Hybrid Energy Fuel Delivery System

Mini Project Report

Submitted by

Anina Elizebeth

Reg. No.: AJC22MCA-2017

In Partial Fulfillment for the Award of the Degree of

MASTER OF COMPUTER APPLICATIONS
(MCA TWO YEAR)
[Accredited by NBA]

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

AMAL JYOTHI COLLEGE OF ENGINEERING
KANJIRAPPALLY

[Affiliated to APJ Abdul Kalam Technological University, Kerala. Approved by AICTE, Accredited by NAAC. Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2022-2024

DEPARTMENT OF COMPUTER APPLICATIONS
AMAL JYOTHI COLLEGE OF ENGINEERING
KANJIRAPPALLY

CERTIFICATE

This is to certify that the Project report, "**HYBRID ENERGY"** is the bonafide work of **ANINA ELIZEBETH (Regno: AJC22MCA-2017)** in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications under APJ Abdul Kalam Technological University during the year 2023-24.

Mr. Ajith G.S
Internal Guide

Ms. Meera Rose Mathew
Coordinator

Rev. Fr. Dr. Rubin Thottupurathu Jose
Head of the Department

DECLARATION

I hereby declare that the project report **"HYBRID ENERGY"** is a bonafide work done at Amal Jyothi College of Engineering, towards the partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications (MCA) from APJ Abdul Kalam Technological University, during the academic year 2023-2024.

Date:	**ANINA ELIZEBETH**
KANJIRAPPALLY	**Reg: AJC22MCA-2017**

ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to all who helped me in completing this project successfully. It has been said that gratitude is the memory of the heart. I wish to express my sincere gratitude to our Manager **Rev. Fr. Dr. Mathew Paikatt** and Principal **Dr. Lillykutty Jacob** for providing good faculty for guidance.

I owe a great depth of gratitude towards our Head of the Department **Rev. Fr. Dr. Rubin Thottupurathu Jose** for helping us. I extend my whole hearted thanks to the project coordinator **Ms. Meera Rose Mathew** for her valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Mr. Ajith G.S** for his inspiration and helping hand.

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project. I convey my hearty thanks to my family for the moral support, suggestions, and encouragement to make this venture a success.

ANINA ELIZEBETH

ABSTRACT

"Hybrid Energy" is a web-based application that changes how people get fuel. It connects users with gas stations in a new way. This application makes it easier for people to get and handle fuel. It has a system that manages different types of users like Admins, Vendors, and Customers. This system makes sure everyone gets a personalized experience.

This application also makes payments safe and integrates them into its system. This means people can trust that their money is safe when they pay. It asks users for their opinions and experiences, making the tool better based on what users say. "Hybrid Energy" is a big change in how fuel gets to people. It makes everything smoother for users – how they interact, make orders, pay, and share feedback. Overall, it makes buying fuel modern, easy, and based on what users need.

.

CONTENT

List of Abbreviation

- IDE - Integrated Development Environment
- HTML - Hyper Text Markup Language
- CSS - Cascading Style Sheet
- SQL - Structured Query Language
- UML - Unified Modelling Language
- JS - JavaScript
- AJAX - Asynchronous JavaScript and XML Environment
- URL - Uniform Resource Locator
- PY – Python
- 1NF - First Normal Form
- 2NF - Second Normal Form
- 3NF - Third Normal Form
- PK - Primary Key
- FK - Foreign Key
- ORM - Object-Relational Mapping
- MVT - Model-View-Template
- MVC - Model-View-Controller
- RDBMS - Relational Database Management System

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The "Hybrid Energy" project is a bold endeavor aimed at revolutionizing fuel delivery systems through an innovative web-based application. Its core objective is to transform how people obtain and manage fuel by redefining the interaction between users and fuel stations. A key focus lies in establishing a strong user management system that caters to various roles such as Admins, Vendors, and Customers, ensuring personalized experiences and efficient workflows.

By strategically situating stations and maintaining a comprehensive database containing station details, services offered, and dynamic fuel pricing, the project facilitates easy order management and delivery coordination. It prioritizes secure payment integration to build trust in financial transactions and actively encourages user engagement through feedback mechanisms.

The ultimate goal of "Hybrid Energy" is to reshape the fuel delivery ecosystem by providing a contemporary, efficient, and user-centered approach to fuel procurement. It aims to set new industry standards by offering a modernized experience that prioritizes user needs and convenience.

1.2 PROJECT SPECIFICATION

Admins:

- Overall control of the website.
- Management of user accounts, and system monitoring.
- Authority to block or delete users or their details when necessary.
- Monitoring the rating of the customers and managing their orders.
- Admin can add the location of the pump that should be viewed in the user home.

Vendors:

- Manage fuel station details, services, and pricing.
- Coordinate orders and deliveries efficiently.

Customers:

- Access to fuel station information.
- Place orders and manage deliveries.
- Secure and convenient payment options.

Technology Stack:

- **Frontend:** HTML, CSS, JS, bootstrap for the user interface.
- **Backend:** Django, SQLite.
- **Payment Integration:** Razor Pay
- **Feedback Mechanism:** Interactive feedback system to gather user insights.
- **Authentication:** Authentication using Google Authentication.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

In the realm of system development, a pivotal initial step is system analysis. This crucial phase involves the collection and examination of data with the primary goal of identifying problems and offering effective solutions. A seamless and productive exchange of information between the system's end-users and the developers plays a vital role in this process. It is imperative to kickstart any system development endeavor with a comprehensive system analysis. The role of a system analyst in this context is akin to that of an investigator. They meticulously scrutinize the existing system's performance, delve into the system's inputs and outputs, and establish the connections between its processes and the overall organizational outcomes. Various techniques, such as surveys and interviews, are employed to gather the necessary information. The key objectives here are to gain a deep understanding of how the system functions, pinpoint areas of concern, and propose effective solutions to address the issues plaguing the business. This initial phase of acquiring and analyzing data for future system studies is commonly referred to as a preliminary study. A thorough and comprehensive preliminary study is crucial to lay a strong foundation for the success of any system development project.

2.2 EXISTING SYSTEM

The current fuel delivery system lacks a unified platform specifically designed to streamline operations and interactions between users and fuel stations. There is a significant absence of an integrated application system dedicated to optimizing the efficiency of fuel station operations. This gap highlights the absence of a robust and all-encompassing platform dedicated to fuel station operations, a void that the 'Hybrid Energy' project seeks to fill. It aims to revolutionize and modernize user-fuel station interactions, providing a comprehensive solution tailored specifically for enhanced fuel procurement and streamlined station management.

2.2.1 NATURAL SYSTEM STUDIED

The natural system study of the "Hybrid Energy" project involves understanding its functionality within the broader ecosystem of fuel delivery and management. It assesses the project's impact on existing systems and its alignment with natural processes.

2.2.2 DESIGNED SYSTEM STUDIED

The designed system for "Hybrid Energy" represents an innovative and comprehensive approach to revolutionize the fuel delivery landscape. This system is centered around a web-based application that facilitates seamless interactions between users and fuel stations, aiming to transform the way individuals procure and manage fuel resources. Key functionalities include a user management system with tailored roles for Admins, Vendors, and Customers, ensuring personalized experiences and streamlined workflows. Through strategically situated stations and a comprehensive database containing station details and dynamic fuel pricing, the system enables easy order management and delivery coordination. Secure payment integration guarantees trust and reliability in financial transactions, while active user engagement mechanisms through feedback systems refine the platform's responsiveness. Employing a technology stack encompassing HTML, CSS, JavaScript, Bootstrap, Django, SQLite, Razorpay, and interactive feedback mechanisms, the system promises enhanced user experiences, operational efficiency, and security, aiming to set new benchmarks in providing a modern, user-driven fuel procurement experience.

2.3 DRAWBACKS OF EXISTING SYSTEM

- In the context of the existing fuel delivery system outlined, several drawbacks can be identified:
- Lack of Unified Platform: The current system lacks a centralized platform that integrates all aspects of fuel delivery. This absence leads to fragmented operations and inefficient communication between users and fuel stations.
- Limited Technological Integration: There is a deficiency in utilizing modern technologies to streamline processes. Without an integrated application system, user-fuel station interactions remain archaic, leading to inefficiencies in order management and delivery coordination.
- Fragmented Service Providers: Fuel-related services are dispersed across different agencies, resulting in a lack of standardized processes. This fragmentation increases costs for users and poses challenges in accessing comprehensive services from a single source.
- Insufficient User Engagement: The current solutions, such as small-scale Instagram pages managed by limited teams, fail to engage users effectively. There is a lack of robust mechanisms to gather user feedback and improve services accordingly.
- Inadequate Scope and Functionality: The existing systems, primarily small-scale operations, lack the breadth of functionalities required for efficient fuel procurement and management. This limitation restricts the system's ability to cater to the diverse needs of users and fuel stations.
- Environmental Implications: The current system may lack measures to address environmental concerns associated with fuel delivery, potentially contributing to higher ecological footprints due to inefficiencies.

Addressing these drawbacks is crucial for improving the efficiency, accessibility, and sustainability of the fuel delivery system, a gap that the "Hybrid Energy" project aims to bridge by providing a comprehensive and modernized solution.

2.4 PROPOSED SYSTEM

The proposed system, "Hybrid Energy," is a revolutionary project set to transform the fuel delivery landscape through an innovative web-based application. This system aims to redefine how individuals procure and manage fuel resources by creating a user-centric platform that facilitates seamless interactions between users and fuel stations. It includes a robust user management system catering to different roles—Admins, Vendors, and Customers—ensuring personalized experiences and streamlined workflows. Strategic station placement and a comprehensive database facilitate efficient order management and delivery coordination. The system emphasizes secure payment integration to build trust in financial transactions and actively engages users through feedback mechanisms for continuous improvement. Designed with modern technologies such as HTML, CSS, JavaScript, Bootstrap, Django, SQLite, and Razorpay, the proposed system promises enhanced user experiences, operational efficiency, and sets new standards for a modern, efficient, and user-driven fuel procurement experience.

2.5 ADVANTAGES OF PROPOSED SYSTEM

1. **Enhanced User Experience:** By redefining user-fuel station interactions, the system provides a more intuitive and user-friendly experience, allowing for efficient fuel procurement and management.
2. **Streamlined Operations:** The inclusion of a robust user management system and strategically placed stations facilitates easy order management and delivery coordination, streamlining overall fuel delivery operations.
3. **Trustworthy Transactions:** Integration of secure payment methods ensures reliability and trust in financial transactions, providing users with a secure platform for monetary exchanges.
4. **Active User Engagement:** Incorporating feedback mechanisms encourages user participation, allowing for continuous improvements and refinement based on user insights, thereby enhancing system responsiveness.
5. **Modern Technological Framework:** Utilizing a range of modern technologies including HTML, CSS, JavaScript, Bootstrap, Django, SQLite, and Razorpay, the system operates on a technologically advanced foundation, ensuring efficiency and adaptability.
6. **Industry Benchmarking:** By aiming to set new industry standards, the proposed system aims to become a model for modern, efficient, and user-centric fuel procurement, potentially influencing future developments in the sector.
7. **Comprehensive Solution:** The system addresses multiple aspects of fuel delivery, encompassing user-fuel station interactions, order processing, payment security, and user feedback, providing a holistic solution to users' needs in the fuel procurement process.

These advantages collectively make the "Hybrid Energy" system a comprehensive and modernized solution in the fuel delivery ecosystem, promising enhanced user experiences, operational efficiency, and technological advancements.

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

This examination is a fundamental step in determining whether a project will achieve the organization's objectives considering the resources, time invested in it. It assists the developer in assessing the potential benefits and long-term possibilities of the project. To ascertain the feasibility and worthiness of further analysis, a feasibility study must be conducted for the proposed system. The feasibility study evaluates how the proposed system would impact the organization, its ability to meet customer demands, and the efficient use of resources. Consequently, a feasibility study is typically conducted before proceeding with the development of a new application. The assessment carefully considers various factors, including technical, financial, and operational viability, as outlined in the feasibility study document.

3.1.1 Economic Feasibility

The economic feasibility analysis is a vital step in assessing the viability of the Hybrid Energy migration support application in terms of cost and time investment. It involves a comprehensive examination of all factors that could impact the project's success. After conducting a thorough cost-benefit analysis, the Hybrid Energy project has been found to be feasible and economical within the pre-assumed budget. The analysis indicates that the potential benefits and revenue generation opportunities of Hybrid Energy align well with the initial investment, making it a financially viable endeavor. A thorough assessment of different cost categories, including computer expenses, software implementation, system analysis, website coding, and database design, was conducted to determine the development cost of the Hybrid Energy migration support application. These costs are one-time expenses that will not recur after the project is completed. By carefully analyzing these cost categories, we can ensure that the development of Hybrid Energy is economically feasible and will lead to a positive return on investment.

3.1.2 Technical Feasibility

In the context of the Hybrid Energy fuel delivery application, technical feasibility refers to the review process that establishes if it is possible to build and implement the product using the technology and resources currently in use. The analysis evaluates the technology used to determine how effective the suggested strategy is. It is essential for detecting and resolving potential project challenges before work is started. Hybrid Energy is created to be user-friendly and self-explanatory, requiring little training. Additionally, users may easily access the system, which reduces the cost of using it.

3.1.3 Behavioral Feasibility

Behavioral feasibility is integral to the success of Hybrid Energy, the fuel delivery system. To ensure its adoption, we aim to gauge the willingness of key stakeholders, including migrants, accommodation providers, institutes, employers, and moderators, to embrace the system. We will conduct surveys and engage in open communication to understand and address any concerns or barriers. User-friendly features and access to training and support resources will be provided to facilitate a smooth transition. Cultural sensitivity is vital, given our target audience, and we will ensure the platform respects users' backgrounds and preferences. We will actively listen to user feedback and maintain a feedback mechanism to continuously improve the system.

3.1.4 Feasibility Study Questionnaire

1. Project Overview?

"Hybrid Energy" is a pioneering endeavor set to revolutionize the fuel delivery landscape through an innovative web-based application. The core objective of this project is to transform how individuals procure and manage fuel resources by redefining the interaction between users and fuel stations. Central to its mission is the establishment of a robust user management system catering to diverse roles—Admins, Vendors, and Customers—ensuring personalized experiences and efficient workflows. Through strategic station placement and a comprehensive database, the project aims to facilitate easy order management and delivery coordination. It prioritizes secure payment integration to build trust in financial transactions and actively encourages user engagement through feedback mechanisms. Ultimately, "Hybrid Energy" aims to reshape the fuel delivery ecosystem by providing a modern, efficient, and user-centered approach to fuel procurement, setting new industry standards in the process.

2. To what extend the system is proposed for?

The proposed system, "Hybrid Energy," is designed comprehensively to cater to various facets of the fuel delivery ecosystem. It aims to address the inefficiencies and gaps present in the current fuel procurement and management systems.

3. Specify the Viewers/Public which is to be involved in the System?

In the "Hybrid Energy" system, several user categories or groups are involved:

1. **Admins:** They have comprehensive control over the system. Admins manage news updates, user accounts, and system monitoring. They possess the authority to oversee the overall system and perform tasks like blocking or deleting users or their details when required.
2. **Vendors:** Vendors play a critical role in managing fuel station details, services, and pricing. Their responsibilities extend to coordinating orders and ensuring efficient deliveries, contributing significantly to the operational aspect of fuel stations within the system.

3. **Customers:** The primary users of the system, customers interact with the platform to access fuel station information, place orders, manage deliveries, and utilize secure and convenient payment options.
4. **General Public/End Users:** While not directly involved in system management, the general public can access certain functionalities offered by the system. They may interact with the platform to obtain information about fuel stations or potentially engage in fuel procurement activities facilitated by the system.

4. List the Modules included in your System?

In the "Hybrid Energy" system, several modules are included to ensure efficient functionality and comprehensive management:

1. **User Management Module:** This module handles user roles and permissions, allowing Admins to manage user accounts and access levels for Admins, Vendors, and Customers.
2. **Fuel Station Management Module:** Dedicated to Vendors, this module enables the management of fuel station details, services offered, pricing, and order coordination.
3. **Order and Delivery Module:** Facilitates order placement, management, and delivery coordination for Customers, ensuring seamless transactions and timely fuel deliveries.
4. **Payment Integration Module:** Ensures secure and convenient payment options for Customers, integrating payment gateways like Razorpay for reliable financial transactions.
5. **Feedback Mechanism Module:** Encourages user engagement by implementing mechanisms for Customers to provide feedback, contributing to continuous system improvement.
6. **System Monitoring Module:** Enables Admins to oversee the overall system functionality, monitor news updates, user activities, and perform necessary actions to maintain system integrity.

These modules collectively form the backbone of the "Hybrid Energy" system, encompassing various functionalities and user roles to streamline fuel procurement and management processes.

5. Identify the users in your project?

1. **Admins:** They hold the highest level of authority within the system. Admins are responsible for managing the overall functionality, overseeing news updates, user accounts, and system monitoring. They have the capability to maintain system integrity by controlling user access and handling potential issues or irregularities within the platform.
2. **Vendors:** These users are responsible for managing fuel station details, services offered, and pricing within the system. Vendors play a crucial role in coordinating orders and ensuring efficient deliveries, maintaining accurate and updated information regarding their respective

fuel stations.

3. **Customers:** The primary end-users of the system, customers engage with the platform to access information about fuel stations, place orders for fuel, manage their deliveries, and utilize secure payment options provided by the system.

6. Who owns the system?

This is a sole idea of Anina Elizebeth, young talented student of Amal Jyothi College of Engineering. She has the ownership of the idea and its web application.

7. System is related to which firm/industry/organization?

The system described, "Hybrid Energy," is affiliated with the fuel delivery industry or sector. It focuses on transforming how fuel resources are procured and managed, potentially impacting various firms or organizations operating within the fuel delivery landscape. This innovative system aims to revolutionize the interaction between users and fuel stations, catering to customers, vendors, and administrators involved in the fuel procurement process. While it's a software system designed to streamline operations within the fuel industry, the exact firm or organization associated with the development and deployment of this system might vary depending on the project stakeholders, investors, or the entity responsible for its implementation.

8. Details of person that you have contacted for data collection?

Pump

https://iocl.com/ (Website)

9. Questionnaire to collect details about the project? (Min 10 questions, include descriptive answers, attach additional docs (e.g., Bill receipts), if any?)

1. Are there any planned enhancements or future features envisioned for the system beyond its initial release?

Yes, future iterations aim to include additional features such as real-time tracking, integration with IoT devices, and further user customization options for an enriched experience.

2. How are payment transactions processed within the system, and what payment gateways are integrated?

Payment transactions are facilitated through Razorpay integration, ensuring secure and reliable financial exchanges between Customers and Vendors.

3. What is the primary objective of Hybrid Energy?

The primary objective of Hybrid Energy is to revolutionize the fuel delivery landscape by creating an innovative, user-centric system that transforms the way individuals procure and manage fuel resources. This system aims to streamline interactions between users and fuel stations, providing a seamless platform for fuel procurement, order management, and delivery coordination.

4. Can you describe the process of user registration and authentication on Hybrid energy?

User Registration:

1. **Registration Form:** Users access the registration form, providing necessary details such as name, email, and password.
2. **Input Validation:** Hybrid Energy validates the entered information, checking for correct formats and unique email IDs.
3. **Email Verification:** An email verification link is sent to the provided email address to confirm its validity and the user's ownership.
4. **Confirmation:** Once the user clicks on the verification link, their email is confirmed, and they are directed to the Hybrid Energy platform.

User Authentication:

1. **Login Page:** Registered users access the platform through a login page by providing their email/username and password.
2. **Credential Verification:** Hybrid Energy authenticates the user-entered credentials against the stored data in its database.
3. **Session Creation:** Upon successful verification, Hybrid Energy creates a session token or cookie, providing temporary access to the platform.
4. **Session Management:** The session token validates subsequent interactions, allowing users to access specific functionalities within Hybrid Energy during the active session.
5. **Session Termination:** Users can log out manually, terminating their session and revoking access to the platform.

5. How the security is ensured in the hybrid energy?

In Hybrid Energy, security is a top priority, safeguarding user data and transactions. The system employs encryption for data protection, robust authentication, and authorization mechanisms for user access control. Secure payment gateways like Razorpay ensure safe financial transactions. Regular security audits, data backups, and disaster recovery plans mitigate vulnerabilities and prevent data loss. Secure communication protocols, user education, and compliance with industry standards strengthen the system's security. Continuous monitoring and incident response strategies maintain a safe environment, ensuring a trustworthy fuel procurement experience for users.

6. How will you measure the success of the project?

The success of the project will be gauged through a holistic assessment of various key performance indicators (KPIs). These include user adoption rates, measured by the active engagement of users with the platform, order volume to ascertain the acceptance and utilization of the system, and customer satisfaction levels evaluated through feedback and ratings. Revenue generation from transactions conducted via the platform will reflect its financial impact, while system uptime and performance will ensure consistent availability and responsiveness. User retention rates, cost savings, or efficiency gains, and security metrics will further contribute to the evaluation. Additionally, tracking market penetration and ensuring compliance with relevant industry standards and regulations will be integral in determining the project's overall success. The collective assessment of these quantitative and qualitative measures will provide a comprehensive understanding of the project's effectiveness in enhancing user experiences, operational efficiency, and security within the fuel delivery domain.

7. What all major thing should considered and implemented to run a pump management system effective?

To run a pump management system effectively, a few critical elements need attention. A user-friendly interface for both staff and customers is essential, enabling easy navigation and clear functionality. Managing inventory levels and automating restocking processes ensures a steady fuel supply without overstocking. Robust payment gateways and security measures instill trust during transactions. Regular maintenance schedules maintain pump performance, while real-time monitoring alerts staff to issues. Implementing CRM tools for customer interactions and compliance with regulations are equally vital. Energy-efficient practices, mobile accessibility, staff training, and continuous support round out the essentials for smooth pump system operation.

8. Are there any external factors or dependencies that may impact the project?

External factors or dependencies can significantly influence a project like pump management systems. For instance, changes in government regulations related to fuel distribution, environmental policies, or safety standards can directly impact operations and necessitate system adaptations. Additionally, advancements in technology or shifts in consumer preferences for alternative energy sources might influence the demand for fuel, requiring the system to be flexible. Dependencies on third-party vendors for hardware, software, or payment processing could affect the project's timeline or functionality if issues arise with these external providers. Moreover, economic conditions, market fluctuations, or unexpected events like natural disasters can also pose challenges that impact the project's progress. Developing contingency plans to address these potential dependencies is crucial for project resilience.

9. What are the major milestones or phases of the project Hybrid energy?

The major phases of the Hybrid Energy project include conceptualization, system design, development, testing, deployment, user training, ongoing maintenance, and future scaling. These stages encompass planning, building, testing, and maintaining the fuel delivery system to ensure its successful implementation and continuous improvement.

10. How would you rate the user-friendliness of Hybrid Energy?

Assessing the user-friendliness of Hybrid Energy can be best gauged through user feedback and usability testing. Employing metrics such as ease of navigation, intuitiveness in placing orders, clarity of information, and the overall experience, I'd rate the system's user-friendliness as a crucial aspect that directly impacts its success. Conducting surveys, user interviews, and analyzing user behavior within the platform can provide a comprehensive understanding of its user-friendliness and help in refining and improving the system for enhanced user experiences.

