts.

1. **Inventory Management**

* API for real-time stock tracking and alerts.
* CRUD operations (Create, Read, Update, Delete) for fuel stock levels.

1. **Reporting & Analytics Backend**

* Generate sales and inventory reports.
* Data aggregation for revenue and customer activity tracking.

1. **Frontend-Backend Integration**

* Connect React.js frontend with backend APIs.
* Handle API calls for authentication, transactions, and inventory updates.

1. **Testing & Optimization**

* API testing using Postman.
* Error handling for invalid requests.
* Optimize database queries for faster response times (<2s).

**User Stories to Work On**

User Story 4 – Add Fuel Station

User Story 6 – Update Fuel Station Details

User Story 7 – Remove Fuel Station

User Story 8 – Register Workers

User Story 9 – Assign Workers to Stations

User Story 11 – Record Fuel Sales

User Story 12 – Generate Sales Report

User Story 13 – Track Fuel Inventory

User Story 14 – Alert for Low Fuel

User Story 15 – Order Fuel Supply

User Story 17 – Redeem Loyalty Points

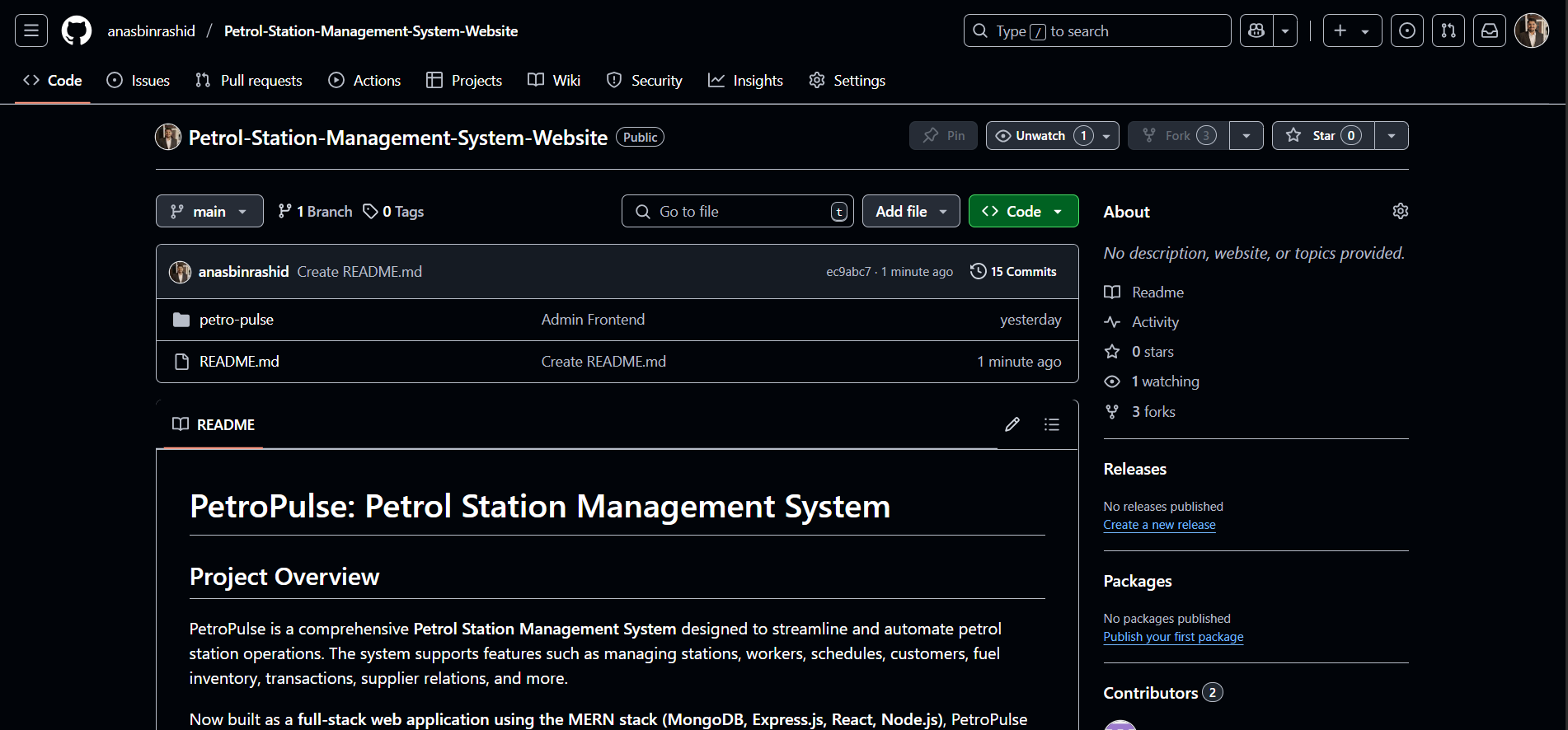
User Story 19 – Manage Pricing for Fuel Types

