AQUA MOTUS

WATER METRO MANAGEMENT SYSTEM

Mini Project Report

Submitted by

ANMIGHA N M

Reg. No.: AJC20MCA-I014

In Partial fulfillment for the Award of the Degree Of

INTEGRATED MASTER OF COMPUTER APPLICATIONS

(INMCA)

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

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

2024-2025

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

CERTIFICATE

This is to certify that the Mini-Project report, "**AQUA MOTUS"** is the bona fide work of **ANMIGHA N M (Regno: AJC20MCA-I014)** 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 2024-25.

Mr. Jinson Devis
Internal Guide

Mr. Binumon Joseph
Coordinator

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

DECLARATION

I hereby declare that the mini-project report **"AQUA MOTUS"** is a bona fide work done at Amal Jyothi College of Engineering Autonomous, towards the partial fulfilment of the requirements for the award of the **Integrated Master of Computer Applications (INMCA)** from **APJ Abdul Kalam Technological University**, during the academic year **2024-2025**.

Date: **ANMIGHA N M**

KANJIRAPPALLY **Reg: AJC20MCA-I014**

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 Director (Administration) **Rev. Fr. Dr. Roy Abraham Pazhayaparampil** 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 **Mr. Binumon Joseph, Assistant Professor** for his 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. Jinson Devis, Assistant Professor** 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.

ANMIGHA N M

ABSTRACT

The mini-project entitled “AQUA MOTUS” is a dynamic web application for managing Ticket and Facilities Management System that significantly elevates user experience through streamlined online ticket booking processes and seamless access to comprehensive information regarding available facilities.

In the proposed system, users can register online to book tickets for boat transportation services and can browse available boat schedules, select routes, and book tickets accordingly. The Station Master manages boats and services, ensuring smooth operations, scheduling, and maintenance. The admin registers boats, station master, updates services, and oversees overall system functionality.

The modules of the system are:

1. **Admin**

- Can login to dashboard.
- Add new stations.
- Add boat details.
- Add new events.
- Register new station masters.
- Can Take Report

2. **Station master**

- Can login to dashboard.
- Assign boats to specific routes and schedules.
- View tickets booked by users.
- Assign boat for event were payment done.
- Cancel tickets have refund (80%).
- Generate and view reports related to services.

3. **User**

- Register account in aqua motus with email verification.
- Can login to dashboard.
- Can book the ticket for available boat schedules and Events.
- View booked tickets with payment option and cancel button.
- Profile View and updation.
- Logout

CONTENT

List of Abbreviation

HTML	–	Hypertext Markup Language
CSS	–	Cascading Style Sheets
JQuery	–	JavaScript Query
JS	–	JavaScript
Django	–	A Python-based Web Framework
DBSQLite	–	Database Structured Query Language Lite
SQL	–	Structured Query Language
ML	–	Machine Learning
NLP	–	Natural Language Processing
NLTK	–	Natural Language Toolkit
SVM	–	Support Vector Machine
LSTM	–	Long Short-Term Memory

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The Aqua Motus project is a dynamic web application designed to streamline ticket booking and facilities management for water metro services, enhancing both user experience and operational efficiency. The system allows users to register, browse boat schedules, select routes, book tickets, and manage payments. Station masters can oversee boat operations, assign schedules, and view booking reports, while administrators manage system-wide functionalities including registering boats, station masters, and events. Utilizing technologies like Python, Django, and SQLite, with machine learning for weather forecasting and route optimization, Aqua Motus aims to deliver a seamless, secure, and scalable solution for water metro transportation management.

1.2 PROJECT SPECIFICATION

The Aqua Motus project is specified as a dynamic web-based application developed using Python for backend logic and Django as the web framework. It integrates an SQLite database for managing user, ticket, and service data, offering robust and secure management of water metro services. The system supports three key user roles: Admin, Station Master, and User. Admins can manage boats, stations, and events, while Station Masters oversee boat assignments, service schedules, and reports. Users can browse routes, book tickets, and manage payments. Additionally, machine learning is incorporated to provide weather forecasting, boat safety measures, and route optimization, ensuring operational efficiency and user safety. The system is designed to handle large-scale operations with minimal technical requirements, making it a cost-effective solution for water metro management.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

The system study for the Aqua Motus project focuses on analysing the feasibility, requirements, and scope necessary to develop an efficient water metro management system. AQUA MOTUS is a dynamic web application designed to streamline ticket booking, facility management, and the overall operation of boat transportation services. The system study evaluates various dimensions, including technical, operational, economic, and behavioural feasibility, to ensure the project's viability.

This analysis is grounded in a detailed understanding of the target audience, including users, station masters, and administrators, and outlines the system's core modules and functionalities. The study also highlights the system's integration with existing technologies, including Python, Django, and machine learning, for a robust backend, alongside SQLite for database management. It delves into key performance indicators, user roles, and operational goals, ensuring that the proposed system not only meets but exceeds the expectations of stakeholders.

By conducting a thorough requirement gathering process and feasibility assessment, the system study aims to lay a strong foundation for developing a scalable, user-friendly, and technically feasible platform that enhances the overall water metro service experience.

2.2 LITERATURE REVIEW

The AQUA MOTUS project studies current water metro operations, analyzing manual processes like ticketing and boat assignments to identify improvements through digital solutions, aiming for enhanced efficiency and eco-friendly transportation. The existing water metro system relies heavily on manual processes, causing delays, inefficiencies, and an increased workload for staff. Limited accessibility forces users to visit stations to view schedules and buy tickets, reducing convenience and user satisfaction. Without an integrated digital system, resource management is challenging, leading to operational inefficiencies and a lack of real-time updates for passengers on boat availability, delays, or cancellations. Additionally, the high dependence on paperwork raises the risk of errors and data loss, while the system’s limited scalability makes it difficult to handle growing passenger demand without significant manual intervention.

Sentiment analysis has become a vital tool across various sectors, particularly in transportation, where it helps organizations understand customer emotions and opinions through reviews on travel-related apps like Red Bus, Make My Trip, and Yatra.com [1]. Research from the last few years has pointed out the use of Sentiment Analysis to evaluate and improve service quality in the

airline industry through categorizing customer comments as positive, negative, or neutral. This process includes data collection, preprocessing, sentiment analysis techniques, and visualization, so that the airlines can assess the quality of their service and change their strategies to better serve the passengers [2]. Additionally, sentiment analysis in public transportation systems offers insights into commuter preferences, aiding in the planning of multimodal journey options and improving overall user experience. This can be achieved by combining sentiment analysis with current innovations such as car-sharing plans and autonomous vehicles, enabling transportation companies to offer more individualized and user-friendly services, which in turn will lead to increased customer satisfaction and ultimately the use of public and shared transportation [3].

With the explosion of social networking sites, there has been an outpouring of unstructured textual data which leads to the necessity for more sophisticated sentiment and emotion analysis techniques [8]. The present techniques, lexicon based, and corpus based, are good in some ways and not so good in others, but recurrent neural networks (RNNs) such as LSTM can capture long term dependencies very well from a large amount of data. Preprocessing and feature extraction problems remain, especially with implicit features in text. Sentiment analysis has become an integral part of understanding user opinions on many different forums, and new methods using part-of-speech tagging seem to be very promising in achieving greater accuracy [9]. A review of sentiment analysis techniques highlights the widespread use of lexicon-based methods and machine learning approaches, such as Naïve Bayes and SVM, particularly for analysing data from platforms like Twitter [10]. This analysis implies that although lexicon-based methods are accurate for smaller corpus, machine learning methods are more appropriate for larger more complicated data, and it calls for the creation of general models for different corpora and for the exploration of other wikis.

Recent developments in natural language processing (NLP) and artificial intelligence (AI), especially with models like BERT and RoBERTa, have greatly enhanced the ability to analyze feelings and opinions in many fields, including aviation [4]. These models have performed very well in categorizing sentiments, especially when looking at reviews from airline customers, with RoBERTa being the most accurate. Techniques like Random Forest, a type of machine learning, have also been successful in sorting out sentiments from social media, giving airlines useful information to improve their services [5]. Moreover, methods like text mining and topic modeling help analyze customer reviews, pinpointing areas where airlines can improve their services and ensuring they meet customer needs better [6]. Using sentiment analysis across various languages and platforms highlights its importance in handling large amounts of unstructured data [7].

Due to the proliferation of digital platforms like blogs and social media, which produce enormous volumes of data that are rich in opinions, sentiment analysis has grown in importance. To determine public opinion on a range of subjects, goods, and services, this procedure entails removing and categorizing subjective information from text. Because of their ease of use and precision, supervised machine learning algorithms such as Naïve Bayes (NB) and Support Vector Machines (SVM) are commonly used in text mining and natural language processing (NLP). These techniques have shown their worth in drawing useful conclusions from huge datasets in applications like social media monitoring and consumer feedback analysis. [11].

Despite its advancements, sentiment analysis faces challenges related to the complexity of human language, including sarcasm, domain specific terminology, and varying contextual meanings. Issues like domain dependence and sentiment difficulties polarity persist. interpretation Addressing these challenges requires expanding datasets, improving algorithms, and exploring new methodologies. Sentiment analysis's ability to provide businesses—by classification actionable insights automating for sentiment and obtaining real-time feedback—enhances decision-making and reduces manual analysis costs. As technology evolves, sentiment analysis is expected to offer increasingly nuanced understandings of public opinion and sentiment across various sectors [12].

Sentiment analysis (SA), which examines textual data from social media and other online platforms, has emerged as a crucial method for comprehending public sentiment. Text may now be effectively classified as positive, negative, or neutral thanks to the development of SA approaches, which include the application of machine learning algorithms like Naïve Bayes and Support Vector Machines (SVM). These techniques are well known for their effectiveness in processing vast amounts of data and deriving insightful conclusions. By handling the enormous volume of data produced on social media platforms, the incorporation of big data technologies, such Hadoop, has further improved sentiment analysis's efficacy. [13].

Additionally, sentiment analysis of social media data, especially from Twitter, has shown significant value in understanding consumer sentiment. Various machine learning approaches, combined with semantic analysis, have been explored to improve sentiment classification accuracy. Research indicates that Naïve Bayes outperforms other methods like Maximum Entropy and SVM when using a unigram model, achieving higher accuracy. Combining semantic analysis with machine learning techniques and expanding the training dataset can further refine sentiment classification. These advancements offer a practical approach for analysing unstructured data from

social media platforms [14].

To implement the sentiment analysis techniques discussed in this review paper, technologies such as Python and its libraries for natural language processing (NLP) are highly suitable. Key tools include NLTK and spaCy for text preprocessing steps like tokenization, lemmatization, and stop-word removal. For implementing machine learning algorithms, libraries like scikit-learn provide support for models like Naïve Bayes and SVM, while deep learning frameworks like TensorFlow and PyTorch enable the use of more advanced models such as LSTM and CNN for nuanced text interpretation. Additionally, transformer-based models such as BERT and RoBERTa, available via Hugging Face's Transformers library, can be used to capture contextual information and handle complex language structures, improving sentiment accuracy. These technologies collectively allow for a comprehensive and efficient sentiment analysis process on passenger reviews, facilitating the extraction of insights into user satisfaction and areas for service improvement in Water Metro services.

Sentiment Analysis of Water Metro Passengers Review: A Comprehensive Review

The paper presents an in-depth review of sentiment analysis techniques as applied to passenger reviews in Water Metro services, a unique and environmentally friendly transportation alternative. Recognizing the importance of understanding passenger sentiment, the paper explores how sentiment analysis can enhance service quality and provide actionable insights into passenger satisfaction.

The study surveys various sentiment analysis methods, including lexicon-based approaches, machine learning algorithms like Naïve Bayes and SVM, and advanced deep learning models such as LSTM and CNN. Each method's advantages and challenges are discussed, particularly in the context of handling large volumes of text data and complex language nuances such as sarcasm and implicit meanings.

Data for analysis can be sourced from social media platforms, online review sites, structured surveys, and mobile app feedback, with preprocessing steps—such as tokenization and feature extraction—playing a critical role in preparing data for accurate sentiment classification. The paper also highlights recent advancements in NLP, such as BERT and RoBERTa, which have enhanced sentiment categorization in various fields, including transportation.

Challenges in sentiment analysis, such as multilingual processing, domain-specific terminology, and polarity interpretation, are identified. The paper concludes by suggesting that future research

should focus on improving algorithmic accuracy and exploring real-time, multilingual sentiment analysis for a more nuanced understanding of passenger experiences.

In conclusion, the Sentiment Analysis of Water Metro Passengers Review paper underscores the potential of sentiment analysis as a powerful tool to enhance service quality in water metro systems. By leveraging techniques ranging from lexicon-based methods to advanced deep learning models like LSTM, CNN, and transformer-based architectures (BERT, RoBERTa), sentiment analysis can provide meaningful insights into passenger satisfaction and areas for improvement. While the field has seen advancements, challenges remain, such as handling multilingual data, accurately interpreting domain-specific terminology, and addressing nuances like sarcasm. Future research should aim to refine these models for higher accuracy and scalability, enabling real-time, multilingual sentiment analysis that captures a more comprehensive understanding of passenger experiences.

2.3 PROPOSED SYSTEM

The proposed system Aqua Motus is a dynamic web application designed to enhance the efficiency and user experience of the water metro service by automating ticket and facility bookings. It offers a centralized platform that allows users to easily access boat schedules, book tickets, and manage bookings online, eliminating the need for physical visits to stations. The system features dedicated modules for administrators, station masters, and users, enabling seamless management of services, routes, and resources. With real-time updates and comprehensive data analytics, the proposed system aims to streamline operations, improve resource allocation, and provide a more accessible and user-friendly experience for passengers, thereby supporting the sustainable growth of the water transportation infrastructure.

2.4 ADVANTAGES OF PROPOSED SYSTEM

- **Enhanced User Experience**: The Aqua Motus system simplifies the ticket booking process, allowing users to easily browse schedules, select routes, and manage bookings online, which leads to increased customer satisfaction and convenience.

- **Operational Efficiency**: By automating various administrative tasks such as service management and ticketing, the system reduces manual workload for station masters and administrators, enabling them to focus on strategic tasks and improving overall operational efficiency.

- **Real-Time Updates**: The system provides real-time information on boat schedules, availability, and weather conditions, helping users make informed decisions and improving safety during travel.

- **Data Analytics and Reporting**: Aqua Motus incorporates data analytics features that enable administrators to track key performance indicators, assess service demand, and generate reports, facilitating data-driven decision-making and resource allocation.

- **Scalability and Flexibility**: The modular design of the system allows for easy updates and scalability, enabling it to adapt to future expansions, additional services, or changes in operational needs without significant overhauls.

- **Cost-Effectiveness**: By streamlining operations and reducing reliance on manual processes, the system lowers operational costs and improves revenue generation opportunities through enhanced service delivery and customer engagement.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

The Aqua Motus project addresses the need for an efficient and user-friendly platform for managing water metro services, focusing on ticket booking, facility management, and operational oversight. The system targets individuals seeking boat transportation services, station masters managing daily operations, and administrators overseeing the entire system.

Feasibility is conducted to identify the best system that meets all requirements. It is both necessary and important to evaluate the feasibility of a project at the earliest possible time[15]. The feasibility study includes an identification description, an evaluation of proposed system and selection of the best system for the job. During the system is to be carried out this is to ensure that the proposed system is not A burden to the shop.

This feasibility study evaluates the practicality and viability of developing the Aqua Motus, a dynamic web application designed to streamline ticket and facility management for water metro systems. The system aims to enhance user experience through simplified ticket booking processes and comprehensive facility information access. The study covers technical, operational, and economic feasibility to determine the project's viability.

The feasibility study should be relatively cheap and quick. The results should inform the decision of whether to go ahead with a more detailed analysis, some understanding of the major requirements for the system is essential.

3.1.1 Economical Feasibility

Economic feasibility is the most frequently used method for evaluating the effectiveness of the candidate system. It is essential because the main goal of the proposed system is to have economically better results along with increased efficiency.

A cost evaluation is weighed against the ultimate income or product. Economic justification is generally the bottom-line consideration that includes cost benefit analysis, long term corporate income strategies and cost of resources needed for development and potential market growth. When compared to the advantage obtained from implementing the system its cost is affordable. The proposed system was developed with available resources since cost input for the software is almost nil the output of the software is always a profit. Hence software is economically feasible.

Revenue Generation:

- Potential revenue streams include transaction fees from ticket sales, advertising revenue, and partnership agreements with service providers.
- The system's ability to handle peak loads efficiently could lead to higher ridership, indirectly contributing to increased revenue.

3.1.2 Technical Feasibility

The technical feasibility centers on the existing system and to what extent it can support the proposed addition. The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. The minimum requirements of the system are met by the average user. The developer system has á modest technical requirement as only minimal or null changes are required for implementing system.

Normally associated with the technical feasibility includes:

- Development risk
- Resource availability
- Technology

The proposed system can work without any additional hardware or software support other than the computer system and networks. So, I analyzed that the proposed system is much more technically feasible than other systems when comparing with the benefits of the new system.

Current Resources and Technology:

- Python serves as the backend logic engine, leveraging its robustness and wide community support.
- Django, a Python web framework, provides a solid foundation for building secure and scalable web applications.
- SQLite database management system is suitable for small to medium-sized applications, offering ease of use and integration with Django.
- Machine Learning enhances system capabilities, including weather forecasting for safe travel planning, boat safety and security measures, and route optimization for efficient service delivery.

Analysis:

- The development team possesses the necessary technical skills in Python, Django, and machine learning, ensuring the successful implementation of the system.
- Existing technologies are stable, widely adopted, and supported by a vast community, minimizing risks associated with technology adoption.
- Integration with other systems, such as weather forecasting APIs and payment gateways, is feasible and beneficial for enhancing user experience and operational efficiency.

3.1.3 Behavioral Feasibility

People are inherently resistant to change, and the computer is known for facilitating changes and an estimate should be made of how strongly the user, staff reacts towards the development of the computerized system. In the existing system more manpower is required, and time factor is more. The more manpower for managing many files for dynamic data replication and more time for search through these files is needed. But in the proposed system, both manpower and time factors are reduced, and unnecessary burden is reduced. Thus, the remaining people are made to engage in some other important work. Also, there is no need to wait in case of downloading the data for the users therefore, the system is behaviorally feasible.

3.1.4 Operational Feasibility

Operational feasibility is a measure of how well a proposed system solves the problems and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.

Alignment with Business Objectives:

- The AQUA MOTUS system directly addresses the need for improved ticket booking and facility management within water metro systems, aligning with the business objectives of reducing wait times and enhancing customer satisfaction.
- The system's modular design allows for flexibility and scalability, adapting to future changes in service offerings or operational needs.

User Acceptance:

- Preliminary market research indicates strong demand for such a system among water metro users, station masters, and administrators.
- The system's user-friendly interface and comprehensive feature set are expected to facilitate quick adoption and acceptance by the target audience.

3.1.5 Feasibility Study Questionnaire

1. What is the status of the Kochi Water Metro project?
   The Kochi Water Metro project is currently operational, serving commuters with boat transportation services across Kochi city. It aims to reduce traffic congestion and provide a sustainable mode of transport.

2. How does the Kochi Water Metro contribute to reducing pollution levels in Kochi city?
   By providing a clean, efficient mode of transportation that reduces reliance on cars, the Kochi Water Metro contributes significantly to lowering air pollution levels in Kochi city, promoting a healthier living environment for residents.

3. What technologies are currently available to support the development of this system?
   The Kochi Water Metro system leverages advanced technologies to support its development and operation, ensuring it is a state-of-the-art and environment-friendly water transport system. Electric Hybrid Boats: The system uses low wash, twin-screw, aluminum-hulled, hybrid electric-powered boats. These boats are developed locally by technical experts/engineers in Kochi, combining the efficiency of electric power with the durability of aluminum hulls.

4. What are the key features of the Kochi Water Metro system?
   Key features include modern, eco-friendly boats, digital ticketing systems, and state-of-the-art terminals equipped with amenities for passengers.

5. How does the Kochi Water Metro integrate with other modes of public transportation?
   It integrates seamlessly with Kochi Metro Rail Limited (KMRL) services, allowing passengers to transfer between rail and water transport easily.

6. What measures are in place to ensure passenger safety on the Kochi Water Metro?
   Safety measures include regular boat inspections, trained crew members, emergency evacuation procedures, and adherence to international maritime standards.
7. How does the Kochi Water Metro contribute to reducing traffic congestion in Kochi city?
   By providing an alternative mode of transport, the Kochi Water Metro helps divert traffic from roads, thereby reducing congestion and improving overall mobility in the city.

8. What plans are there for future expansions or improvements to the Kochi Water Metro?
   Future plans include expanding the network to cover more areas, introducing new boat models, and upgrading terminal facilities to enhance passenger experience.

9. How has the public responded to the introduction of the Kochi Water Metro?
   Public response has been generally positive, with increasing numbers of commuters opting for water transport over traditional road-based options. Feedback suggests improvements in convenience and reliability.

10. How does the Kochi Water Metro compare to other forms of public transportation in terms of cost-effectiveness?
    Compared to traditional road-based public transportation, the Kochi Water Metro offers a more cost-effective solution by reducing traffic congestion and providing a faster, cleaner mode of transport, thus lowering overall operational costs and improving efficiency.

11. What are the key performance indicators (KPIs) used to evaluate the success of the Kochi Water Metro?
    Key performance indicators for the Kochi Water Metro include ridership numbers, passenger satisfaction rates, reduction in travel times, and environmental impact metrics such as carbon emissions savings.

12. What measures are in place to ensure the safety of passengers during adverse weather conditions?
    During adverse weather conditions, the Kochi Water Metro implements a series of safety measures, including adjusting schedules, rerouting boats where necessary, and providing passengers with real-time updates via mobile apps and announcements onboard.

3.2 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor - intel i5
RAM - 1 6 G B
Hard disk - S S D

3.2.2 Software Specification

Front End	-	HTML, CSS, JavaScript, Bootstrap
Backend	-	Python, Django Framework, DB SQLite
Client on PC	-	Windows 7 and above.
Technologies used	-	JS, HTML5, AJAX, J Query, PHP, CSS, Python, Django

3.3 SOFTWARE DESCRIPTION

3.3.1 Django

Django is a high-level Python web framework designed to support the development of robust and secure web applications efficiently[18]. It follows the Model-View-Template (MVT) architectural pattern, which provides a systematic way of organizing code for maintainability and scalability. Django simplifies web development by offering built-in modules and utilities that streamline common tasks, such as URL routing, database interaction, and form processing. It promotes the "Don't Repeat Yourself" (DRY) principle, allowing developers to reuse components across different projects and minimizing redundancy in code. Django also offers a powerful admin interface, automatically generated from the models defined by the developer. This admin interface allows for easy management of data and models, making it an essential tool for administrators and site managers.

Django's Object-Relational Mapping (ORM) system is another one of its core features, enabling developers to interact with databases using Python code rather than writing raw SQL queries. This improves development speed and reduces the likelihood of errors. The ORM simplifies querying, updating, and deleting records in the database, providing a clean, Pythonic way of interacting with data. Django also has strong support for user authentication, permission management, and role based access control, ensuring that user data and interactions are secure. Furthermore, the framework provides out-of-the-box protection against common security threats like cross-site scripting (XSS), SQL injection, and cross-site request forgery (CSRF), making it an excellent choice for developing secure web applications. With its modular design, Django can be easily extended to meet specific project needs, supporting both small-scale applications and enterprise-level projects with ease.

3.3.2 Python

Python is the primary programming language used in Django-based applications, renowned for its simplicity, readability, and versatility [17]. Its clean syntax allows developers to write efficient code quickly, which improves productivity, making Python ideal for both beginners and experienced developers. Python's vast ecosystem of libraries and frameworks extends its functionality to various domains, such as web development, data science, artificial intelligence, and automation. The Python Standard Library itself offers built-in modules for handling tasks such as file I/O, regular expressions, and data manipulation. Python's compatibility across multiple platforms also makes it easy to deploy applications on Windows, macOS, enhancing its flexibility.

The readability of Python not only improves the development experience but also ensures that maintaining and updating code is straightforward, even in complex applications. Python's active and vast developer community continually contributes to its rich ecosystem, providing libraries and tools that allow developers to integrate new technologies into their projects quickly. In the case of web development, Python's simplicity complements Django's high-level abstraction, allowing the focus to remain on building feature-rich applications rather than handling low-level programming challenges. The combination of Python's ease of use and Django's structured framework offers a powerful development environment that allows for faster prototyping, cleaner code, and more efficient scaling of applications over time.

3.3.3 DB SQLite

DB SQLite is the database engine used by Django by default and is particularly well-suited for small to medium-sized projects. Unlike other relational databases that require a separate server process (such as PostgreSQL or MySQL), DBSQLite operates directly from a single database file on the disk, which simplifies setup and configuration. This makes it an ideal choice for development environments and lightweight applications where database performance is not the critical bottleneck. With DBSQLite, there's no need to install or maintain a separate database server, which significantly reduces complexity, especially in the early stages of development. Despite its simplicity, DBSQLite adheres to the principles of ACID (Atomicity, Consistency, Isolation, Durability), ensuring that transactions are handled in a reliable and safe manner.

Even though it is lightweight, DBSQLite supports most SQL operations, including complex queries, joins, and indexing, making it fully functional for many applications. It integrates seamlessly with Django's ORM, allowing developers to define their data models in Python without worrying about writing SQL queries manually. This means that the database interaction layer is highly abstracted, enabling rapid development while maintaining a strong, logical structure for handling data. However, as DBSQLite is primarily designed for local development, it may not be as efficient for handling large-scale production applications with high concurrent user traffic. For larger projects, it is common to switch to more robust databases like PostgreSQL or MySQL during production deployment, but DBSQLite remains an excellent choice for prototyping, development, and smaller-scale applications.

CHAPTER 4

SYSTEM DESIGN

4.1 INTRODUCTION

Any engineered system or product's development process begins with design. A creative process is design. The secret to an efficient system is a decent design. Design is the process of thoroughly outlining a process or system so that it can be physically executed by using a variety of methodologies and concepts. It can be defined as the process of utilizing several approaches and concepts to precisely specify a component, a procedure, or a system to enable its physical realization. Regardless of the development paradigm employed, software design forms the technical core of the software engineering process. The system design creates the necessary architectural detail.

4.2 UML DIAGRAM

The components of the principles of object-oriented programming are represented by the language known as the Unified Modeling Language (UML), which is utilized in the industry of software engineering[19]. It serves as the standard definition of the entire software architecture or structure. Complex algorithms are solved and interacted with in Object- Oriented Programming by treating them as objects or entities. Anything can be one of these things. It could either be a bank manager or the bank itself. The thing can be a machine, an animal, a vehicle, etc. The issue is how we connect with and control them, even though they are capable of and ought to execute duties. Interacting with other objects, sending data from one object to another, manipulating other objects, etc., are examples of tasks. There could be hundreds or even thousands of objects in a single piece of software.

UML include the following diagrams:

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

4.2.1 Use Case Diagram

A use case diagram is a visual representation of the interactions between system components. A approach for identifying, outlining, and organizing system requirements is called a use case. The word "system" here refers to a thing that is being created or run, like a website for mail-order product sales and services. UML (Unified Modeling Language), a standard language for the modelling of real-world objects and systems, uses use case diagrams. The planning of general requirements, the validation of a hardware design, the testing and debugging of a software product in development, the creation of an online help reference, or the completion of a job focused on customer support are all examples of system objectives. For instance, use cases in a product sales context can involve customer service, item ordering, catalogue updating, and payment processing.