3.2 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor	-	Intel I3 or above
RAM	-	4 G B
Hard disk	-	2 5 6 G B

3.2.2 Software Specification

Front End	-	HTML5, Bootstrap, CSS
Back End	-	Django Framework, Python
Database	-	SQLite
Client on PC	-	Windows 7 and above.
Technologies used	-	JS, HTML5, AJAX, J Query, Python, CSS, Django, SQLite

3.3 SOFTWARE DESCRIPTION

3.3.1 DJANGO FRAMEWORK:

The Django Framework is a popular and robust web framework for Python developers. It is revered for its simplicity, clean code, and rapid development capabilities. Django is built on the Model-Template-Views (MTV) architectural pattern, which shares similarities with the ModelView-Controller (MVC) pattern used in other frameworks. One of its standout features is the Object-Relational Mapping (ORM) system, simplifying database interactions by representing database tables as Python objects. This abstraction eliminates the need for writing raw SQL queries, making database operations more straightforward.

Django also offers a built-in administrative interface, making content management a breeze. Its URL routing system allows developers to define clean and user-friendly URLs for their web applications. Furthermore, Django provides comprehensive support for form handling, data validation, and user authentication, reducing the complexity of common web development tasks.

Security is a top priority in Django, with built-in protections against common web vulnerabilities like Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). With a thriving community and an array of reusable packages, Django is a versatile choice for web development projects of varying sizes and complexities.

3.3.2 SQLite:

SQLite is a lightweight, self-contained, and serverless relational database management system. Unlike traditional databases, it does not require a separate server but is embedded directly within the application. One of its standout features is the self-contained nature of SQLite databases; the entire database is stored in a single file, simplifying management, backups, and transfers. Despite its lightweight design, SQLite is ACID-compliant, meaning it ensures data integrity by supporting transactions, making it suitable for multi-user environments.

SQLite is cross-platform and compatible with various operating systems, making it a versatile choice for applications targeting different platforms. Its small code size and minimal resource usage make it suitable for resource-constrained environments, such as mobile devices. SQLite is known for its speed and efficiency, especially in read-heavy workloads. It is commonly used in mobile applications, desktop applications, and embedded systems, where a full-fledged database server might be excessive, and portability is essential.

CHAPTER 4
SYSTEM DESIGN

4.1 INTRODUCTION

Design is the first step into the development phase for any engineered product or system. Design is a creative process. A good design is the key to effective system. The term "design" is defined as "the process of applying various techniques and principles for the purpose of defining a process or a system in sufficient detail to permit its physical realization". It may be defined as a process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit its physical realization. Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm that is used. The system design develops the architectural detail required to build a system or product. As in the case of any systematic approach, this software too has undergone the best possible design phase fine tuning all efficiency, performance and accuracy levels. The design phase is a transition from a user-oriented document to a document to the programmers or database personnel. System design goes through two phases of development: Logical and Physical Design.

4.2 UML DIAGRAM

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997. UML is a pictorial language used to make software blueprints. UML can be described as a general-purpose visual modeling language to visualize, specify, construct, and document software system. UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object-oriented analysis and design. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system. The visual effect of the UML diagram is the most important part of the entire process. All the other elements are used to make it complete. They enhance communication by providing a visual representation that is easily understood by technical and non-technical stakeholders alike. They aid in analysis and design by capturing system requirements, relationships, and interactions. UML diagrams also facilitate software development by serving as a blueprint for developers to implement and test systems.

UML includes the following nine diagrams.

- Use case diagram
- Sequence diagram
- State chart diagram
- Activity diagram
- Class diagram
- Object diagram
- Component diagram
- Deployment diagram

4.2.1 USE CASE DIAGRAM

A use case diagram is a graphic depiction of the interactions among the elements of a system. A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. In this context, the term "system" refers to something being developed or operated, such as a mail-order product sales and service Web site. Use case diagrams are employed in UML (Unified Modeling Language), a standard notation for the modeling of real-world objects and systems. System objectives can include planning overall requirements, validating a hardware design, testing and debugging a software product under development, creating an online help reference, or performing a consumer-service-oriented task. For example, use cases in a product sales environment would include item ordering, catalog updating, payment processing, and customer relations. A use case diagram contains four components.

• The boundary, which defines the system of interest in relation to the world around it.

• The actors, usually individuals involved with the system defined according to their roles.

• The use cases, which are the specific roles are played by the actors within and around the system.

• The relationships between and among the actors and the use cases.

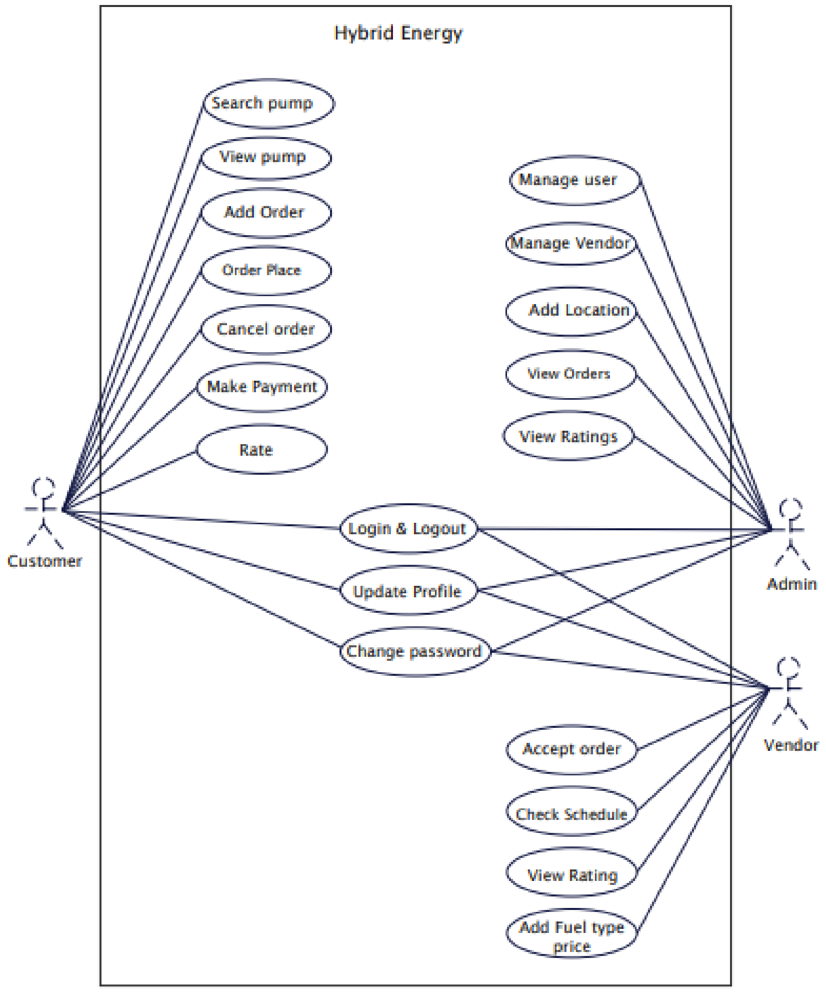


Fig.1 Use Case diagram for 'Hybrid Energy'

4.2.2 SEQUENCE DIAGRAM

The chronological order of interactions between various system components is shown in a sequence diagram, a form of interaction diagram. It demonstrates how several things communicate with one another over the course of a series of messages. These images are sometimes referred to as event scenarios or event scenarios diagrams. In software engineering, sequence diagrams are frequently used to describe and comprehend the needs of both new and old systems. They support the visualization of object control relationships and the detection of systemic issues.

Sequence Diagram Notations –

i. **Actors** - In UML, a role that interacts with the system and its objects is represented by an actor. Actors frequently exist outside of the system that the UML diagram is intended to portray. Actors can play a variety of roles, including those of external topics or human users. A stick person notation is used in UML diagrams to represent actors. Depending on the situation that is being modelled; a sequence diagram may have more than one actor.

ii. **Lifelines** - A lifeline in a sequence diagram is a vertical dashed line that represents the lifespan of an object participating in the interaction. Each lifeline represents an individual participant in the sequence of events and is labeled with the name of the participant. The lifeline shows the timeline of events for the participant and is drawn as a vertical line extending from the participant's activation point to its deactivation point.

iii. **Messages** - Messages are a key component of sequence diagrams, representing the interactions and communication between objects or components in a system. They can be categorized into synchronous and asynchronous messages, create, and delete messages, self-messages, reply messages, found messages, and lost messages. Guards are also used to model conditions and restrictions on message flow.

iv. **Guards** - Guards in UML are used to model conditions and are employed to restrict the flow of messages when a certain condition is met. This feature is essential for letting software developers know about any constraints or limitations associated with a system or a particular process.

Uses of sequence diagram –

- Modeling and visualizing the logic of complex functions, operations, or procedures.
- Showing details of UML use case diagrams.
- Understanding the detailed functionality of current or future systems.
- Visualizing how messages and tasks move between objects or components in a system.

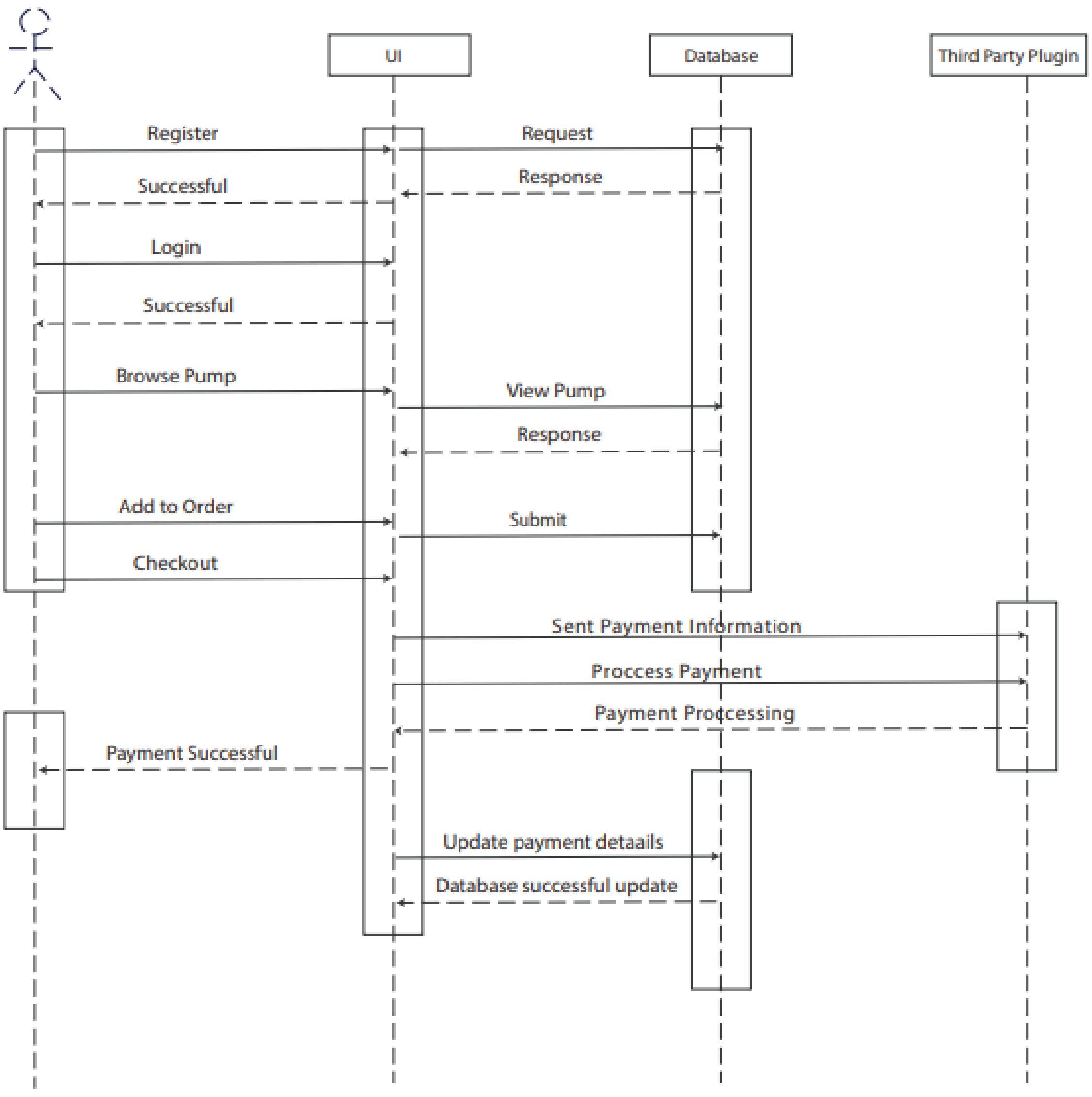


Fig.2 Sequence diagram for ‘Hybrid Energy’

4.2.3 State Chart Diagram

A state diagram is a visual representation, often created using the Unified Modeling Language (UML), that shows the different states that an object can exist in and how it can transition between those states. It is also referred to as a state machine diagram or state chart diagram. The State Chart Diagram is a behavioral diagram in UML that describes the behavior of a system or object over time. It includes various elements such as:

- Initial State - This state represents the starting point of the system or object and is denoted by a solid black circle.
- State - This element describes the current state of the system or object at a specific point in time and is represented by a rectangle with rounded corners.
- Transition - This element shows the movement of the system or object from one state to another

and is represented by an arrow.

- Event and Action - An event is a trigger that causes a transition to occur, and an action is the behavior or effect of the transition.
- Signal - A message or trigger caused by an event that is sent to a state, causing a transition to occur.
- Final State - The State Chart Diagram ends with a Final State element, which is represented by a solid black circle with a dot inside. It indicates that the behavior of the system or object has completed.

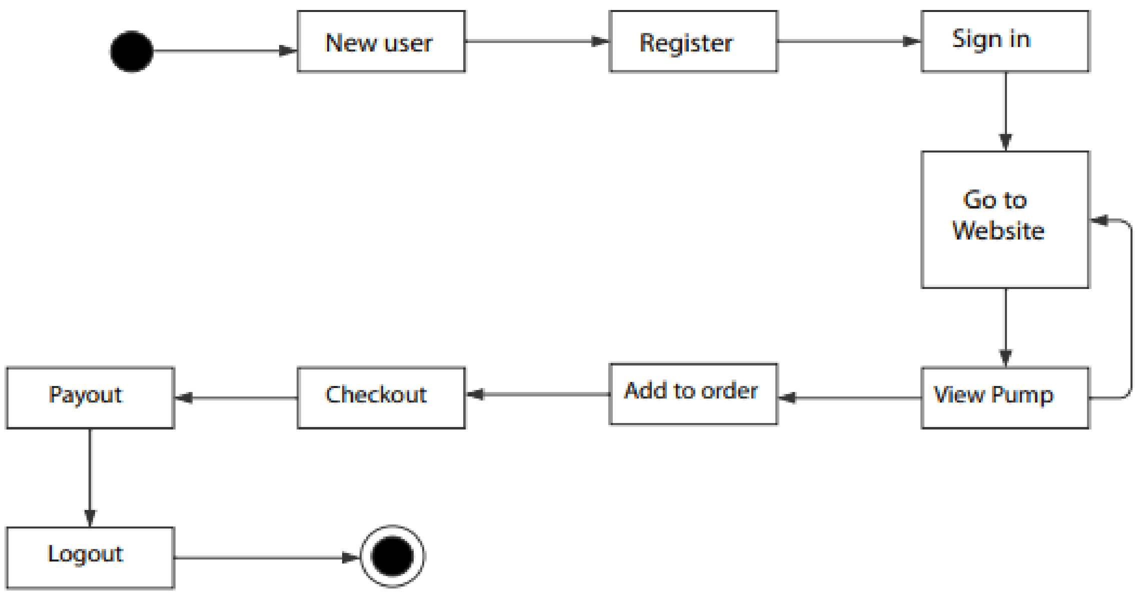


Fig.3 State chart diagram for ‘Hybrid Energy’

4.2.4 Activity Diagram

Activity diagrams depict how different levels of abstraction of activities are linked to provide a service. Typically, an event should be completed by some activities, particularly when the activity is intended to do multiple separate goals that need coordination. Another typical requirement is how the events in a single use case interact with one another, particularly in use cases where operations may overlap and require coordination. It may also be used to show how a collection of interrelated use cases interacts to reflect business operations. In an activity diagram, activities are represented as rounded rectangles, and arrows indicate the flow of control between activities. Decision points are depicted as diamonds, where different paths or branches can be taken based on conditions or criteria. Forks and joins represent parallel or concurrent execution of activities. Swim lanes can also be used to group related activities performed by specific actors or roles. Activity diagrams are valuable for modelling and analyzing complex processes or workflows. They help to visualize the sequence of steps, dependencies, and decision points within a system, providing a clear understanding of how the process operates. This aids in identifying potential bottlenecks, inefficiencies, or areas for optimization.

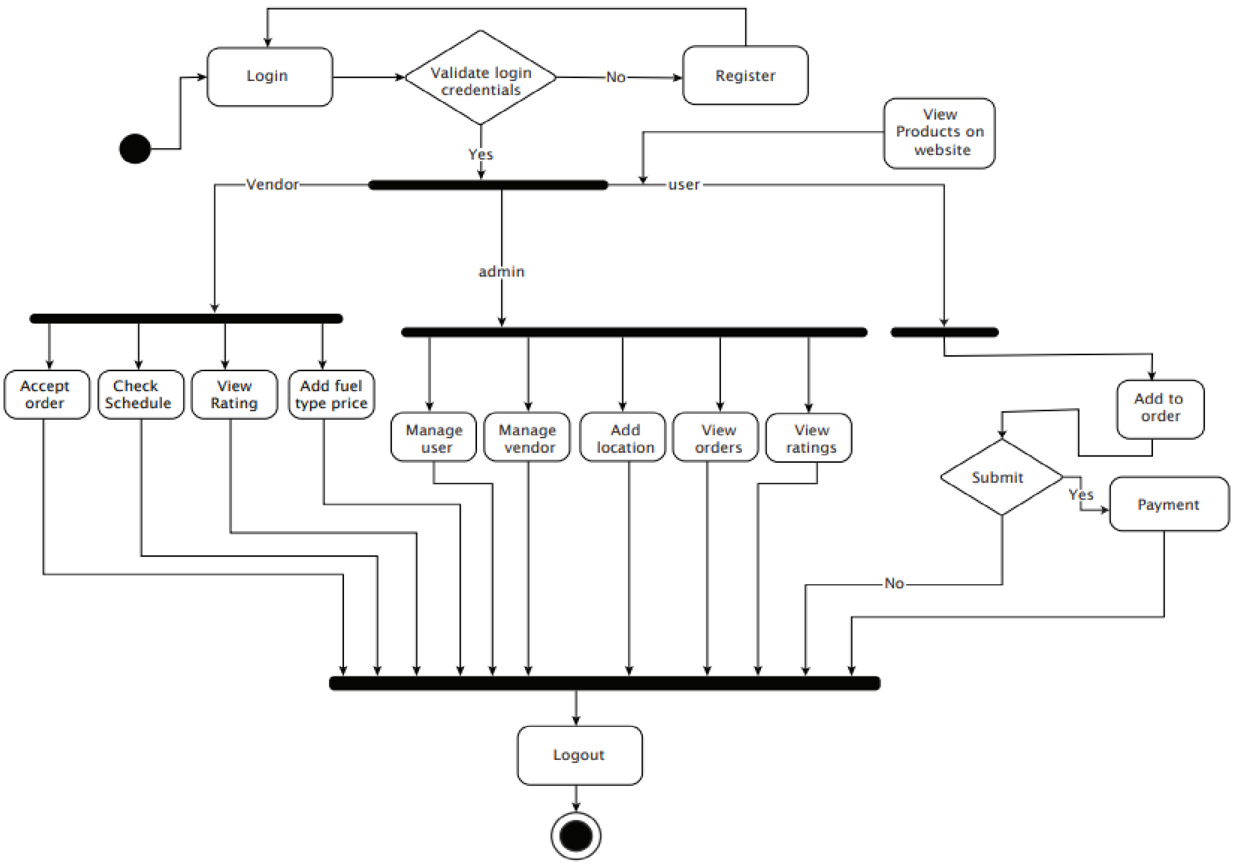


Fig.4 Activity diagram for ‘Hybrid Energy’

4.2.5 Class Diagram

Class diagram is a static diagram. It represents the static view of the application. Class diagrams are useful for visualizing, describing, and documenting various system components as well as for writing executable code for software applications. A class diagram describes the constraints imposed on the system together with the properties and operations of a class. The only UML diagrams that can be directly converted into object-oriented languages are class diagrams, which are extensively utilized in the designing of object-oriented systems. An assortment of classes, interfaces, affiliations, partnerships, and limitations are displayed in a class diagram. It also goes by the name "structural diagram”. Class diagrams are widely used in software development to design and document the structure of object-oriented systems. They help to identify and organize classes, their attributes, and behaviors, providing a blueprint for software implementation. Class diagrams also assist in understanding and communicating the relationships and interactions between different classes. Key elements in a class diagram include associations, which represent relationships between classes, such as one-to-one, one-to-many, or many-to-many associations. Inheritance is depicted using arrows to indicate the "is-a" relationship, where one class inherits attributes and methods from another. Aggregation and composition relationships are used to represent whole-part relationships between classes. Class diagrams are an essential tool for visualizing the static structure of a system and capturing its essential elements. They aid in system design, modelling, and documentation, allowing software developers and stakeholders to understand the system's architecture, dependencies, and responsibilities of different classes.