# **Supporting Information**

## **GitHub Repository**

**Repository Link:** [*https://github.com/anasbinrashid/Petrol-Station-Management-System-Website*](https://github.com/anasbinrashid/Petrol-Station-Management-System-Website)

**Repository Screenshot**

****

**Commits Screenshot**

**A screenshot of a computer

AI-generated content may be incorrect.**

## **Trello Snapshots**

**Initial Snapshot**

**A screenshot of a computer

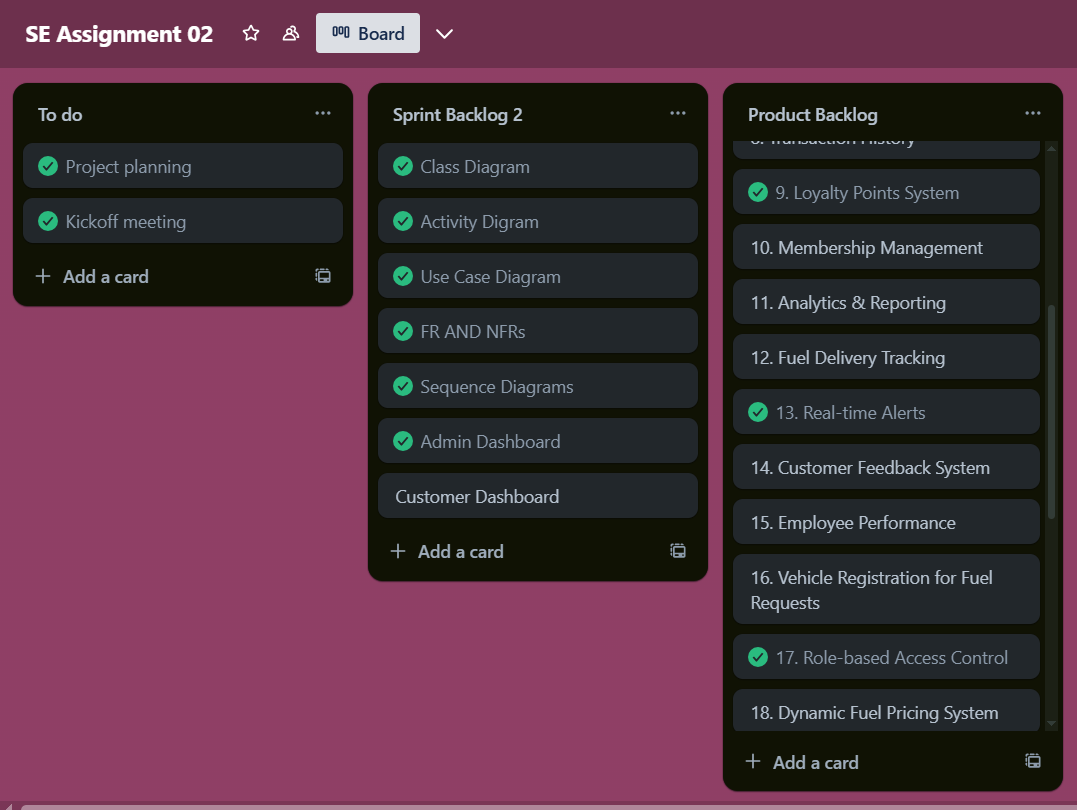
AI-generated content may be incorrect.**

**Middle of the Sprint Snapshot**

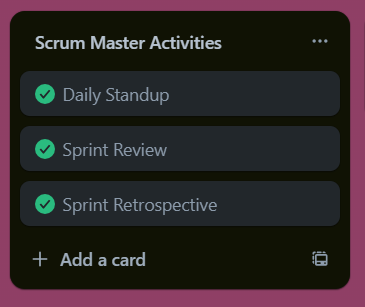
**A screenshot of a computer

AI-generated content may be incorrect.**

**End of the Sprint**

****

**Scrum Master Activities**

****