There are four elements in a use case diagram.

- The boundary, which defines the system of interest in relation to the world around it.
- The actors, usually individuals involved with the system, are 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.

Use case diagrams are drawn to capture the functional requirements of a system. After identifying the above items, we must use the following guidelines to draw an efficient use case diagram.

- The name of a use case is very important. The name should be chosen in such a way so that it can identify the functionalities performed.
- Give a suitable name for actors.
- Show relationships and dependencies clearly in the diagram.
- Do not try to include all types of relationships, as the main purpose of the diagram is to identify the requirements.
- Use notes whenever required to clarify some important points.

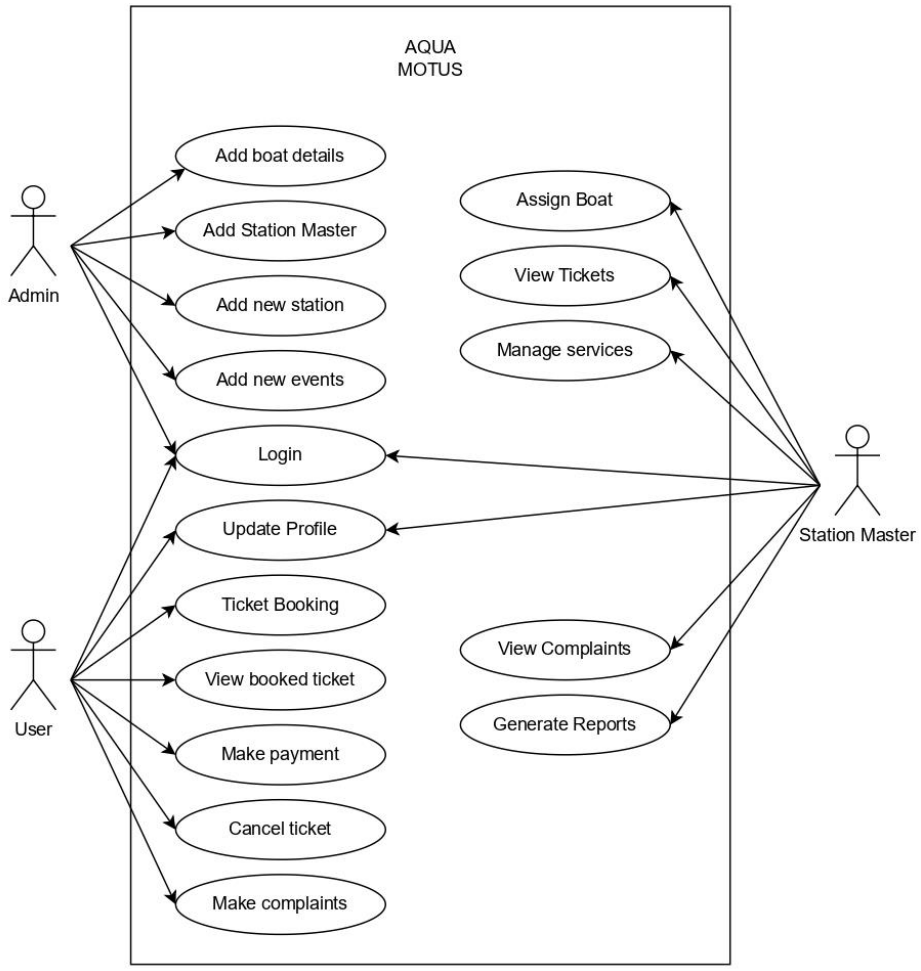

Figure 1: Use case diagram

4.2.2 Sequence Diagram

A sequence diagram essentially shows how things interact with one another sequentially, or the order in which these interactions occur. A sequence diagram can also be referred to as event diagrams or event scenarios. Sequence diagrams show the actions taken by the components of a system in chronological order. Businesspeople and software engineers frequently use these diagrams to record and comprehend the requirements for new and current systems.

Sequence Diagram Notations:

- **Actors** – In a UML diagram, an actor represents a particular kind of role that interacts with the system and its objects. An actor is always beyond the purview of the system that we want to use the UML diagram to represent. We employ actors to portray a variety of roles, including those of human users and other outside subjects. In a UML diagram, an actor is represented using a stick person notation. In a sequence diagram, there might be several actors.

- **Lifelines** – A lifeline is a named element in a sequence diagram that represents an individual participant. So, in a sequence diagram, each incident is represented by a lifeline. A sequence diagram's lifeline elements are at the top.

Messages – Messages are used to show how objects communicate with one another. The messages are displayed on the lifeline in chronological sequence. Arrows are how messages are represented. A sequence diagram's main components are lifelines and messages.

Messages can be broadly classified into the following categories:

- Synchronous messages
- Asynchronous Messages
- Create message
- Delete Message
- Self-Message
- Reply Message
- Found Message
- Lost Message

Guards – To model conditions we use guards in UML. They are used when we need to restrict the flow of messages on the pretext of a condition being met. Guards play an important role in letting software developers know the constraints attached to a system or a particular process.

Uses of sequence diagrams:

- Used to model and visualize the logic behind a sophisticated function, operation or procedure.
- They are also used to show details of UML use case diagrams.
- Used to understand the detailed functionality of current or future systems.
- Visualize how messages and tasks move between objects or components in a system.

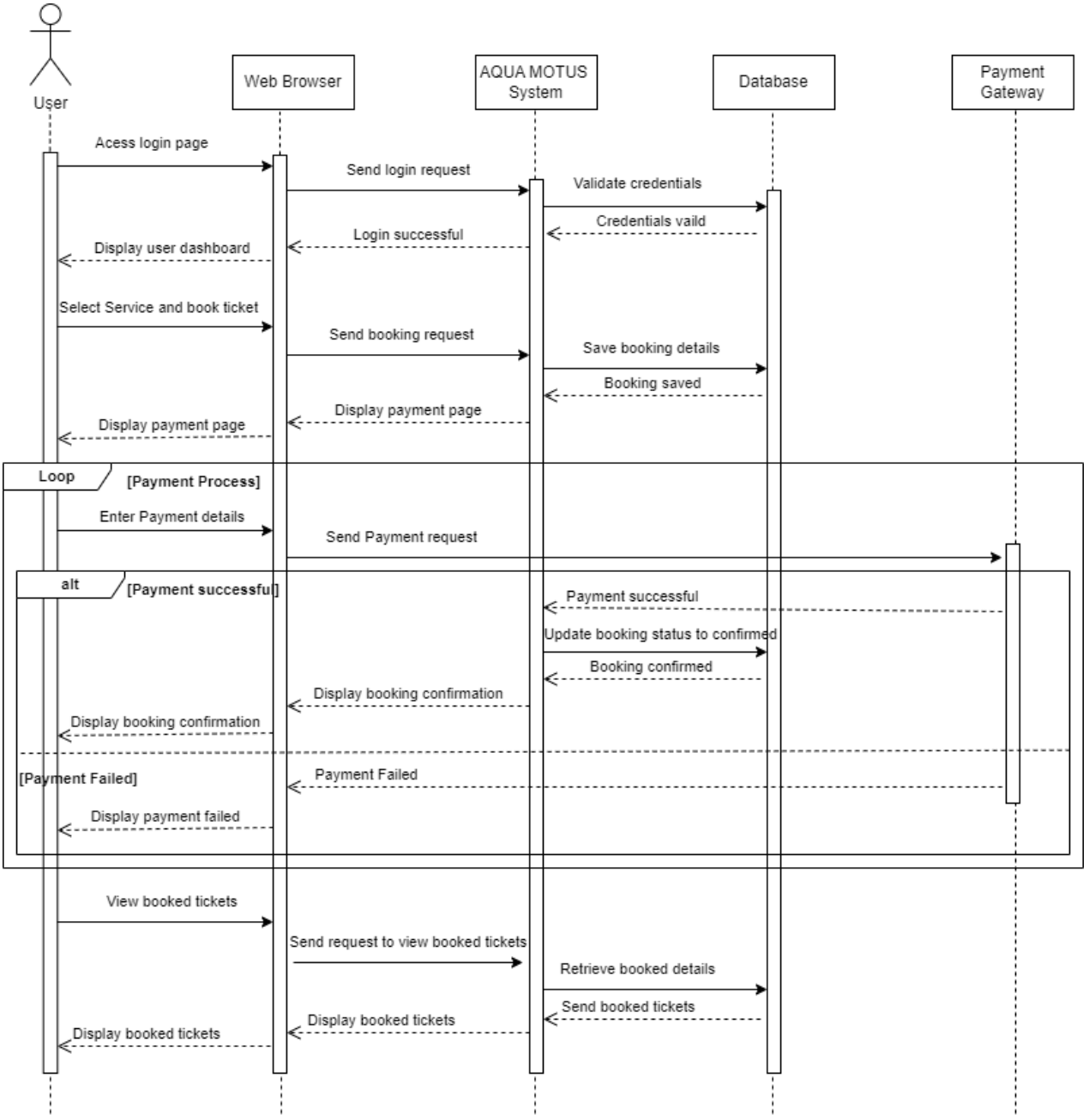

Figure 2: Sequence Diagram

4.2.3 State Chart Diagram

A state diagram, also known as a state machine diagram or state chart diagram, is an illustration of the states an object can attain as well as the transitions between those states in the Unified Modeling Language (UML).

4.2.4 Activity Diagram

Another crucial UML diagram for describing the system's dynamic elements is the activity diagram. An activity diagram is essentially a flowchart that shows how one activity leads to another. The action might be referred to as a system operation. One operation leads to the next in the control flow. This flow may be parallel, contemporaneous, or branched. Activity diagrams use many features, such as fork, join, etc., to cope with all types of flow control. An activity diagram is a behavioral diagram i.e., it depicts the behavior of a system. An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.

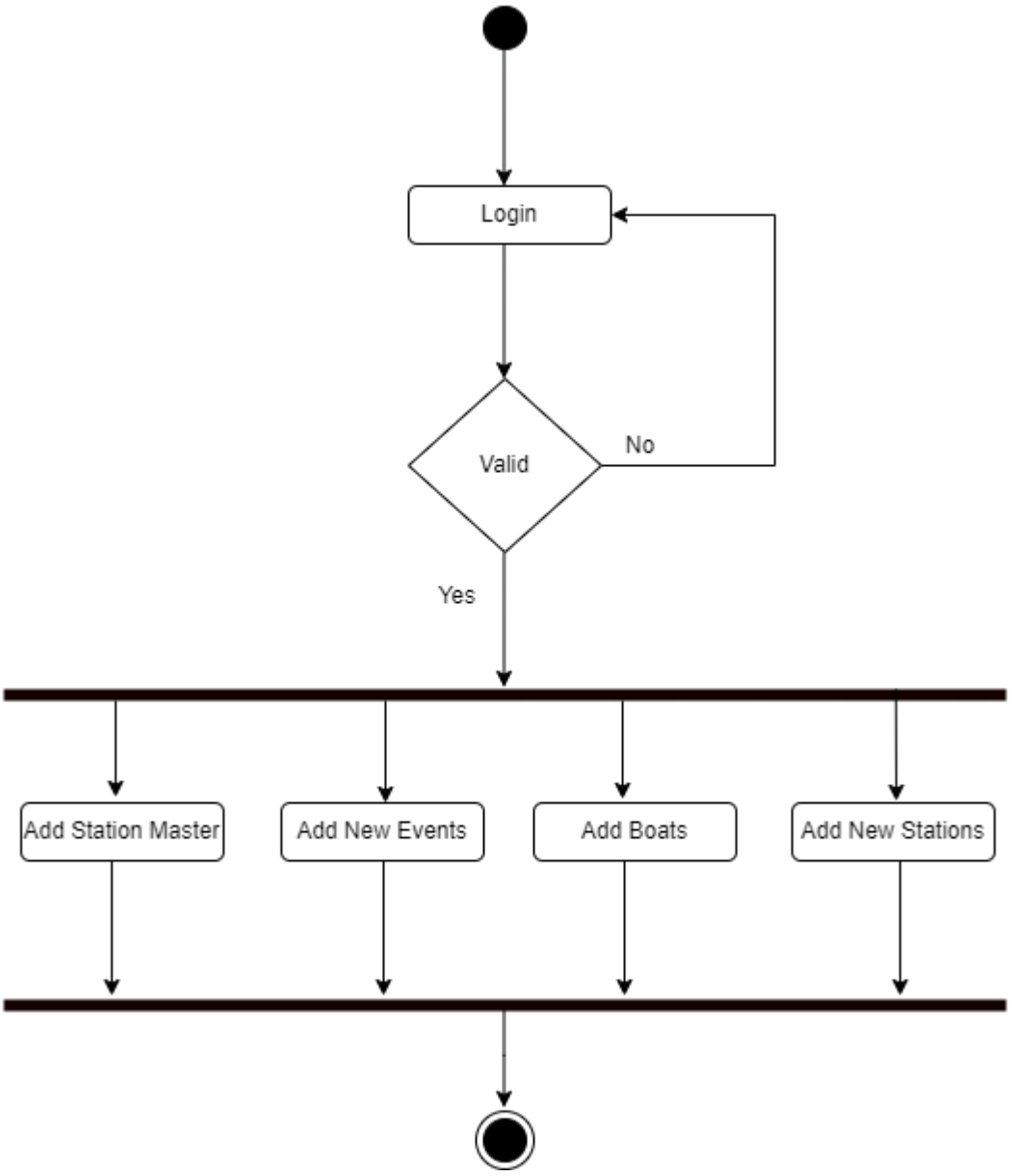

Figure 3: Activity Diagram of Admin

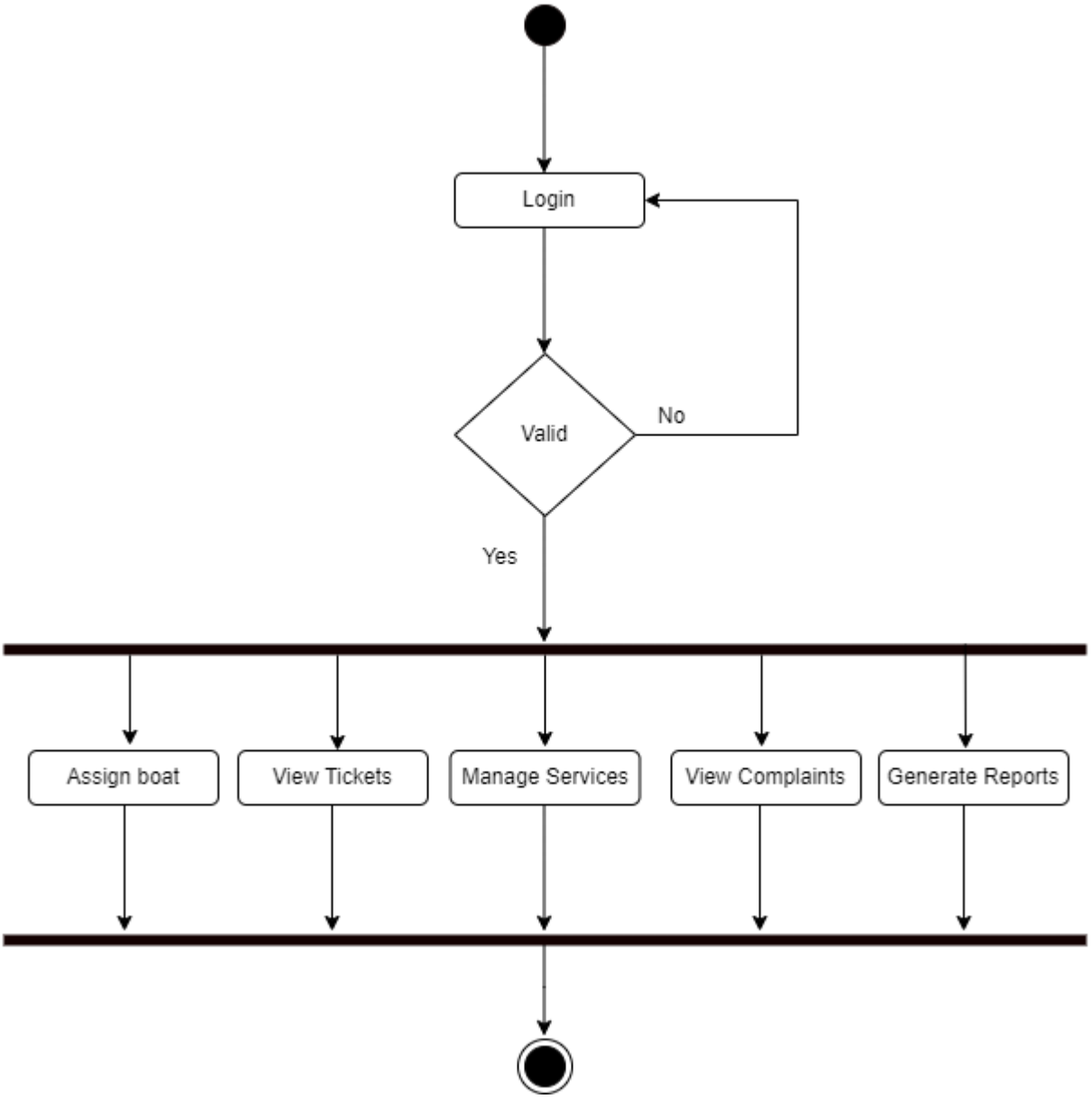

Figure 4: Activity Diagram of Station Master

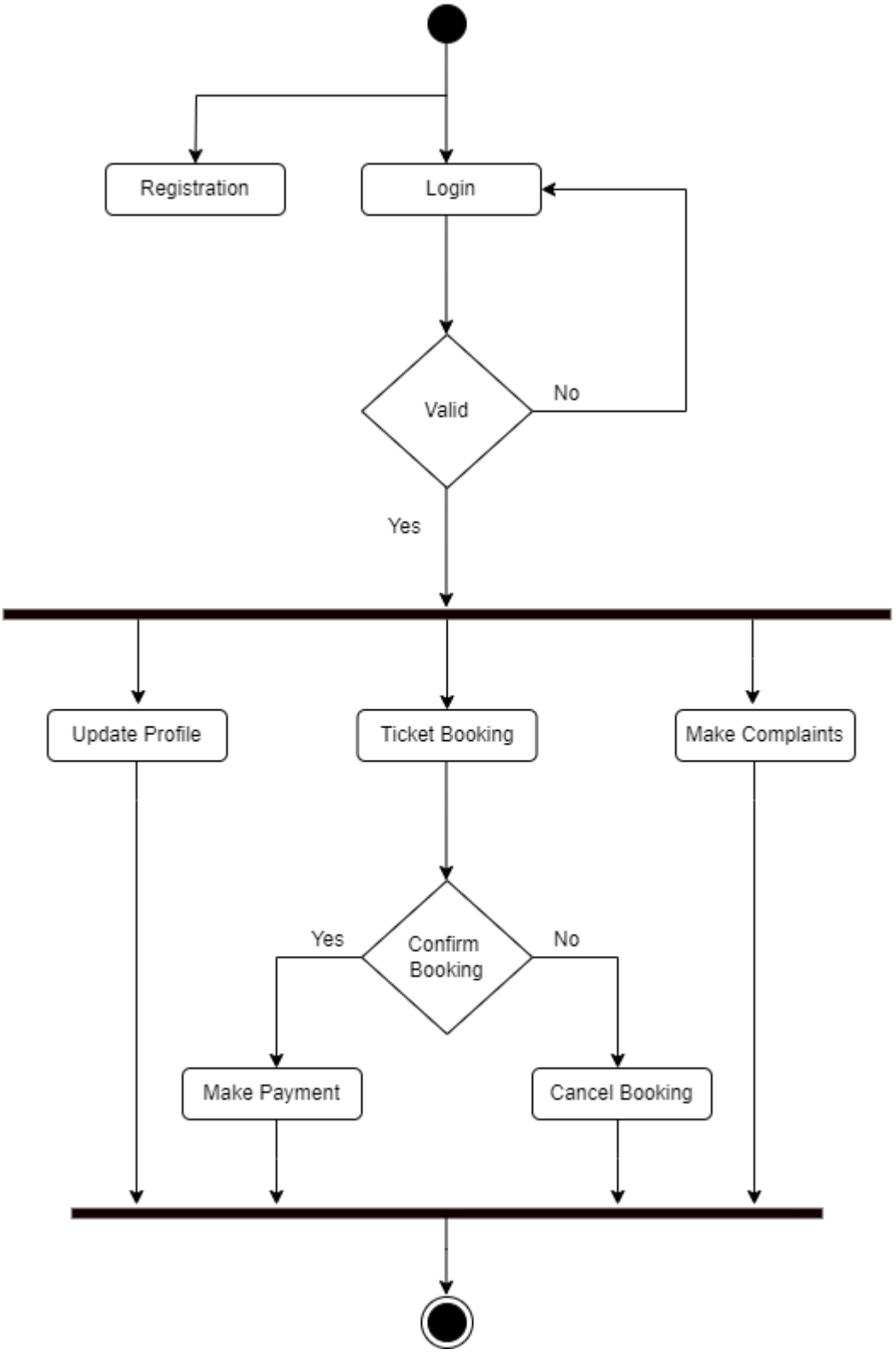

Figure 5: Activity Diagram of User

4.2.5 Class Diagram

Class diagrams is a Static diagram. It represents the application's static view. Class diagrams are used to create executable code for software applications as well as for visualizing, explaining, and documenting various elements of systems. The characteristics and functions of a class are described in a class diagram, along with the restrictions placed on the system. Because they are the only UML diagrams that can be directly transferred to object-oriented languages, class diagrams are frequently employed in the modelling of object-oriented systems. A collection of classes, interfaces, affiliations, collaborations, and constraints are displayed in a class diagram.

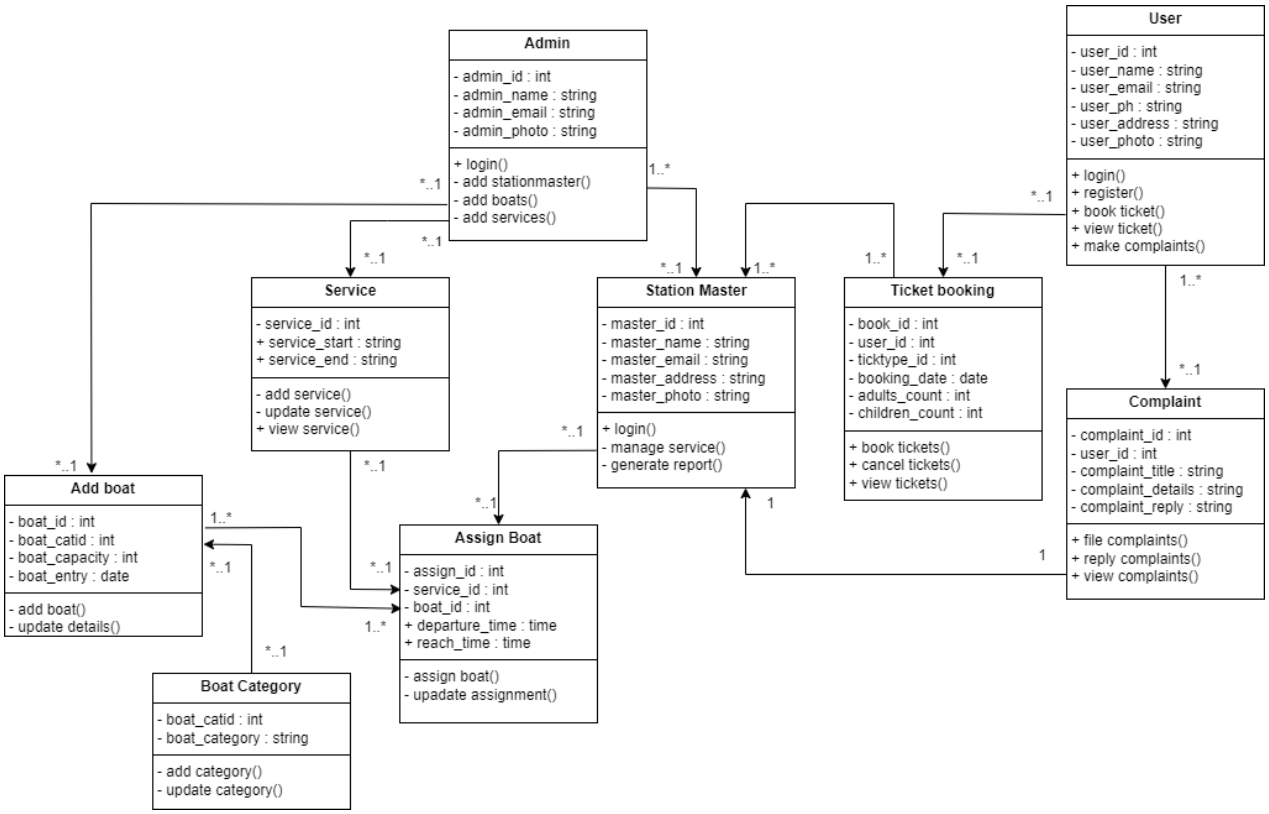

Figure 6: Class Diagram

4.2.6 Object Diagram

Since class diagrams are the source of object diagrams, class diagrams are a prerequisite for object diagrams. An instance of a class diagram is represented by an object diagram. Class and object diagrams both use the same fundamental ideas. The static view of a system is also represented by object diagrams, but this static view represents a momentary snapshot of the system. To represent a group of items and their connections as an instance, object diagrams are employed.

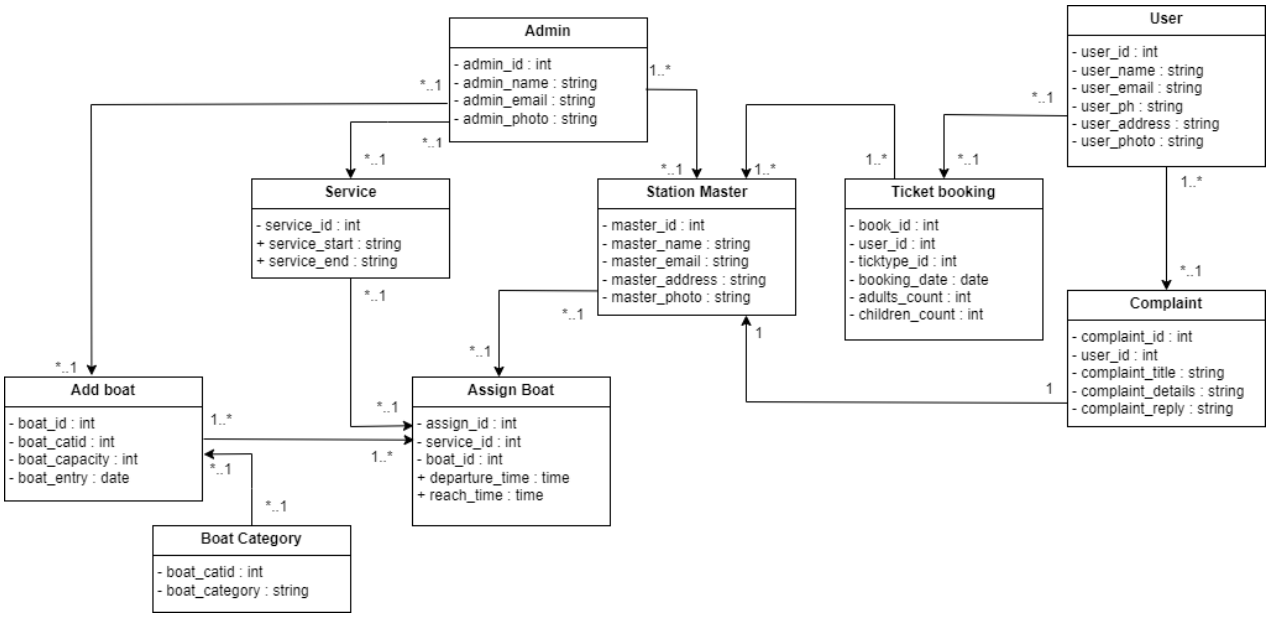

Figure 7: Object diagram

4.2.7 Component Diagram

Component diagrams come in a variety of behaviors and personalities. The physical parts of the system are represented using component diagrams. Executables, libraries, files, documents, and other items that are physically present in a node are just a few examples. Component diagrams are used to show how the components of a system are connected and arranged. These diagrams can also be used to construct systems that can be run.