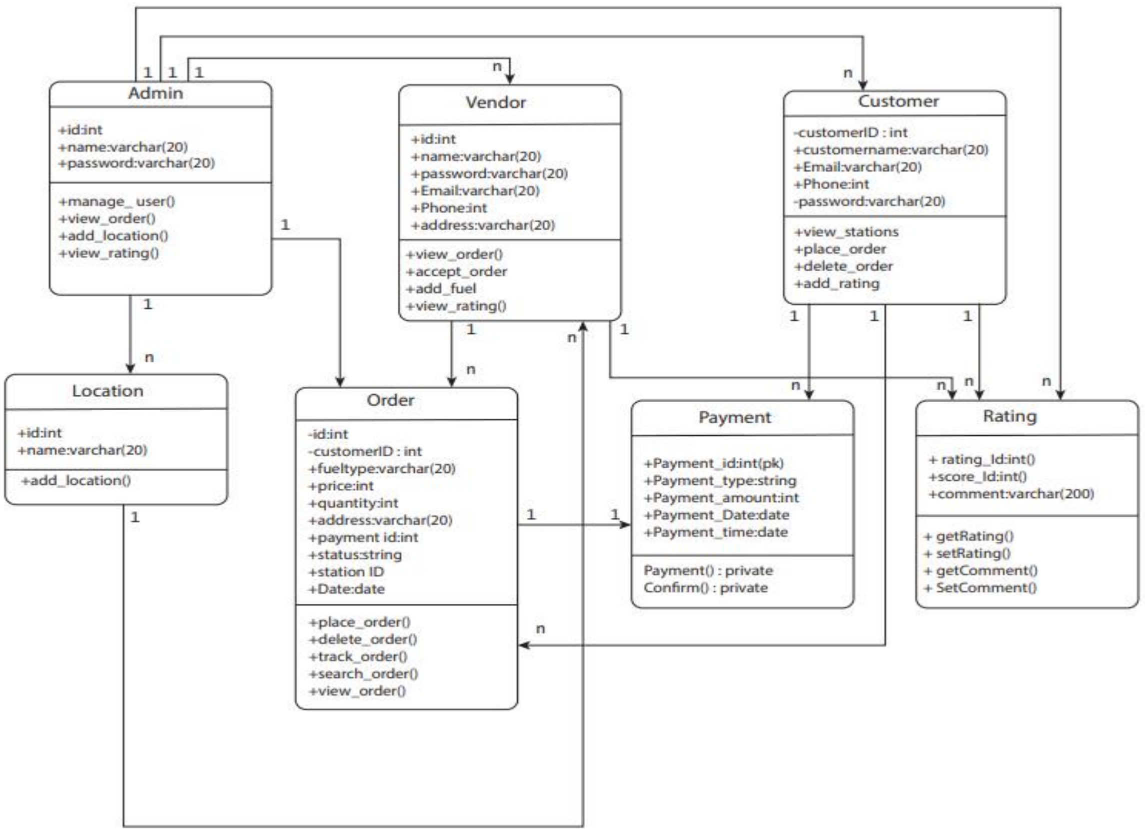


Fig.5 Class diagrams for ‘Hybrid Energy’

4.2.6 Object Diagram

Class diagrams and object diagrams are closely related in object-oriented modeling. Object diagrams are instances of class diagrams, which represent a snapshot of the system at a given moment in time. Both types of diagrams use the same concepts and notation to represent the structure of a system. While class diagrams are used to model the structure of the system, including its classes, attributes, and methods, object diagrams represent a group of objects and their connections at a specific point in time. An object diagram is a type of structural diagram in UML that shows instances of classes and their relationships. The main components of an object diagram include:

- Object: An object is an instance of a class that represents a specific entity in the system. It is represented as a rectangle with the object name inside.

- Class: A class is a blueprint or template for creating objects that defines its attributes and methods. It is represented as a rectangle with three compartments for the class name, attributes, and methods.
- Link: A link is a relationship between two objects that represents a connection or association. It is represented as a line connecting two objects with optional labels.
- Attribute: An attribute is a property or characteristic of an object that describes its state. It is represented as a name-value pair inside the object rectangle.
- Value: A value is a specific instance or setting of an attribute. It is represented as a value inside the attribute name-value pair.
- Operation: An operation is a behavior or action that an object can perform. It is represented as a method name inside the class rectangle.
- Multiplicity: Multiplicity represents the number of instances of a class that can be associated with another class.

Object diagrams help to visualize the relationships between objects and their attributes in a system. They are useful for understanding the behavior of a system at a specific point in time and for identifying potential issues or inefficiencies in the system.

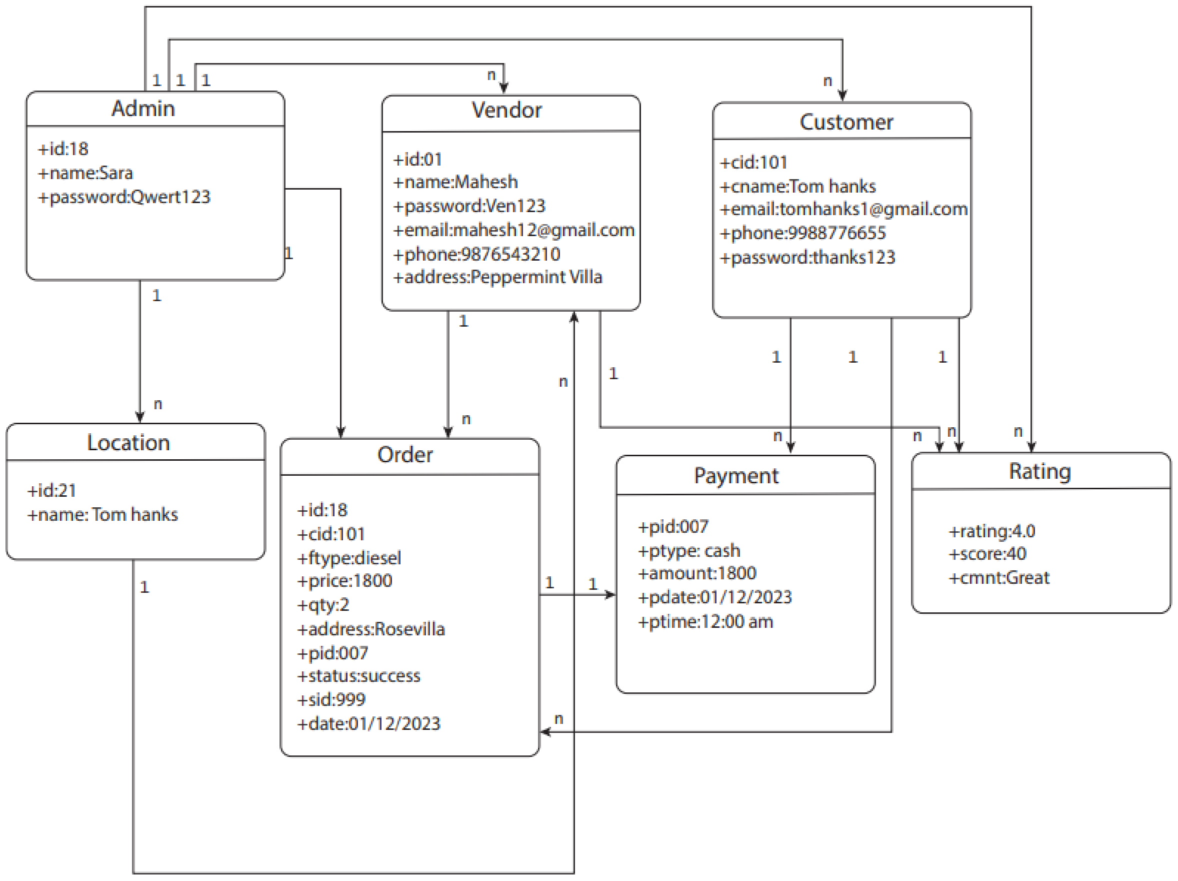


Fig.6 Object diagrams for ‘Hybrid Energy’

4.2.7 Component Diagram

A component diagram in UML illustrates how various components are interconnected to create larger components or software systems. It is an effective tool for representing the structure of complex systems with multiple components. By using component diagrams, developers can easily visualize the internal structure of a software system and understand how different components work together to accomplish a specific task. Its key components include:

- Component: A modular and encapsulated unit of functionality in a system that offers interfaces to interact with other components. It is represented as a rectangle with the component name inside.
- Interface: A contract between a component and its environment or other components, specifying a set of methods that can be used by other components. It is represented as a circle with the interface name inside.
- Port: A point of interaction between a component and its environment or other components. It is represented as a small square on the boundary of a component.
- Connector: A link between two components that enables communication or data exchange. It is represented as a line with optional adornments and labels.
- Dependency: A relationship between two components where one component depends on another for its implementation or functionality. It is represented as a dashed line with an arrowhead pointing from the dependent component to the independent component.
- Association: A relationship between two components that represents a connection or link. It is represented as a line connecting two components with optional directionality, multiplicity, and role names.
- Provided/Required Interface: A provided interface is an interface that a component offers to other components, while a required interface is an interface that a component needs from other components to function properly. These are represented by lollipops and half circles respectively.

Component diagrams are useful for modeling the architecture of a software system, and can help identify potential issues and improvements in the design. They can also be used to communicate the structure and behavior of a system to stakeholders, such as developers and project managers.

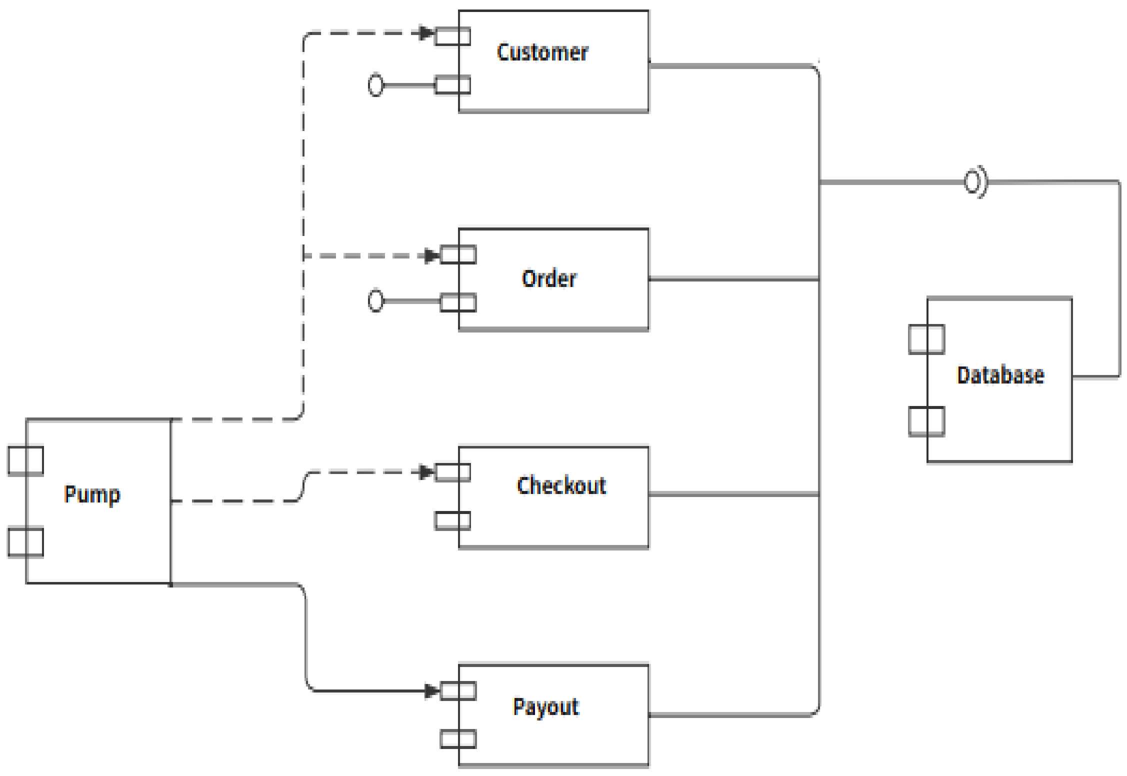


Fig.7 Component Diagram for ‘Hybrid Energy’

4.2.8 Deployment Diagram

A deployment diagram is a type of UML diagram that focuses on the physical hardware used to deploy software. It provides a static view of a system's deployment and involves nodes and their relationships. The deployment diagram maps the software architecture to the physical system architecture, showing how the software will be executed on nodes. Communication paths are used to illustrate the relationships between the nodes. Unlike other UML diagram types, which focus on the logical components of a system, the deployment diagram emphasizes the hardware topology. The key components of a deployment diagram are:

- Node - A node is a physical or virtual machine on which a component or artifact is deployed. It is represented by a box with the node's name inside.
- Component - A component is a software element that performs a specific function or provides a specific service. It is represented by a rectangle with the component's name inside.
- Artifact - An artifact is a physical piece of data that is used or produced by a component. It is represented by a rectangle with the artifact's name inside.
- Deployment Specification - A deployment specification describes how a component or artifact is deployed on a node. It includes information about the location, version, and configuration parameters of the component or artifact.
- Association - An association is a relationship between a node and a component or artifact that

represents a deployment dependency. It is represented by a line connecting the two components with optional directionality, multiplicity, and role names.

- Communication Path - A communication path represents the connection between nodes, such as a network connection or communication channel. It is represented by a line with optional labels and adornments.

Deployment diagrams help in visualizing the physical architecture of a system and identifying any potential issues or bottlenecks in the deployment process. They also aid in planning the deployment strategy and optimizing the use of hardware resources.

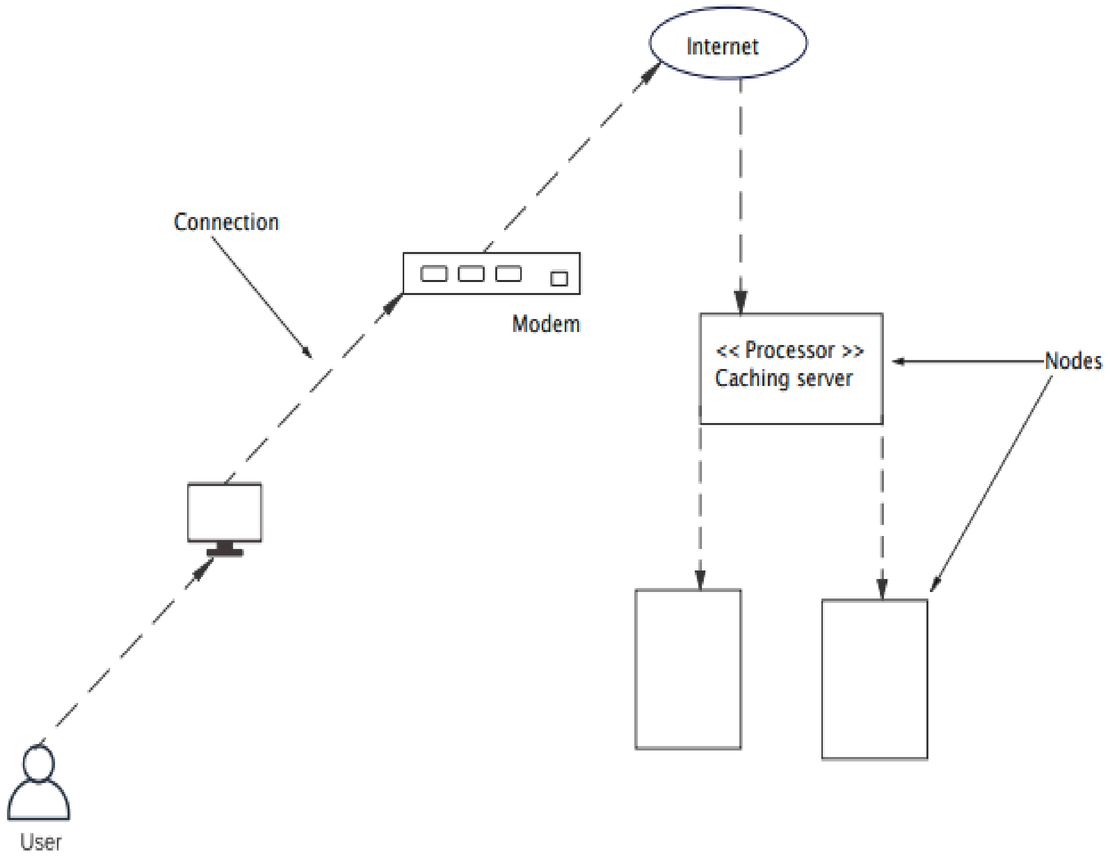
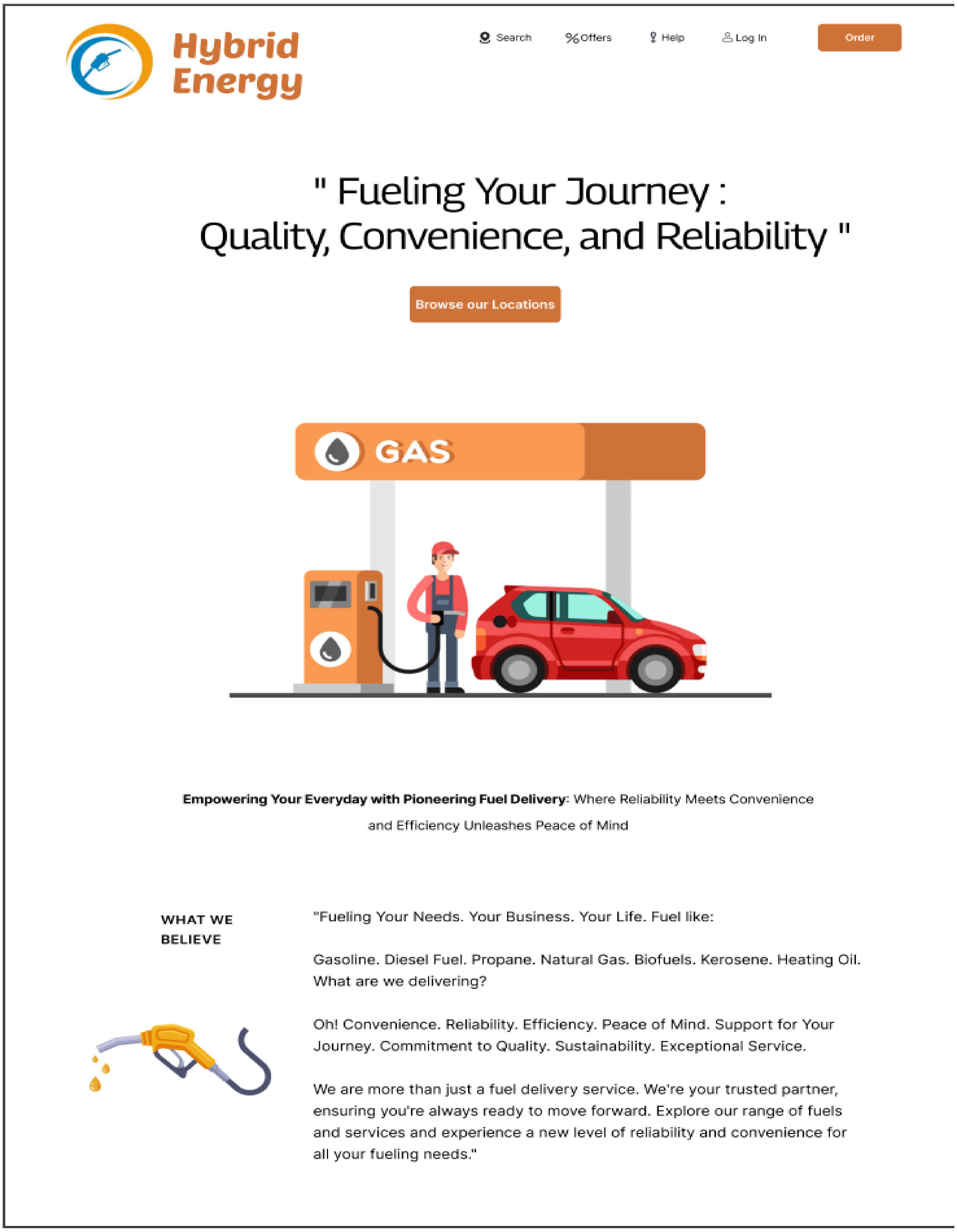


Fig.8 Deployment diagram for 'Hybrid Energy'

4.3 USER INTERFACE DESIGN USING FIGMA

Form Name: Home page



" Fueling Your Journey :
Quality, Convenience, and Reliability "

Empowering Your Everyday with Pioneering Fuel Delivery: Where Reliability Meets Convenience and Efficiency Unleashes Peace of Mind

WHAT WE BELIEVE

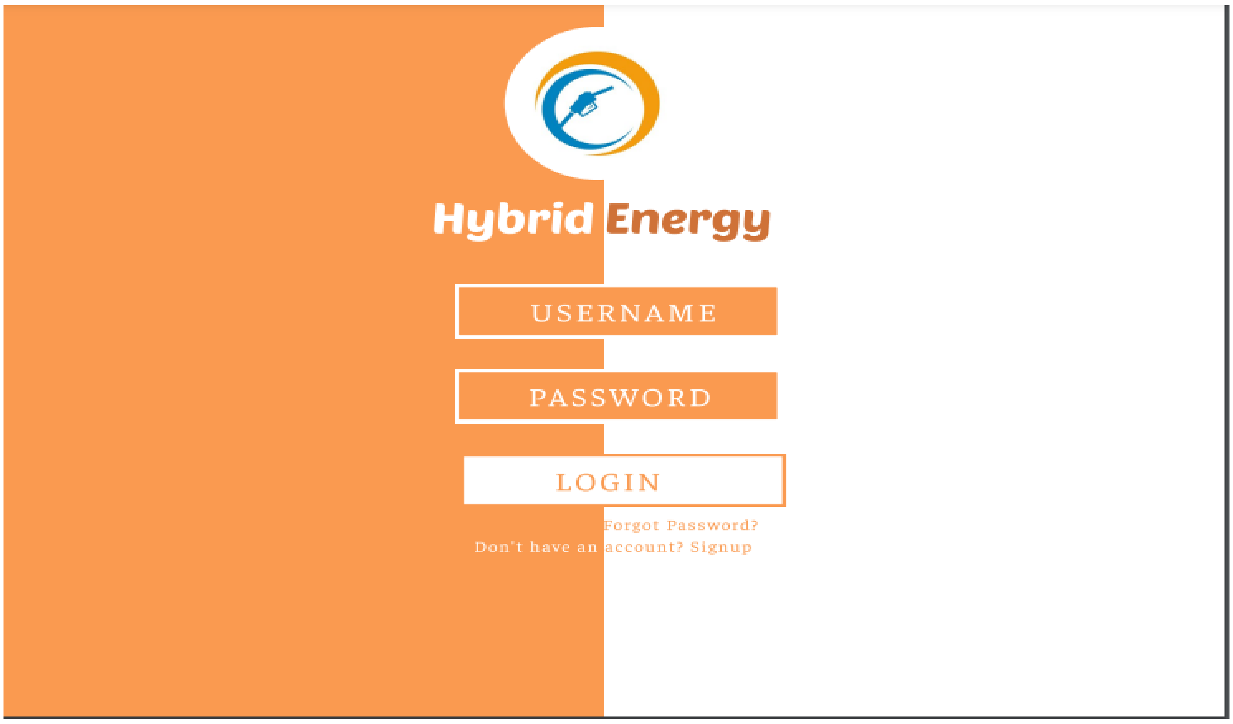
"Fueling Your Needs. Your Business. Your Life. Fuel like:

Gasoline. Diesel Fuel. Propane. Natural Gas. Biofuels. Kerosene. Heating Oil. What are we delivering?