4.2.8 Deployment Diagram

An execution architecture of a system, containing nodes like hardware or software execution environments, and the middleware linking them, is shown in a deployment diagram, a form of UML diagram. Typically, deployment diagrams are used to represent the actual hardware and software of a system. By using it, you can comprehend how the hardware will physically deliver the system. In contrast to other UML diagram types, which primarily depict the logical components of a system, deployment diagrams assist in describing the hardware structure of a system.

4.2.9 Collaboration Diagram

A collaboration diagram, also known as a communication diagram, is a type of behavior diagram in Unified Modeling Language (UML) that shows the interactions between objects and their

messages in a system. It is used to visualize the interactions and communication patterns between objects in a system. In a collaboration diagram, the objects that participate in the interaction are represented by rectangles, and the messages exchanged between them are represented by arrows. The sequence of messages is shown from top to bottom, with the initial message at the top and the final message at the bottom.

Collaboration diagrams can be used to model any system that involves multiple objects and interactions between them, such as a software system, a business process, or a physical system. They are useful for identifying the objects that participate in the interaction, and for understanding the sequence of messages exchanged between them.

4.3 USER INTERFACE DESIGN USING FIGMA

Form Name: Login

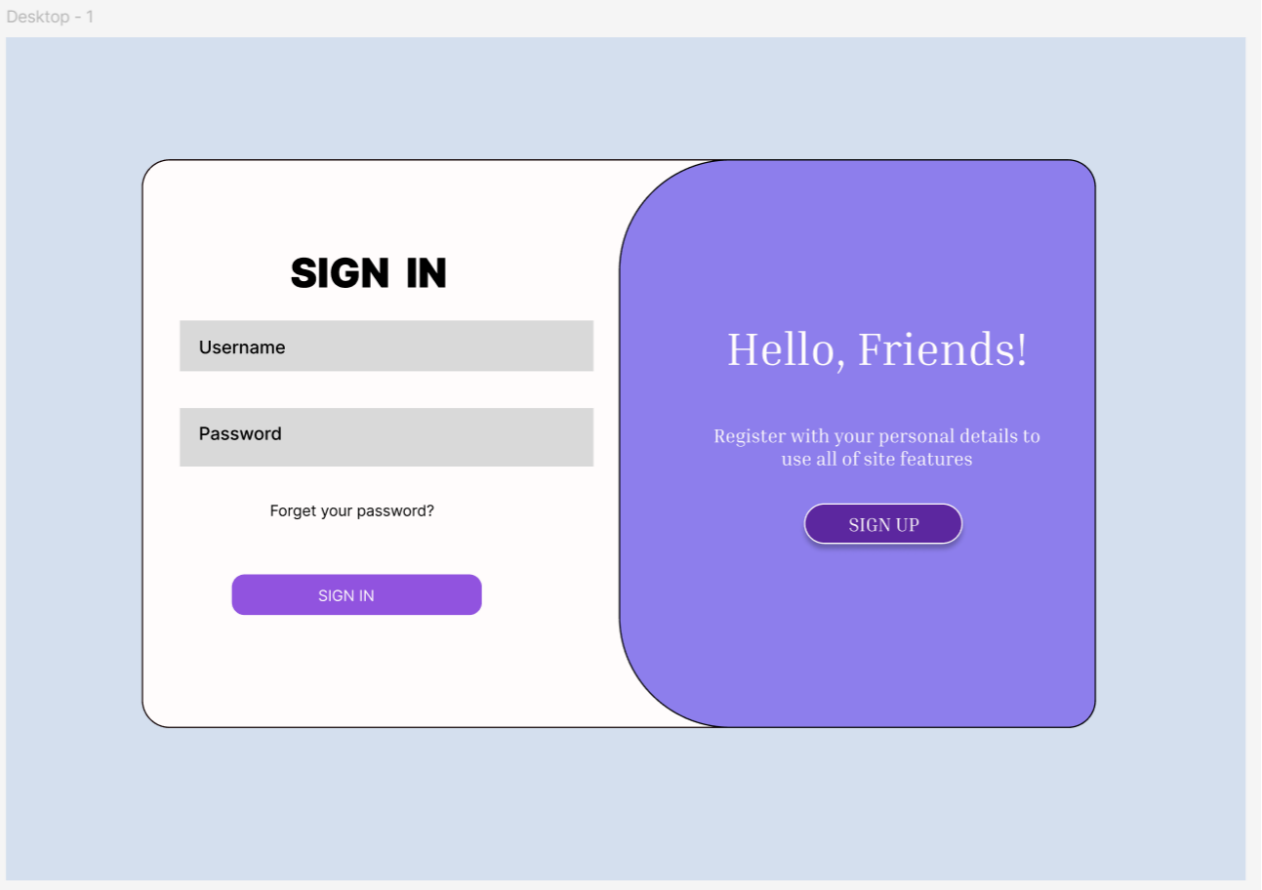

Figure 8: Login Page Design

Form Name: Registration

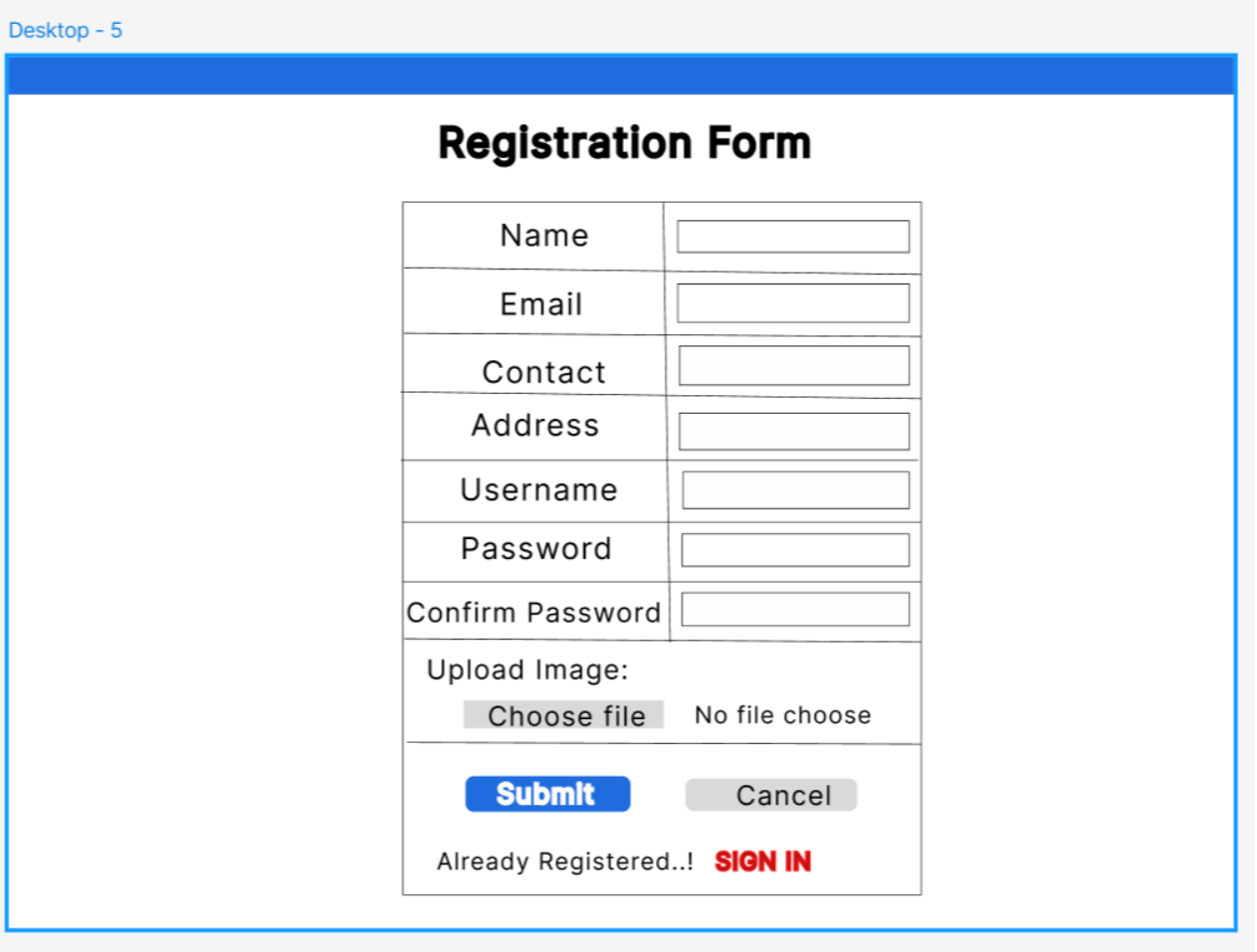

Figure 9: Registration Page Design

Form Name: Station Master Registration by Admin

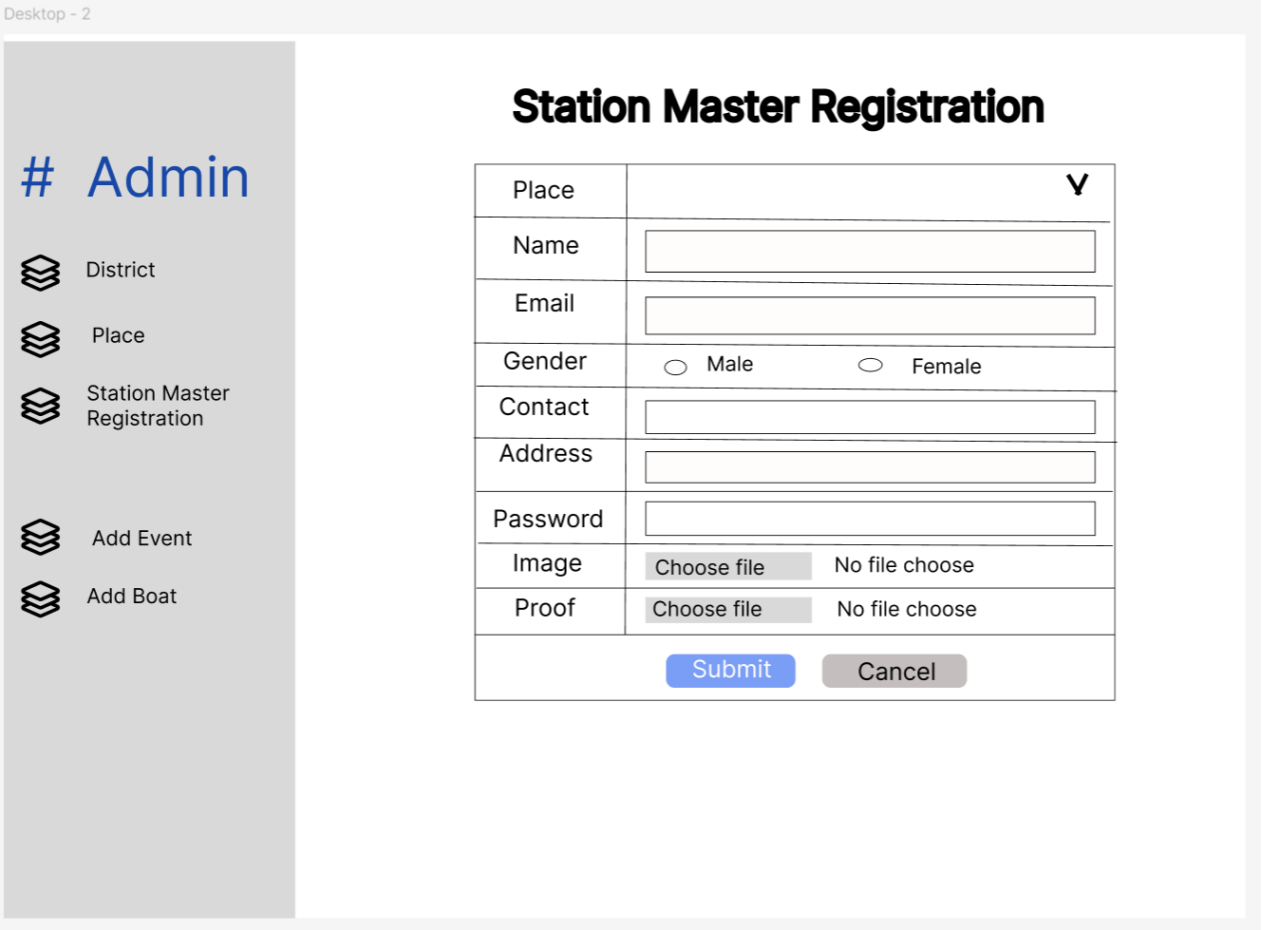

Figure 10: Station Master Registration Design

Form Name: Assign Boat Service by Station Master

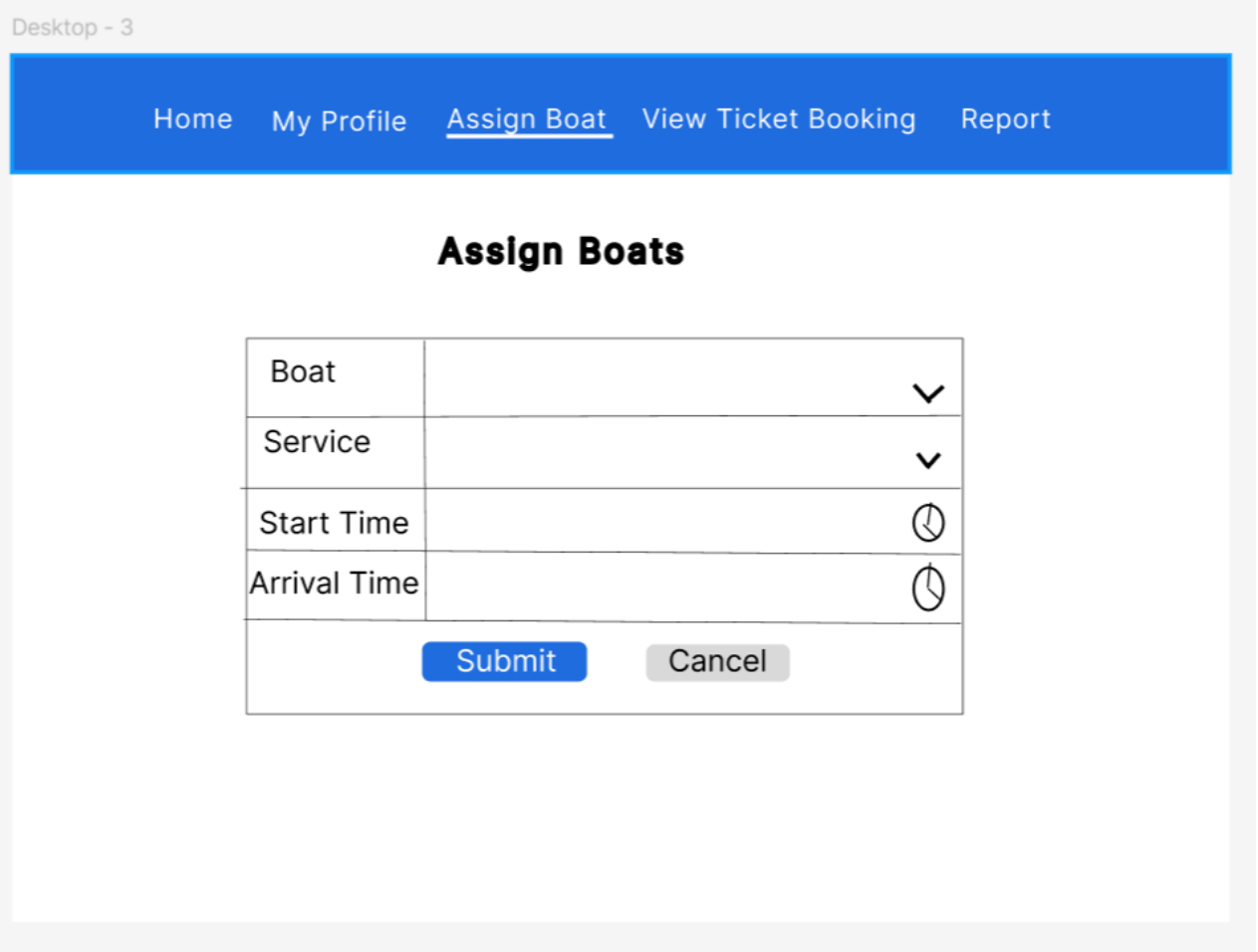

Figure 11: Assign Boat Design

Form Name: Ticket Book by User

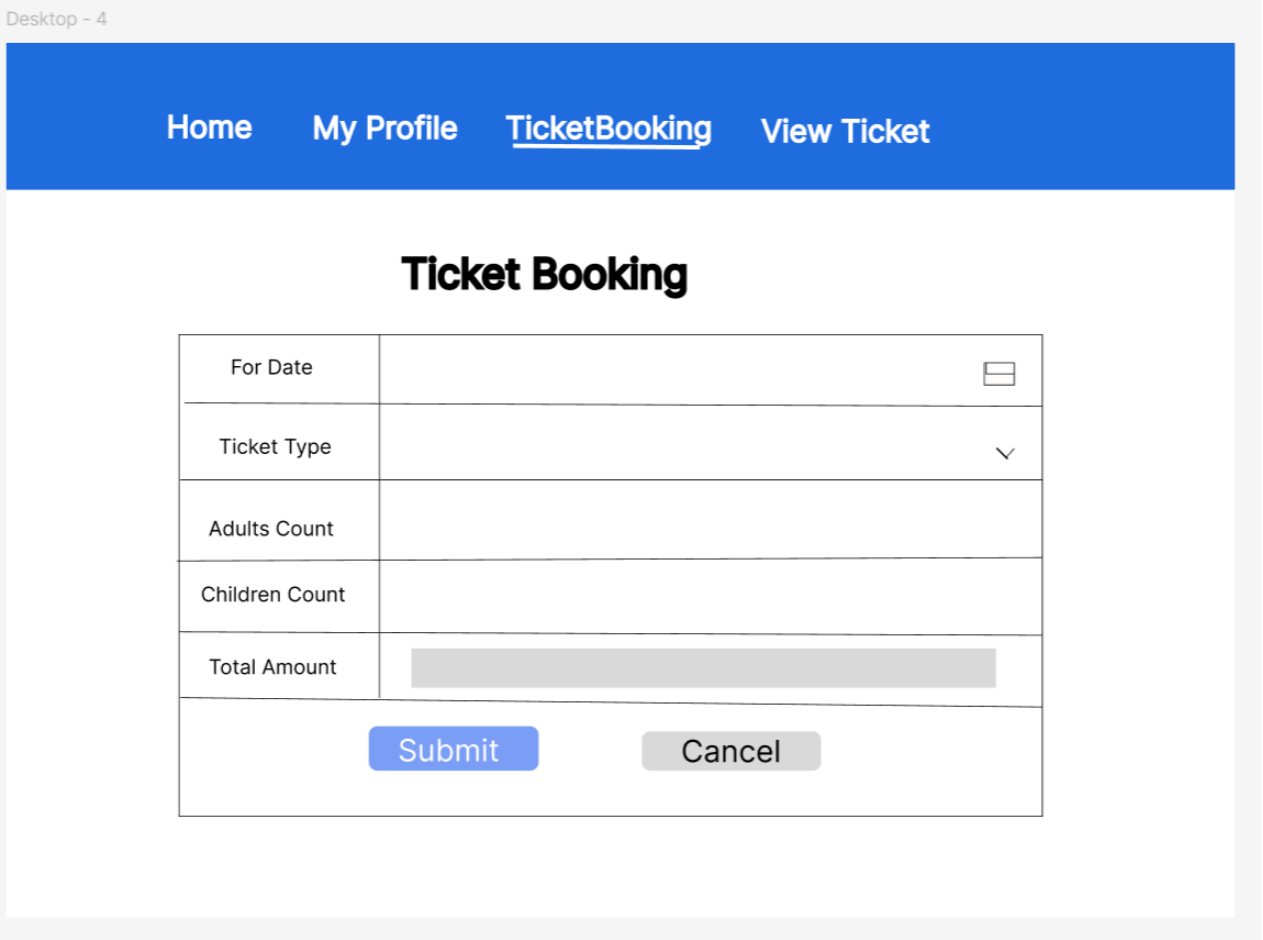

Figure 12: Ticket Book Page Design

4.4 DATABASE DESIGN

A database is a structured system with the capacity to store information and allows users to retrieve stored information quickly and effectively[16]. Any database's primary goal is its data, which demands protection. There are two stages to the database design process. The user needs are obtained in the first step, and a database is created to as clearly as possible meet these criteria. This process, known as information level design, is carried out independently of all DBMS. The design for the specific DBMS that will be used to construct the system in issue is converted from an information level design to a design in the second stage. Physical Level Design is the stage where the characteristics of the particular DBMS that will be used are discussed. Parallel to the system design is a database design. The database's data arrangement aims to accomplish the two main goals listed below.

- Data Integrity
- Data Independence

4.4.1 Relational Database Management System (RDBMS)

In a relational model, the database is shown as a set of relations. Each relation resembles a file or table of records with values. A row is referred to as a tuple, a column heading is referred to as an attribute, and the table is referred to as a relation in formal relational model language. A relational database is made up of a number of tables, each with its own name. In a story, each row represents a group of associated values. Relations, Domains & Attributes. A relation is a table. Tuples are the units of a table's rows. An ordered group of n elements is a tuple. Attributes are referred to as columns. Every table in the database has relationships already established between them. This guarantees the integrity of both referential and entity relationships. A group of atomic values make up a domain D. Choosing a data type from which the domain's data values are derived is a typical way to define a domain. To make it easier to understand the values of the domain, it is also helpful to give it a name. Each value in a relation is atomic and cannot be broken down.

Relationships

- Table relationships are established using Key. The two main keys of prime importance Primary Key & Foreign Key. Entity Integrity and Referential Integrity Relationships can be established with these keys.
- Entity Integrity enforces that no Primary Key can have null values.
- Referential Integrity enforces that no Primary Key can have null values.

- Referential Integrity for each distinct Foreign Key value, there must exist a matching Primary Key value in the same domain. Other keys are Super Key and Candidate Keys.

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. Data normalization is a formal process. structures in ways that encourage integrity and remove duplication. Normalization is a method of dividing large datasets into smaller ones and removing superfluous fields. Into a smaller table. Additionally, it serves to prevent additions, deletions, and updates. Anomalies. Keys and relationships are two notions commonly used in data modelling. A table row is uniquely identified by its key. key uniquely identifies a row in a table. There are two types of keys, primary key and foreign key. A 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 means placing things in their natural form, as the name suggests. By using normalization, the application developer aims to establish a coherent arrangement of the data into appropriate tables and columns, where names may be quickly related to the data by the user. By removing recurring groups in front of the data, normalization prevents data redundancy, which puts a heavy strain on the computer's resources. include:

- Normalize the data.
- Choose proper names for the tables and columns.
- Choose the proper name for the data.

First Normal Form

According to the First Normal Form, each attribute's domain must only include atomic values, and each attribute's value in a tuple must be a single value from that domain. In other words, 1NF forbids using relationships as attribute values within tuples or relations within relations. Single atomic or indivisible values are the only attribute values that are permitted under 1NF. The data must first be entered into First Normal Form. This can be accomplished by separating data into tables of a similar type in each table. Each table is given a Primary Key or Foreign Key as per the requirement of the project. In this we form new relations for each non-atomic attribute or nested relation. This eliminated repeating groups of data. A relation is said to be in first normal form if only if it satisfies the constraints that contain the primary key only.

Example:

The "Student_Info" table before normalization:

Student_ID	Name	Subjects	Marks
101	Ammu	Maths, Bio, Mala	90
102	Achu	Chem, Phy	85
103	Anju	Hindi, Maths	80

The "Student_Info" table after 1NF:

Student_ID	Name	Subjects	Marks
101	Ammu	Maths	90
101	Ammu	Bio	90
101	Ammu	Mala	90
102	Achu	Chem	85
102	Achu	Phy	85
103	Anju	Hindi	80
103	Anju	Maths	80

Second Normal Form

No non-key attribute should be functionally dependent on a portion of the main key for relations where the primary key has several attributes, according to Second Normal Form. This involves breaking down each partial key into its dependent characteristics and setting up a new relation for each one. Keep the original primary key and any properties that are entirely dependent on it in your database. This procedure aids in removing data that depends only on a small portion of the key. If and only if a relation satisfies all the requirements for first normal form for the primary key and all of the non-primary key qualities of the relation are completely dependent on the primary key alone, then that relation is said to be in second normal form.

Example:

The "Student_Info" table after 2NF:

Table 1: students

Student_ID	Student_Name
101	Ammu
102	Achu
103	Anju

Table 2: subjects

Subject_ID	Subject_Name
1	Maths
2	Chemistry
3	Hindi

Table 3: student_marks

Student_ID	Subject_ID	Marks
101	1	90
101	2	90
101	3	90
102	2	85
102	3	85
103	1	80
103	2	80

Third Normal Form

Relation should not have a non-key attribute that is functionally determined by another non-key attribute or by a collection of non-key attributes, according to the Third Normal Form. The primary key should not be transitively dependent, in other words. The non- key attributes that functionally determine other non-key attributes are decomposed in this way put up in relation. This procedure is used to eliminate anything not wholly dependent on the Primary Key. Only when a relation is in second normal form and, more importantly, when its non-key characteristics do not depend on those of other non-key attributes, is it considered to be in third normal form.

Example:

The "Student_Info" table after 3NF:

Table 1: students

Student_ID	Student_Name
101	Ammu
102	Achu
103	Anju

Table 2: subjects

Subject_ID	Subject_Name
1	Maths
2	Chemistry
3	Hindi

Table 3: student_subjects

Student_ID	Subject_ID
101	1
101	2
101	3
102	2
102	3
103	1
103	2

Table 4: student_marks

Student_ID	Subject_ID	Marks
101	1	90
101	2	90
101	3	90
102	2	85
102	3	85
103	1	80
103	2	80

Fourth Normal Form

The fourth normal form (4NF) is a database normalization rule that further refines data modeling by addressing multi-valued dependencies. When a table contains multiple independent sets of repeating data, we can break it down into smaller tables, with each table containing only one set of related data. This reduces data redundancy and improves data consistency by ensuring that each table represents a single, well-defined concept or entity. To achieve 4NF, we need to ensure that all multi-valued dependencies are removed from the table, and that each table contains only attributes that are functionally dependent on the primary key.

Fifth Normal Form

5NF is indeed the highest level of normalization in relational database design, and it deals with complex data models that involve multiple overlapping multi-valued dependencies. In 5NF, tables are decomposed into smaller tables in order to eliminate any possible redundancy caused by overlapping dependencies, while ensuring that there is no loss of data.

The goal of 5NF is to ensure that each table represents a single entity or relationship, and that the data is organized in a way that minimizes redundancy, eliminates anomalies, and improves data integrity. By breaking down tables into smaller, more specialized tables, 5NF helps to eliminate potential issues with update anomalies, insertion anomalies, and deletion anomalies, which can occur when data is not properly normalized.

4.4.3 Sanitization

Data sanitization means that we remove all dangerous characters from an input string before passing it to the SQL engine. SQL injection occurs when an attacker is able to query or modify a database due to poor input sanitization. Python filters are used to sanitize and validate external input. The Python filters extension has many functions needed for checking user input, and is designed to do data sanitization easier and quicker. Hence all the user inputs are sanitized/filtered before passing it to the database.

4.4.4 Indexing

Indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed. It is a data structure technique which is used to quickly locate and access data in a database. Indexes are created using one or more database columns. Indexing is also done to optimize various query responses.

4.5 TABLE DESIGN

1. Table Name: tbl_district

Description: District details

Primary key: id

Foreign key: none

Sl.no	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Integer	Primary Key	Id of district
2	district_name	Varchar(50)	Not null	Name of district
3	status	Integer	Not null	Active or Inactive

2. Table Name: tbl_place

Description: Place details

Primary key: id

Foreign Key: district_id references table tbl_district

Sl.no	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Integer	Primary key	Id of place
2	district_id	Integer	Foreign Key	Id of district
3	place_name	Varchar(50)	Not null	Name of place
4	status	Integer	Not null	Active or Inactive

3. Table Name: feedback

Description: Feedback details

Primary key: id

Foreign Key: user_id references table customdetails

Sl.no	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Integer	Primary key	Id of feedback
2	user_id	Integer	Foreign Key	Id of user
3	title	Varchar(50)	Not null	Title of feedback
4	description	Varchar(100)	Not null	Details of feedback
5	reply	Varchar(50)	Null	Reply of feedback
6	date	Date	Not null	Date of feedback
7	status	Integer	Not null	Active or Deactive
8	Sentiment_score	Float	Not null	Sentiment score for analysis

4. Table Name: tbl_ticketbooking

Description: Ticket booking details

Primary key: id

Foreign Key: service_id references table tbl_services, user_id references table customdetails

Sl.no	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Integer	Primary key	Id of booking
2	user_id	Integer	Foreign Key	Id of User
3	service_id	Integer	Foreign Key	Id of service
4	date	Date	Not null	Booking date
5	adults_count	Integer	Null	Count of adults
6	children_count	Integer	Null	Count of children’s
7	ticket_number	Varchar(20)	Not null	Number of the ticket booking
8	book_amount	Decimal	Not null	Amount of the ticket
9	payment_id	Varchar(100)	Null	Id of the payment done
10	refund_amount	Decimal(10,2)	Null	Refund amount
11	payment	Integer	Not null	0 for pending, 1 for complete

5. Table Name: tbl_eventbooking

Description: Event Ticket booking details

Primary key: id

Foreign Key: service_id references table tbl_services, user_id references table customdetails, event_startpoint and event_endpoint references tbl_place, assign reference tbl_boat

Sl.no	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Integer	Primary key	Id of booking
2	user_id	Integer	Foreign Key	Id of User
3	event_type_id	Integer	Foreign Key	Id of service
4	assign	Integer	Foreign Key	Boat assign for the event
5	event_startpoint	Integer	Foreign Key	Starting point of the event
6	event_endpoint	Integer	Foreign Key	Departure point of the event
7	adults_count	Integer	Null	Count of adults
8	children_count	Integer	Null	Count of children’s
9	event_number	Varchar(20)	Not null	Number of the ticket booking
10	event_stattime	Time	Not null	Start time of the event
11	duration	Duration	Not null	Duration of the event
12	boat_deck	Varchar(50)	Not null	Single duck or Double duck
13	event_date	Date	Not null	Booking date
14	payment_id	Varchar(100)	Null	Id of the payment
15	refund_amount	Demical(10,2)	Null	Refund amount
16	status	Integer	Not null	Paid or not paid nor assign boat or cancel

6. Table Name: tbl_service

Description: Service details

Primary key: id

Foreign key: service_startpoint_id, service_endpoint_id references tbl_place, Assignboat_boat_id reference tbl_boat

Sl.no	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Integer	Primary key	Id of service
2	service_startpoint_id	Integer	Foreign Key	Service start point id
3	service_endpoint_id	Integer	Foreign Key	Service end point id
4	Assignboat_boat_id	Integer	Foreign Key	Boat of the service assign
5	assignboat_starttime	Time	Not null	Start time of the service
6	duration	Duration	Not null	Time duration of the service
7	rate	Float	Not null	Rate of the service
8	status	Integer	Not null	Active or Inactive

7. Table Name: tbl_boat

Description: Boat details

Primary key: id

Foreign key: None

Sl.no	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Integer	Primary key	Id of boat
2	boat_name	Varchar(50)	Not null	Name of the boat
3	boat_capacity	Integer	Not null	Boat capacity
4	boat_entrydate	Date	Not null	Boat entry date
5	boat_service	Varchar(50)	Not null	Public or Tourism
6	boat_deck	Varchar(50)	Not null	Single or Double deck
7	status	Integer	Not null	Active or Inactive

8. Table Name: tbl_eventtype

Description: Event details

Primary key: id

Foreign key: None

Sl.no	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Integer	Primary key	Id of event type
2	event_type	Varchar(50)	Not null	Name of the event type
3	event_rate	Integer	Not null	Rate of the event
4	event_details	Varchar(50)	Not null	Details of the event
5	status	Integer	Not null	Active or Inactive

9. Table Name: tbl_stationmaster

Description: Station Master details

Primary key: id

Foreign Key: master_place_id references table tbl_place, user_id references table customdetails

Sl.no	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Integer	Primary key	Id of station master
2	User_id	Integer	Foreign Key	Id of Abstract Table(customdetails)
3	master_place_id	Integer	Foreign Key	Id of Place
4	master_gender	Varchar(50)	Not null	Gender of station master
5	master_contact	Integer (12)	Not null	Phone number of station master
6	master_address	Varchar(50)	Not null	Address of station master
7	master_photo	Varchar(50)	Not null	Photo of station master
8	master_proof	Varchar(50)	Not null	Proof of station master
9	status	Integer	Not null	Active or Inactive

10. Table Name: profile

Description: User details

Primary key: id

Foreign Key: user_id references table customdetails (Abstract Table)

Sl.no	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Integer	Primary key	Id of user
2	user_id	Integer	Foreign Key	Id of Abstract Table(customdetails)
3	user_gender	Varchar(50)	Not null	Gender of user
4	user_contact	Integer (15)	Not null	User phone number
5	user_address	Varchar(50)	Not null	Address user
6	user_photo	Varchar(50)	Not null	Photo user

11. Table Name: customdetails

Description: Abstract table with login details

Primary key: id

Foreign Key: None

Sl.no	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Integer	Primary key	Id of login
2	username	Varchar(50)	Not null	Username for login
3	password	Varchar(50)	Not null	User password for login
4	first_name	Varchar(50)	Not null	First Name of the user
5	last_name	Varchar(50)	Not null	Last Name of the user
6	email	Varchar(40)	Not null	Email of the user
7	role	Varchar(20)	Not null	Role indicates type of the login – Admin, Station Master, User

CHAPTER 5

SYSTEM TESTING

5.1 INTRODUCTION

Software testing is the procedure of meticulously monitoring the way software is used to see if it works as expected. Software testing is usually used in conjunction with the words validation and verification[20]. A product, including software, is validated by being examined or evaluated to see if it conforms to all pertinent requirements. Software testing, one sort of verification, also makes use of reviews, analyses, inspections, and walkthroughs. Validation is the process of ensuring that what has been specified corresponds to what the user wants. Other procedures that are typically connected to software testing include static analysis and dynamic analysis. Without running the code, static analysis examines the software's source code to look for errors and gather statistics. Dynamic analysis looks at the behavior of software while it is in use to provide information like execution traces, timing profiles, and test coverage specifics. Running a program with the intention of finding any faults is how a program is tested. Testing is a group of tasks that can be organized in advance and completed in a systematic way. Individual modules are tested first, followed by the integration of the entire computer-based system. There are numerous regulations that can be utilized as testing objectives, and testing is essential for the achievement of system testing objectives. The following:

- A test case with a high likelihood of detecting an unknown fault qualifies as a good test case.
- A test that finds an error that has not yet been found is successful.

A test that effectively accomplishes the goals will reveal software problems. Testing also reveals that the software functions appear to function in accordance with the specification and that the performance requirements appear to have been met. There are three ways to test programs.

- For accuracy
- For effective implementation
- Regarding computational complexity

Testing for correctness is meant to ensure that a program performs exactly a site was intended to. This is much harder than it might initially seem, especially for big programs.

5.2 TEST PLAN

A test plan suggests a number of required steps that need be taken in order to complete various testing methodologies. The activity that is to be taken is outlined in the test plan. A computer program, its documentation, and associated data structures are all created by software developers.

It is always the responsibility of the software developers to test each of the program's separate components to make sure it fulfils the purpose for which it was intended. In order to solve the inherent issues with allowing the builder evaluate what they have developed, there is an independent test group (ITG). Testing's precise goals should be laid forth in quantifiable language. So that the mean time to failure, the cost to find and fix the defects, remaining defect density or frequency of occurrence and test work-hours per regression test all should be stated within the test plan.

The levels of testing include:

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

5.2.1 Unit Testing

Unit testing concentrates verification efforts on the software component or module, which is the smallest unit of software design. The component level design description is used as a guide when testing crucial control paths to find faults inside the module's perimeter. the level of test complexity and the untested area determined for unit testing. Unit testing is white-box focused, and numerous components may be tested simultaneously. To guarantee that data enters and exits the software unit under test properly, the modular interface is tested. To make sure that data temporarily stored retains its integrity during each step of an algorithm's execution, the local data structure is inspected. Boundary conditions are tested to ensure that all statements in a module have been executed at least once. Finally, all error handling paths are tested. Before starting any other test, tests of data flow over a module interface are necessary. All other tests are irrelevant if data cannot enter and depart the system properly. An important duty during the unit test is the selective examination of execution pathways. Error circumstances must be foreseen in good design, and error handling paths must be put up to cleanly reroute or halt work when an error does arise. The final step of unit testing is boundary testing. Software frequently fails at its limits.

5.2.2 Integration Testing

Integration testing is a methodical approach for creating the program's structure while also carrying out tests to find interface issues. The goal is to construct a program structure that has been determined by design using unit tested components. The program as a whole is tested. Correction is challenging since the size of the overall program makes it challenging to isolate the causes. As

soon as these mistakes are fixed, new ones arise, and the process repeats itself in an apparently unending cycle. All of the modules were integrated after unit testing was completed in the system to check for any interface inconsistencies. A distinctive program structure also developed when discrepancies in program structures were eliminated.

5.2.3 Validation Testing or System Testing

The testing process comes to an end here. This involved testing the entire system in its entirety, including all forms, code, modules, and class modules. Popular names for this type of testing include system tests and black box testing. The functional requirements of the software are the main emphasis of the black box testing approach. That example, using Black Box testing, a software engineer can create sets of input conditions that will fully test every program requirement. The following sorts of problems are targeted by black box testing: erroneous or missing functions, interface errors, data structure or external data access errors, performance errors, initialization errors, and termination errors.

5.2.4 Output Testing or User Acceptance Testing

The system considered is tested for user acceptance; here it should satisfy the firm's need. The software should keep in touch with perspective system; user at the time of developing and making changes whenever required. This done with respect to the following points:

- Input Screen Designs,
- Output Screen Designs,

The above testing is done taking various kinds of test data. Preparation of test data plays avital role in the system testing. After preparing the test data, the system under study is tested using that test data. While testing the system by which test data errors are again uncovered and corrected by using above testing steps and corrections are also noted.

5.2.5 Automation Testing

Software and other computer goods are tested automatically to make sure they abide by tight guidelines. In essence, it's a test to ensure that the hardware or software performs exactly as intended. It checks for errors, flaws, and any other problems that might occur throughout the creation of the product. Any time of day can be used to do automation testing. It looks at the software using scripted sequences. It then summarizes what was discovered, and this data can be compared to results from earlier test runs.

5.2.6 Selenium Testing

An open-source program called Selenium automates web browsers. It offers a single interface that enables you to create test scripts in several different programming languages, including Ruby, Java, NodeJS, PHP, Perl, Python, and C #. Web application testing for cross browser compatibility is automated using the Selenium testing tool. Whether they are responsive, progressive, or standard, it is utilized to assure high-quality web apps. Selenium is a free software program.

Test Case 1:

Code

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
driver=webdriver.Chrome()
driver.maximize_window()
driver.get("http://127.0.0.1:8000/Guest/Login/")
name_field = driver.find_element('name','Email')
password_field = driver.find_element('name','Password')
name_field.send_keys('aquamotus2024@gmail.com')
password_field.send_keys('aqua123')
time.sleep(3)
driver.find_element("name", "login").click()
if (driver.current_url=="http://127.0.0.1:8000/WebAdmin/Homepage/"):
    print("\n\n\n\nTest Successfull!!!\n\n\n ")
else:
    print("Test failed!!!")
time.sleep(3)
driver.quit()
```

Screenshot

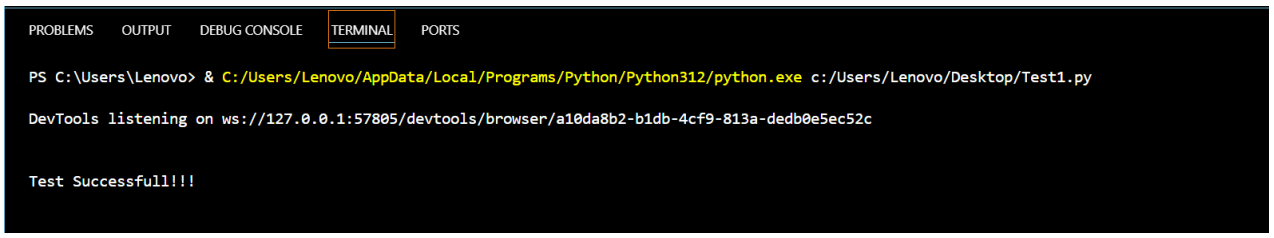

Figure 13: Test Successfully

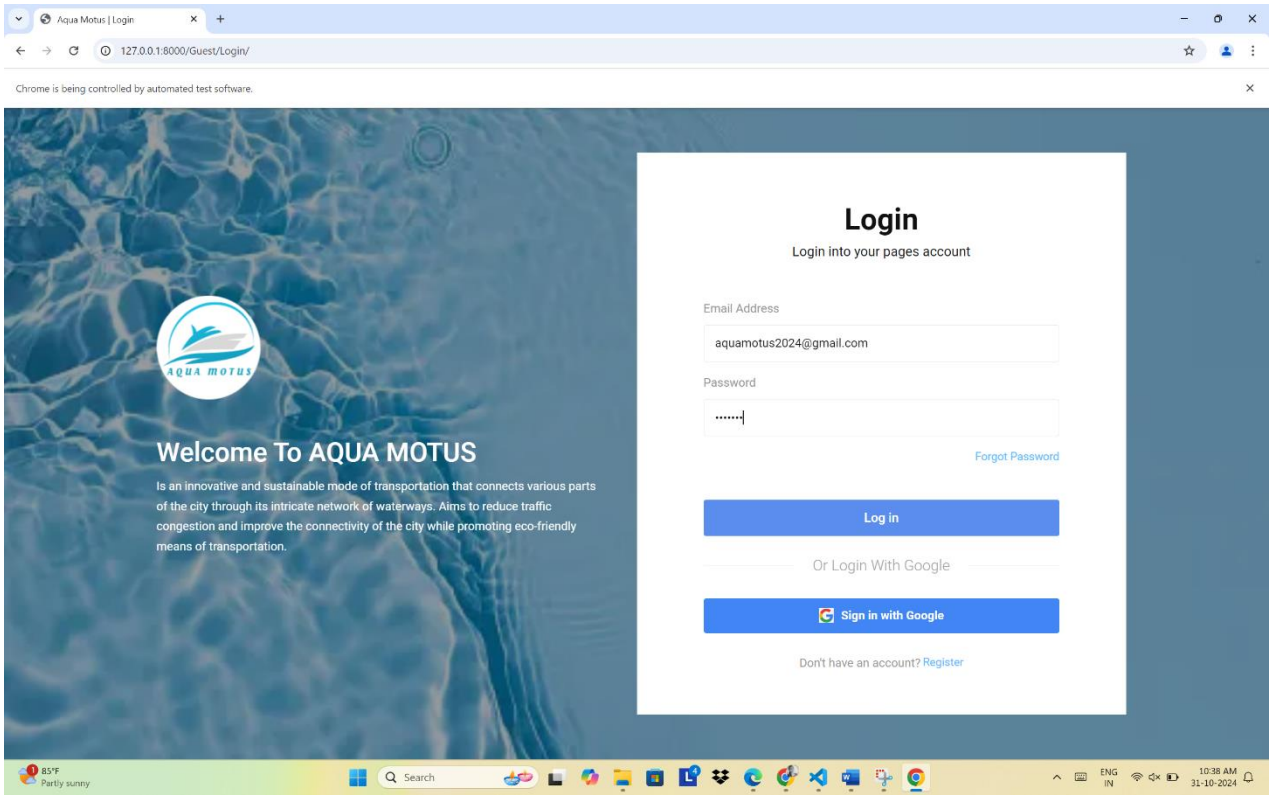

Figure 14: Login page test

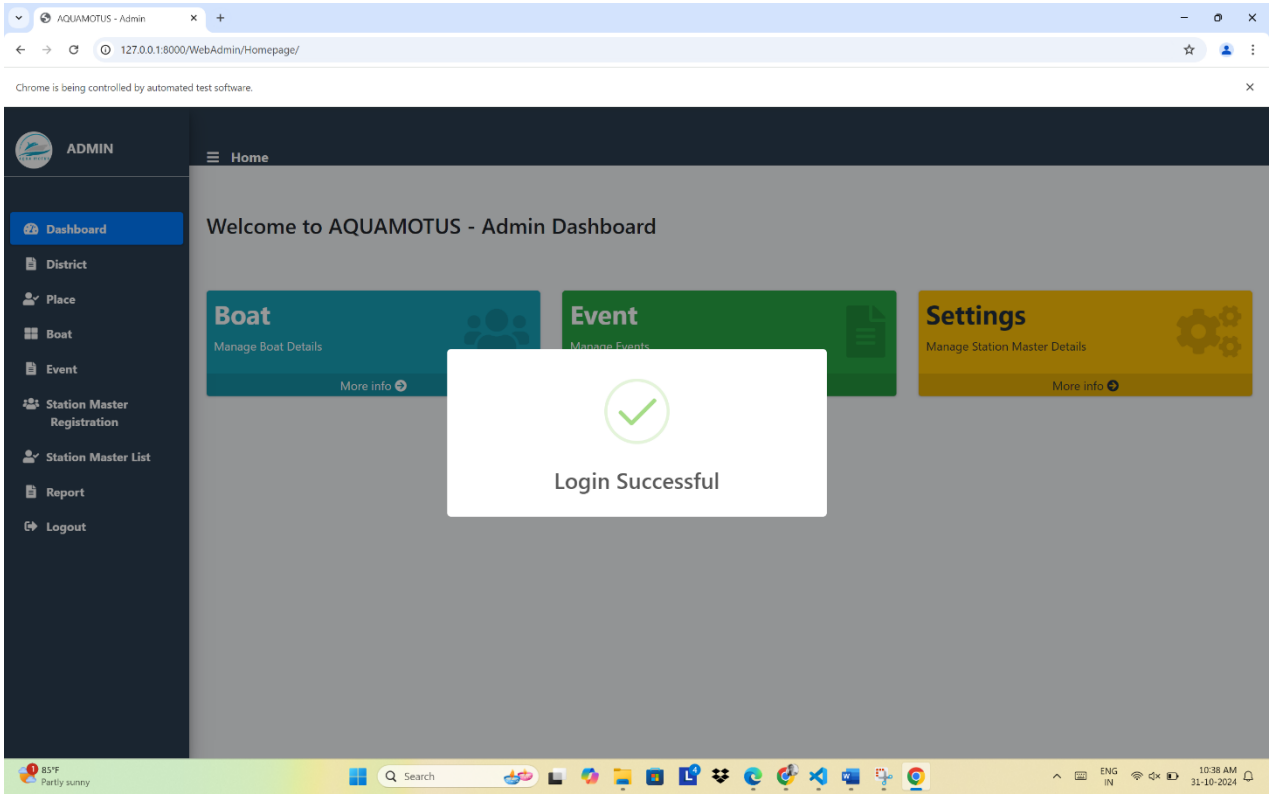

Figure 15: Login admin dashboard

Test Report

Test Case 1

Project Name: AQUA MOTUS - WATERMETRO					
Login Test Case					
Test Case ID: Test_1			**Test Designed By:** Anmigha N M		
Test Priority(Low/Medium/High): High			**Test Designed Date:** 10/10/2024		
Module Name: Login Page			**Test Executed By:** Mr. Jinson Devis		
Test Title: Login Page Test			**Test Execution Date:** 10/10/2024		
Description: Verify Login with valid username and password					
Pre-Condition: Admin has valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/Fail)**
1	Navigate to the login page		Display Login page	Login page displayed	Pass
2	Valid username Entered	aquamotus2024@gmail.com	Admin should enter the main page	Admin entered to the main page	Pass
3	Valid password entered	aqua123			
4	Click on Login Button				
Post-Condition: Username and password is checked with database values for successful login					

Test Case 2:

Code

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
driver=webdriver.Chrome()
driver.maximize_window()
driver.get("http://127.0.0.1:8000/Guest/Login/")
name_field = driver.find_element('name','Email')
password_field = driver.find_element('name','Password')
name_field.send_keys('aquamotus@gmail.com')
password_field.send_keys('aqua12')
time.sleep(3)
driver.find_element("name", "login").click()
if (driver.current_url=="http://127.0.0.1:8000/Guest/Login/"):
    print("\n\n\n\nTest Successfull!!!\n\n\n ")
else:
    print("\n\nTest failed!!!\n\n")
time.sleep(3)
driver.quit()
```

Screenshot

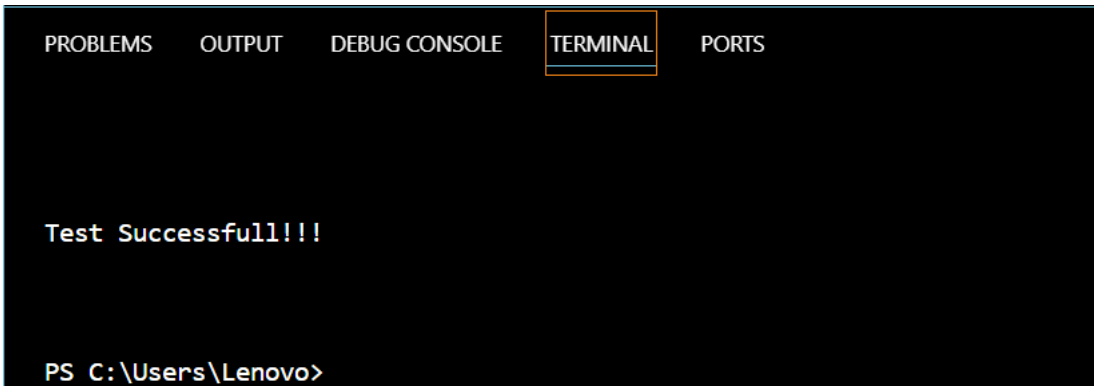

Figure 16: Test Successfully

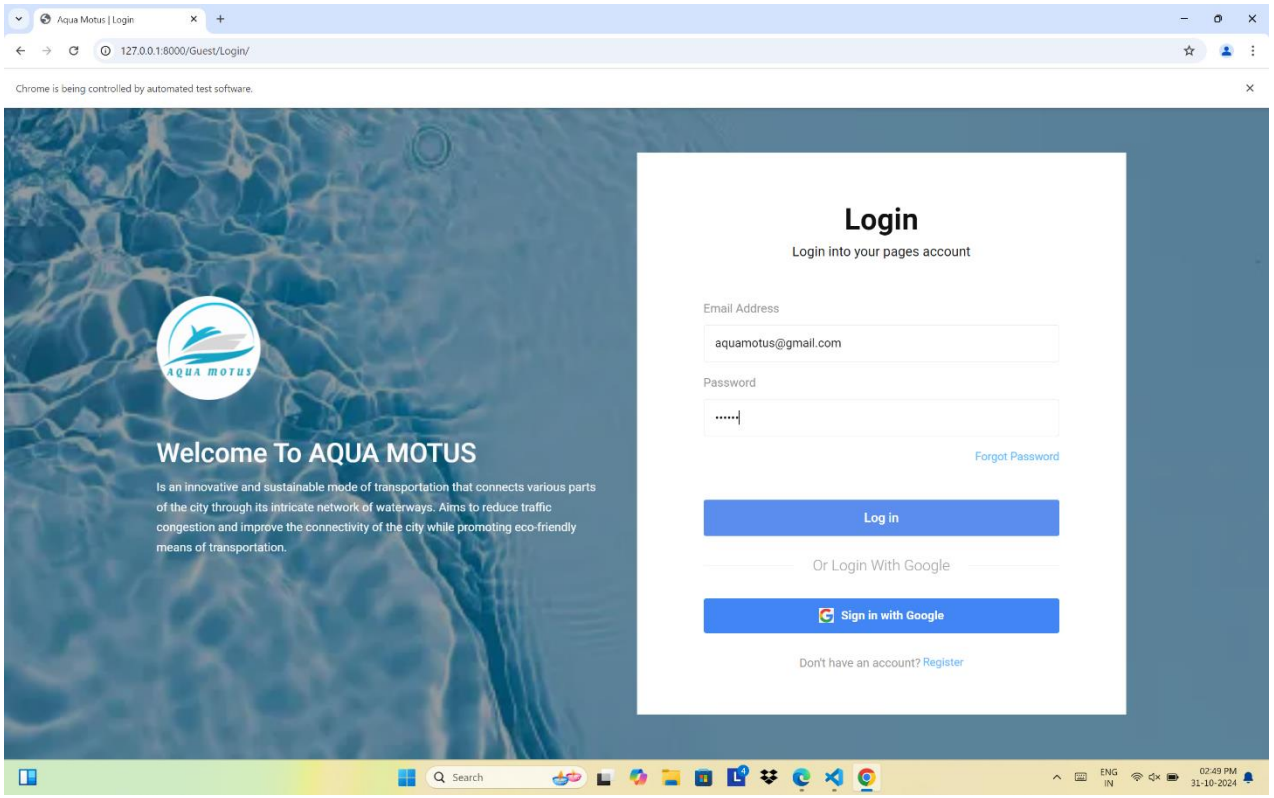

Figure 17: Login Test

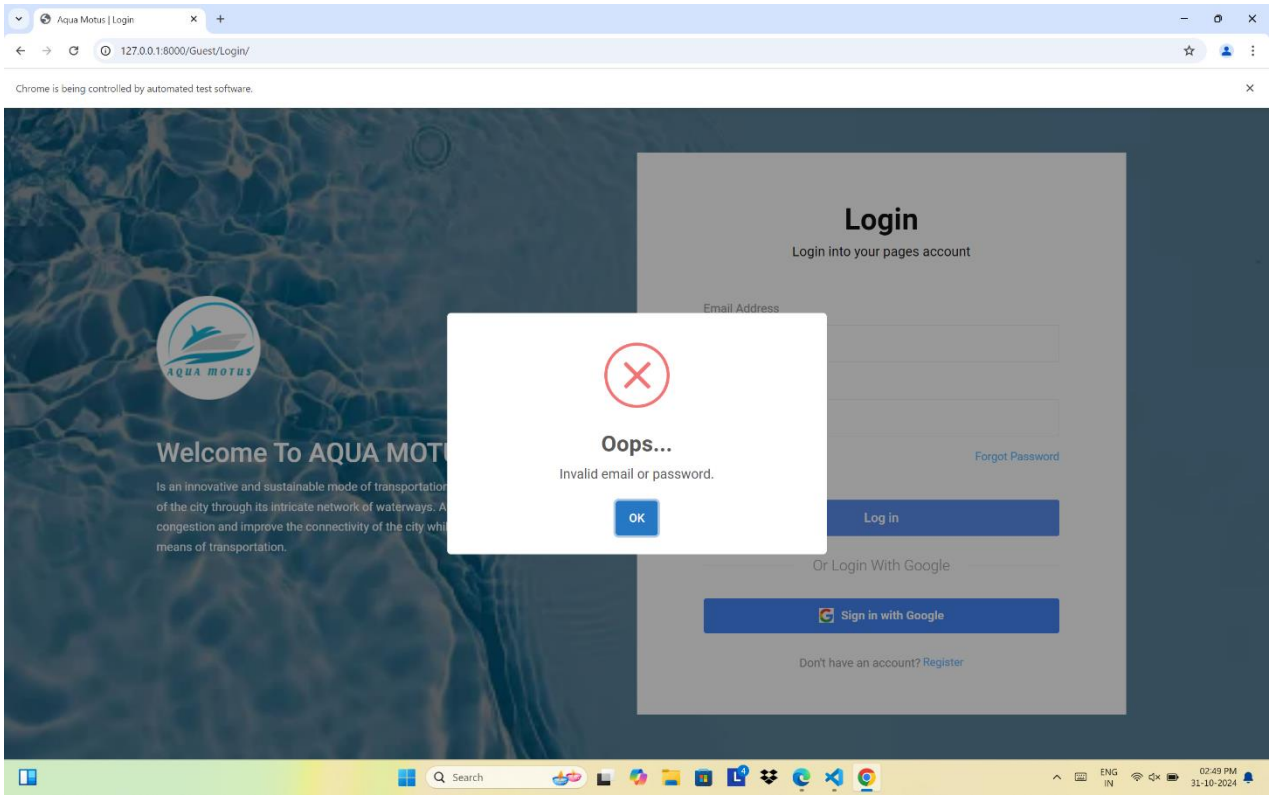

Figure 18: Show alert to Login

Test Report

Test Case 2

Project Name: AQUA MOTUS - WATERMETRO					
Login Test Case					
Test Case ID: Test_2			**Test Designed By:** Anmigha N M		
Test Priority (Low/Medium/High): High			**Test Designed Date:** 10/10/2024		
Module Name: Login Page			**Test Executed By:** Mr. Jinson Devis		
Test Title: Login Page Test			**Test Execution Date:** 10/10/2024		
Description: Verify Login with invalid username and password is not possible					
Pre-Condition: Admin not allowed to login with invalid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/Fail)**
1	Navigate to the login page		Display Login page	Login page displayed	Pass
2	Invalid username Entered	aquamotus@gmail.com	Admin should not enter to main page	Admin stay in login page itself	Pass
3	Invalid password entered	aqua123			
4	Click on Login Button				
Post-Condition: Username and password is not in database, so login is not possible					