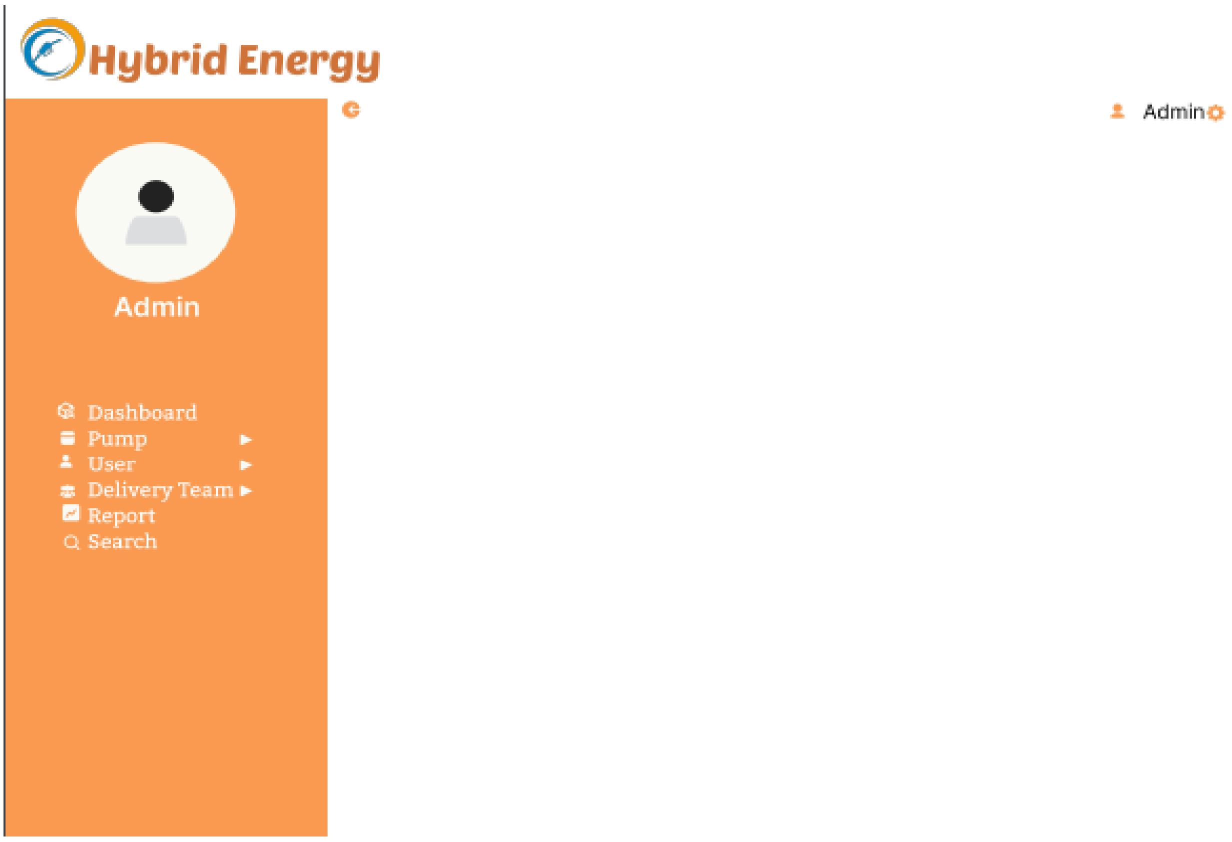
Oh! Convenience. Reliability. Efficiency. Peace of Mind. Support for Your Journey. Commitment to Quality. Sustainability. Exceptional Service.

We are more than just a fuel delivery service. We're your trusted partner, ensuring you're always ready to move forward. Explore our range of fuels and services and experience a new level of reliability and convenience for all your fueling needs."

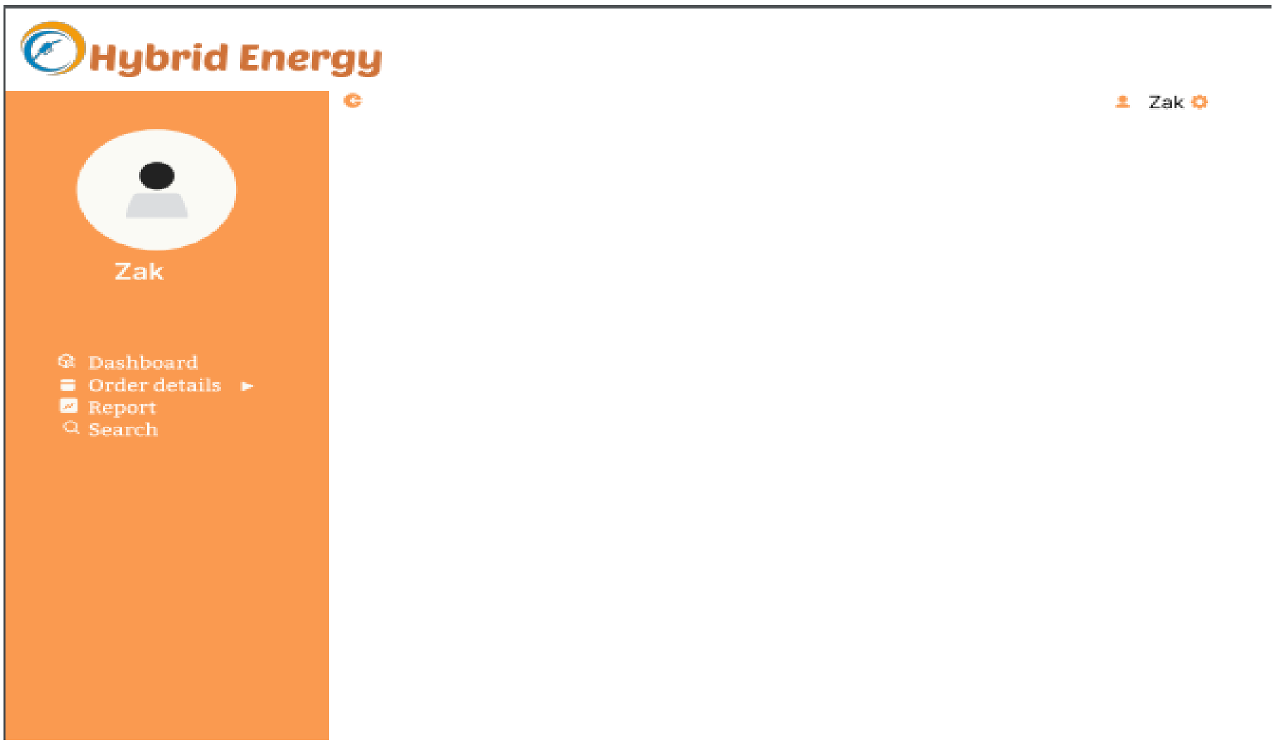
Form Name: Login



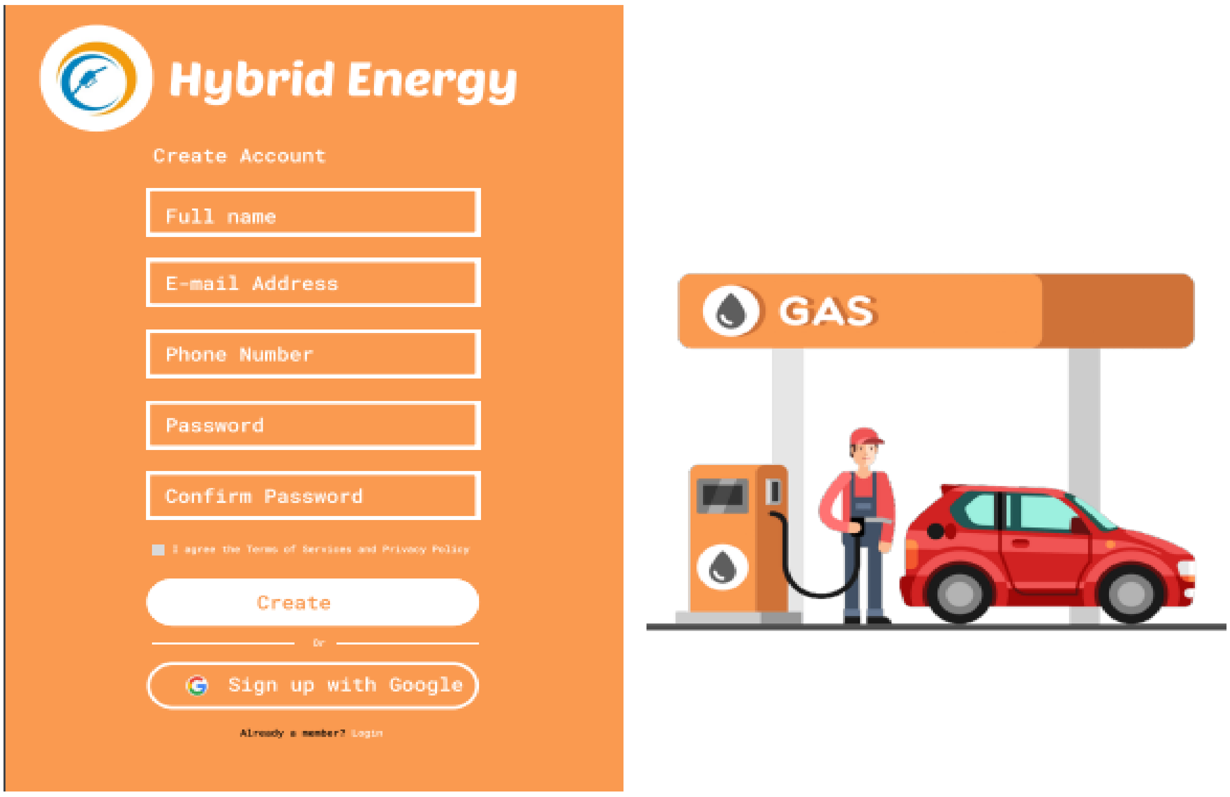
Form Name: Admin Dashboard



Form Name: Pump Dashboard



Form Name: Register Form



4.4 DATABASE DESIGN

A database is an organized collection of information that's organized to enable easy accessibility, administration, and overhauls. The security of information could be a essential objective of any database. The database design process comprises of two stages. In the first stage, user requirements are gathered to create a database that meets those requirements as clearly as possible. This is known as information-level design and is carried out independently of any DBMS. In the second stage, the design is converted from an information-level design to a specific DBMS design that will be used to construct the system. This stage is known as physical-level design, where the characteristics of the specific DBMS are considered. Alongside system design, there is also database design, which aims to achieve two main goals: data integrity and data independence.

4.4.1 Relational Database Management System (RDBMS)

Data is represented via a relational model as a set of relationships. Every relationship is like a table of values or datasets. In relational model terminology, rows are called tuples, column headings are called attributes, and tables are relations. A relational file consists of tables, each of which is given a unique name. A row in the chart represents a group of related values.

4.4.2 Normalization

The simplest possible grouping of data is used to put them together so that future changes can be made with little influence on the data structures. The formal process of normalizing data structures in a way that reduces duplication and fosters integrity. Using the normalization technique, superfluous fields are removed and a huge table is divided into several smaller ones. Anomalies in insertion, deletion, and updating are also prevented by using it. Keys and relationships are two notions used in the standard form of data modelling. A row in a table is uniquely identified by a key. Primary keys and foreign keys are two different kinds of keys. Primary key is an element, or set of components, in a table that serves as a means of distinguishing between records from the same table. A column in a table known as a foreign key is used to uniquely identify records from other tables. Up to the third normal form, all tables have been normalized. Normalization is a process in database design that aims to organize data into proper tables and columns, making it easily correlated to the data by the user. This process eliminates data redundancy that can be a burden on computer resources. The main steps involved in normalization include:

- Normalizing the data
- Choosing appropriate names for tables and columns
- Choosing the correct names for the data

By following these steps, a developer can create a more efficient and organized database that is easier to manage and maintain.

First Normal Form

The First Normal Form (1NF) requires that each attribute in a table must contain only atomic or indivisible values. It prohibits the use of nested relations or relations within relations as attribute values within tuples. To satisfy 1NF, data must be moved into separate tables where the data is of similar type in each table, and each table should have a primary key or foreign key as required by the project. This process eliminates repeating groups of data and creates new relations for each non-atomic attribute or nested relation. A relation is in 1NF only if it satisfies the constraints that contain the primary key only

Second Normal Form

Second normal form (2NF) is a rule in database normalization that states that non-key attributes should not be functionally dependent on only the part of the primary key in a relation that has a composite primary key. In other words, each non-key attribute should depend on the entire primary key, not just a part of it. To achieve this, we need to decompose the table and create new relationships for each subkey along with their dependent attributes. It is important to maintain the relationship with the original primary key and all attributes that are fully functionally dependent on it. A relation is said to be in 2NF only if it satisfies all the 1NF conditions for the primary key and every non-primary key attribute of the relation is fully dependent only on the primary key.

Third Normal Form

Third normal form (3NF) requires that a relation have no non-key attribute that is functionally determined by another non-key attribute or set of non-key attributes. This means that there should be no transitive dependency on the primary key. To achieve 3NF, we decompose the relation and set up a new relation that includes non-key attributes that functionally determine other non-key attributes. This helps eliminate any dependencies that do not just rely on the primary key. A relation is considered a relation in 3NF if it satisfies the conditions of 2NF and, moreover, the non-key attributes of the relation are not dependent on any other non-key attribute.

4.4.3 Sanitization

Data sanitization entails removing or erasing data from a storage device on purpose and permanently to make sure it cannot be restored. Normally, when data is removed from storage media, the medium is not truly wiped, and an attacker who gains access to the device can recover the data. Serious questions about security and data privacy are raised by this. Sanitization involves cleaning storage media so that no data remains on the device and that no data can be recovered even with cutting-edge forensic techniques.

4.4.4 Indexing

By reducing the number of disc accesses needed when a query is completed, indexing helps a database perform better. It is a data structure method used to locate and access data in a database rapidly. Several database columns are used to generate indexes.

- The search key, which is the first column, contains a duplicate of the table's primary key or potential primary key. These values are kept in sorted order so that it is easy to get the corresponding data. Note: The data may or may not be kept in sorted order.
- The address of the disc block on which that specific key value is stored is held by a series of pointers in the second column, which is designated as the Data Reference or Pointer.

4.5 TABLE DESIGN

1. **Table Name: CustomUser**

   Primary key: **User_ID**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	user_ID	INT (10)	PRIMARY_KEY	User_ID
2	username	CharField (32)	UNIQUE	Username
3	password	EmailField	NOT_NULL	Password
4	email	CharField (32)	UNIQUE	Email ID
5	is_customer	BooleanField	NOT_NULL	Role as Customer
6	is_vendor	BooleanField	NOT_NULL	Role as Vendor
7	is_superuser	BooleanField	NOT_NULL	Role as Admin

2. **Table Name: UserProfile**

   Primary key: **PID**

   Foreign key**:User_id** references **tbl_CustomUser**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	PID	INT (10)	PRIMARY_KEY	P_ID
2	profile_picture	ImageField	NOT_NULL	Profile image
3	date_added	DateTimeField	NOT_NULL	Date Added
4	User_id	INT (10)	FOREIGN KEY	Reference to CustomUser

3. **Table Name: FuelStation**

Primary key: **FSID**

Foreign key**:User_id** references **tbl_CustomUser, Loc_ID** references **LocationDetails**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	FSID	INT (10)	PRIMARY_KEY	FSID
2	User_id	INT (10)	FOREIGN KEY	Reference to CustomUser
3	station_name	CharField (32)	NOT_NULL	Station name
4	ownername	CharField (32)	NOT_NULL	Owner name
5	address	CharField (102)	NOT_NULL	Address
6	Loc_ID	INT (10)	FOREIGN KEY	Reference to LocationDetails
7	email	EmailField	UNIQUE	Email ID
8	Phone_number	CharField (32)	NOT_NULL	Phone number
9	Gst_number	CharField (32)	UNIQUE	Gst number
10	Logo_image	ImageField	NOT_NULL	Logo image

4. **Table Name: Fuel**

Primary key: **FID**

Foreign key**: FSID** references **tbl_FuelStation**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	FID	INT (10)	PRIMARY_KEY	FID
2	Fuel Type	CharField (32)	NOT_NULL	Fuel type
3	Price	DecimalField	NOT_NULL	Price
4	FSID	INT (10)	FOREIGN KEY	Reference to FuelStation

5. **Table Name: LocationDetails**

Primary key: **Loc_ID**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Loc_ID	INT (10)	PRIMARY_KEY	Loc_ID
2	Location name	CharField (32)	UNIQUE	Location name

6. **Table Name: Order**

Primary key: **OID**

Foreign key**: User_id** references **tbl_CustomUser,FSID** references **tbl_FuelStation, FID** references **tbl_Fuel**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	OID	INT (10)	PRIMARY_KEY	OID
2	User_id	INT (10)	FOREIGN KEY	Reference to CustomUser
3	FSID	INT (10)	FOREIGN KEY	Reference to FuelStation
4	FID	INT (10)	FOREIGN KEY	Reference to Fuel
5	quantity	DecimalField	NOT_NULL	Quantity
6	Total_price	DecimalField	NOT_NULL	Total price
7	Order_date	DateTimeField	NOT_NULL	Date
8	is_delivered	BooleanField	NOT_NULL	Status
9	is_accepted	BooleanField	NOT_NULL	Status
10	is_ordered	BooleanField	NOT_NULL	Status
11	is_rejected	BooleanField	NOT_NULL	Status
12	is_active	BooleanField	NOT_NULL	Status
13	delivery_address	CharField (32)	NOT_NULL	Delivery Address
14	payment_method	CharField (32)	NOT_NULL	Payment Method

7. **Table Name: Payment**

Primary key: **Pay_ID**

Foreign key**: User_id** references **tbl_CustomUser,OID** references **tbl_Order**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Pay_ID	INT (10)	PRIMARY_KEY	Pay_ID
2	OID	INT (10)	FOREIGN KEY	Reference to Order
3	is_paid	BooleanField	NOT_NULL	Status
4	razor_pay_order_id	CharField (32)	NOT_NULL	Razorpay OrderID
5	User_id	INT (10)	FOREIGN KEY	Reference to CustomUser
6	amount	DecimalField	NOT_NULL	Amount
7	payment_date	DateTimeField	NOT_NULL	Date

8. Table Name: Rating

Primary key: **RID**

Foreign key**: User_id** references **tbl_CustomUser,FSID** references **tbl_FuelStation**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	RID	INT (10)	PRIMARY_KEY	RID
2	User_id	INT (10)	FOREIGN KEY	Reference to CustomUser
3	FSID	INT (10)	FOREIGN KEY	Reference to FuelStation
4	value	INT (10)	NOT_NULL	Value
5	comment	TextField	NOT_NULL	Comment

CHAPTER 5

SYSTEM TESTING

5.1 INTRODUCTION

Software Testing is the process of executing software in a controlled manner, in order to answer the question - Does the software behave as specified? Software testing is often used in association with the term's verification and validation. Validation is the checking or testing of items, includes software, for conformance and consistency with an associated specification. Software testing is just one kind of verification, which also uses techniques such as reviews, analysis, inspections, and walkthroughs. Validation is the process of checking that what has been specified is what the user actually wanted.

5.2 TEST PLAN

An extensive document that describes the testing strategy and approach for a specific project or product is called a test plan. Selenium is one of the most popular tools for automating tests when it comes to web application testing. A test plan suggests a number of required steps that need be taken in order to complete various testing methodologies. Therefore, the test plan should include information on the mean time to failure, the cost to locate and correct the flaws, the residual defect density or frequency of occurrence, and the number of test work hours required for each regression test. The testing levels include:

- Unit testing
- Integration Testing
- Data validation Testing
- Output Testing

Overall, defining the scope, identifying the test environment, defining the test cases, generating the test scripts, running the tests, recording the results, analyzing the results, and reporting the results are all steps in the Selenium test plan creation process. By doing the following actions, you can make sure that your testing is thorough and efficient and that you are able to spot and fix any issues before they become serious ones.

5.2.1 Unit Testing

Unit testing is a software testing technique that focuses on verifying individual components or modules of the software design. The purpose of unit testing is to test the smallest unit of software design and ensure that it performs as intended. Unit testing is typically white-box focused, and multiple components can be tested simultaneously. The component-level design description is used as a guide during testing to identify critical control paths and potential faults within the module's perimeter. During unit testing, the modular interface is tested to ensure that data enters and exits the software unit under test properly. The local data structure is inspected to ensure that data temporarily stored retains its integrity during each step of an algorithm's execution. Boundary conditions are tested to ensure that all statements in a module have been executed at least once, and all error handling paths are tested to ensure that the software can handle errors correctly. Before any other testing can take place, it is essential to test data flow over a module interface. If data cannot enter and exit the system properly, all other tests are irrelevant. Another crucial duty during unit testing is the selective examination of execution pathways to anticipate potential errors and ensure that error handling paths are set up to reroute or halt work when an error occurs. Finally, boundary testing is conducted to ensure that the software operates correctly at its limits. In the Hybrid Energy System, unit testing was carried out by treating each module as a distinct entity and subjecting them to a variety of test inputs. Any issues with the internal logic of the modules were fixed, and each module was tested and run separately after coding. Unused code was eliminated, and it was confirmed that every module was functional and produced the desired outcome.

5.2.2 Integration Testing

Integration testing is a systematic approach that involves creating the program structure while simultaneously conducting tests to identify interface issues. The objective is to construct a program structure based on the design that uses unit-tested components. The entire program is then tested. Correcting errors in integration testing can be challenging due to the size of the overall program, which makes it difficult to isolate the causes of the errors. As soon as one set of errors is fixed, new ones may arise, and the process may continue in an apparently endless cycle. Once unit testing is complete for all modules in the system, they are integrated to check for any interface inconsistencies. Any discrepancies in program structures are resolved, and a unique program structure is developed.

5.2.3 Validation Testing or System Testing

The final stage of the testing process involves testing the entire software system as a whole, including all forms, code, modules, and class modules. This is commonly referred to as system testing or black box testing. The focus of black box testing is on testing the functional requirements of the software. A software engineer can use this approach to create input conditions that will fully test each program requirement. The main types of errors targeted by black box testing include incorrect or missing functions, interface errors, errors in data structure or external data access, performance errors, initialization errors, and termination errors.

5.3.4 Output Testing or User Acceptance Testing

User approval of the system under consideration is tested; in this case, it must meet the needs of the company. When developing, the program should stay in touch with the user and perspective system to make modifications as needed. With regard to the following points, this was done: • Input Screen Designs, • Output Screen Designs, The aforementioned testing is carried out using a variety of test data. The preparation of test data is essential to the system testing process. The system under investigation is then put to the test using the prepared test data. When testing the system, test data issues are found again and fixed using the testing procedures described above. The fixes are also logged for use in the future.

5.2.5 Automation Testing

A test case suite is executed using specialized automated testing software tools as part of the software testing technique known as automation testing. The test stages are meticulously carried out by a human performing manual testing while seated in front of a computer. Additionally, the automation testing software may generate thorough test reports, compare expected and actual findings, and enter test data into the System Under Test. Software test automation necessitates significant financial and material inputs. Repeated execution of the same test suite will be necessary during subsequent development cycles.