Test Case 3:

Code

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException
import time
from selenium import webdriver
from selenium.webdriver.chrome.options import Options

chrome_options = Options()
chrome_options.add_argument("--ignore-certificate-errors")
chrome_options.add_argument("--disable-web-security")
chrome_options.add_argument("--allow-insecure-localhost")

driver = webdriver.Chrome(options=chrome_options)
driver.maximize_window()

driver.get("http://127.0.0.1:8000/Guest/Login/")

name_field = driver.find_element(By.NAME, 'Email')
password_field = driver.find_element(By.NAME, 'Password')
name_field.send_keys('anmigha@gmail.com')
password_field.send_keys('Anmigha@12')
time.sleep(1)
driver.find_element(By.NAME, "login").click()

time.sleep(3)
if driver.current_url == "http://127.0.0.1:8000/User/HomePage/":
    print("\n\nLogin Test Successful!\n\n")
else:
    print("Login Test Failed!")
    driver.quit()
    exit()

driver.get("http://127.0.0.1:8000/User/feedback/")

time.sleep(2)

title_field = driver.find_element(By.ID, 'txt_title')
description_field = driver.find_element(By.ID, 'txt_description')
title_field.send_keys('Great Service')
description_field.send_keys('The Water Metro service is very convenient and well-
managed.')

driver.find_element(By.CSS_SELECTOR, "button[type='submit']").click()
```

```
try:
    alert = WebDriverWait(driver, 5).until(
        EC.visibility_of_element_located((By.CLASS_NAME, "alert-container"))
    )
    if alert:
        print("\n\nFeedback submitted successfully!\n\n")
    else:
        print("Feedback submission failed!")
except TimeoutException:
    print("Feedback submission success message not found!")

time.sleep(2)  # Wait for the page to reload
feedback_items = driver.find_elements(By.CLASS_NAME, "feedback-item")

feedback_found = False
for item in feedback_items:
    if "Great Service" in item.text and "The Water Metro service is very
convenient and well-managed." in item.text:
        feedback_found = True
        print("\n\nFeedback is displayed in the list.\n\n")
        break

if not feedback_found:
    print("Feedback not found in the list.")

driver.quit()
```

Screenshot

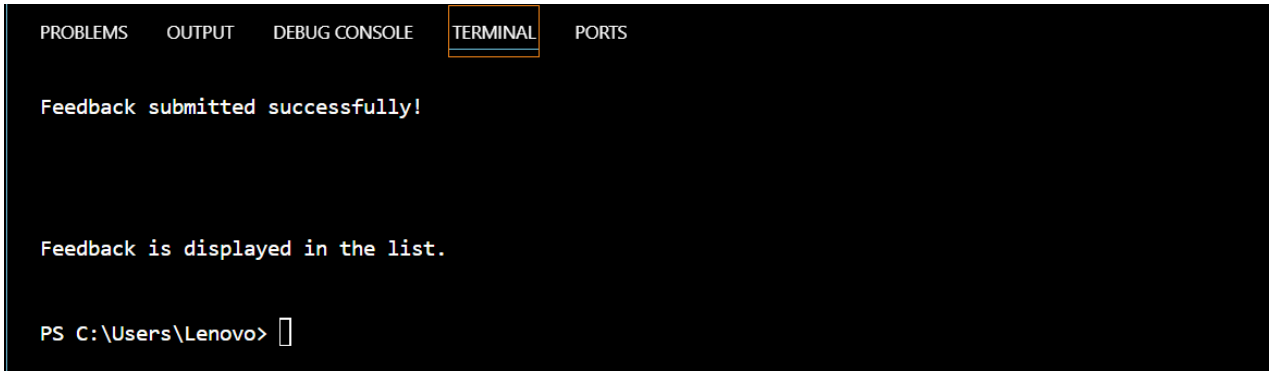

Figure 19: Test Successfully

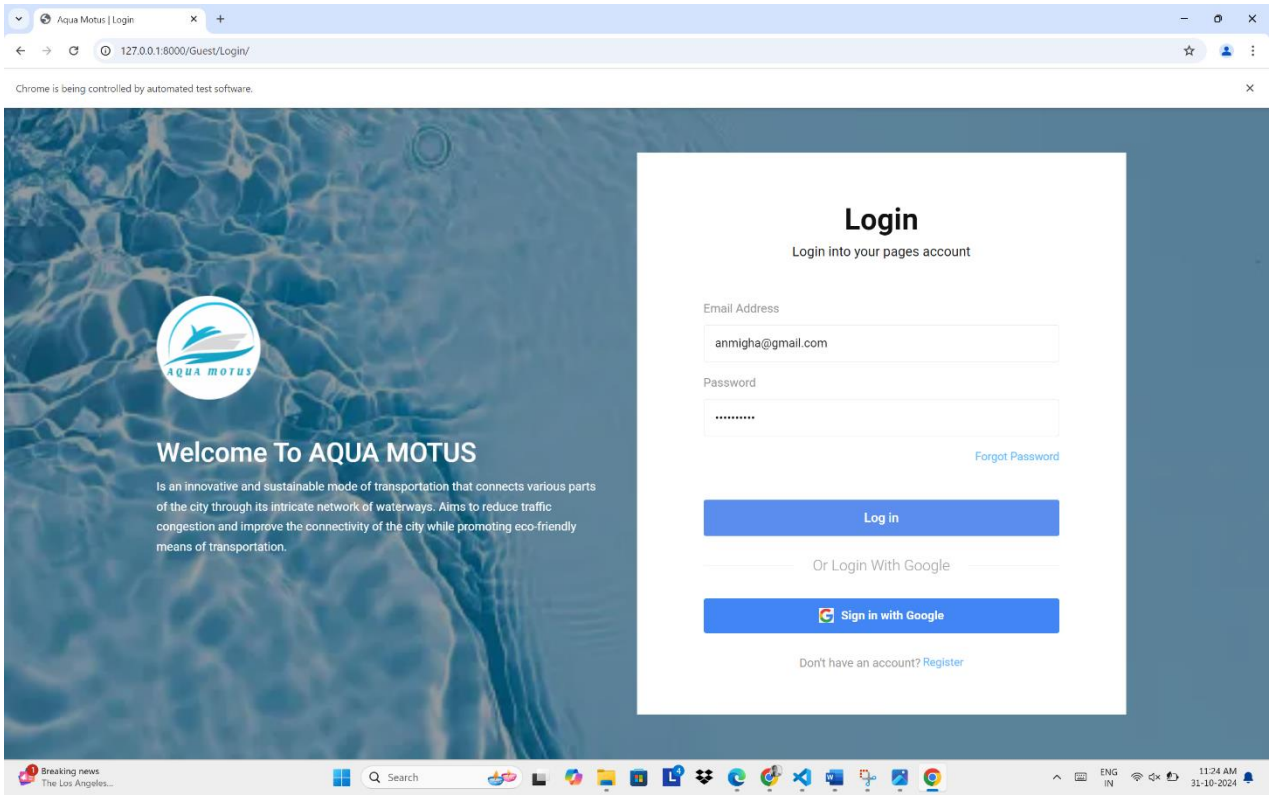

Figure 20: Login Page

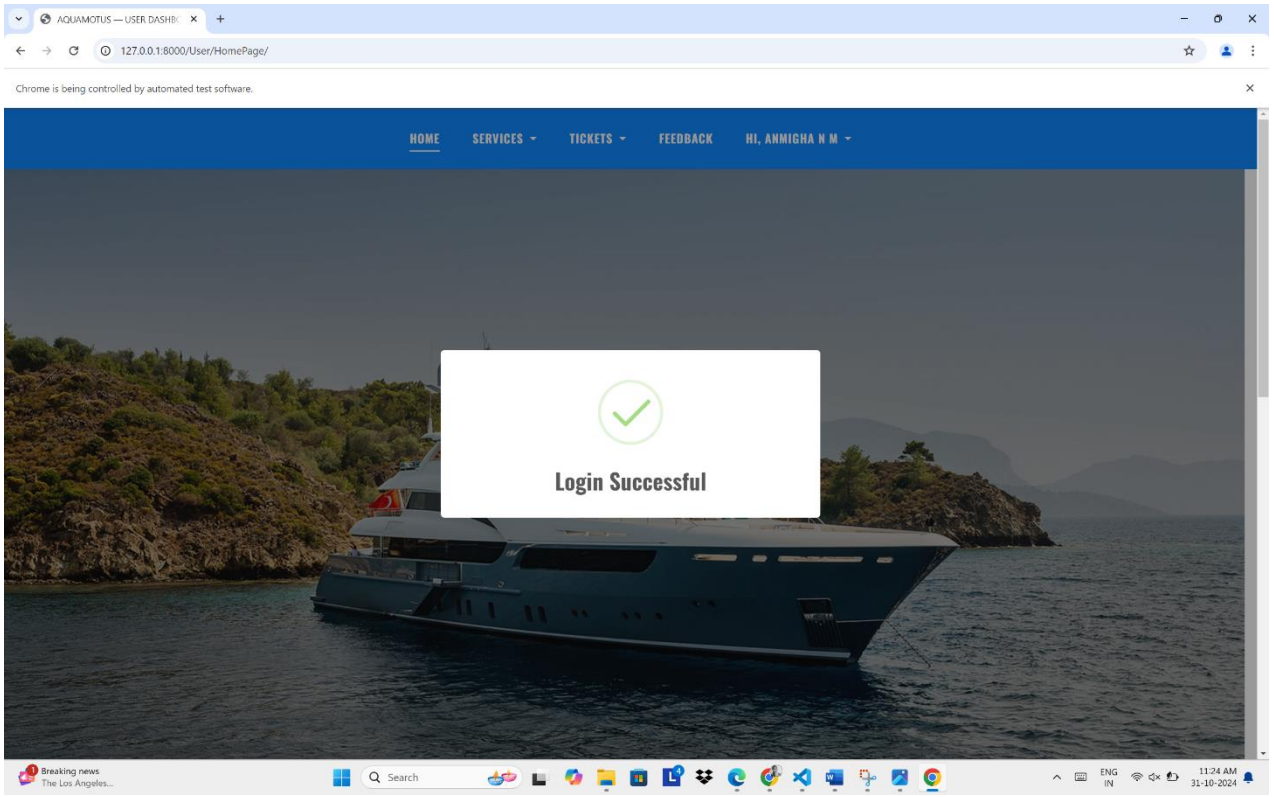

Figure 21: User Dashboard

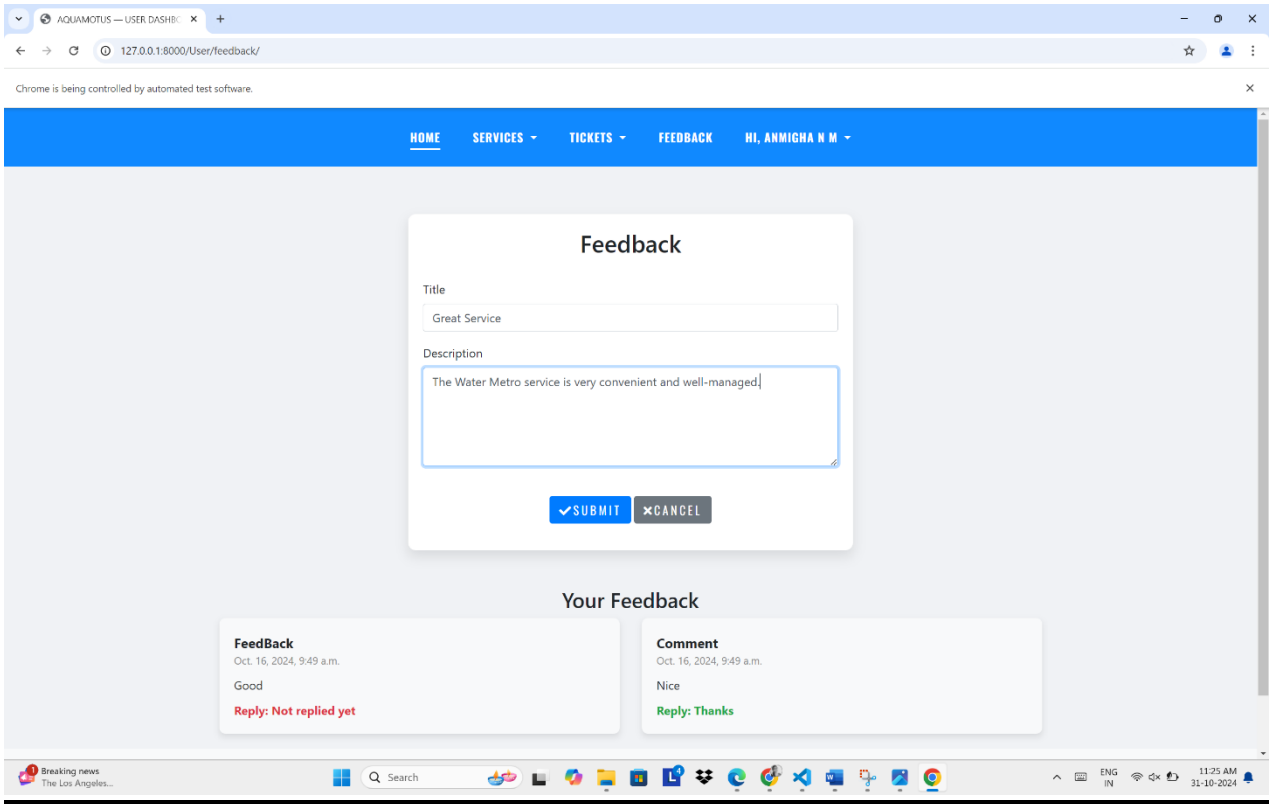

Figure 22: Feedback page Test

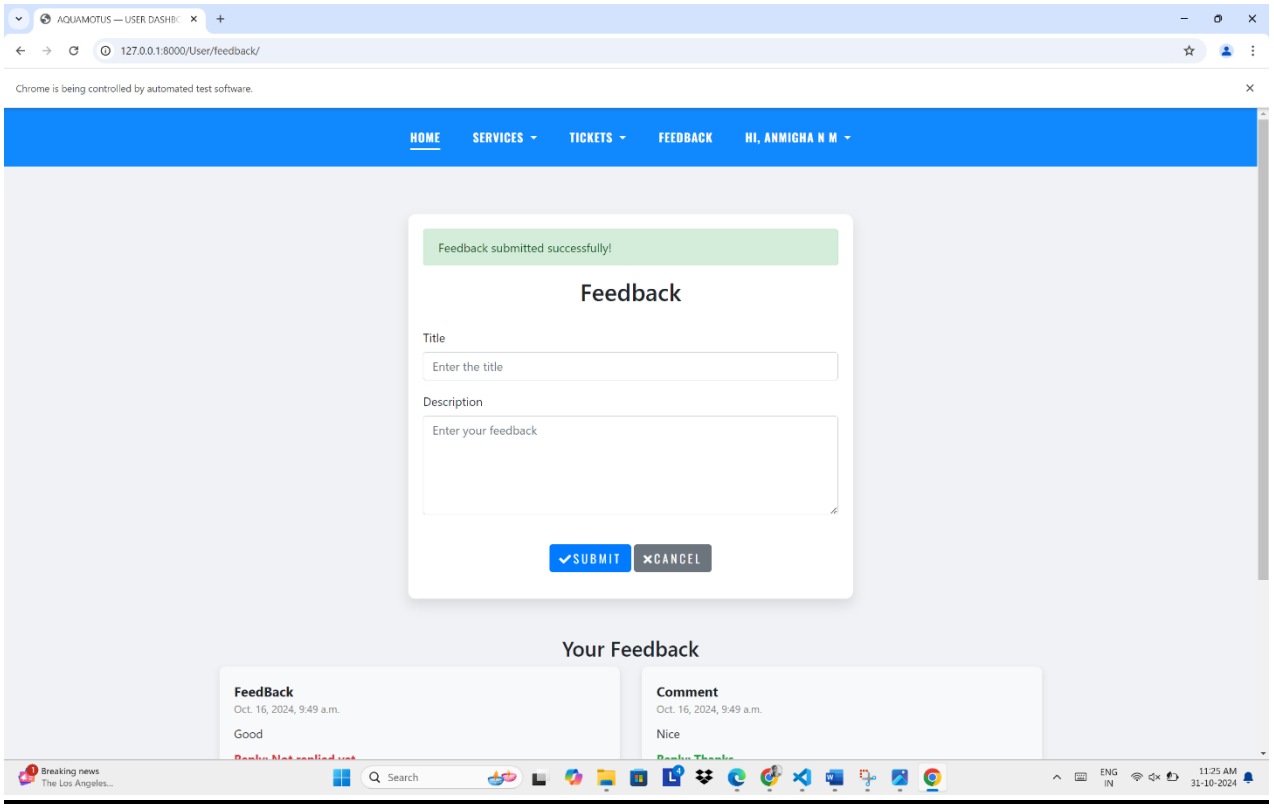

Figure 23: Submitted successfully

Test report

Test Case 3

Project Name: AQUAMOTUS - WATERMETRO					
Feedback Test Case					
Test Case ID: Test_3			**Test Designed By:** Anmigha N M		
Test Priority (Low/Medium/High): High			**Test Designed Date:** 10/10/2024		
Module Name: Feedback			**Test Executed By:** Mr. Jinson Devis		
Test Title: Feedback of the user			**Test Execution Date:** 10/10/2024		
Description: User Feedback page					
Pre-Condition: Login to the user page and enter the feedback. It display in the list					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/Fail)**
1	Navigate to the login page		Display login page	Login page displayed	Pass
2	Valid Username entered	anmigha@gmail.com	User can successfully login and dashboard should be visible	User successfully login and entered into the user dashboard	Pass
3	Valid Password entered	Anmigha@12			
4	Click on the Login Button				
5	Navigate to the feedback page		Feedback page should be visible	Feedback page visible	Pass
6	Valid Title can be enter	Great Service	User should successfully entered feedback and content should be visible	Users enter the feedback successfully and content is visible	Pass
7	Valid Description can be enter	The Water Metro service is very convenient and well-managed.			
9	Click on submit Button				
Post-Condition: Feedback of the user stored and display it in user page					

Test Case 4:

Code

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.support.ui import Select
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.keys import Keys
import time

driver=webdriver.Chrome()
driver.maximize_window() # Replace with your actual path

try:
    driver.get("http://127.0.0.1:8000/Guest/Login/")
    name_field = driver.find_element('name','Email')
    password_field = driver.find_element('name','Password')
    name_field.send_keys('anmigha24@gmail.com')
    password_field.send_keys('Anupama@12')
    time.sleep(3)
    driver.find_element("name", "login").click()
    if (driver.current_url=="http://127.0.0.1:8000/StationMaster/HomePage/"):
        print("\n\n\n\nTest Successfull!!!\n\n\n ")
    else:
        print("Test failed!!!")
    time.sleep(3)

    driver.get("http://127.0.0.1:8000/StationMaster/Services/")
    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.ID, "servicesForm"))
    )
    # Interact with the 'Boat' dropdown
    boat_select = Select(driver.find_element(By.ID, "assignboat_boat"))
    boat_select.select_by_index(1)  # Select the first option after "select.."

    start_point_select = Select(driver.find_element(By.ID, "txt_start"))
    start_point_select.select_by_index(1)  # Select the first valid option

    end_point_select = Select(driver.find_element(By.ID, "txt_end"))
    end_point_select.select_by_index(2)  # Select another valid option for testing

    start_time = driver.find_element(By.NAME, "assignboat_starttime")
    start_time.clear()
    start_time.send_keys("10:00")
    start_time.send_keys(" AM")

    duration_input = driver.find_element(By.NAME, "duration")
```

```
    duration_input.clear()
    duration_input.send_keys("01:30:00")  # Example duration

    rate_input = driver.find_element(By.ID, "txt_rate")
    rate_input.clear()
    rate_input.send_keys("100")

    submit_button = driver.find_element(By.CSS_SELECTOR, "input[type='submit']")
    submit_button.click()

    time.sleep(2)

    try:
        startpoint_error = driver.find_element(By.ID, "startpoint_error").text
        endpoint_error = driver.find_element(By.ID, "endpoint_error").text
        rate_error = driver.find_element(By.ID, "rate_error").text

        assert startpoint_error == "", "Start Point validation failed!"
        assert endpoint_error == "", "End Point validation failed!"
        assert rate_error == "", "Rate validation failed!"

        print("All form validations passed and form submitted successfully.")
    except Exception as e:
        print(f"\n\n\nForm validation errors: {str(e)}\n\n\n")
    # Check Cancel button functionality
    cancel_button = driver.find_element(By.ID, "cancelButton")
    cancel_button.click()

    time.sleep(2)

finally:
    # Close the browser after testing
    driver.quit()
```