5.2.6 Selenium Testing

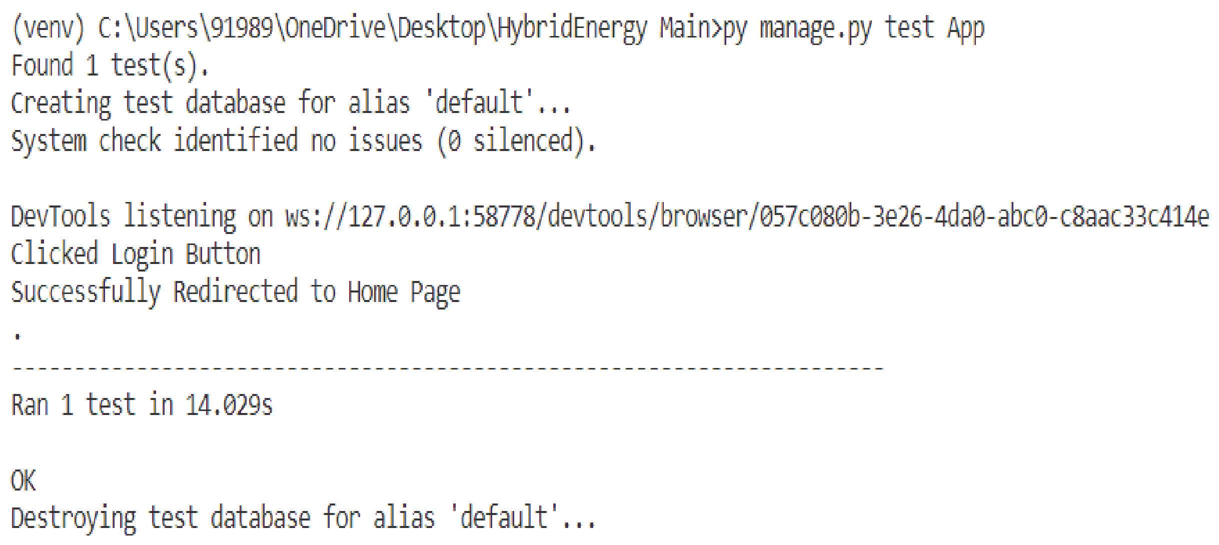
Selenium is a free (open-source) automated testing framework used to verify web applications across different browsers and platforms. You can use multiple programming languages to create Selenium test scripts like Java, C#, Python, etc. Testing done using Selenium testing tool is usually referred to as Selenium testing Since Selenium is a collection of different tools, it also had different developers. Below are the key people who have made significant contributions to the Selenium project. Moreover, Selenium's extensibility allows integration with other testing frameworks, tools, and technologies. It can be combined with testing frameworks like TestNG or JUnit for advanced test management and reporting. It integrates well with popular build tools, source control systems, and defect tracking systems.

Test Case 1: Customer-Login

Code

```
from django.test import TestCase
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
class LoginTestCase(TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome()  # Initialize Chrome WebDriver
        self.driver.implicitly_wait(10)   # Implicit wait for elements

    def tearDown(self):
        self.driver.quit()  # Close the browser session after each test

    def test_login(self):
        self.driver.get('http://127.0.0.1:8000/login/')
        username_input = self.driver.find_element(By.NAME, 'username')
        password_input = self.driver.find_element(By.NAME, 'password')
        login_button = self.driver.find_element(By.ID, 'logout-link')
        print('Clicked Login Button')
        username_input.send_keys('amil')
        password_input.send_keys('Amil@123')
        login_button.click()
        WebDriverWait(self.driver, 10).until(EC.url_contains('userhome'))
        print("Successfully Redirected to Home Page")
```

Screenshot



```
(venv) C:\Users\91989\OneDrive\Desktop\HybridEnergy Main>py manage.py test App
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

DevTools listening on ws://127.0.0.1:58778/devtools/browser/057c080b-3e26-4da0-abc0-c8aac33c414e
Clicked Login Button
Successfully Redirected to Home Page
.
----------------------------------------------------------------------
Ran 1 test in 14.029s

OK
Destroying test database for alias 'default'...
```

Test Report

Test Case 1	
Project Name: HYBRID ENERGY	
Login Test Case	
Test Case ID: Test_1	**Test Designed By:** Anina Elizebeth
Test Priority(Low/Medium/High): High	**Test Designed Date:** 25-11-2023
Module Name: Login	**Test Executed By:** Mr. Ajith G.S
Test Title: Customer Login	**Test Execution Date:** 27-11-2023
Description: Verify login with valid username and password	
Pre-Condition: User has valid username and password	

Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/ Fail)**
1	Navigation to Login Page		Dashboard should be displayed	Login page displayed	Pass
2	Provide Valid Username	Username:anina	User should be able to login	User Logged in and navigated to User Dashboard	Pass
3	Provide Valid Password	Password: Anina@123			
4	Click on Login button				

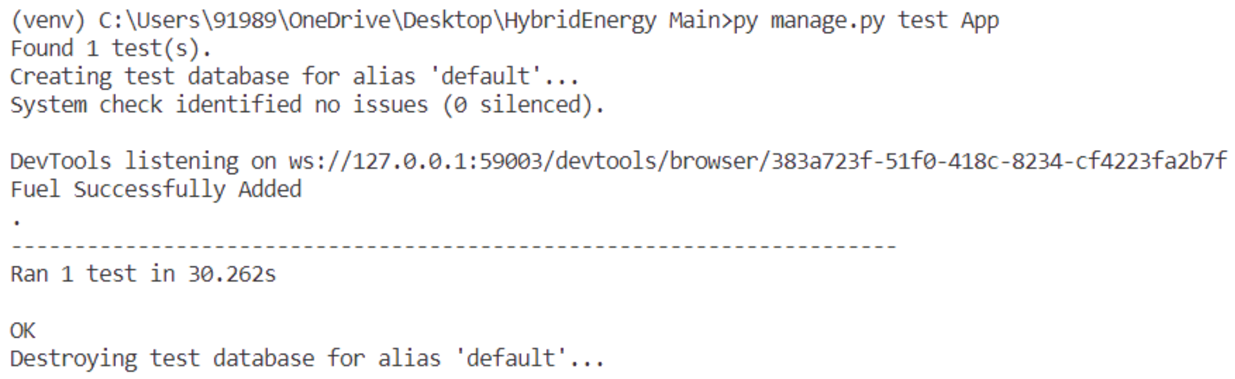
Post-Condition: User is validated with database and successfully login into account. The Account session details are logged in database.

Test Case 2: Add Fuel

Code

```
from django.test import TestCase
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time  # Add the time library

class WebAppTests(TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome()
        self.driver.implicitly_wait(10)

    def tearDown(self):
        self.driver.quit()

    def login(self):
        self.driver.get('http://127.0.0.1:8000/login/')
        username_input = self.driver.find_element(By.NAME, 'username')
        password_input = self.driver.find_element(By.NAME, 'password')
        login_button = self.driver.find_element(By.ID, 'logout-link')
        username_input.send_keys('gayathri')  # Replace with your actual username
        password_input.send_keys('Gayathri@123')  # Replace with your actual password
        login_button.click()
        WebDriverWait(self.driver,
10).until(EC.url_to_be('http://127.0.0.1:8000/pumphome/'))

    def add_fuel(self):
        self.driver.get('http://127.0.0.1:8000/fuel/')
        fuel_type = self.driver.find_element(By.NAME, 'fueltype')
        fuel_type.send_keys('Diesel')
        price = self.driver.find_element(By.NAME, 'price')
        price.send_keys('50')
        add_fuel_button                    =                    self.driver.find_element(By.XPATH,
"//button[contains(text(),'Add Fuel')]")
        add_fuel_button.click()
        WebDriverWait(self.driver,
10).until(EC.presence_of_element_located((By.XPATH,
"//table//tr/td[contains(text(),'Diesel')]")))

    def test_login_and_add_fuel(self):
        self.login()
        self.add_fuel()
```

Screenshot


```
(venv) C:\Users\91989\OneDrive\Desktop\HybridEnergy Main>py manage.py test App
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

DevTools listening on ws://127.0.0.1:59003/devtools/browser/383a723f-51f0-418c-8234-cf4223fa2b7f
Fuel Successfully Added
.
----------------------------------------------------------------------
Ran 1 test in 30.262s

OK
Destroying test database for alias 'default'...
```

Test report

Test Case 2

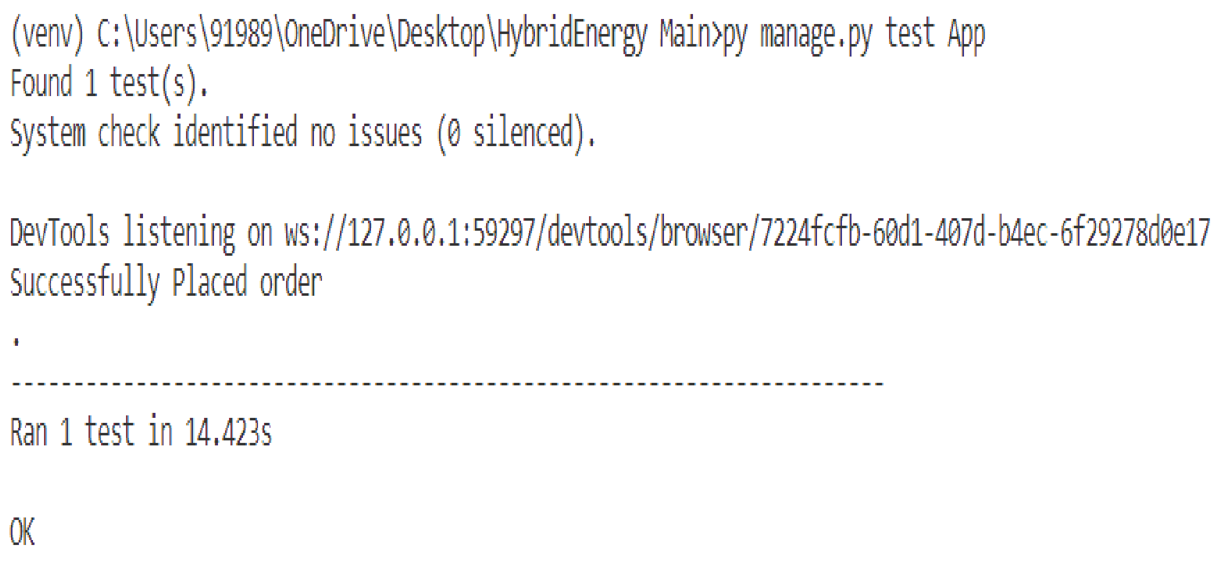
Project Name: HYBRID ENERGY					
Add Fuel Test Case					
Test Case ID: Test_2			**Test Designed By:** Anina Elizebeth		
Test Priority(Low/Medium/High): High			**Test Designed Date:** 25-11-2023		
Module Name: Add Fuel			**Test Executed By:** Mr. Ajith G.S		
Test Title : Add Fuel type and price in the form			**Test Execution Date:** 27-11-2023		
Description: Add Fuel by Vendor from dashboard					
Pre-Condition : Vendor has Fuel to add					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigation to Fuel Page		Dashboard should be displayed	Dashboard is displayed	Pass
2	Add a Fuel type	Fuel Type: Petrol	Fuel details should be added	Fuel details are added	Pass
3	Add a Fuel price	Fuel Price: 50			
4	Click on the Submit button				
Post-Condition: Vendor has added the Fuel. Fuel details are added to the database.					

Test Case 3: Place Order

Code

```
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support.ui import Select
from selenium.webdriver.support import expected_conditions as EC

class UserInterfaceTests(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome()
        self.base_url = "http://localhost:8000"
        self.driver.implicitly_wait(50)

    def test_login_and_place_order(self):
        self.driver.get(self.base_url + '/login')
        self.assertIn("Login Form", self.driver.title)
        username = self.driver.find_element(By.NAME, 'username')
        password = self.driver.find_element(By.NAME, 'password')
        username.send_keys("godwin")
        password.send_keys("Godwin@123")
        self.driver.find_element(By.ID, "logout-link").click()
        self.driver.implicitly_wait(5)
        self.assertIn("userhome", self.driver.current_url)
        place_order_button = self.driver.find_element(By.ID, "placeorder")
        self.driver.execute_script("arguments[0].scrollIntoView();", place_order_button)
        place_order_button = WebDriverWait(self.driver, 
10).until(EC.element_to_be_clickable((By.ID, "placeorder")))
        place_order_button.click()
        self.assertIn("place_order", self.driver.current_url)
        fuel_type_dropdown = Select(self.driver.find_element(By.ID, 'fuel_type'))
        fuel_type_dropdown.select_by_visible_text('petrol')
        quantity_dropdown = Select(self.driver.find_element(By.ID, 'quantity'))
        quantity_dropdown.select_by_value('1')
        delivery_point_input = self.driver.find_element(By.ID, 'delivery_point')
        delivery_point_input.send_keys('123 Example St, City')

        payment_method_dropdown=Select(self.driver.find_element(By.ID,
'payment_method'))
        payment_method_dropdown.select_by_visible_text('Credit Card')
        place_order_button = self.driver.find_element(By.ID, 'submit-button')
        self.driver.execute_script("arguments[0].scrollIntoView();", place_order_button)
        place_order_button.click()
        self.assertIn("ordersummary", self.driver.current_url)

    def tearDown(self):
        self.driver.quit()

if __name__ == "__main__":
    unittest.main()
```

Screenshot



```
(venv) C:\Users\91989\OneDrive\Desktop\HybridEnergy Main>py manage.py test App
Found 1 test(s).
System check identified no issues (0 silenced).

DevTools listening on ws://127.0.0.1:59297/devtools/browser/7224fcfb-60d1-407d-b4ec-6f29278d0e17
Successfully Placed order
.
----------------------------------------------------------------------
Ran 1 test in 14.423s

OK
```

Test report

Test Case 3

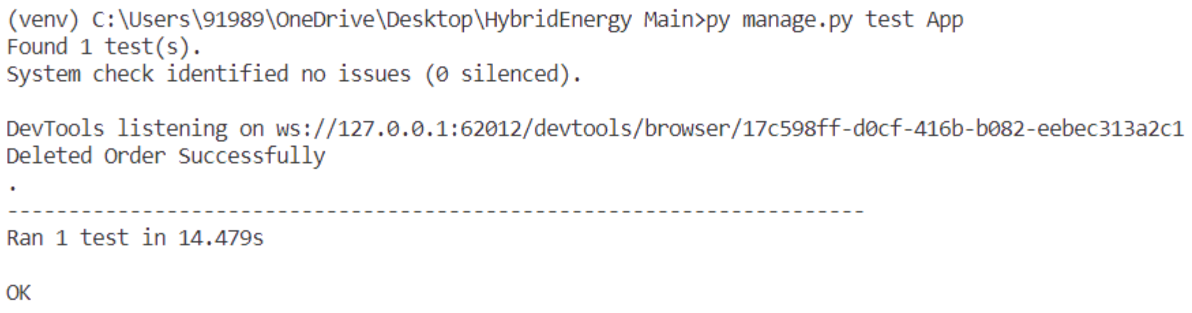
Project Name: HYBRID ENERGY					
Place Order Test Case					
Test Case ID: Test_3			**Test Designed By:** Anina Elizebeth		
Test Priority(Low/Medium/High): High			**Test Designed Date:** 25-11-2023		
Module Name: Place order			**Test Executed By:** Mr. Ajith G.S		
Test Title : Place Order			**Test Execution Date:** 27-11-2023		
Description: Place order by the Customer					
Pre-Condition: Customer has order to place					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/ Fail)**
1	Navigation to Customer Dashboard		Dashboard should be displayed	Dashboard is displayed	Pass
2	Select a Fuel type	Fuel Type: Petrol	Order details should be added	Order placed	Pass
3	Select Quantity	Quantity: 2			
4	Add Delivery Point	Delivery Point: Rose Villa,West fort ,Thrissur			
5	Select Payment method	Payment Method:Pay Pal			
6	Click on Place Order Button				
Post-Condition: New Order Placed Successfully.					

Test Case 4: Delete Order

Code

```
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By

class PlaceOrderAfterLoginTest(unittest.TestCase):
  def setUp(self):
    self.driver = webdriver.Chrome()
    self.base_url = "http://localhost:8000"
    self.driver.maximize_window()
    self.driver.implicitly_wait(10)

  def test_login_and_place_order(self):
    self.driver.get(self.base_url + '/login')
    self.assertIn("Login Form", self.driver.title)

    username = self.driver.find_element(By.NAME, 'username')
    password = self.driver.find_element(By.NAME, 'password')
    username.send_keys("godwin")
    password.send_keys("Godwin@123")
    self.driver.find_element(By.ID, "logout-link").click()
    self.driver.implicitly_wait(5)
    self.driver.get(self.base_url + '/customer_unorders')
    place_order_button = self.driver.find_element(By.CLASS_NAME, 'order-action-form')
    place_order_button.click()
    self.assertIn("customer_orders", self.driver.current_url)

  def tearDown(self):
    self.driver.quit()

if __name__ == "__main__":
  unittest.main()
```

Screenshot



```
(venv) C:\Users\91989\OneDrive\Desktop\HybridEnergy Main>py manage.py test App
Found 1 test(s).
System check identified no issues (0 silenced).

DevTools listening on ws://127.0.0.1:62012/devtools/browser/17c598ff-d0cf-416b-b082-eebec313a2c1
Deleted Order Successfully
.
----------------------------------------------------------------------
Ran 1 test in 14.479s

OK
```

Test report

Test Case 4

Project Name: HYBRID ENERGY					
Delete Order Test Case					
Test Case ID: Test_4			**Test Designed By:** Anina Elizebeth		
Test Priority (Low/Medium/High): High			**Test Designed Date:** 25-11-2023		
Module Name: Delete order			**Test Executed By:** Mr. Ajith G.S		
Test Title : Delete Order			**Test Execution Date:** 27-11-2023		
Description: Delete order by the Customer					
Pre-Condition : Customer has order to Delete					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigation to Customer Dashboard		Dashboard should be displayed	Dashboard is displayed	Pass
2	Select an Order		Order details should be deleted	Order deleted	Pass
3	Click on delete button				
Post-Condition: Order Deleted Successfully.					

CHAPTER 6

IMPLEMENTATION

6.1 INTRODUCTION

The implementation phase of a project is where the design is transformed into a functional system. It is a crucial stage in ensuring the success of the new system, as it requires gaining user confidence that the system will work effectively and accurately. User training and documentation are key concerns during this phase. Conversion may occur concurrently with user training or at a later stage. Implementation involves the conversion of a newly revised system design into an operational system. During this stage, the user department bears the primary workload, experiences the most significant upheaval, and has the most substantial impact on the existing system. Poorly planned or controlled implementation can cause confusion and chaos. Whether the new system is entirely new, replaces an existing manual or automated system, or modifies an existing system, proper implementation is essential to meet the organization's needs. System implementation involves all activities required to convert from the old to the new system. The system can only be implemented after thorough testing is done and found to be working according to specifications. System personnel evaluate the feasibility of the system. Implementation requires extensive effort in three main areas: education and training, system testing, and changeover. The implementation phase involves careful planning, investigating system and constraints, and designing methods to achieve changeover.

6.2 IMPLEMENTATION PROCEDURES

Software implementation is the process of installing the software in its actual environment and ensuring that it satisfies the intended use and operates as expected. In some organizations, the software development project may be commissioned by someone who will not be using the software themselves. During the initial stages, there may be doubts about the software, but it's important to ensure that resistance does not build up.

This can be achieved by:

- Ensuring that active users are aware of the benefits of the new system, building their confidence in the software.
- Providing proper guidance to the users so that they are comfortable using the application. Before viewing the system, users should know that the server program must be running on the server. Without the server object up and running, the intended process will not take place.

6.2.1 User Training

User training is designed to prepare the user for testing and converting the system. To achieve the objective and benefits expected from computer-based system, it is essential for the people who will be involved to be confident of their role in the new system. As system becomes more complex, the need for training is more important. By user training the user comes to know how to enter data, respond to error messages, interrogate the database, and call up routine that will produce reports and perform other necessary functions.

6.2.2 Training on the Application Software

After providing the necessary basic training on computer awareness, it is essential to provide training on the new application software to the user. This training should include the underlying philosophy of using the new system, such as the flow of screens, screen design, the type of help available on the screen, the types of errors that may occur while entering data, and the corresponding validation checks for each entry, and ways to correct the data entered. Additionally, the training should cover information specific to the user or group, which is necessary to use the system or part of the system effectively. It is important to note that this training may differ across different user groups and levels of hierarchy.

6.2.3 System Maintenance

The maintenance phase is a crucial aspect of the software development cycle, as it is the time when the software is actually put to use and performs its intended functions. Proper maintenance is essential to ensure that the system remains functional, reliable, and adaptable to changes in the system environment. Maintenance activities go beyond simply identifying and fixing errors or bugs in the system. It may involve updates to the software, modifications to its functionalities, and enhancements to its performance, among other things. In essence, software maintenance is an ongoing process that requires continuous monitoring, evaluation, and improvement of the system to meet changing user needs and requirements.

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

The Hybrid Energy project embodies a transformative approach to modernize the fuel delivery landscape, focusing on user-centric interactions between customers and fuel stations. By integrating innovative technologies and fostering seamless communication, it strives to revolutionize the fuel procurement experience. With its robust user management, efficient order coordination, secure transactions, and active user engagement, Hybrid Energy sets out to address existing system limitations and cater to diverse user needs. Through continual refinement based on user feedback and technological advancements, it aspires to establish new industry benchmarks for a more efficient, secure, and user-driven fuel procurement ecosystem.

7.2 FUTURE SCOPE

The future scope of Hybrid Energy seems promising and expansive. As technology advances and user preferences evolve, Hybrid Energy can incorporate more intuitive interfaces and features, potentially integrating AI-driven predictive analysis for better fuel demand forecasting. It could expand its network of fuel stations, ensuring a wider reach and accessibility for users. Moreover, integrating renewable energy sources and exploring environmentally sustainable practices could align with future trends and eco-conscious consumer preferences. Collaboration with electric vehicle charging stations or diversifying into alternative energy sources might also broaden its offerings. Ultimately, continual innovation and adaptability to emerging technologies will be pivotal in Hybrid Energy's future success and sustainability.

.

CHAPTER 8
BIBLIOGRAPHY

REFERENCES:

- Gary B. Shelly, Harry J. Rosenblatt, “System Analysis and Design”, 2009.
- Roger S Pressman, “Software Engineering”, 1994
- IEEE Std 1016 Recommended Practice for Software Design Descriptions.