Screenshot

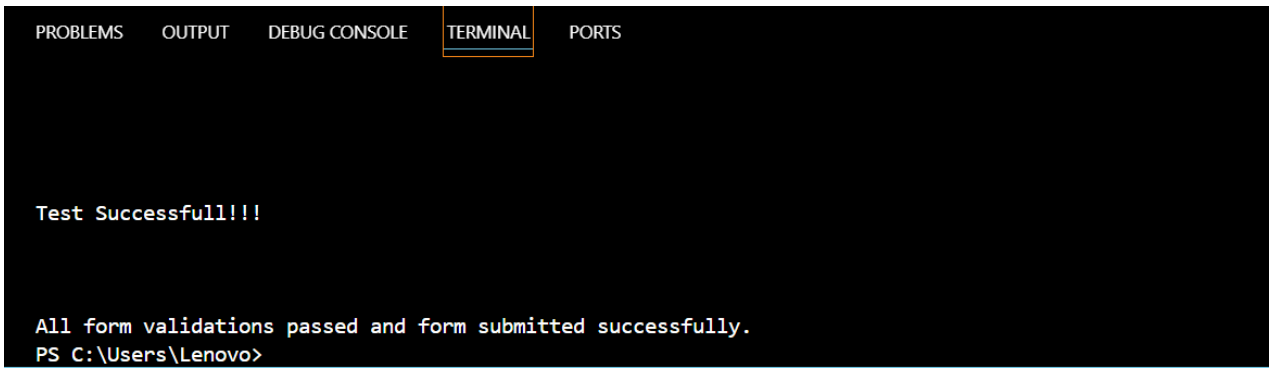

Figure 24: Test Successfully

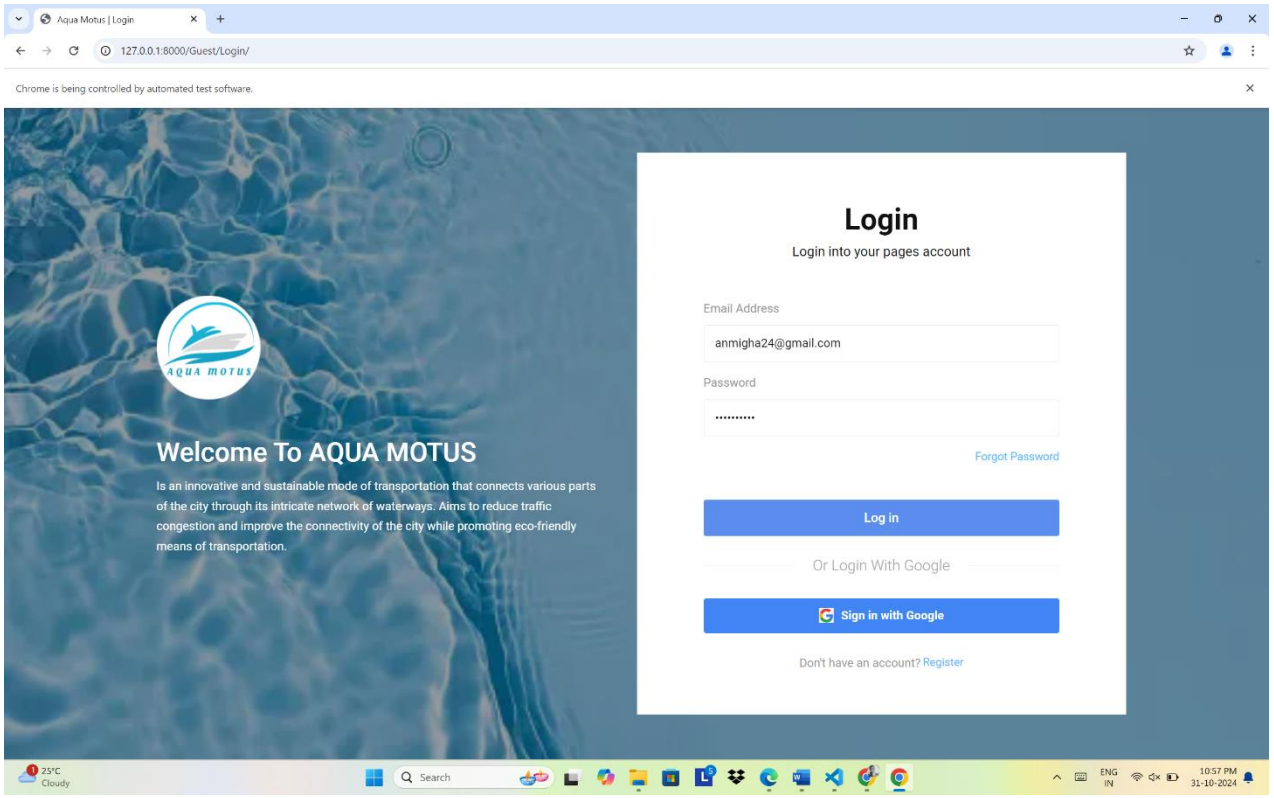

Figure 25: Login page

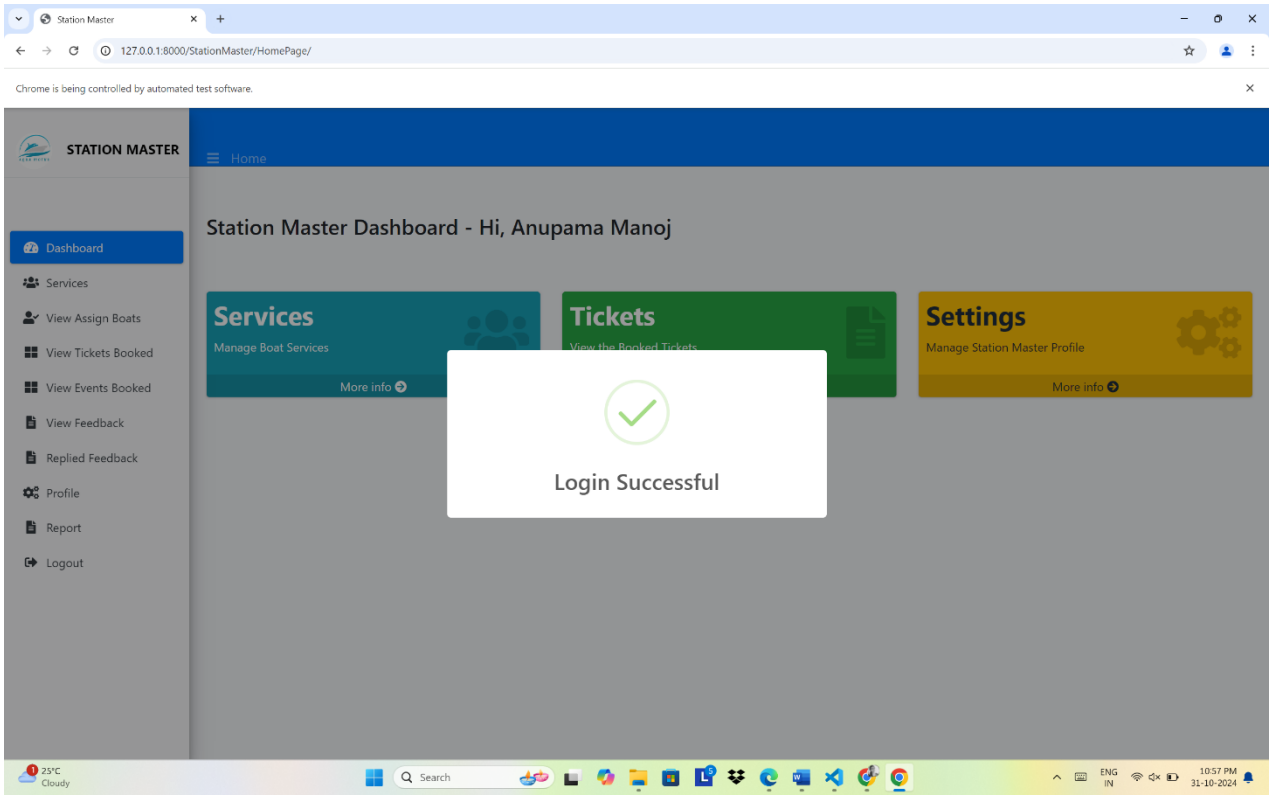

Figure 26: Station Master Dashboard

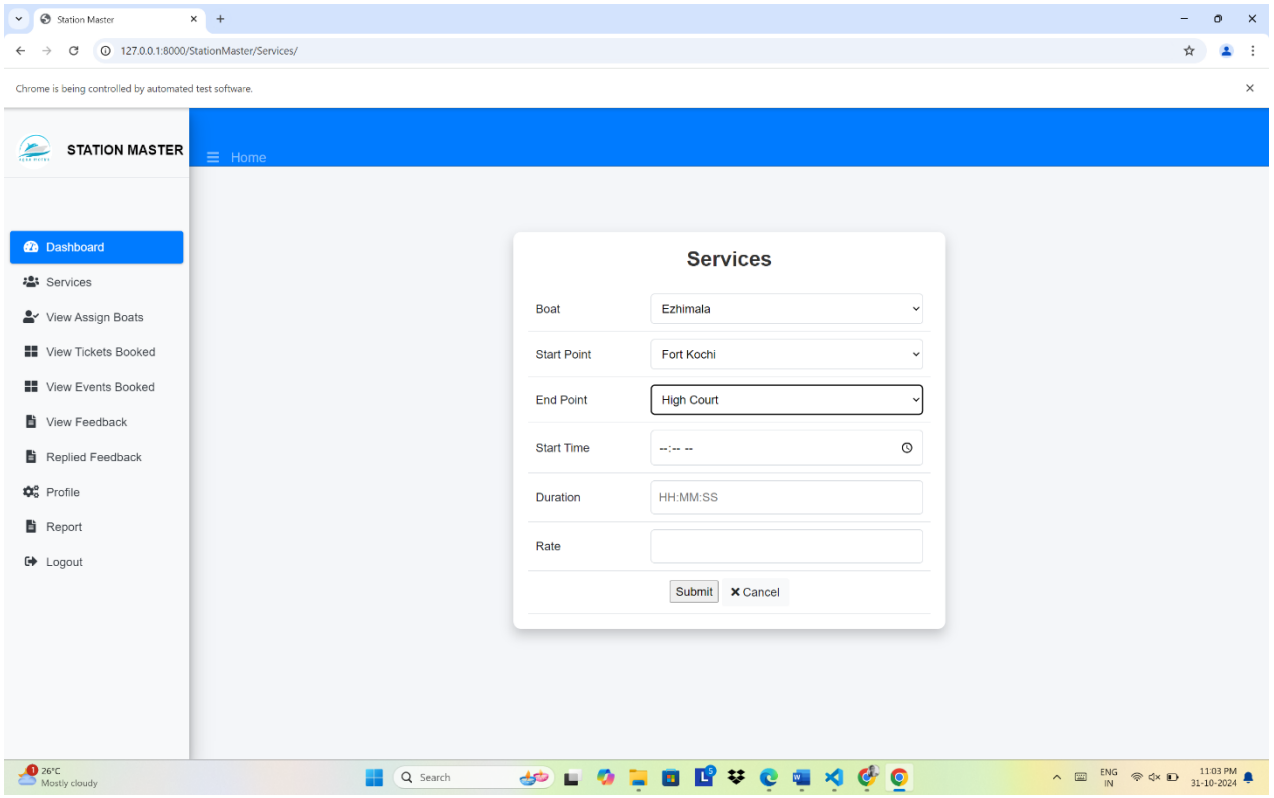

Figure 27: Boat Service page

Test Report

Test Case 4

Project Name: AQUAMOTUS - WATERMETRO					
Service Test Case					
Test Case ID: Test_4			**Test Designed By:** Anmigha N M		
Test Priority (Low/Medium/High): High			**Test Designed Date:** 10/10/2024		
Module Name: Service			**Test Executed By:** Mr. Jinson Devis		
Test Title: Assign Service			**Test Execution Date:** 10/10/2024		
Description: Add service by station master					
Pre-Condition: Login to the station master dashboard and add service					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/Fail)**
1	Navigate to the login page		Display login page	Login page displayed	Pass

2	Valid Username entered	Anmigha24@gmail.com	Station master can successfully login and dashboard should be visible	Station master successfully login and entered into the station master dashboard	Pass
3	Valid Password entered	Anupama@12			
4	Click on the Login Button				
5	Navigate to the service page		Service page should be visible	Service page visible	Pass
6	Valid Boat can be entered	Ezhimala	Station master should successfully enter service	Station master enter the service successfully	Pass
7	Valid Start point selected	Fort Kochi			
8	Valid End point selected	High Court			
9	Valid Start time selected	10 AM			
10	Valid Duration enter	00:30:00			
11	Valid Rate enter	40			
12	Click on submit Button				
Post-Condition: Add services and store it					

CHAPTER 6

IMPLEMENTATION

6.1 INTRODUCTION

The project's implementation phase is where the conceptual design is transformed into a functional system. It can be regarded as the most important stage in creating a successful new system since it gives users assurance that the system will operate as intended and be reliable and accurate. User documentation and training are its main concerns. Usually, conversion happens either during or after the user's training. Implementation is the process of turning a newly revised system design into an operational one, and it simply refers to placing a new system design into operation. The user department now bears most of the workload, faces the most disruption, and has the biggest influence on the current system. If the implementation is not carefully planned or controlled, it can create chaos and confusion. Implementation encompasses all the steps used to switch from the old system to the new one. The new system could be entirely different, take the place of an existing manual or automated system, or it could be modified to work better. A reliable system that satisfies organizational needs must be implemented properly. System implementation refers to the process of actually using the built system. This comprises all the processes involved in switching from the old to the new system. Only after extensive testing and if it is determined that the system is operating in accordance with the standard scan it be put into use. The system personnel check the feasibility of the system. The more complex the system being implemented, the more involved will be the system analysis and design effort required to implement the three main aspects: education and training, system testing and changeover. The implementation state involves the following tasks:

- Careful planning.
- Investigation of system and constraints.
- Design methods to achieve the changeover.

6.2 IMPLEMENTATION PROCEDURES

Software implementation refers to the complete installation of the package in its intended environment, as well as to the system's functionality and satisfaction of its intended applications. The software development project is frequently commissioned by someone who will not be using it. People have early reservations about the software, but we must watch out that they do not become more resistant by making sure that:

- The active user must be aware of the benefits of using the new system.
- Their confidence in the software is built up.
- Proper guidance is imparted to the user so that he is comfortable in using the application.

Before going ahead and viewing the system, the user must know that for viewing the result, the server program should be running in the server. If the server object is not up running on the server, the actual process won't take place.

6.2.1 User Training

The purpose of user training is to get the user ready to test and modify the system. The people who will be involved must have faith in their ability to contribute to the goal and benefits anticipated from the computer-based system. Training is more necessary as systems get more complicated. The user learns how to enter data, handle error warnings, query the database, call up routines that will generate reports, and execute other important tasks through user training.

6.2.2 Training on the Application Software

The user will need to receive the essential basic training on computer awareness after which the new application software will need to be taught to them. This will explain the fundamental principles of how to use the new system, including how the screens work, what kind of help is displayed on them, what kinds of errors are made while entering data, how each entry is validated, and how to change the date that was entered. Then, while imparting the program's training on the application, it should cover the information required by the particular user or group to operate the system or a certain component of the system.

6.2.3 System Maintenance

Software maintenance is the modification of a software product aner delivery to correct faults, to improve performance or other attributes. Maintenance is the ease with which a program can be corrected if any error is encountered, adapted if its environment changes or enhanced if the customer desires a change in requirement. Maintenance follows conversation to extend that changes are necessary to maintain satisfactory operations relative to changes in the user's environment.

Maintenance often includes minor enhancements or corrections to problems that surface in the system's operation. Maintenance is also done based on fixing the problems reported, changing the interface with other software or hardware enhancing the software. Categories of maintenance:

Corrective Maintenance

Corrective maintenance is the most used maintenance approach, but it is easy to see its limitations. When equipment fails, it often leads to downtime in production, and sometimes damages other

parts. In most cases, this is expensive. Also, if the equipment needs to be replaced, the cost of replacing it alone can be substantial. Reliability of systems maintained by this type of maintenance is unknown and cannot be measured. Corrective maintenance is possible since the consequences of failure or wearing out are not significant and the cost of this maintenance is not great

Adaptive Maintenance

Modification of a software product performed after delivery to keep a are product usable m a changed or changing environment. Adaptive maintenance includes any work initiated as a consequence of moving the software to a different hardware or software platform. It is a change driven by the need to accommodate modifications in the environment of software system. The environment in this context refers to the totality of all conditions and influences which act from outside upon the system.

Perfective Maintenance

Modification of a software product alter delivery to improve performance or maintainability. This term is used to describe changes undertaken to expand the existing requirements of the system. A successful piece or software lends to be subjected to a the Succession of changes resulting in an increase in us requirements. This is based an premise that as the software becomes useful, the user experiment with new cases beyond the of Scope for which it was initially developed. Vxpansi01 n requirements can take the form enhancement of existing system functionality and improvement in computational efficiency.

Preventive Maintenance

Preventive maintenance is a schedule of planned maintenance actions aimed at the prevention of breakdowns and failures. The primary goal of preventive maintenance is to prevent the failure of equipment before it actually occurs. It is designed to preserve and enhance equipment reliability by replacing worn components before they actually fail. Preventive maintenance activities include equipment checks, partial or complete overhauls at specified periods

Long-term benefits of preventive maintenance include:

- Improved system reliability.
- Decreased cost of replacement.
- Decreased system downtime.

6.2.4 Hosting

Render is a powerful cloud hosting platform that simplifies deploying and managing web applications by automating infrastructure tasks like scaling, load balancing, and SSL management. Ideal for projects like "AQUA MOTUS," Render offers an easy-to-use interface where you can connect your GitHub repository, set environment variables, and manage configurations, all within a few clicks. With support for static and media file handling and the option to use SQLite for simple data storage, Render ensures reliable performance and security for development environments. Its straightforward configuration, automatic deployment, and scalability make it a practical choice for hosting dynamic applications. However, for production, migrating to a persistent database is recommended due to SQLite's limitations on cloud platforms.

Procedure

Step 1: Sign up/log in to Render and create a new Web Service.
Step 2: Connect your GitHub repo containing the "AQUA MOTUS" project.
Step 3: Set necessary environment variables in Render settings.
Step 4: Update Django to use Render's dynamic PORT.
Step 5: Configure static file handling with Whitenoise.
Step 6: Add build (pip install -r requirements.txt) and start commands (python manage.py runserver 0.0.0.0:$PORT).
Step 7: Deploy and monitor logs for any issues.

Hosted Website : https://render.com/

Hosted Link : https://watermetro-jsgm.onrender.com

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

The proposed system is a very effective plus efficient GUI-based component. This software is well tested; it works properly to meet the user requirements as described in the project. Currently the system is web-based, giving all the required services, information and user functions. Various future enhancements will be made. It reduces manual paper works.

The WaterMetro Ticketing and Facilities Management System represents a significant advancement in the realm of urban transportation, offering a seamless blend of convenience, efficiency, and innovation. By transitioning from traditional manual processes to a digital platform, the system not only simplifies ticket booking but also provides users with access to comprehensive information about WaterMetro services and facilities.

Through the implementation of online ticket booking, QR code ticketing, and a user-friendly information portal, the system enhances the overall user experience, making it easier and more convenient for passengers to navigate the WaterMetro network. Furthermore, the integration of advanced technologies such as IoT, machine learning, and AR opens new possibilities for future enhancements, promising even greater levels of personalization, accessibility, and efficiency.

As urban populations continue to grow and cities face increasing challenges related to transportation and sustainability, the WaterMetro Ticketing and Facilities Management System stands poised to evolve and adapt, addressing the evolving needs of users and stakeholders alike. By embracing innovation and leveraging data-driven insights, the system has the potential to not only revolutionize water-based transportation but also contribute to the broader goals of urban mobility and environmental stewardship.

In conclusion, the WaterMetro Ticketing and Facilities Management System represents a transformative step forward in urban transportation, laying the groundwork for a more connected, accessible, and sustainable future. Through ongoing collaboration, innovation, and engagement with users and stakeholders, the system is poised to continue evolving, delivering value to communities and shaping the future of urban mobility.

7.2 FUTURE SCOPE

- **Integration with Smart Technologies:**

  Integrate the system with IoT (Internet of Things) devices to provide real-time updates on water levels, weather conditions, and vessel statuses.

  Implement smart ticketing solutions such as contactless payments, NFC (Near Field Communication) technology, or biometric authentication for faster and more secure transactions.

- **Personalized Recommendations:**

  Utilize machine learning algorithms to analyze user preferences and behavior, offering personalized recommendations for routes, facilities, and activities.

  Implement loyalty programs or rewards systems to incentivize frequent users and enhance customer retention.

- **Accessibility Features:**

  Enhance accessibility features within the application to cater to users with disabilities, including support for screen readers, voice commands, and alternative input methods.

  Provide multilingual support to accommodate users from diverse linguistic backgrounds, improving inclusivity and usability.

- **Augmented Reality (AR) Integration:**

  Introduce AR features within the application to provide users with immersive experiences, such as virtual tours of WaterMetro stations, interactive maps, and augmented information overlays.

- **Enhanced Facilities Management:**

  Implement predictive maintenance algorithms to proactively identify and address potential issues with infrastructure, vessels, or facilities, minimizing downtime and disruptions.

  Introduce smart facility management solutions, such as energy-efficient lighting systems, automated waste management, and real-time monitoring of environmental parameters.

- **Expanded Service Offerings:**

  Introduce additional services and partnerships, such as onboard entertainment, food and beverage options, guided tours, or ticket bundles for local attractions.

  Collaborate with other transportation networks (e.g., buses, trains) to offer seamless multimodal journeys and integrated ticketing solutions.

CHAPTER 8

BIBLIOGRAPHY

REFERENCES:

1. Gupta, A., Dwivedi, D., & Singh, J. (2022). Sentiment analysis of travelling passengers using machine learning. *International Journal of Advance Research and Innovative Ideas in Education, 8*(3), 16776-1685. ISSN(O)-2395-4396. Available at: www.ijariie.com
2. Idris, S. L., & Mohamad, M. (2022). A study on sentiment analysis on airline quality services: A conceptual paper. *International Journal of Advanced Research in Computer Science and Software Engineering, 12*(6), 345-355.
3. Kumar, G. L., Reddy, I. K., & Koduru, S. (2021). *Sentiment Analysis in Transportation System*. International Journal of Creative Research Thoughts (IJCRT), 9(11), IJCRT2111075. ISSN: 2320-2882. Retrieved from www.ijcrt.org.
4. Li, Z., Yang, C., & Huang, C. (2024). A Comparative Sentiment Analysis of Airline Customer Reviews Using Bidirectional Encoder Representations from Transformers (BERT) and Its Variants. *Mathematics, 12*(1), 53. https://doi.org/10.3390/math12010053
5. Raihen, M. N., & Akter, S. (2024). Sentiment analysis of passenger feedback on U.S. airlines using machine learning classification methods. *World Journal of Advanced Research and Reviews, 23*(01), 2260–2273. https://doi.org/10.30574/wjarr.2024.23.1.2183
6. Farzadnia, S., & Vanani, I. R. (2022). Identification of opinion trends using sentiment analysis of airlines passengers' reviews. *Journal of Air Transport Management, 102*, 102232. https://doi.org/10.1016/j.jairtraman.2022.102232
7. Redhu, S., Srivastava, S., Bansal, B., & Gupta, G. (2018). Sentiment Analysis Using Text Mining: A Review. *International Journal on Data Science and Technology, 4*(2), 49-53. https://doi.org/10.11648/j.ijdst.20180402.12
8. Nandwani, P., & Verma, R. (2021). A review on sentiment analysis and emotion detection from text. *Springer-Verlag GmbH Austria, part of Springer Nature*. https://doi.org/10.1007/s00542-021-06384-z
9. Sultana, N., Kumar, P., Patra, M. R., Chandra, S., & Alam, S. K. (2024). Sentiment analysis for product review. *Department of Computer Science and Engineering, Calcutta Institute of Technology, India*.
10. Drus, Z., & Khalid, H. (2019). Sentiment analysis in social media and its application: Systematic literature review. In *The Fifth Information Systems International Conference 2019*. Azman Hashim International Business School, Kuala Lumpur, Malaysia. Available at https://doi.org/10.1016/j.jairtraman.2022.102232

11. Wankhade, M., Rao, A. C. S., & Kulkarni, C. (2022). A survey on sentiment analysis methods, applications, and challenges. Artificial Intelligence Review, 55, 5731–5780. https://doi.org/10.1007/s10462-021-09983-0
12. Grana, P. A. (2022). Sentiment analysis of text using machine learning models. *International Research Journal of Modernization in Engineering Technology and Science*, 4(5), 2952. e-ISSN: 2582-5208. Retrieved from www.irjmets.com
13. Aqlan, A. A. Q., Manjula, B., & Naik, R. L. (2019). A study of sentiment analysis: Concepts, techniques, and challenges. In *Proceedings of International Conference on Computational Intelligence and Data Engineering* (Vol. 28, pp. 147-156). Springer Nature Singapore Pte Ltd. https://doi.org/10.1007/978-981-13-6459-4_16
14. Ashique, M., Kumar, S., Vij, A., & Panwar, S. (2021). Sentiment analysis using machine learning approaches of Twitter data and semantic analysis. *Turkish Journal of Computer and Mathematics Education*, 12(6), 5181-5192.
15. Elias M Award. “SYSTEM ANALYSIS AND DESIGN”
16. R Elmarsi and B Navathe. “FUNDAMENTALS OF DATABASE SYSTEM”
17. John Zelle. “PYTHON PROGRAMMING AN INTRODUCTION TO COMPUTER SCIENCE”
18. P. Thakur and P. Jadon, "Django: Developing web using Python," 2023 3rd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), Greater Noida, India, 2023, pp. 303-306, doi: 10.1109/ICACITE57410.2023.10183246.
19. Sundaramoorthy, S. (2022). UML Diagramming: A Case Study Approach (1st ed.). Auerbach Publications. https://doi.org/10.1201/9781003287124
20. Eran Kinsbruner; Gleb Bahmutov, A Frontend Web Developer’s Guide to Testing: Explore leading web test automation frameworks and their future driven by low-code and AI , Packt Publishing, 2022.

WEBSITES:

- http://chat.openai.com/
- http://tutorialspoint.com/
- http://github.com/
- http://www.w3schools.com/

CHAPTER 9

APPENDIX

9.1 SAMPLE CODE

StationMaster\views.py

```
from django.shortcuts import render,redirect,get_object_or_404
from StationMaster.models import *
from WebAdmin.models import *
from User.models import *
from django.urls import reverse
from datetime import timedelta
import re
from django.http import HttpResponseBadRequest
import razorpay
from django.conf import settings
from django.http import JsonResponse
from django.views.decorators.cache import never_cache
from django.contrib.auth.decorators import login_required

# Create your views here.

@never_cache
@login_required
def homepage(request):
    master = tbl_stationmaster.objects.get(user=request.user)  # Adjust based on
your model's structure
    return render(request, 'StationMaster/MasterHome.html', {'masterdata':
master})

@never_cache
@login_required
def myprofile(request):
    user1 = request.user
    # Assuming the user exists in tbl_stationmaster
    master = tbl_stationmaster.objects.get(user=user1)
    return
render(request,'StationMaster/MyProfile.html',{'masterdata':master,'master':user1}
)

@never_cache
@login_required
def editprofile(request):
    user1 = request.user
    user = tbl_stationmaster.objects.get(user=user1)
    if request.method == "POST":
        user1.first_name = request.POST.get("txt_name")
        user1.last_name = request.POST.get("txt_lastname")
        user1.email = request.POST.get("txt_email")
        user1.username = request.POST.get("txt_email")
        user.master_contact = request.POST.get("txt_contact")
```

```
        user.master_address = request.POST.get("txt_address")
        user1.save()
        user.save()
        request.session['profile_updated'] = True
        return redirect('WebStationMaster:MyProfile')
    else:
        profile_updated = request.session.pop('profile_updated', False)
        return render(request, 'StationMaster/EditProfile.html', {'master': user,
'masterdata': user1, 'profile_updated': profile_updated})

@never_cache
@login_required
def changepassword(request):
    user1 = request.user
    master = tbl_stationmaster.objects.get(user=user1)

    if request.method == "POST":
        currentpass = request.POST.get("txt_currentpassword")

        # Use check_password to verify the current password
        if user1.check_password(currentpass):  # Updated line
            newpass = request.POST.get("txt_newpassword")
            conpass = request.POST.get("txt_confirmpassword")

            if newpass == conpass:
                user1.set_password(newpass)  # Use set_password to hash the new
password
                user1.save()
                msg = "Password changed successfully."
                return render(request, 'Guest/Login.html')
            else:
                msg = "New password and confirm password do not match."
                return render(request, 'StationMaster/ChangePassword.html',
{'msg': msg, 'masterdata':master,'master':user1})
        else:
            msg = "Current password is incorrect."
            return render(request, 'StationMaster/ChangePassword.html', {'msg':
msg, 'masterdata':master,'master':user1})
    else:
        return render(request, 'StationMaster/ChangePassword.html',
{'masterdata':master,'master':user1})

@never_cache
@login_required
def services(request):
    places = tbl_place.objects.filter(status=1)
    boats = tbl_boat.objects.filter(status=1,boat_service = 'Public Transport Boat
Services')
    services = tbl_services.objects.all()
```

```
    error_message = ""

    if request.method == "POST":
        startpoint_id = request.POST.get("services_startpoint")
        endpoint_id = request.POST.get("services_endpoint")
        rate = request.POST.get("rate")
        duration = request.POST.get("duration")
        starttime = request.POST.get("assignboat_starttime")
        boat_id = request.POST.get("assignboat_boat")

        if not (startpoint_id and endpoint_id and rate and duration and starttime
and boat_id):
            error_message = "Please fill all fields."
        else:
            try:
                rate = float(rate)
                startpoint = tbl_place.objects.get(id=startpoint_id)
                endpoint = tbl_place.objects.get(id=endpoint_id)
                boat = tbl_boat.objects.get(id=boat_id)
                duration_parts = duration.split(':')
                duration_seconds = int(duration_parts[0]) * 3600 +
int(duration_parts[1]) * 60 + int(duration_parts[2])
                duration = timedelta(seconds=duration_seconds)
                tbl_services.objects.create(
                    assignboat_boat=boat,
                    services_startpoint=startpoint,
                    services_endpoint=endpoint,
                    rate=rate,
                    duration=duration,
                    assignboat_starttime=starttime
                )
                return redirect('WebStationMaster:Services')
            except ValueError:
                error_message = "Please enter a valid rate."
            except Exception as e:
                error_message = str(e)

    return render(request, 'StationMaster/Services.html', {
        'places': places,
        'boats': boats,
        'services': services,
        'error_message': error_message
    })


def parse_duration(duration_str):
    """Convert a string in 'HH:MM:SS' format to a timedelta object."""
    if not duration_str:
        return None
```

```
    try:
        parts = re.split('[:]', duration_str)
        if len(parts) != 3:
            raise ValueError("Duration must be in 'HH:MM:SS' format.")
        hours, minutes, seconds = map(int, parts)
        return timedelta(hours=hours, minutes=minutes, seconds=seconds)
    except ValueError:
        raise ValueError("Invalid duration format. Use 'HH:MM:SS'.")

@never_cache
@login_required
def update_service(request, did):
    places = tbl_place.objects.filter(status=1)
    boats = tbl_boat.objects.filter(status=1)
    updata = get_object_or_404(tbl_services, id=did)
    error_message = ""

    if request.method == "POST":
        startpoint_id = request.POST.get("services_startpoint")
        endpoint_id = request.POST.get("services_endpoint")
        rate = request.POST.get("rate")
        duration_str = request.POST.get("duration")
        starttime = request.POST.get("assignboat_starttime")
        boat_id = request.POST.get("assignboat_boat")

        if not (startpoint_id and endpoint_id and rate and duration_str and
starttime and boat_id):
            error_message = "Please fill all fields."
        else:
            try:
                rate = float(rate)
                duration = parse_duration(duration_str)
                updata.services_startpoint =
tbl_place.objects.get(id=startpoint_id)
                updata.services_endpoint = tbl_place.objects.get(id=endpoint_id)
                updata.rate = rate
                updata.duration = duration
                updata.assignboat_starttime = starttime
                updata.assignboat_boat = tbl_boat.objects.get(id=boat_id)
                updata.save()
                return redirect('WebStationMaster:View_Services')
            except ValueError as e:
                error_message = str(e)
            except tbl_place.DoesNotExist:
                error_message = "Invalid start or end point."
            except tbl_boat.DoesNotExist:
                error_message = "Invalid boat."

    return render(request, 'StationMaster/Services.html', {
```

```
        'places': places,
        'boats': boats,
        'udata': updata,
        'error_message': error_message
    })


def toggle_service_status(request,did):
    service = get_object_or_404(tbl_services, id=did)
    service.status = 1 if service.status == 0 else 0
    service.save()
    return redirect(reverse('WebStationMaster:View_Services'))

@never_cache
@login_required
def view_services(request):
    services = tbl_services.objects.all()
    return render(request, 'StationMaster/AssignBoatsView.html', {
        'services': services
    })

@never_cache
@login_required
def viewticketbooking(request):
    bookings = tbl_ticketbooking.objects.all()
    for booking in bookings:
        if booking.refund_amount:
            booking.refund_amount_display = booking.refund_amount / 100
    return render(request, "StationMaster/ViewTicketBooking.html", {"booking":
bookings})

@never_cache
@login_required
def vieweventbooking(request):
    bookings = tbl_eventbooking.objects.all()
    for booking in bookings:
        if booking.refund_amount:
            booking.refund_amount_display = booking.refund_amount / 100
    return render(request, "StationMaster/ViewEventBooking.html", {"booking":
bookings})

def assign_boat(request):
    if request.method == 'POST':
        booking_id = request.POST.get('booking_id')
        boat_id = request.POST.get('boat_id')

        booking = get_object_or_404(tbl_eventbooking, id=booking_id)
        boat = get_object_or_404(tbl_boat, id=boat_id)
```

```
        # Assign the selected boat to the booking
        booking.assign = boat
        booking.status = 2  # Change status to assigned
        booking.save()

        return redirect('WebStationMaster:vieweventbooking')

def get_active_boats(request):
    boat_deck = request.GET.get('boat_deck')
    # Filter boats that are active and in the services 'Tourism' or 'Recreational
Boat Services'
    boats = tbl_boat.objects.filter(status=1,boat_deck = boat_deck,
boat_service__in=['Tourism and Recreational Boat Services'])
    boat_list = [{'id': boat.id, 'boat_name': boat.boat_name , 'boat_capacity':
boat.boat_capacity} for boat in boats]
    return JsonResponse({'boats': boat_list})


@never_cache
@login_required
def viewcomplaints(request):
    complaintdata = Feedback.objects.filter(status=0)
    return render(request, 'StationMaster/ViewComplaint.html', {'Data':
complaintdata})

@never_cache
@login_required
def repliedcomplaints(request):
    replydata = Feedback.objects.filter(status=1)
    return render(request, 'StationMaster/RepliedComplaints.html', {'ReplyData':
replydata})

@never_cache
@login_required
def reply(request, did):
    data = Feedback.objects.get(id=did)
    if request.method == "POST":
        reply = request.POST.get('txt_reply')
        data.reply = reply
        data.status = 1
        data.save()
        return redirect('WebStationMaster:ViewComplaint')
    else:
        return render(request, 'StationMaster/reply.html', {'data': data})

# Razorpay client setup
razorpay_client = razorpay.Client(auth=(settings.RAZORPAY_KEY_ID,
settings.RAZORPAY_KEY_SECRET))
```

```
def initiate_refund(request):
    # Fetch the booking object using booking_id
    booking_id = request.POST.get('booking_id')

    if not booking_id:
        return HttpResponseBadRequest("Booking ID is missing")

    # Fetch the booking object using booking_id
    booking = get_object_or_404(tbl_eventbooking, id=booking_id)

    if booking.status != 3:
        return HttpResponseBadRequest("Refund is not allowed for this booking
status")

    event_type_rate = booking.event_type.event_rate

    # Ensure event_type_rate is numeric and calculate 80% refund amount
    try:
        event_type_rate = float(event_type_rate)
        refund_amount = int(event_type_rate * 80 / 100 * 100)  # Convert to paise
    except (ValueError, TypeError):
        return HttpResponseBadRequest("Invalid rate value")

    booking.refund_amount = refund_amount
    booking.save()

    if request.method == "POST":
        # Process the refund via Razorpay
        razorpay_order = razorpay_client.order.create({
            "amount": refund_amount,
            "currency": "INR",
            "payment_capture": "1"
        })

        request.session['booking_id'] = booking_id
        request.session['razorpay_order_id'] = razorpay_order['id']

        # Render the payment page for refund
        context = {
            "razorpay_order_id": razorpay_order['id'],
            "razorpay_key": settings.RAZORPAY_KEY_ID,
            "amount": refund_amount,
            "booking_number": booking.event_number,
        }
        return render(request, 'StationMaster/payment_process.html', context)
    else:
        return HttpResponseBadRequest("Invalid request method")

def payment_callback(request):
```

```
    if request.method == "POST":
        # Fetch the booking ID stored in session during payment process
        booking_id = request.session.get('booking_id')
        if not booking_id:
            return HttpResponseBadRequest("Booking ID not found in session")

        # Fetch the booking object using booking_id
        booking = get_object_or_404(tbl_eventbooking, id=booking_id)

        # Update the booking record and set status to 4 (refunded)
        booking.status = 4  # Assuming 4 indicates refund completed
        booking.save()

        # Clear the session data related to payment process
        del request.session['booking_id']
        del request.session['razorpay_order_id']

        # Redirect to the ViewEventBooking page or a success page
        return redirect('WebStationMaster:vieweventbooking')

    return HttpResponseBadRequest("Invalid request method")

def refund(request):
    # Fetch the booking object using booking_id
    booking_id = request.POST.get('booking_id')

    if not booking_id:
        return HttpResponseBadRequest("Booking ID is missing")

    # Fetch the booking object using booking_id
    booking = get_object_or_404(tbl_ticketbooking, id=booking_id)

    if booking.payment != 2:
        return HttpResponseBadRequest("Refund is not allowed for this booking
status")

    rate = booking.service.rate
    adults  = booking.adults_count
    childrens  = booking.childrens_count
    event_type_rate = int((rate*float(adults))) + int((rate*float(childrens)))


    # Ensure event_type_rate is numeric and calculate 80% refund amount
    try:
        event_type_rate = float(event_type_rate)
        refund_amount = int(event_type_rate * 80 / 100 * 100)  # Convert to paise
    except (ValueError, TypeError):
        return HttpResponseBadRequest("Invalid rate value")
```

```
    booking.refund_amount = refund_amount
    booking.save()

    if request.method == "POST":
        # Process the refund via Razorpay
        razorpay_order = razorpay_client.order.create({
            "amount": refund_amount,
            "currency": "INR",
            "payment_capture": "1"
        })

        request.session['booking_id'] = booking_id
        request.session['razorpay_order_id'] = razorpay_order['id']

        # Render the payment page for refund
        context = {
            "razorpay_order_id": razorpay_order['id'],
            "razorpay_key": settings.RAZORPAY_KEY_ID,
            "amount": refund_amount,
            "booking_number": booking.ticket_number,
        }
        return render(request, 'StationMaster/payment.html', context)
    else:
        return HttpResponseBadRequest("Invalid request method")

def callback(request):
    if request.method == "POST":
        # Fetch the booking ID stored in session during payment process
        booking_id = request.session.get('booking_id')
        if not booking_id:
            return HttpResponseBadRequest("Booking ID not found in session")

        # Fetch the booking object using booking_id
        booking = get_object_or_404(tbl_ticketbooking, id=booking_id)

        # Update the booking record and set status to 4 (refunded)
        booking.payment = 3  # Assuming 4 indicates refund completed
        booking.save()

        # Clear the session data related to payment process
        del request.session['booking_id']
        del request.session['razorpay_order_id']

        # Redirect to the ViewEventBooking page or a success page
        return redirect('WebStationMaster:ViewTicketBooking')

    return HttpResponseBadRequest("Invalid request method")

@never_cache
```

```
@login_required
def report(request):
    booking = tbl_ticketbooking.objects.all()
    eventbooking = tbl_eventbooking.objects.all()
    combined_list = []
    counter = 1
    for item in booking:
        combined_list.append({
            'type': 'booking',
            'data': item,
            'counter': counter
        })
        counter += 1

    # Add eventbooking items to combined list with a counter
    for item in eventbooking:
        combined_list.append({
            'type': 'eventbooking',
            'data': item,
            'counter': counter
        })
        counter += 1
    return render(request, 'StationMaster/Report.html', {"combined_list":
combined_list})


def ajaxreport(request):
    # Get filter parameters from the request
    fdate = request.GET.get("fdate", "")
    tdate = request.GET.get("tdate", "")
    status = request.GET.get("status", "")
    ticket_type = request.GET.get("ticketType", "")

    # Initialize empty queryset
    booking = tbl_ticketbooking.objects.none()
    eventbooking = tbl_eventbooking.objects.none()
    combined_list = []
    counter = 1
    # Filter based on ticket type
    if ticket_type == "1":  # Public Transport Boat Services (ticket booking)
        booking = tbl_ticketbooking.objects.all()
        if fdate:
            booking = booking.filter(date__gte=fdate)
        if tdate:
            booking = booking.filter(date__lte=tdate)
        if status:
            booking = booking.filter(payment=status)
```

```
    elif ticket_type == "2":  # Tourism and Recreational Boat Services (event
booking)
        eventbooking = tbl_eventbooking.objects.all()
        if fdate:
            eventbooking = eventbooking.filter(event_date__gte=fdate)
        if tdate:
            eventbooking = eventbooking.filter(event_date__lte=tdate)
        if status:
            eventbooking = eventbooking.filter(status=status)

    else:  # Show both ticket and event bookings (All)
        booking = tbl_ticketbooking.objects.all()
        eventbooking = tbl_eventbooking.objects.all()
        if fdate:
            booking = booking.filter(date__gte=fdate)
            eventbooking = eventbooking.filter(event_date__gte=fdate)
        if tdate:
            booking = booking.filter(date__lte=tdate)
            eventbooking = eventbooking.filter(event_date__lte=tdate)
        if status:
            booking = booking.filter(payment=status)
            eventbooking = eventbooking.filter(status=status)

    for item in booking:
        combined_list.append({
            'type': 'booking',
            'data': item,
            'counter': counter
        })
        counter += 1

    # Add eventbooking items to combined list with a counter
    for item in eventbooking:
        combined_list.append({
            'type': 'eventbooking',
            'data': item,
            'counter': counter
        })
        counter += 1
    # Render the filtered results in the AjaxReport template
    return render(request, "StationMaster/AjaxReport.html", {"combined_list":
combined_list})
```