WEBSITES:

- www.w3schools.com
- https://getbootstrap.com/
- https://www.djangoproject.com/
- https://jquery.com/
- https://chat.openai.com

CHAPTER 9

APPENDIX

9.1 Sample Code

Login.html

```
{% load socialaccount %}
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.5.2/dist/css/bootstrap.min.css" >
   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.5.2/dist/js/bootstrap.min.js"></script>
   <link rel="stylesheet" href="{% static 'css/loginstyle.css'%}">
   <title>Login Form</title>
   <link rel="icon" href="{% static 'images/logo.png'%}" sizes="32x32" type="image/png">
   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.4/css/all.min.css">
</head>
<body>
{% if messages %}
   {% for message in messages %}
   <div class="alert" role="alert">
      {{ message }}
   </div>
   {% endfor %}
{% endif %}
<form method="POST">
      {% csrf_token %}
   <div class="login-container">
      <div class="left-side">
         <div class="form-container">
            <div class="logo">
               <img src="{% static 'images/logo.png'%}" alt="Logo">
            </div>
            <div class="title">
               <h1>Hybrid Energy</h1>
            </div>
            <div class="background"></div>
            <div class="input-container">
               <br>
               <br>
               <input type="text" placeholder="USERNAME" name="username">
               <input type="password" placeholder="PASSWORD" name="password">
               <button id="logout-link">LOGIN</button>
```

```
                    <p class="forgot-password"><a href="{% url 'reset_password' %}">Forgot
Password?</a></p>
                    <p class="signup">Don't have an account? <span><a href="/registerUser">Signup for
Customer</a>/<a href="/registerPump">Pump</a></span></p>

                    <div class="google"style="border: 2px solid  #FA9A50; padding: 10px; border-radius:
5px; text-align: center;">
                        <p>For Customers</p>
                        <button style="background-color:  #FA9A50;color: #ffffff; border: none; padding:
10px 20px; text-align: center; text-decoration: none; display: inline-block; font-size: 16px; margin:
4px 2px; cursor: pointer; border-radius: 4px;">
                            <a href="{% provider_login_url 'google' %}" style="text-decoration: none; color:
#ffffff;"><i class="fab fa-google" style="margin-right: 5px;"></i>Sign in with Google</a>
                        </button>
                      </div>
                </div>
                <br>
            </div>
        </div>
    </form>
    <div class="right-side">
        <div class="image-container">
            <img class="background-image" src="{% static 'images/main.png'%}" alt="Background
Image">
        </div>
    </div>
</div>
<script>
    document.addEventListener("DOMContentLoaded", function () {
      var logoutButton = document.getElementById("logout-link");
        logoutButton.addEventListener("click", function () {
            setTimeout(function () {
                location.reload();
            }, 1000); // Reload after 1 second (1000 milliseconds)
        });
    });
</script>
</body>
<style>
    .google:hover {
      background-color: #ab1313; /* Background color on hover */
      color: #ffffff; /* Text color on hover */
    }
  </style>
```

```
</html>
```

Views.py

```
def login_user(request):
    if request.user.is_authenticated:
        return redirect('/')
    if request.method == 'POST':
        username = request.POST.get("username")
        password = request.POST.get("password")
        if username and password:
            user = authenticate(request, username=username, password=password)
            if user is not None:
                auth_login(request, user)
                if user.is_customer:
                    request.session["username"] = user.username
                    return redirect('/userhome')
                elif user.is_vendor:
                    request.session["username"] = user.username
                    return redirect('/pumphome')
                elif user.is_superuser:
                    request.session["username"] = user.username
                    return redirect('/adminhome')
            else:
                messages.error(request, "Invalid credentials.")
                return redirect('/login')
        else:
            messages.error(request, "Please provide both username and password.")
            return redirect('/login')
    return render(request, 'login.html')
```

Customer Order.html

```
{% extends "userBase.html" %}
{% block title %}Customer Orders{% endblock %}
{% block content %}
    {% if messages %}
        {% for message in messages %}
            <div class="alert" role="alert">
                {{ message }}
            </div>
        {% endfor %}
    {% endif %}
    <div class="container">
        <div class="order-section">
            <h2>Orders List</h2>
            <div class="search-container">
            <input type="text" id="searchInput" placeholder="Search for Order ID..."
onkeyup="searchOrders()">
         </div>
         <table class="order-table">
            <thead>
               <tr class="order-table-header">
                  <th>#No</th>
                  <th>Order ID</th>
                  <th>Fuel Station</th>
                  <th>Date & Time</th>
                  <th>Fuel Type</th>
                  <th>Total Price</th>
                  <th>Status</th>
                  <th>Rate</th>
               </tr>
            </thead>
            <tbody>
               {% for order in ordered_orders %}
                  <tr class="order-row {% cycle 'odd' 'even' %}">
                     <td>{{ forloop.counter }}</td>
                     <td>{{ order.id }}</td>
                     <td>{{ order.station }}</td>
                     <td>{{ order.order_date }}</td>
                     <td>{{ order.fuel_type }}</td>
                     <td>{{ order.total_price }}</td>
                     <td>
                        <span class="order-status {% if order.is_accepted %}accepted{% elif
order.is_rejected %}rejected{% else %}not-accepted{% endif %}">
                           {% if order.is_accepted %}
                           Accepted
                           {% if order.payment_set.first.is_paid %}
                              <p>Paid</p>
                           {% else %}
                              <form action="{% url 'pay' order.id %}" method="post">
                                 {% csrf_token %}
                                 <input type="hidden" name="order_id" value="{{ order.id }}">
                                 <button type="submit" class="payment-btn">Pay</button>
                              </form>
                           {% endif %}
                        {% elif order.is_rejected %}
                           Rejected
                        {% else %}
                           Not Accepted
                        {% endif %}
                        </span>
                    </td>
                    <td><a href="{% url 'station_detail' order.station.id %}" class="amazon-rating-
btn">Rate</a></td>
                </tr>
                {% empty %}
                <tr>
                <td colspan="2">No Orders available.</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>


    <div class="pagination">
        <span class="step-links">
            {% if ordered_orders.has_previous %}
                <a href="?page=1">&laquo; first</a>
                <a href="?page={{ ordered_orders.previous_page_number }}">previous</a>
            {% endif %}

            <span class="current">
                Page {{ ordered_orders.number }} of {{ ordered_orders.paginator.num_pages }}.
            </span>

            {% if ordered_orders.has_next %}
                <a href="?page={{ ordered_orders.next_page_number }}">next</a>
                <a href="?page={{ ordered_orders.paginator.num_pages }}">last &raquo;</a>
            {% endif %}
        </span>
    </div>

  <style>
     /* Styles for the payment button */
.payment-btn {
  display: inline-block;
  padding: 10px 20px;
  font-size: 16px;
  font-weight: bold;
  text-align: center;
  text-decoration: none;
  border-radius: 6px;
  color: #fff;
  background-color: #4CAF50; /* Green color, you can change this */
  border: 1px solid #4CAF50;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);
  transition: background-color 0.3s ease, border-color 0.3s ease, color 0.3s ease;
}
.payment-btn:hover {
  background-color: #45a049;
  border-color: #45a049;
}
.payment-btn:active {
  transform: translateY(1px);
}
.amazon-rating-btn {
  display: inline-block;
  padding: 8px 15px;
  font-size: 14px;
  font-weight: bold;
  text-align: center;
  text-decoration: none;
  border-radius: 4px;
  color: #fff;
  background-color: #f0c14b;
  border: 1px solid #a88734;
  box-shadow: 0 2px 2px rgba(0, 0, 0, 0.1);
  transition: background-color 0.3s ease;
}
.amazon-rating-btn:hover {
  background-color: #ddb347;
}
       .search-container {
        margin-bottom: 20px;
    }
    #searchInput {
        width: 300px;
        padding: 10px;
        border: 1px solid #ccc;
        border-radius: 4px;
        font-size: 16px;
    }
        .alert {
            padding: 15px;
            margin-bottom: 20px;
            border: 1px solid transparent;
            border-radius: 4px;
            background-color: #f8d7da;
            color: #721c24;
        }
        .container {
            width: 90%;
            margin: 0 auto;
        }
        .order-section {
            margin-top: 20px;
            border-radius: 8px;
            padding: 20px;
            background-color:  #fff;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
        }
        .order-table {
            width: 100%;
            border-collapse: collapse;
        }
        .order-table-header {
            background-color: #f8f8f8;
            font-weight: bold;
        }
        .order-table th,
        .order-table td {
            padding: 12px;
            text-align: left;
            border-bottom: 1px solid #e0e0e0;
            font-size: 14px;
        }
        .order-table td:first-child {
            width: 5%;
        }
        .order-row:hover {
            background-color: #f0f0f0;
        }
        .order-status {
            padding: 8px 16px;
            border-radius: 4px;
            font-weight: bold;
            display: inline-block;
        }
        .order-status.accepted {
            background-color: #D4EDDA;
            color: #155724;
        }
        .order-status.rejected {
            background-color: #F8D7DA;
            color: #721C24;
        }
        .order-status.not-accepted {
            background-color: #FFF3CD;
            color: #856404;
        }
    </style>
    <script>
        function searchOrders() {
            let input, filter, table, tr, td, i, txtValue;
            input = document.getElementById('searchInput');
            filter = input.value.toUpperCase();
            table = document.querySelector('.order-table');
            tr = table.getElementsByTagName('tr');
                for (i = 0; i < tr.length; i++) {
                td = tr[i].getElementsByTagName('td')[1]; // Column index where Order ID is located
                    if (td) {
                    txtValue = td.textContent || td.innerText;
                    if (txtValue.toUpperCase().indexOf(filter) > -1) {
                        tr[i].style.display = '';
                    } else {
                        tr[i].style.display = 'none';
                    }
                }
            }
        }
    </script>
{% endblock %}
```

Views.py

```
def customer_orders(request):
    ordered_orders = Order.objects.filter(customer=request.user, is_ordered=True).order_by('-
order_date')
payment_details = Payment.objects.filter(order__in=ordered_orders).values_list('order_id',
flat=True)
    for order in ordered_orders:
        order.is_paid = order.id in payment_details
    items_per_page = 10
    paginator = Paginator(ordered_orders, items_per_page)
    page = request.GET.get('page')
    try:
        orders = paginator.page(page)
```

```
   except PageNotAnInteger:
         orders = paginator.page(1)
   except EmptyPage:
       orders = paginator.page(paginator.num_pages)
   context = {
      'ordered_orders': orders,
   }
   return render(request, 'customerOrders.html', context)
```

PlaceOrder.html

```
{% extends "userBase.html" %}
{% block title %}Place Order{% endblock %}
{% block content %}
{% if messages %}
   {% for message in messages %}
      <div class="alert" role="alert">
         {{ message }}
      </div>
   {% endfor %}
{% endif %}
<div class="container">
   <div class="heading_container">
      <h2>Place an Order at {{ pump.station_name }}</h2>
      <p>Address: {{ pump.address }}</p>
      <p>Location: {{ pump.location.name }}</p>
      <!-- <p>Phone: {{ request.user.phone }}</p>
      <p>Email: {{ request.user.email }}</p> -->
      <!-- Order placement form -->
      <form method="post" class="order-form">
         {% csrf_token %}
         <div class="form-group">
            <label for="fuel_type">Fuel Type:</label>
            <select name="fuel_type_id" id="fuel_type" class="select-field">
               <option value="">Select Fuel Type</option>
               {% for fuel in fuel_types %}
                  <option value="{{ fuel.id }}" data-price="{{ fuel.price }}">{{ fuel.fueltype
}}</option>
               {% endfor %}
            </select>
         </div>

         <div class="form-group">
            <label for="selected_price">Price:</label>
            <span id="selected_price" class="price-span"></span>
         </div>
```

```
            <div class="form-group">
                <label for="quantity">Quantity:</label>
                <select name="quantity" id="quantity" class="select-field">
                    <option value="">Select Quantity</option>
                    <option value="1">1 liter</option>
                    <option value="2">2 liters</option>
                    <option value="3">3 liters</option>
                    <option value="4">4 liters</option>
                </select>
            </div>
            <div class="form-group">
                <label for="delivery_point">Delivery Point:</label>
                <input type="text" name="delivery_point" id="delivery_point" class="text-field">
            </div>
            <div class="form-group">
                <label for="payment_method">Payment Method:</label>
                <select name="payment_method" id="payment_method" class="select-field">
                    <option value="">Select Payment Method</option>
                    <option value="Credit Card">Credit Card</option>
                    <option value="Debit Card">Debit Card</option>
                    <option value="COD">COD</option>
                    <option value="PayPal">PayPal</option>
                </select>
            </div>
            <button type="submit" id="submit-button"class="submit-button">Place Order</button>
        </form>
    </div>
</div>
<style>
    body {
        font-family: Arial, sans-serif;
        background-color: #f0f0f0;
        margin: 0;
        padding: 0;
    }
    .container {
        max-width: 600px;
        margin: 20px auto;
        padding: 20px;
        background-color: #fff;
        border-radius: 10px;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    }
```

```
.heading_container {
   text-align: center;
   margin-bottom: 20px;
}
.order-form {
   /* margin-top: 20px; */
   display: flex;
   flex-direction: column;
   gap: 10px;
}
.form-group {
   margin-bottom: 10px;
   display: flex;
   align-items: center;
}

.form-group label {
   flex: 1;
}
.select-field,
.text-field {
   flex: 2;
   padding: 12px;
   border: 1px solid #ccc;
   border-radius: 5px;
   font-size: 16px;
   box-sizing: border-box;
   width: 100%;
}
.submit-button {
   padding: 12px 24px;
   background-color: #f90;
   color: #fff;
   border: none;
   border-radius: 5px;
   cursor: pointer;
   font-size: 16px;
   transition: background-color 0.3s;
   width: 100%;
   display: block;
}
.submit-button:hover {
   background-color: #e6801b;
}
```

```
</style>
<script>
  document.querySelector('.order-form').addEventListener('submit', function(event) {
    event.preventDefault();
    const fuelType = document.getElementById('fuel_type');
    const selectedPrice = document.getElementById('selected_price');
    const quantity = document.getElementById('quantity');
    const deliveryPoint = document.getElementById('delivery_point');
    const paymentMethod = document.getElementById('payment_method');
    let isValid = true;
    function validateField(field, errorMessage) {
      if (field.value.trim() === '') {
        isValid = false;
        alert(errorMessage);
      }
    }
    validateField(fuelType, 'Please select a fuel type');
    validateField(quantity, 'Please select a quantity');
    validateField(deliveryPoint, 'Please enter a delivery point');
    validateField(paymentMethod, 'Please select a payment method');
    if (isValid) {
      this.submit(); // Submit the form
    }
  });
  document.getElementById('fuel_type').addEventListener('change', function() {
    const selectedPrice = this.options[this.selectedIndex].getAttribute('data-price');
    document.getElementById('selected_price').innerText = `${selectedPrice} per liter`;
  });
</script>
{% endblock content %}
```

Views.py

```
def place_order(request, pump_id):
  if request.method == 'POST':
    fuel_type_id = request.POST.get('fuel_type_id')
    quantity = Decimal(request.POST.get('quantity', '0'))
    delivery_point = request.POST.get('delivery_point')
    payment_method = request.POST.get('payment_method')
    selected_fuel = get_object_or_404(Fuel, pk=fuel_type_id)
    price_per_liter = selected_fuel.price
    total_price = quantity * price_per_liter  # Calculate total price
    if request.user.is_authenticated:
      customer = request.user
    else:
            customer = None
        station = get_object_or_404(FuelStation, pk=pump_id)
        order = Order.objects.create(
            customer=customer,
            fuel_type=selected_fuel,
            station=station,
            quantity=quantity,
            total_price=total_price,
            delivery_address=delivery_point,
            payment_method=payment_method,
        )
        return redirect('ordersummary',order_id=order.id)
    fuel_types = Fuel.objects.filter(station=pump_id)
    pump = get_object_or_404(FuelStation, pk=pump_id)
    context = {
        'fuel_types': fuel_types,
        'pump': pump
            }
    return render(request, 'place_order.html', context)
```

Vendor Fueladd.html

```
{% extends "base.html" %}
{% block title %}Fuel{% endblock %}
{% block content %}
{% if messages %}
    {% for message in messages %}
        <div class="alert alert-danger" role="alert">
            {{ message }}
        </div>
    {% endfor %}
{% endif %}
<div class="container mt-4">
    <h2 class="mb-4" style="color: #4285f4;">Fuel Management</h2>
    <div class="fuel-form-container mb-4">
        <form method="POST" id="fuelForm">
            {% csrf_token %}
            <div class="form-row">
                <div class="col-md-6 mb-3">
                    <label for="id_fueltype" style="font-weight: bold; color: #4285f4;">Fuel
Type:</label>
                    <input type="text" name="fueltype" id="id_fueltype" class="form-control" required>
                    <span class="error" id="fueltypeError"></span>
                </div>
                <div class="col-md-6 mb-3">
                    <label for="id_price" style="font-weight: bold; color: #4285f4;">Price (per
liter):</label>
                    <input type="text" name="price" id="id_price" class="form-control" required>
                    <span class="error" id="priceError"></span>
                </div>
            </div>
            <button type="submit" class="btn btn-primary">Add Fuel</button>
        </form>
    </div>
    <div class="fuel-records-container">
        <h2 class="mb-3" style="color: #4285f4;">Fuel Records</h2>
        <div class="table-responsive">
            <table class="table table-bordered table-hover">
                <thead>
                    <tr>
                        <th style="color: #4285f4;">Fuel Type</th>
                        <th style="color: #4285f4;">Price</th>
                        <th style="color: #4285f4;">Date Recorded</th>
                        <th style="color: #4285f4;">Actions</th>
                    </tr>
                </thead>
                <tbody>
                    {% for fuel in fuels %}
                    <tr>
                        <form method="POST" action="{% url 'update_fuel' fuel.id %}">
                            {% csrf_token %}
                            <td>{{ fuel.fueltype }}</td>
                            <td>
                                <input type="number" id="price_{{ fuel.id }}" name="price" value="{{
fuel.price }}" class="form-control">
                            </td>
                            <td>{{ fuel.profile_modified_at }}</td>
                            <td>
                                <button type="submit" class="btn btn-primary" name="update_fuel" value="{{
fuel.id }}">Update</button>
                            </td>
                        </form>
                    </tr>
                    {% endfor %}
                </tbody>
            </table>
        </div>
    </div>
</div>
```

```
<script>
document.addEventListener("DOMContentLoaded", function() {
   const inputField = document.getElementById('id_fueltype');
   inputField.addEventListener('input', function(event) {
     let enteredValue = event.data;
     if (enteredValue && !/^[A-Za-z]+$/.test(enteredValue)) {
       inputField.value = inputField.value.slice(0, -1);
     }
   });
});
document.addEventListener("DOMContentLoaded", function() {
      const inputField = document.getElementById('id_price');
      inputField.addEventListener('input', function(event) {
         let enteredValue = event.data;
         if (enteredValue && !/^[0-9.]*$/.test(enteredValue)) {
            inputField.value = inputField.value.replace(/[^0-9.]/g, '');
         }
      });
   });
document.addEventListener("DOMContentLoaded", function() {
   const fuelForm = document.getElementById('fuelForm');
   const fuelTypeField = document.getElementById('id_fueltype');
   const priceField = document.getElementById('id_price');
   const priceError = document.getElementById('priceError');
   fuelTypeField.addEventListener('input', function(event) {
      let enteredValue = event.target.value;
      if (enteredValue && !/^[A-Za-z]+$/.test(enteredValue)) {
         fuelTypeField.value = fuelTypeField.value.slice(0, -1);
      }
   });
   priceField.addEventListener('input', function(event) {
      let enteredValue = event.target.value.trim();
      if (enteredValue === "0") {
         priceError.innerText = "Price cannot be zero.";
         priceField.classList.add("error");
      } else if (!/^(\d{1,4}(\.\d{1,2})?)?$/.test(enteredValue)) {
         priceError.innerText = "Please enter a valid price with at most 4 digits before the decimal.";
         priceField.classList.add("error");
      } else {
         priceError.innerText = "";
         priceField.classList.remove("error");
      }
   });
   fuelForm.addEventListener('submit', function(event) {
        let enteredPrice = priceField.value.trim();
        if (enteredPrice === "0" || !/^(\d{1,4}(\.\d{1,2})?)?$/.test(enteredPrice)) {
            event.preventDefault(); // Prevent form submission if price is invalid
        }
    });
});
</script>
<style>
    .container {
        max-width: 800px;
        margin: auto;
    }
    .form-control {
        width: 100%;
        border-radius: 8px;
        border: 1px solid #ccc;
        padding: 10px;
        box-sizing: border-box;
    }
    .fuel-form-container {
        background-color: #f7f7f7;
        padding: 20px;
        border-radius: 10px;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }
    .fuel-records-container {
        background-color: #f7f7f7;
        padding: 20px;
        border-radius: 10px;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        margin-top: 20px;
    }
    .btn-primary {
        background-color: #4285f4;
        color: #fff;
        border: none;
        border-radius: 8px;
        padding: 10px 20px;
    }
    .btn-primary:hover {
        background-color: #0056b3;
    }
    .table {
        margin-top: 20px;
        font-size: 16px;
        border-radius: 8px;
        overflow: hidden;
    }
    .table th,
    .table td {
        text-align: center;
        padding: 12px;
    }
    .table-bordered th,
    .table-bordered td {
        border: 1px solid #ddd;
    }
    .table-hover tbody tr:hover {
        background-color: #f0f5fe;
    }
    .alert {
        background-color: #f8d7da;
        border-color: #f5c6cb;
        color: #721c24;
    }
</style>
{% endblock content %}
```

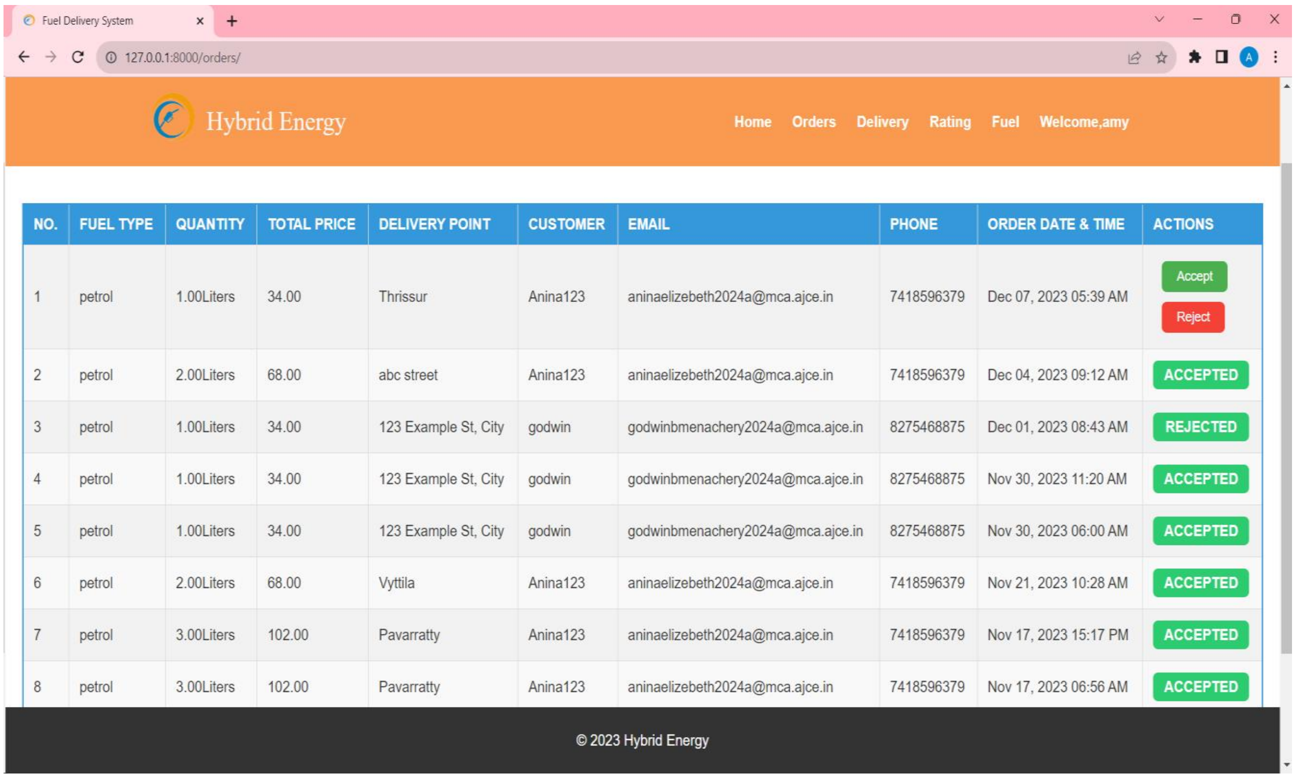
Views.py

```
def fuel(request):
    existing_fuel = None
    if request.user.is_vendor:
        station_id = request.user.id
        station = FuelStation.objects.get(user=station_id)
        if request.method == 'POST':
            fueltype = request.POST.get('fueltype')
            price = request.POST.get('price')
            existing_fuel = Fuel.objects.filter(fueltype=fueltype, station=station).first()
            if existing_fuel:
                messages.error(request, "A fuel of the same type already exists for this station.")
                return redirect('fuel')
            else:
                fuel = Fuel.objects.create(
                    fueltype=fueltype,
                    price=price,
                    station=station
                )
        fuels = Fuel.objects.filter(station=station)
        context = {
            'fuels': fuels,        }
        return render(request, 'fuel.html', context)
```

9.2 Screen Shots

1. PUMP

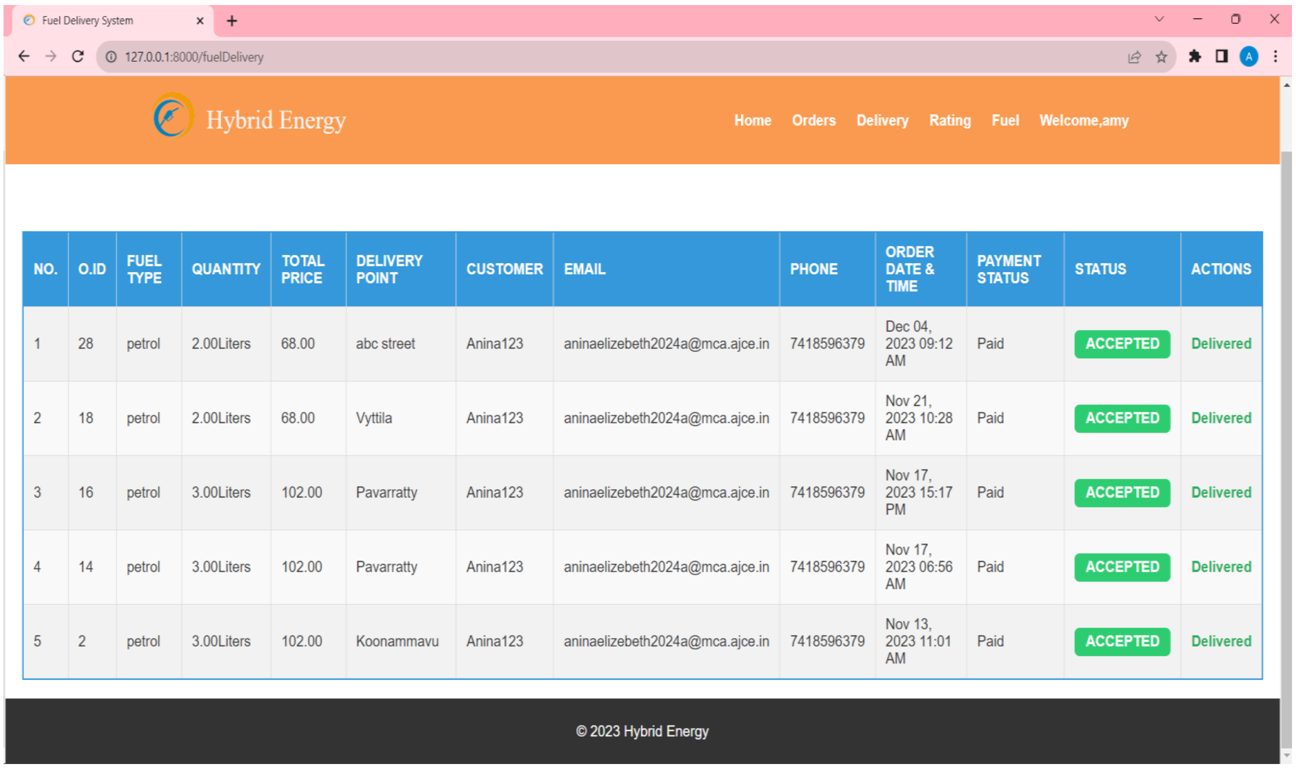
1.1 ORDER LISTING PAGE:



NO.	FUEL TYPE	QUANTITY	TOTAL PRICE	DELIVERY POINT	CUSTOMER	EMAIL	PHONE	ORDER DATE & TIME	ACTIONS
1	petrol	1.00Liters	34.00	Thrissur	Anina123	aninaelizebeth2024a@mca.ajce.in	7418596379	Dec 07, 2023 05:39 AM	Accept Reject
2	petrol	2.00Liters	68.00	abc street	Anina123	aninaelizebeth2024a@mca.ajce.in	7418596379	Dec 04, 2023 09:12 AM	ACCEPTED
3	petrol	1.00Liters	34.00	123 Example St, City	godwin	godwinbmenachery2024a@mca.ajce.in	8275468875	Dec 01, 2023 08:43 AM	REJECTED
4	petrol	1.00Liters	34.00	123 Example St, City	godwin	godwinbmenachery2024a@mca.ajce.in	8275468875	Nov 30, 2023 11:20 AM	ACCEPTED
5	petrol	1.00Liters	34.00	123 Example St, City	godwin	godwinbmenachery2024a@mca.ajce.in	8275468875	Nov 30, 2023 06:00 AM	ACCEPTED
6	petrol	2.00Liters	68.00	Vyttila	Anina123	aninaelizebeth2024a@mca.ajce.in	7418596379	Nov 21, 2023 10:28 AM	ACCEPTED
7	petrol	3.00Liters	102.00	Pavarratty	Anina123	aninaelizebeth2024a@mca.ajce.in	7418596379	Nov 17, 2023 15:17 PM	ACCEPTED
8	petrol	3.00Liters	102.00	Pavarratty	Anina123	aninaelizebeth2024a@mca.ajce.in	7418596379	Nov 17, 2023 06:56 AM	ACCEPTED

© 2023 Hybrid Energy

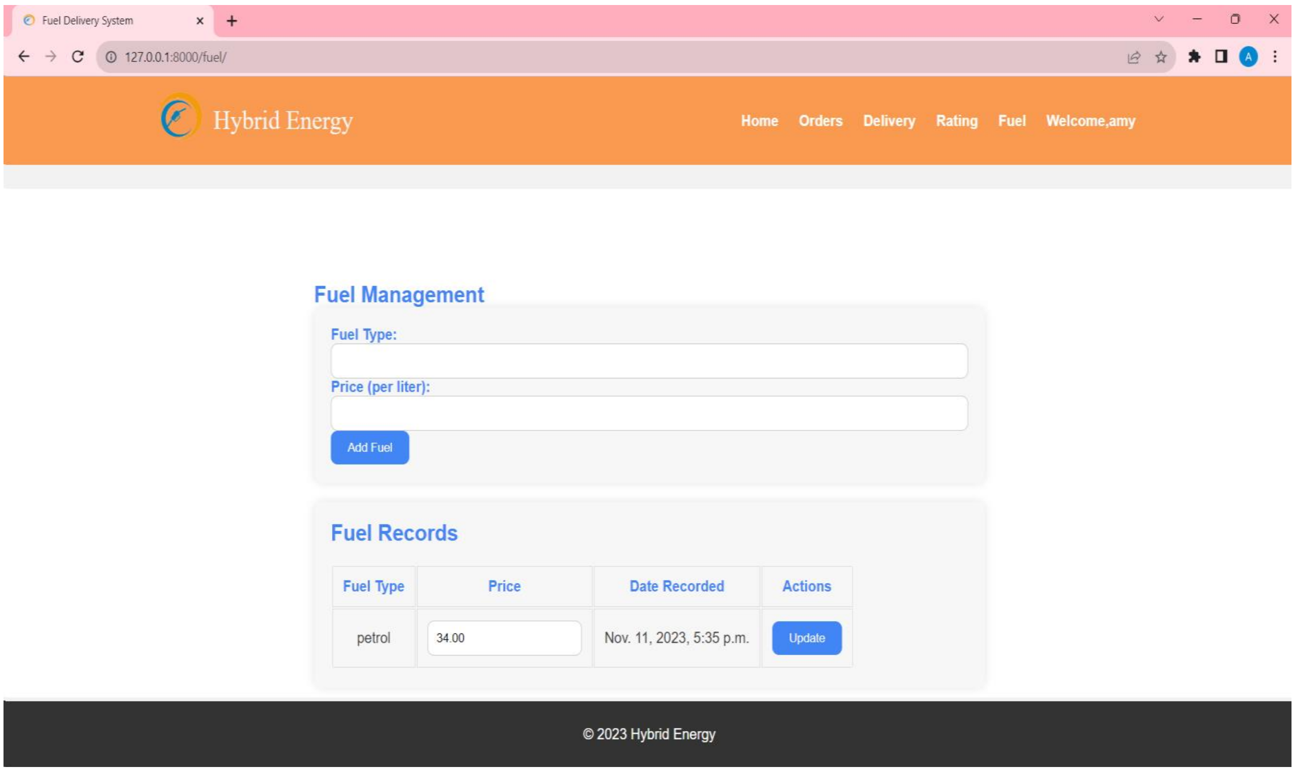
1.2 DELIVERY ORDERS LISTING PAGE:



NO.	O.ID	FUEL TYPE	QUANTITY	TOTAL PRICE	DELIVERY POINT	CUSTOMER	EMAIL	PHONE	ORDER DATE & TIME	PAYMENT STATUS	STATUS	ACTIONS
1	28	petrol	2.00Liters	68.00	abc street	Anina123	aninaelizebeth2024a@mca.ajce.in	7418596379	Dec 04, 2023 09:12 AM	Paid	ACCEPTED	Delivered
2	18	petrol	2.00Liters	68.00	Vyttila	Anina123	aninaelizebeth2024a@mca.ajce.in	7418596379	Nov 21, 2023 10:28 AM	Paid	ACCEPTED	Delivered
3	16	petrol	3.00Liters	102.00	Pavarratty	Anina123	aninaelizebeth2024a@mca.ajce.in	7418596379	Nov 17, 2023 15:17 PM	Paid	ACCEPTED	Delivered
4	14	petrol	3.00Liters	102.00	Pavarratty	Anina123	aninaelizebeth2024a@mca.ajce.in	7418596379	Nov 17, 2023 06:56 AM	Paid	ACCEPTED	Delivered
5	2	petrol	3.00Liters	102.00	Koonammavu	Anina123	aninaelizebeth2024a@mca.ajce.in	7418596379	Nov 13, 2023 11:01 AM	Paid	ACCEPTED	Delivered

© 2023 Hybrid Energy

1.3 FUEL TYPE ADDING PAGE:



Fuel Management

Fuel Type:

Price (per liter):

Add Fuel

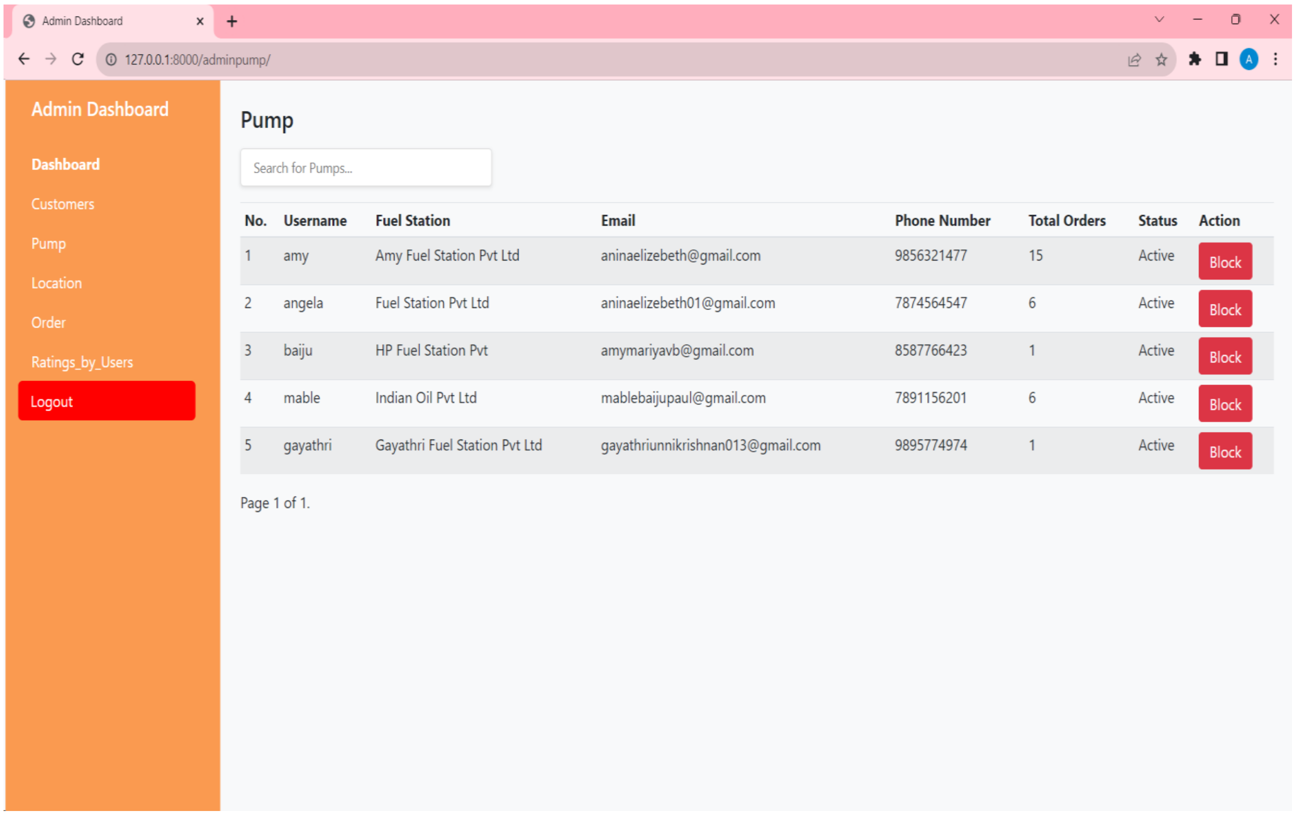
Fuel Records

Fuel Type	Price	Date Recorded	Actions
petrol	34.00	Nov. 11, 2023, 5:35 p.m.	Update

© 2023 Hybrid Energy

2 ADMIN

2.1 Vendors List



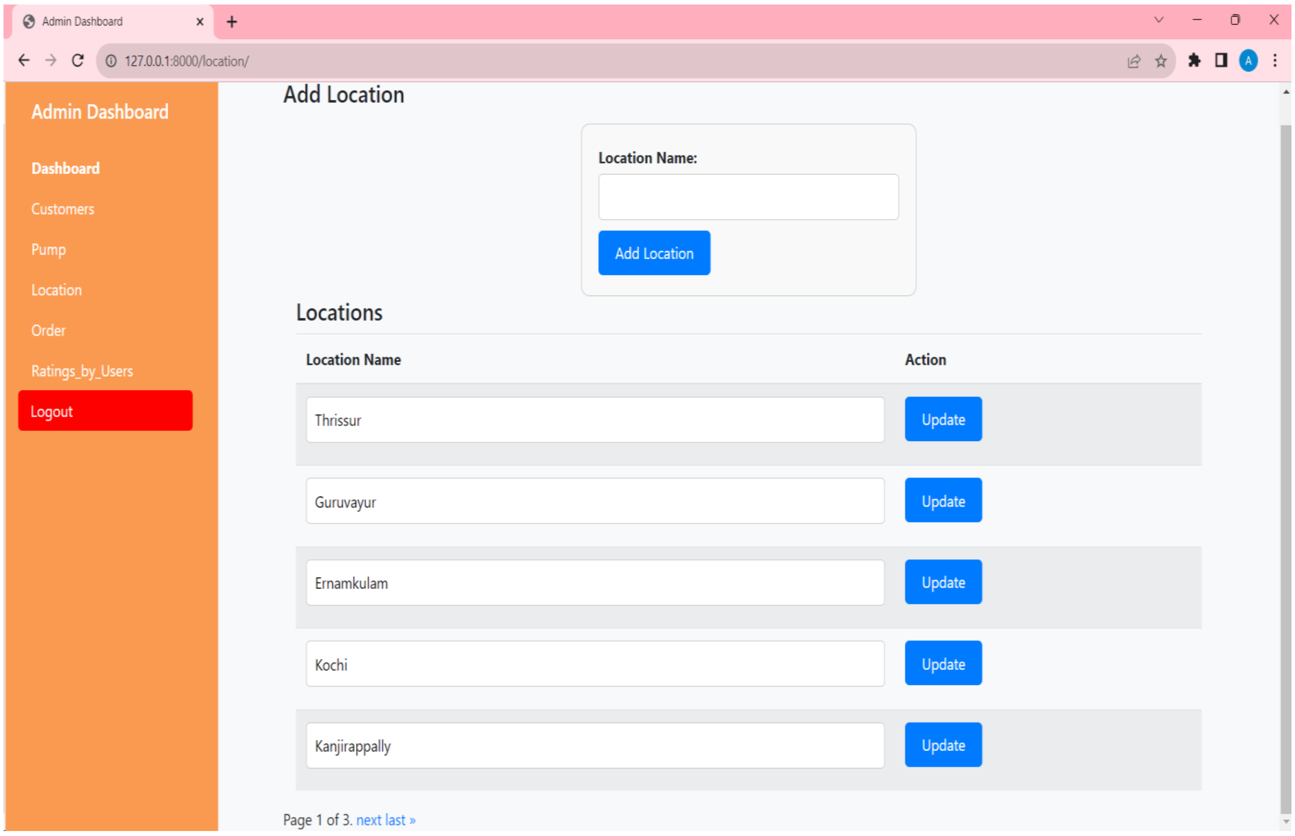
Pump

Search for Pumps...

No.	Username	Fuel Station	Email	Phone Number	Total Orders	Status	Action
1	amy	Amy Fuel Station Pvt Ltd	aninaelizebeth@gmail.com	9856321477	15	Active	Block
2	angela	Fuel Station Pvt Ltd	aninaelizebeth01@gmail.com	7874564547	6	Active	Block
3	baiju	HP Fuel Station Pvt	amymariyavb@gmail.com	8587766423	1	Active	Block
4	mable	Indian Oil Pvt Ltd	mablebaijupaul@gmail.com	7891156201	6	Active	Block
5	gayathri	Gayathri Fuel Station Pvt Ltd	gayathriunnikrishnan013@gmail.com	9895774974	1	Active	Block

Page 1 of 1.

2.2 Location Add

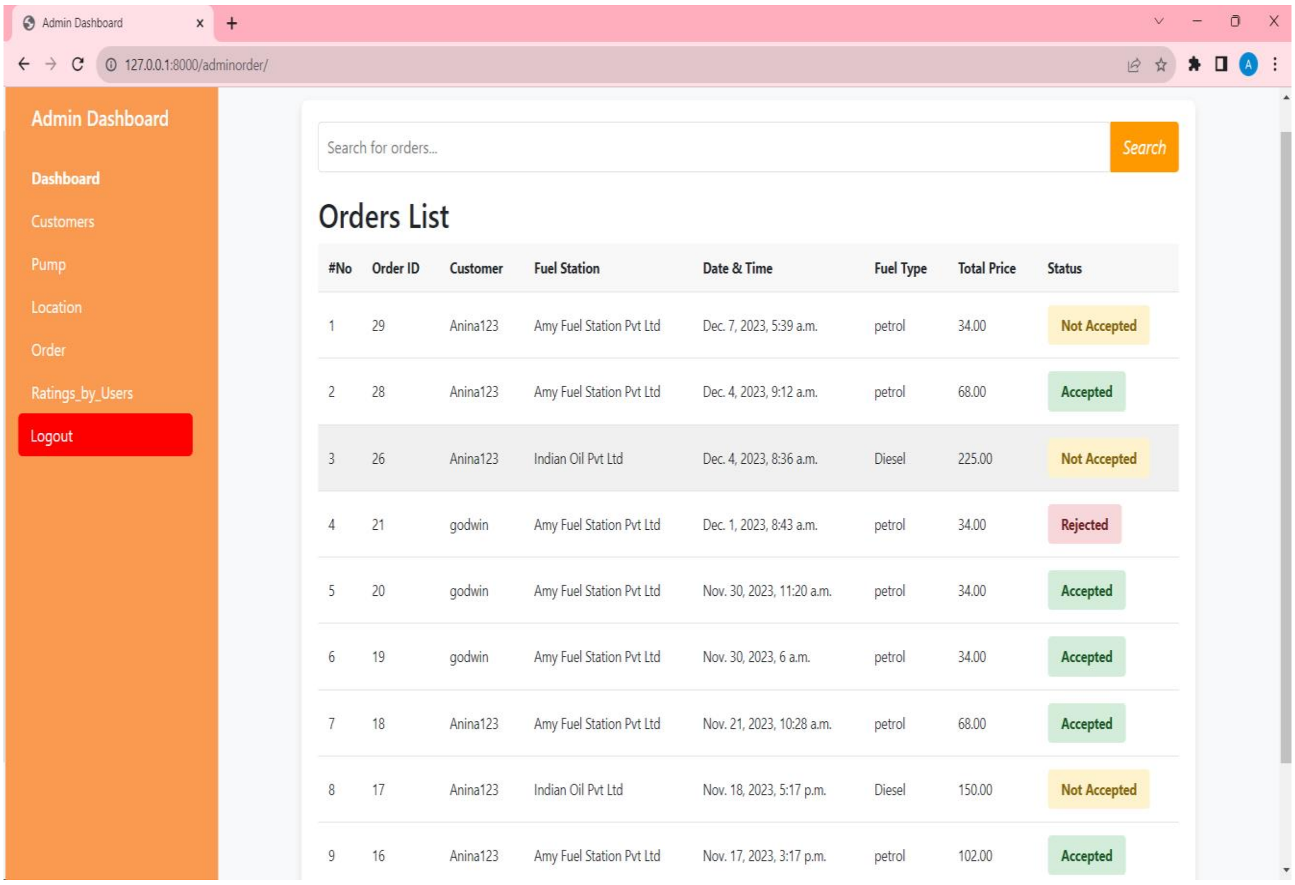


Locations

Location Name	Action
Thrissur	Update
Guruvayur	Update
Ernamkulam	Update
Kochi	Update
Kanjirappally	Update

Page 1 of 3. next last »

2.3 Order List

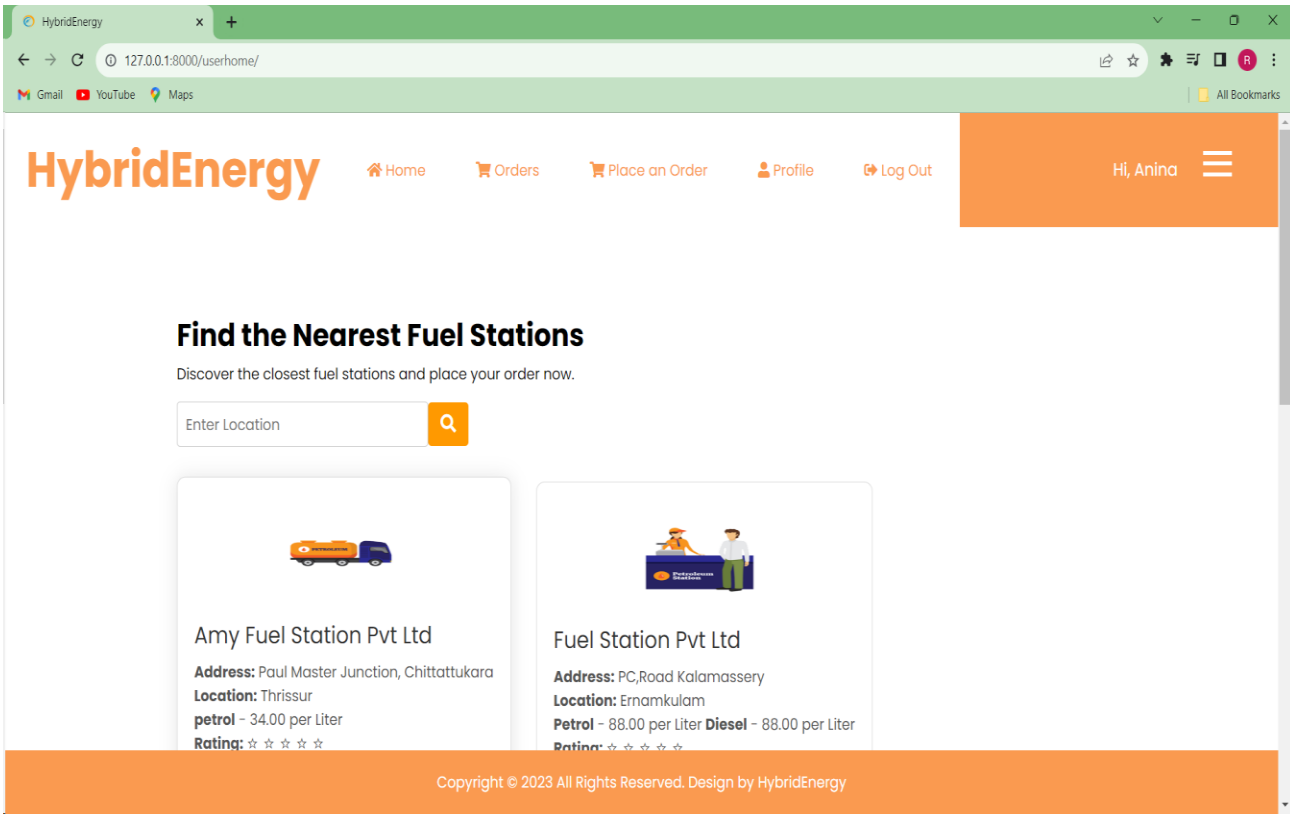


Orders List

#No	Order ID	Customer	Fuel Station	Date & Time	Fuel Type	Total Price	Status
1	29	Anina123	Amy Fuel Station Pvt Ltd	Dec. 7, 2023, 5:39 a.m.	petrol	34.00	Not Accepted
2	28	Anina123	Amy Fuel Station Pvt Ltd	Dec. 4, 2023, 9:12 a.m.	petrol	68.00	Accepted
3	26	Anina123	Indian Oil Pvt Ltd	Dec. 4, 2023, 8:36 a.m.	Diesel	225.00	Not Accepted
4	21	godwin	Amy Fuel Station Pvt Ltd	Dec. 1, 2023, 8:43 a.m.	petrol	34.00	Rejected
5	20	godwin	Amy Fuel Station Pvt Ltd	Nov. 30, 2023, 11:20 a.m.	petrol	34.00	Accepted
6	19	godwin	Amy Fuel Station Pvt Ltd	Nov. 30, 2023, 6 a.m.	petrol	34.00	Accepted
7	18	Anina123	Amy Fuel Station Pvt Ltd	Nov. 21, 2023, 10:28 a.m.	petrol	68.00	Accepted
8	17	Anina123	Indian Oil Pvt Ltd	Nov. 18, 2023, 5:17 p.m.	Diesel	150.00	Not Accepted
9	16	Anina123	Amy Fuel Station Pvt Ltd	Nov. 17, 2023, 3:17 p.m.	petrol	102.00	Accepted

3 CUSTOMER

3.1 HOME



Find the Nearest Fuel Stations

Discover the closest fuel stations and place your order now.

Amy Fuel Station Pvt Ltd

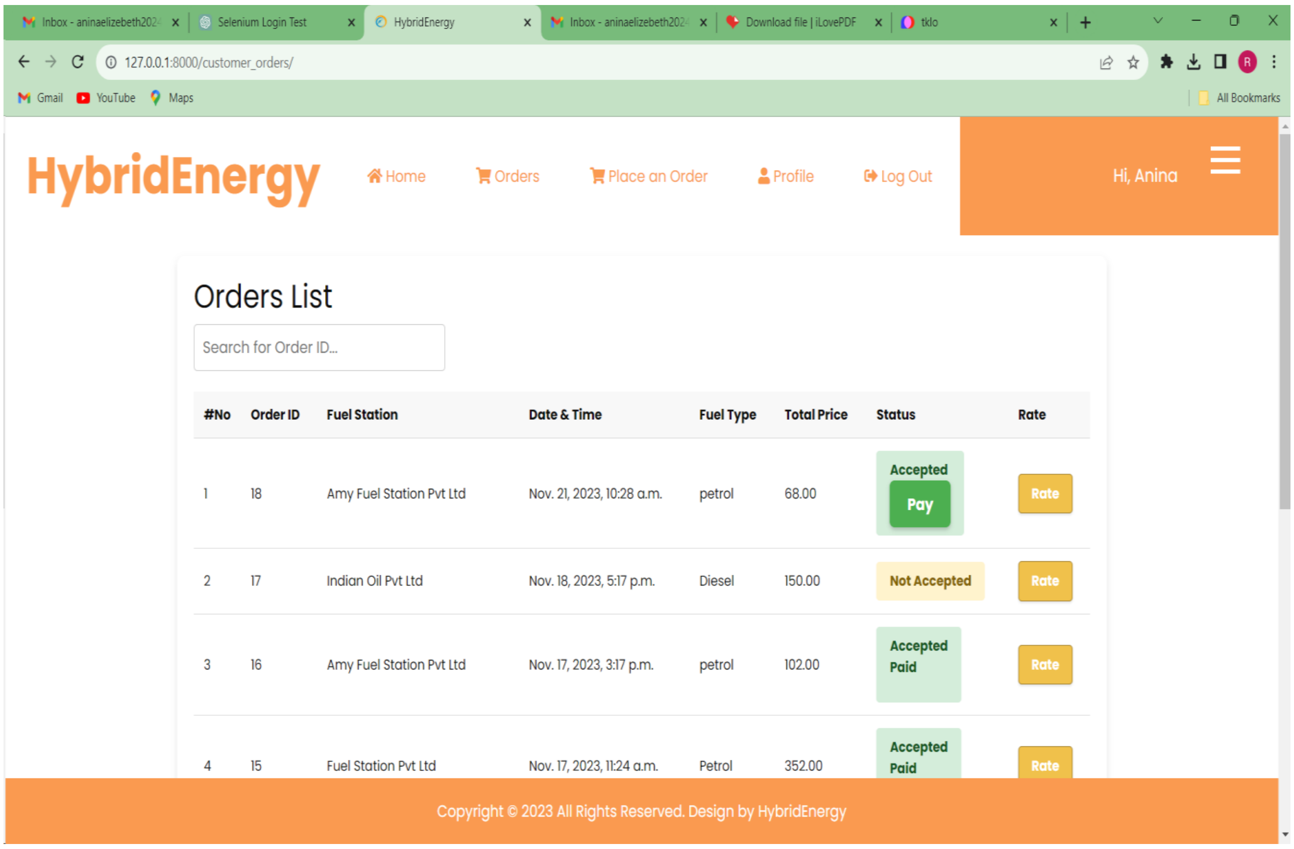
Address: Paul Master Junction, Chittattukara
Location: Thrissur
petrol - 34.00 per Liter
Rating: ☆ ☆ ☆ ☆ ☆

Fuel Station Pvt Ltd

Address: PC,Road Kalamassery
Location: Ernamkulam
Petrol - 88.00 per Liter **Diesel** - 88.00 per Liter
Rating: ☆ ☆ ☆ ☆ ☆

Copyright © 2023 All Rights Reserved. Design by HybridEnergy

3.2 Order List

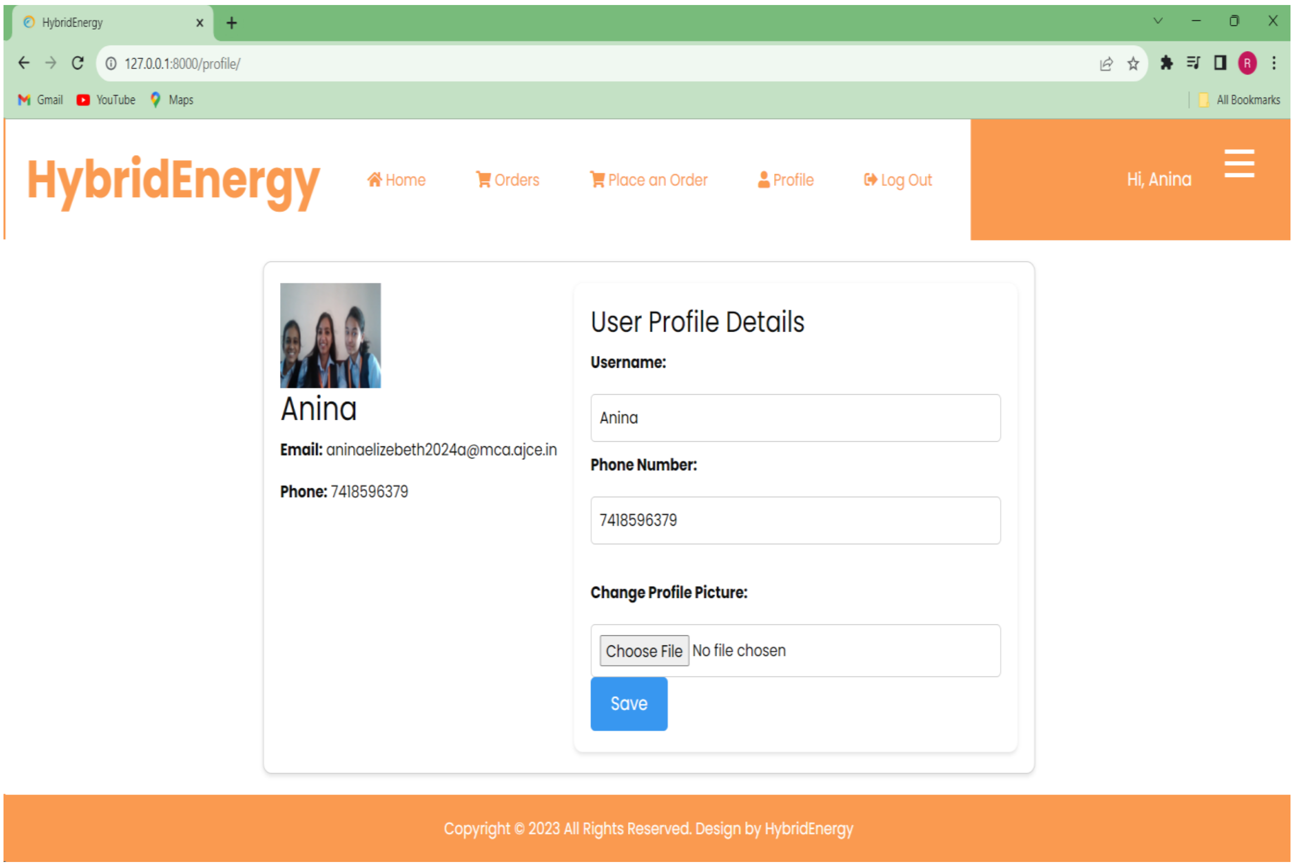


Orders List

#No	Order ID	Fuel Station	Date & Time	Fuel Type	Total Price	Status	Rate
1	18	Amy Fuel Station Pvt Ltd	Nov. 21, 2023, 10:28 a.m.	petrol	68.00	Accepted Pay	Rate
2	17	Indian Oil Pvt Ltd	Nov. 18, 2023, 5:17 p.m.	Diesel	150.00	Not Accepted	Rate
3	16	Amy Fuel Station Pvt Ltd	Nov. 17, 2023, 3:17 p.m.	petrol	102.00	Accepted Paid	Rate
4	15	Fuel Station Pvt Ltd	Nov. 17, 2023, 11:24 a.m.	Petrol	352.00	Accepted Paid	Rate

Copyright © 2023 All Rights Reserved. Design by HybridEnergy

3.3 PROFILE



Anina

Email: aninaelizebeth2024a@mca.ajce.in

Phone: 7418596379

User Profile Details

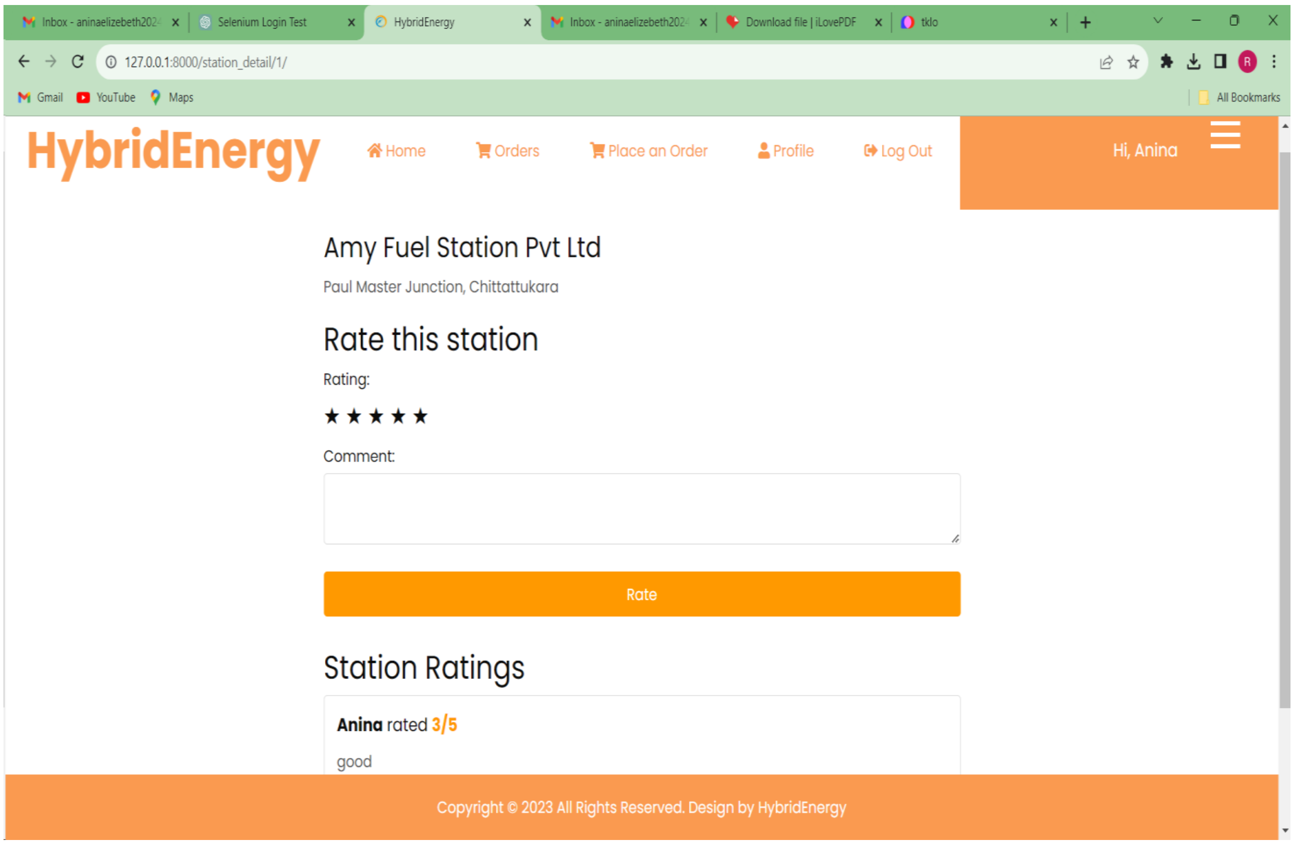
Username: Anina

Phone Number: 7418596379

Change Profile Picture: No file chosen

Copyright © 2023 All Rights Reserved. Design by HybridEnergy

3.4 Rating Page



Amy Fuel Station Pvt Ltd

Paul Master Junction, Chittattukara

Rate this station

Rating:

Comment:

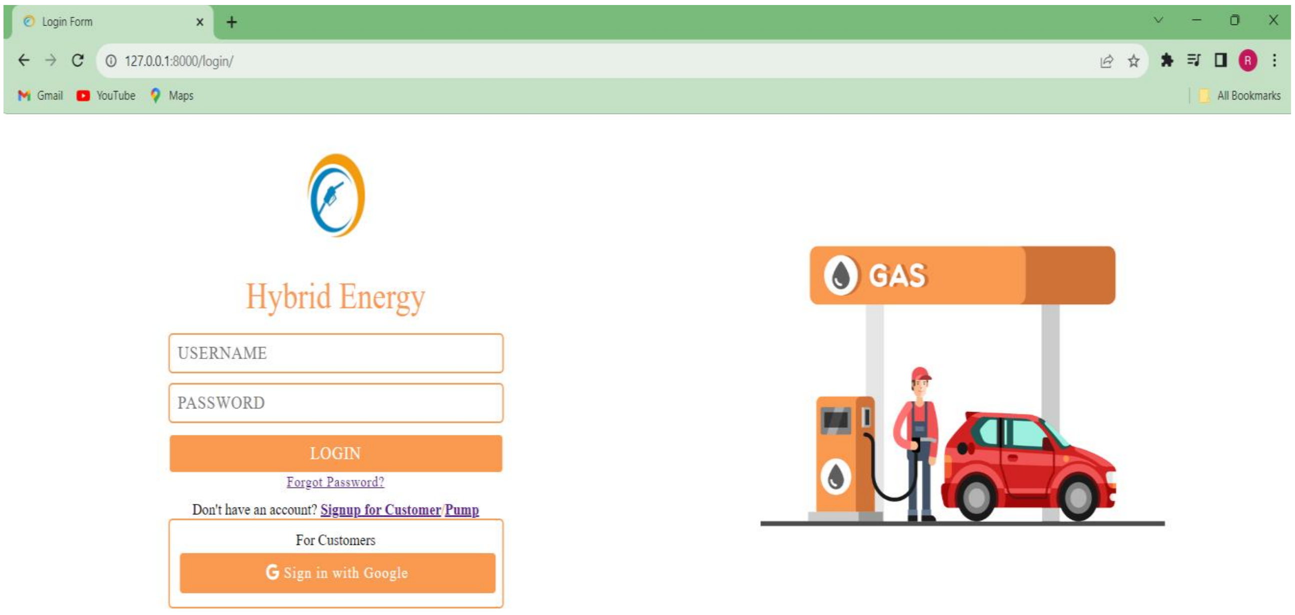
Station Ratings

Anina rated 3/5

good

Copyright © 2023 All Rights Reserved. Design by HybridEnergy

4 LOGIN PAGE



Hybrid Energy

USERNAME

PASSWORD

LOGIN

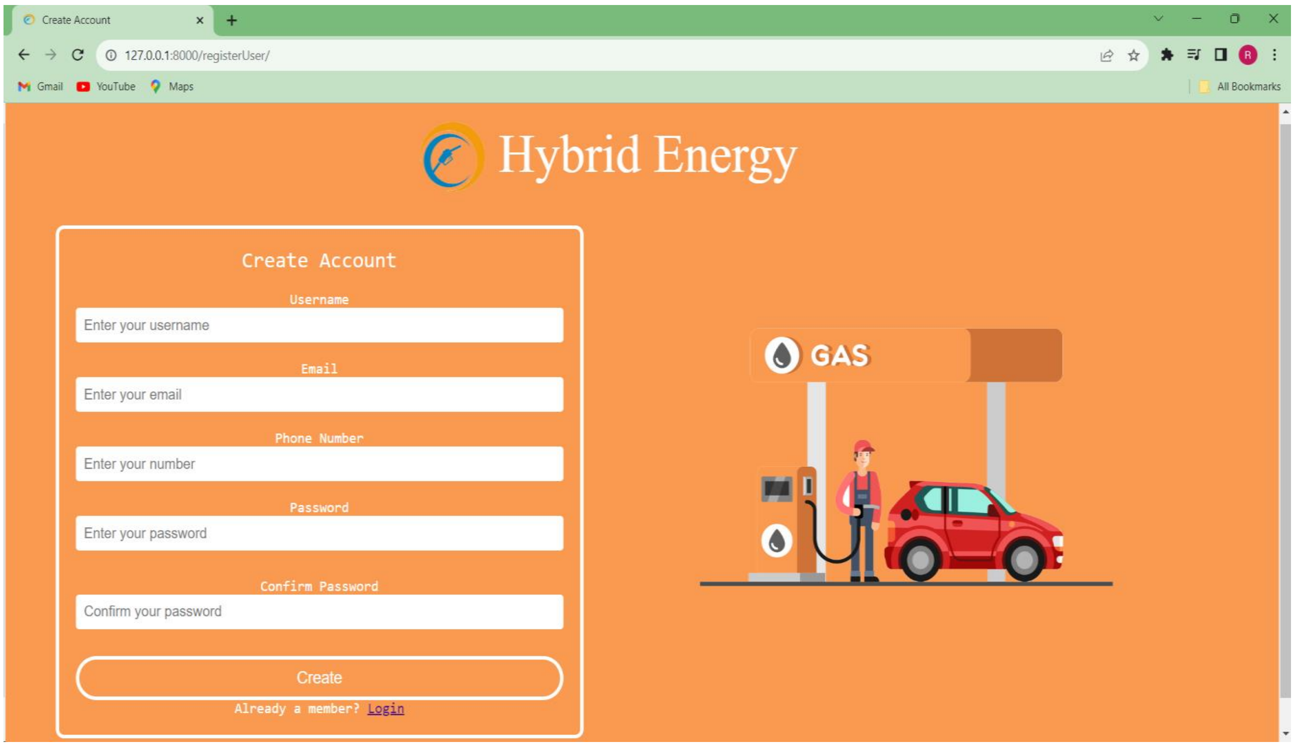
Forgot Password?

Don't have an account? Signup for Customer/Pump

For Customers

Sign in with Google

5 REGISTERATION PAGE



Hybrid Energy

Create Account

Username

Enter your username

Email

Enter your email

Phone Number

Enter your number

Password

Enter your password

Confirm Password

Confirm your password

Create

Already a member? Login