9.2 SCREEN SHOTS

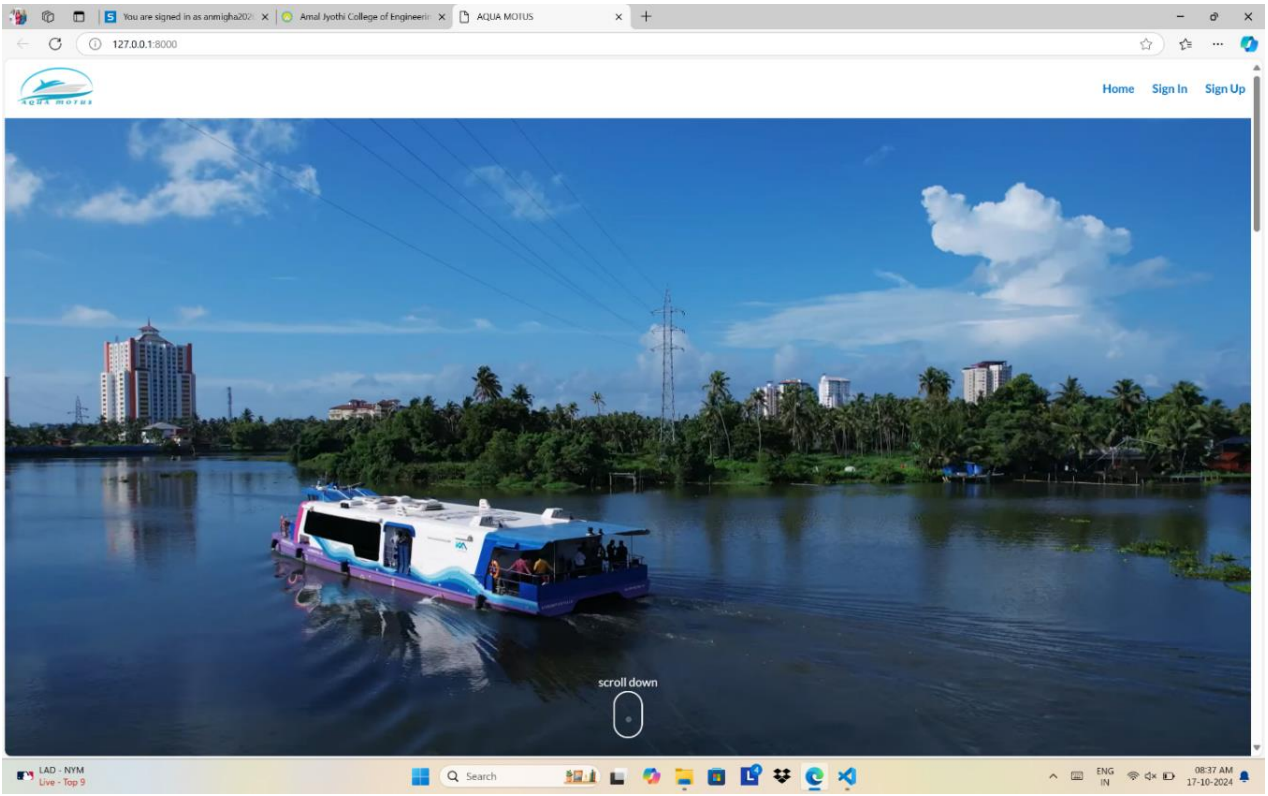

Figure 28: Home page

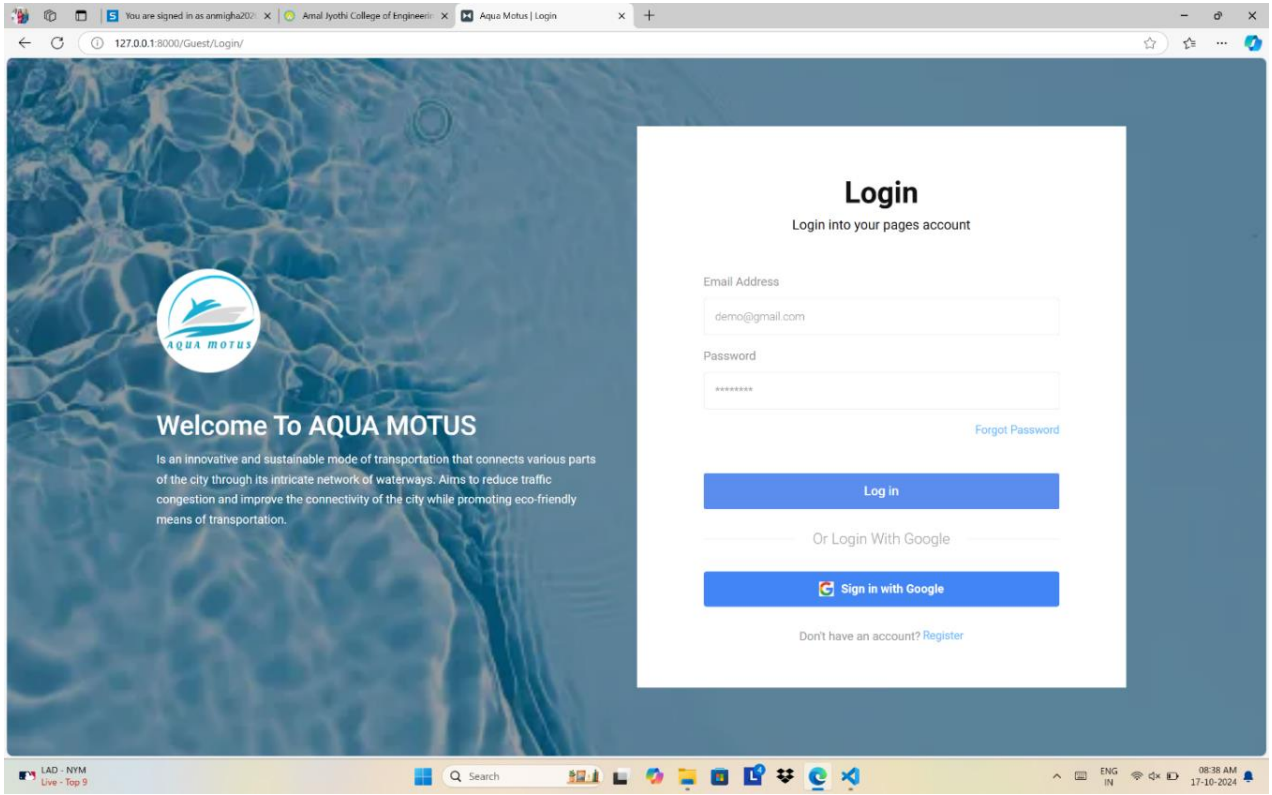

Figure 29: Login page

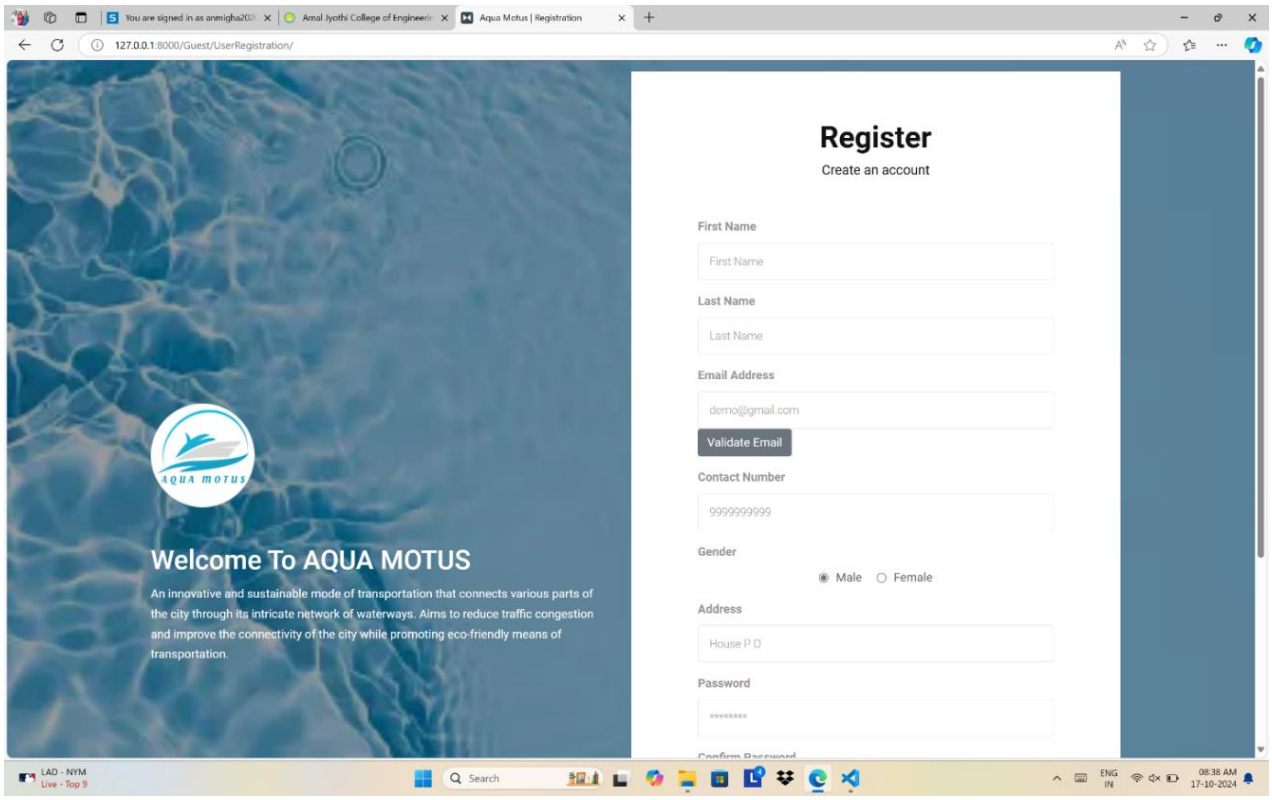

Figure 30: Registration Page

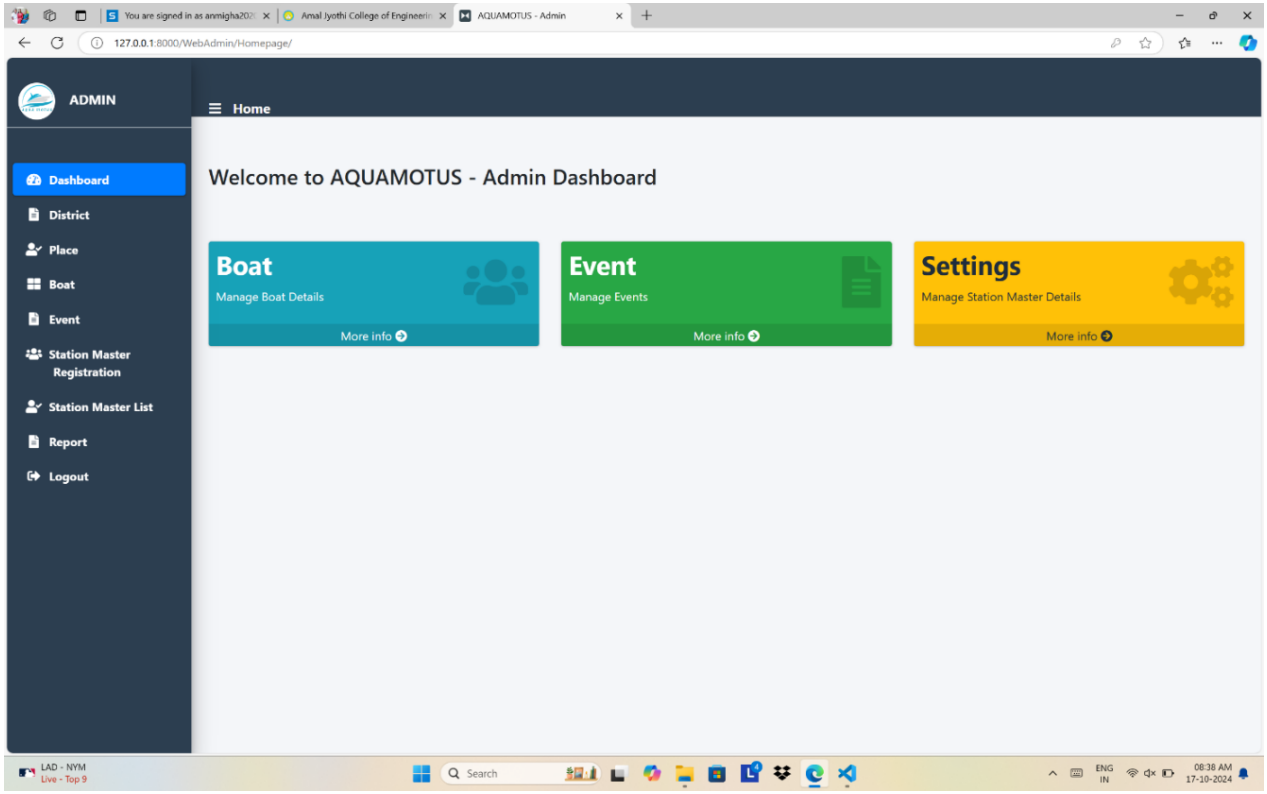

Figure 31: Admin Dashboard

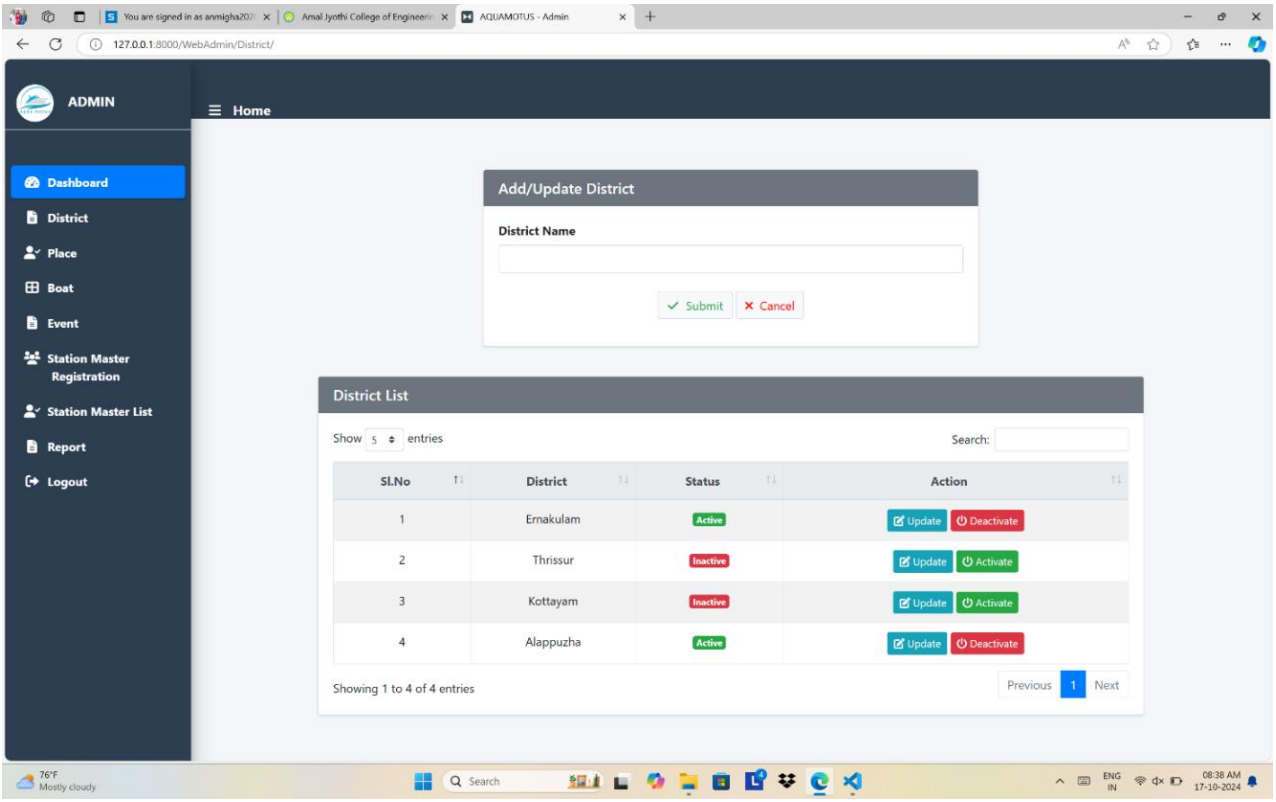

Figure 32: Add District

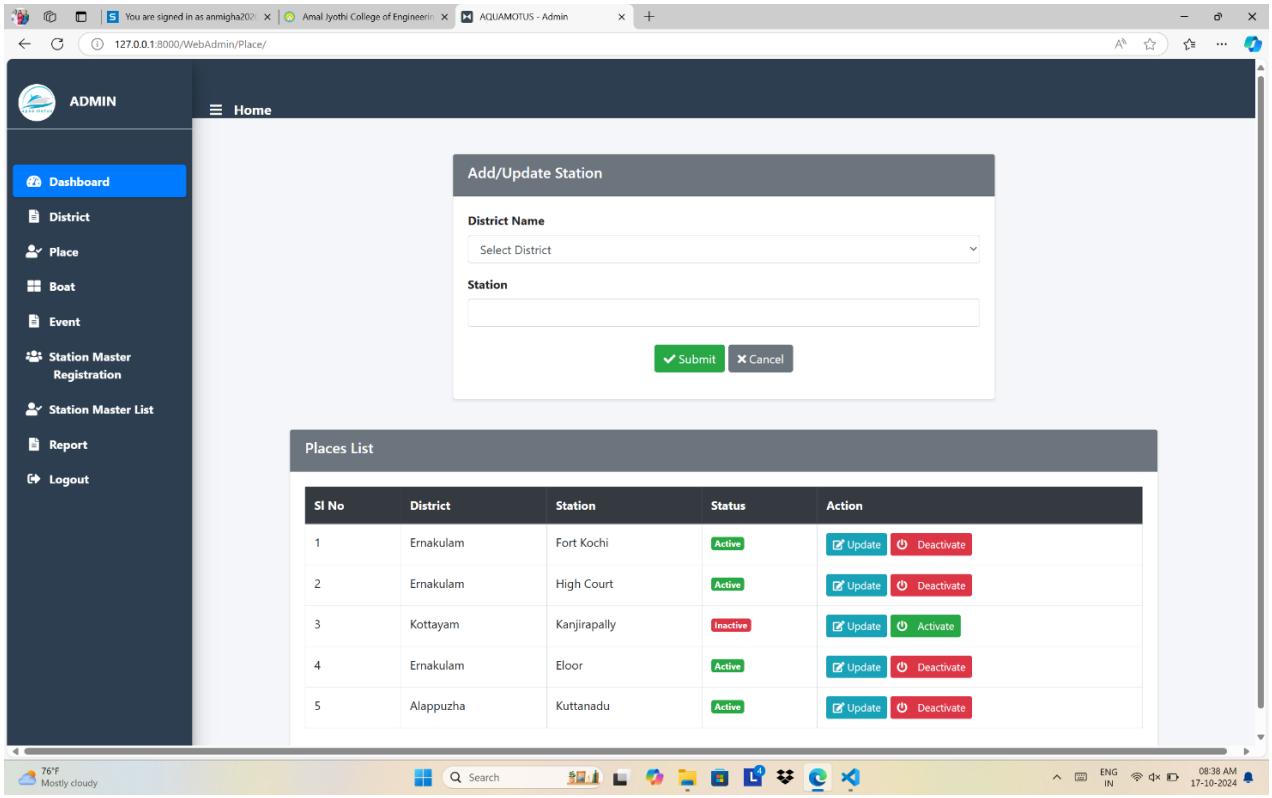

Figure 33: Add Place

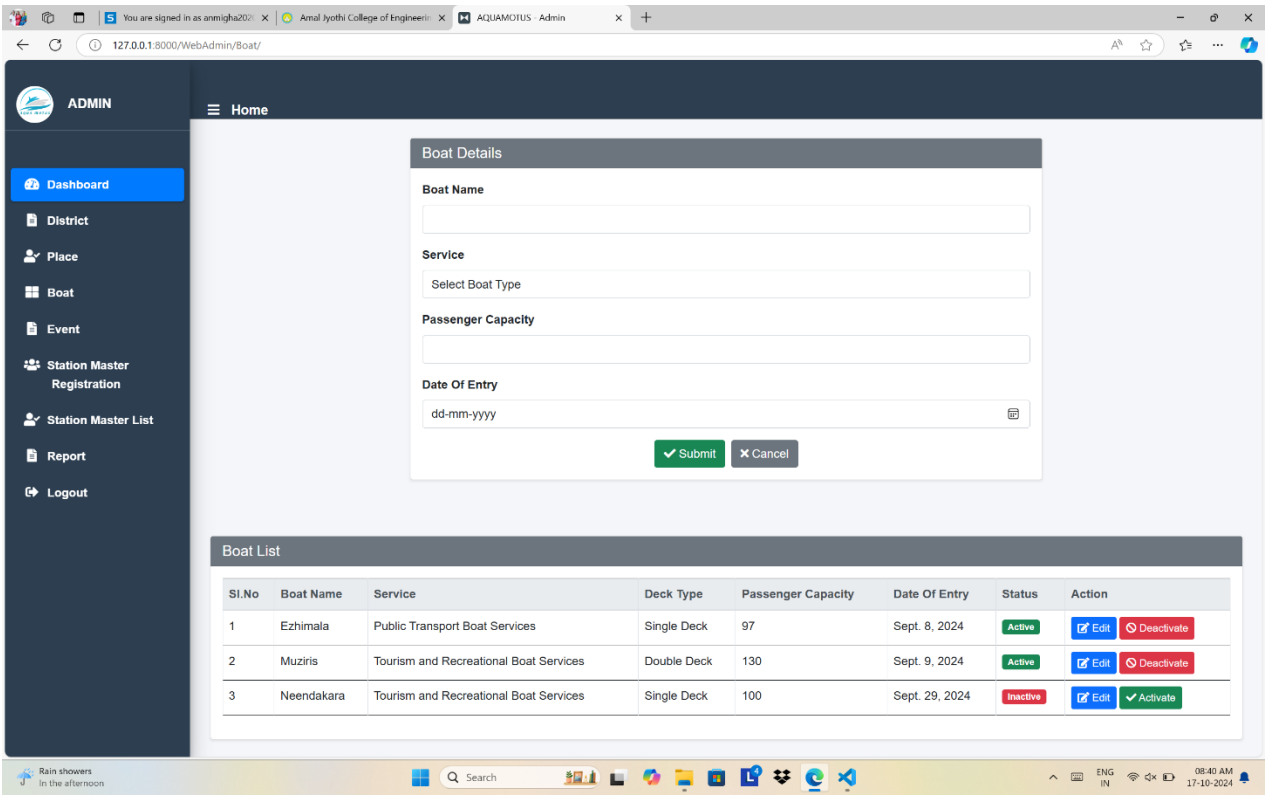

Figure 34: Add Boat

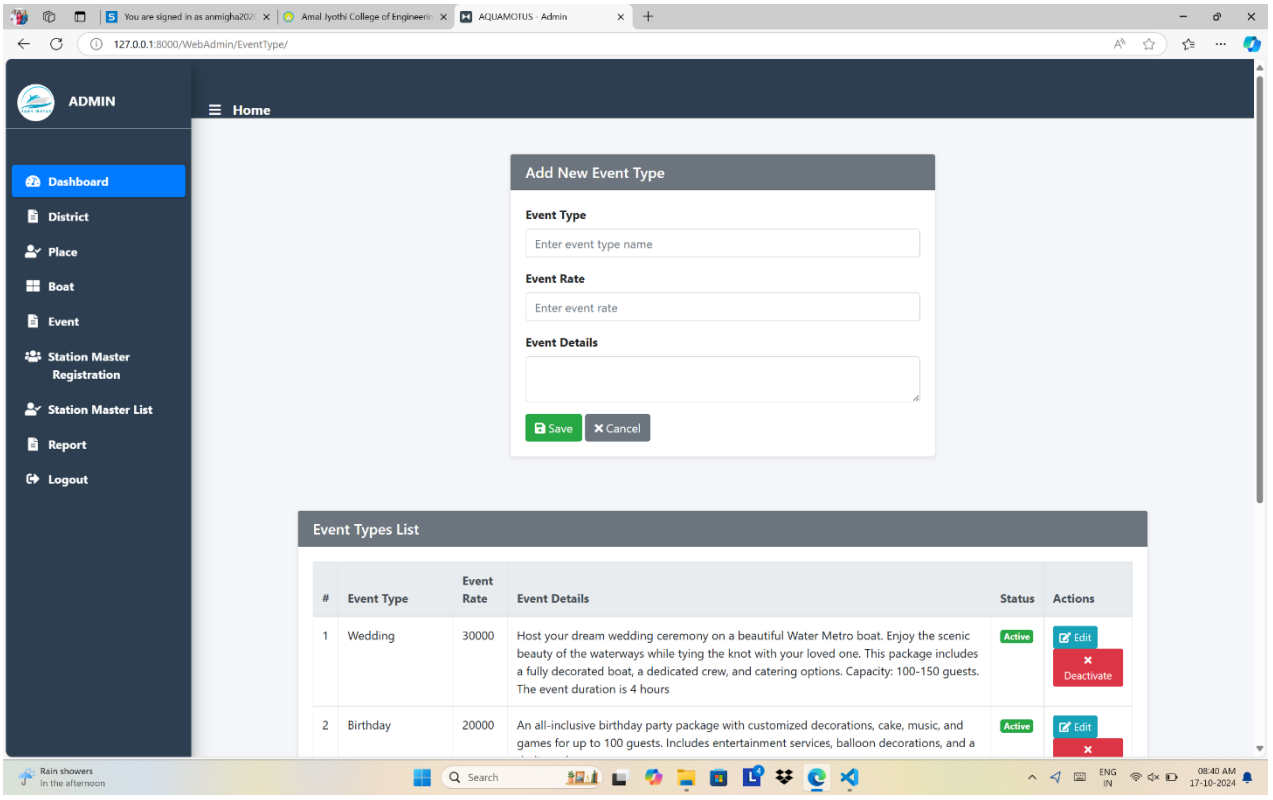

Figure 35: Add Event

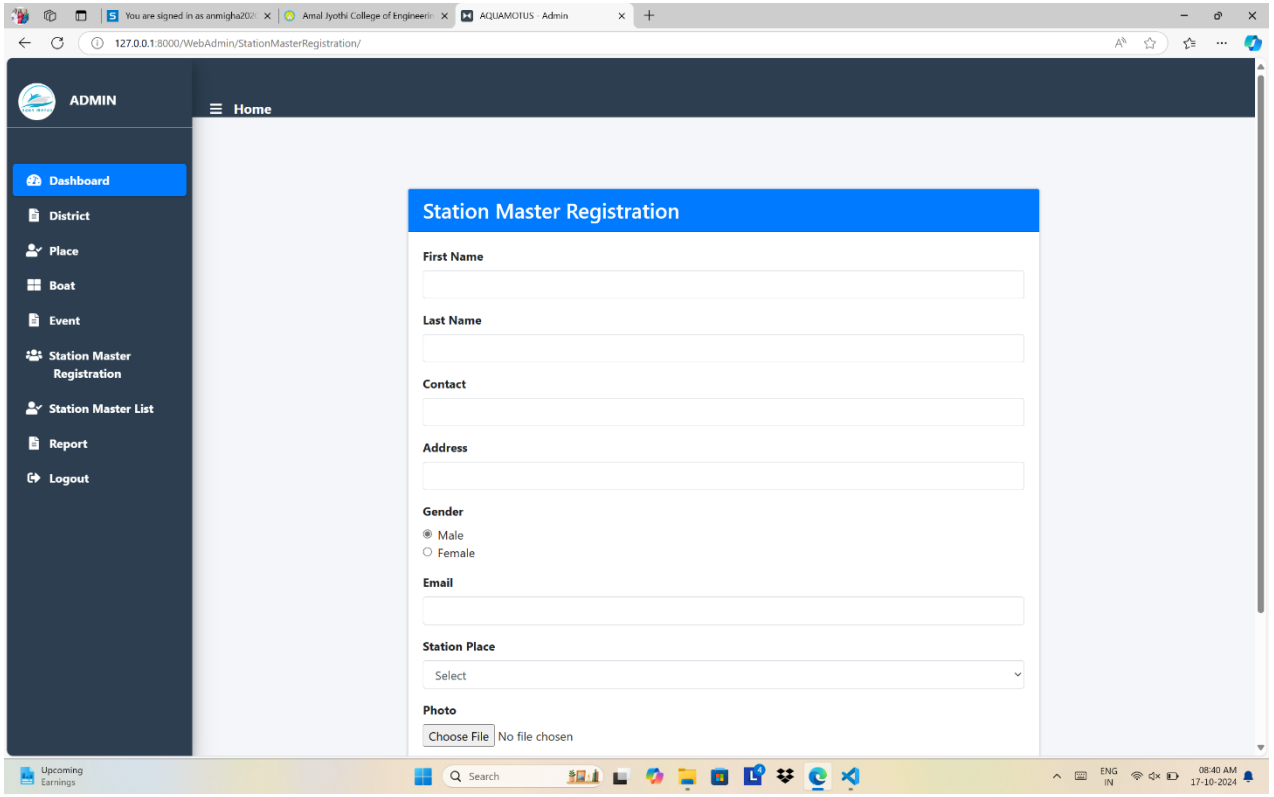

Figure 36: Station Master Registration

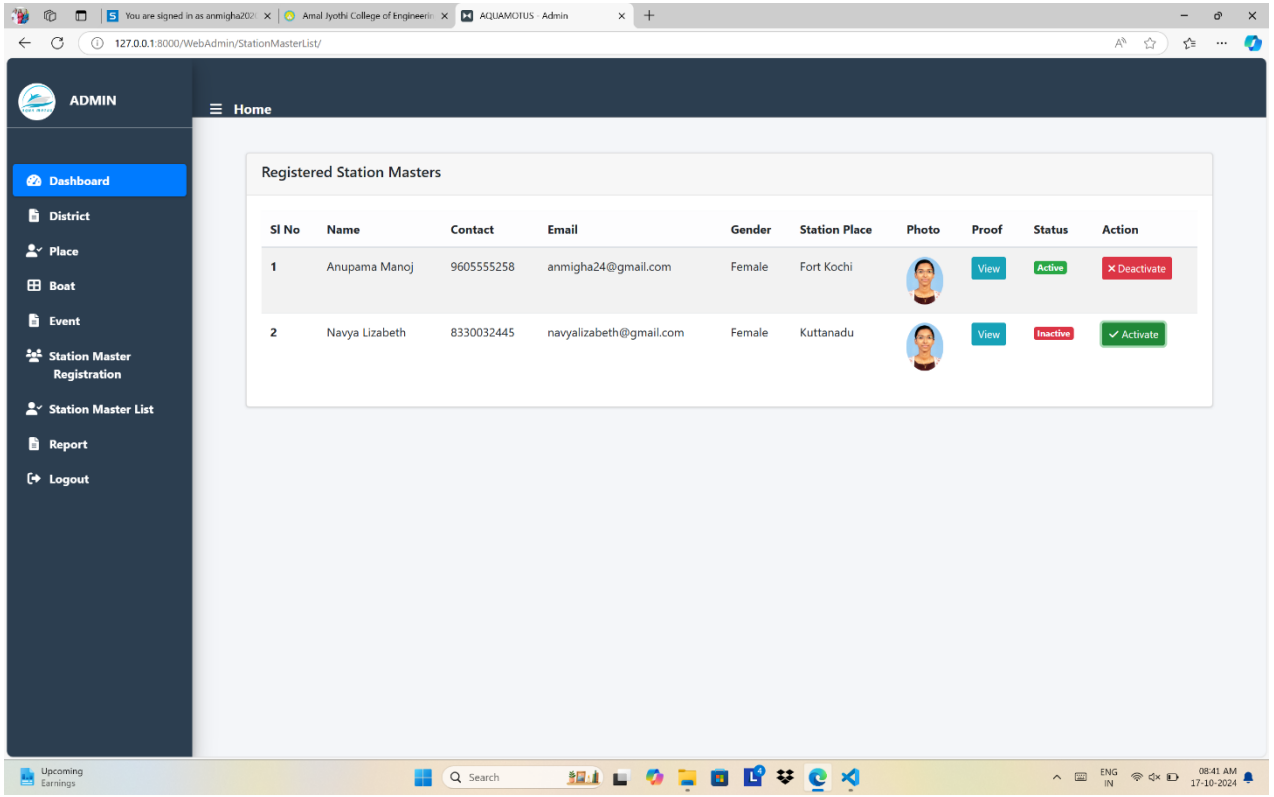

Figure 37: List Registered Station Masters

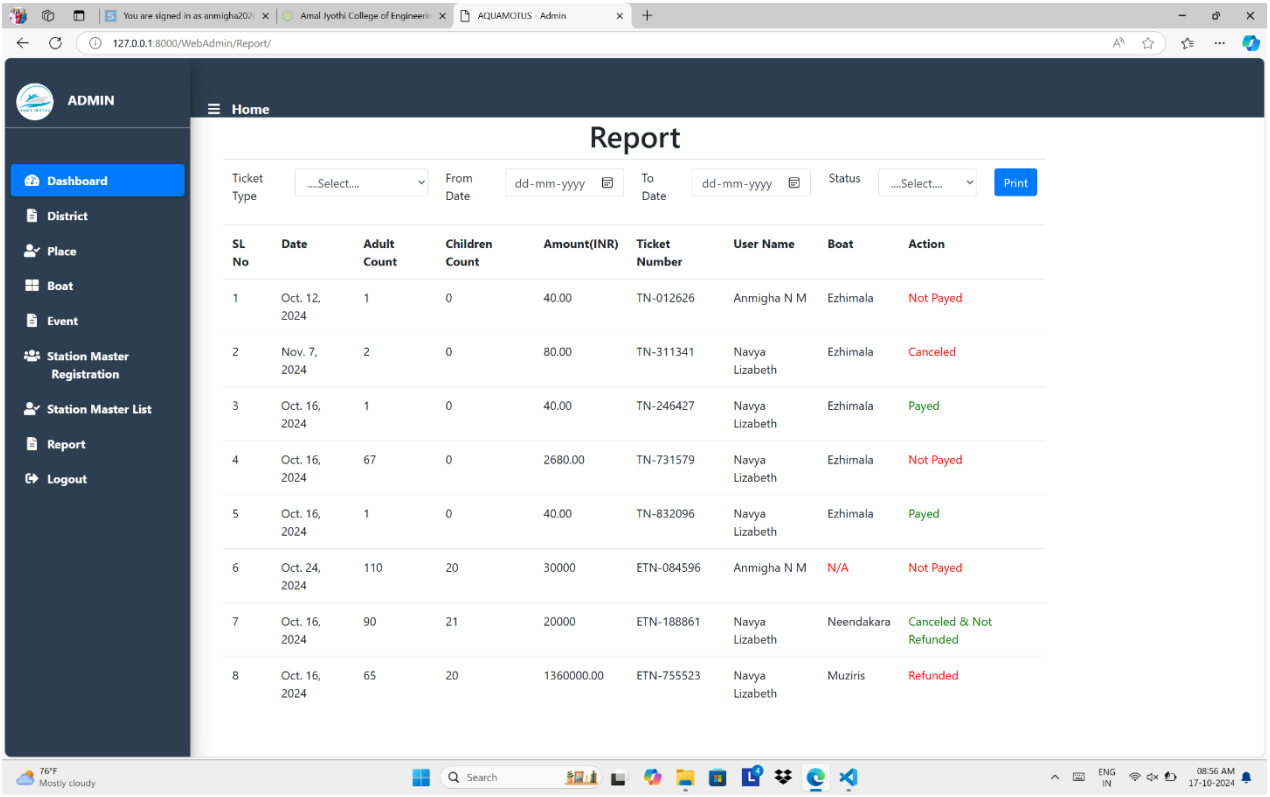

Figure 38: Report

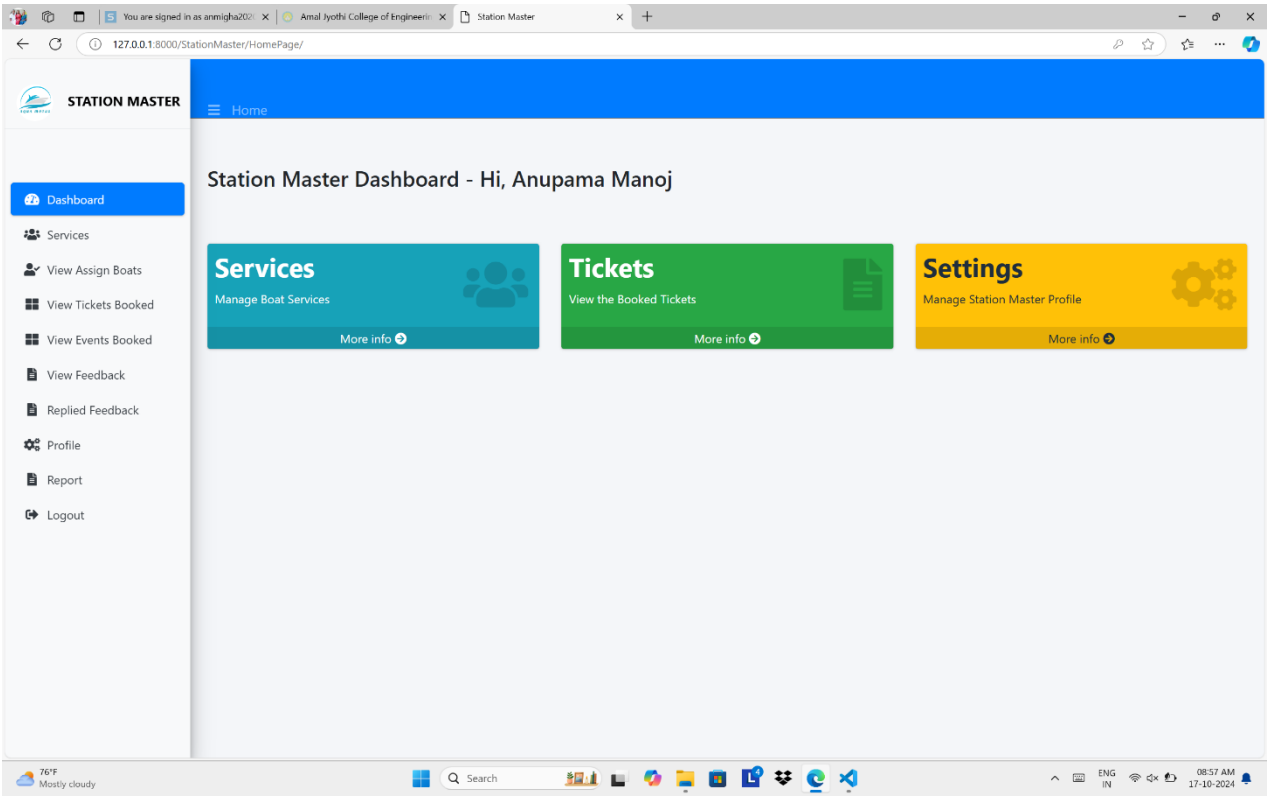

Figure 39: Station Master Dashboard

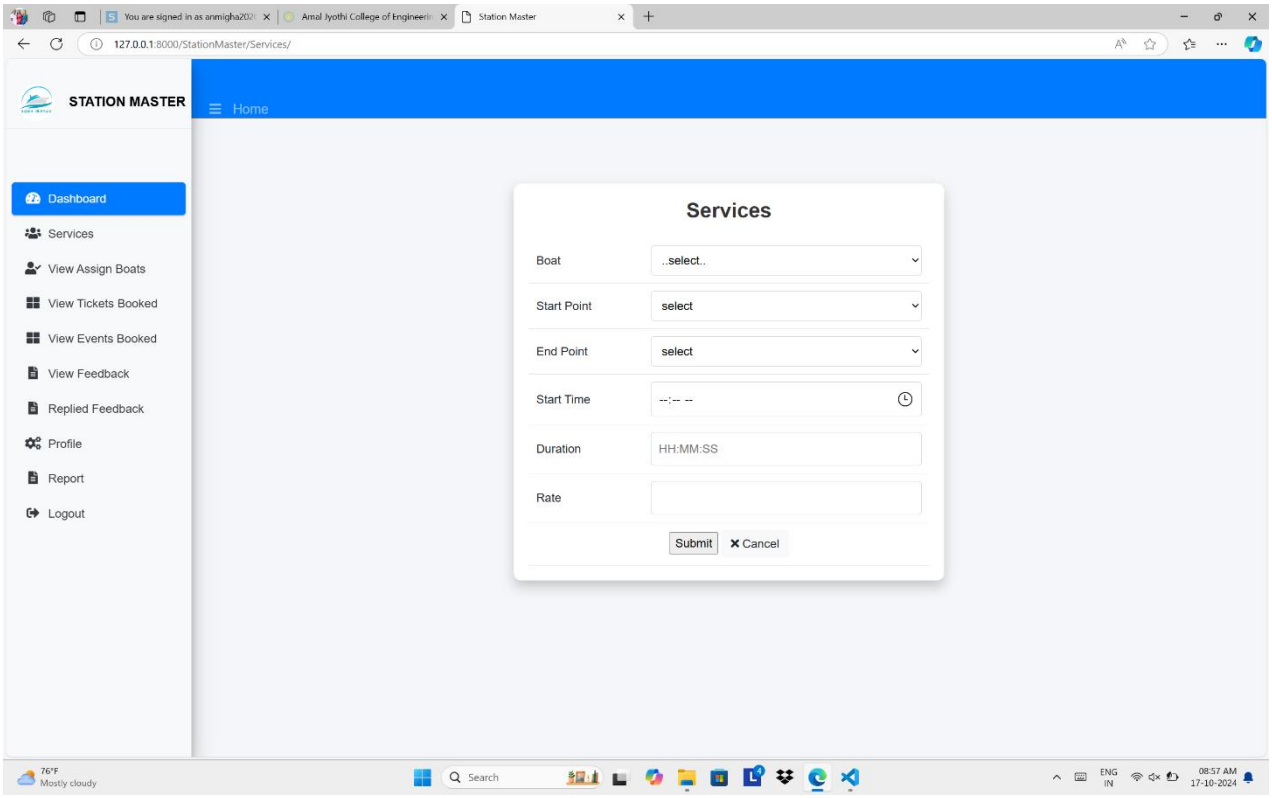

Figure 40: Add Services

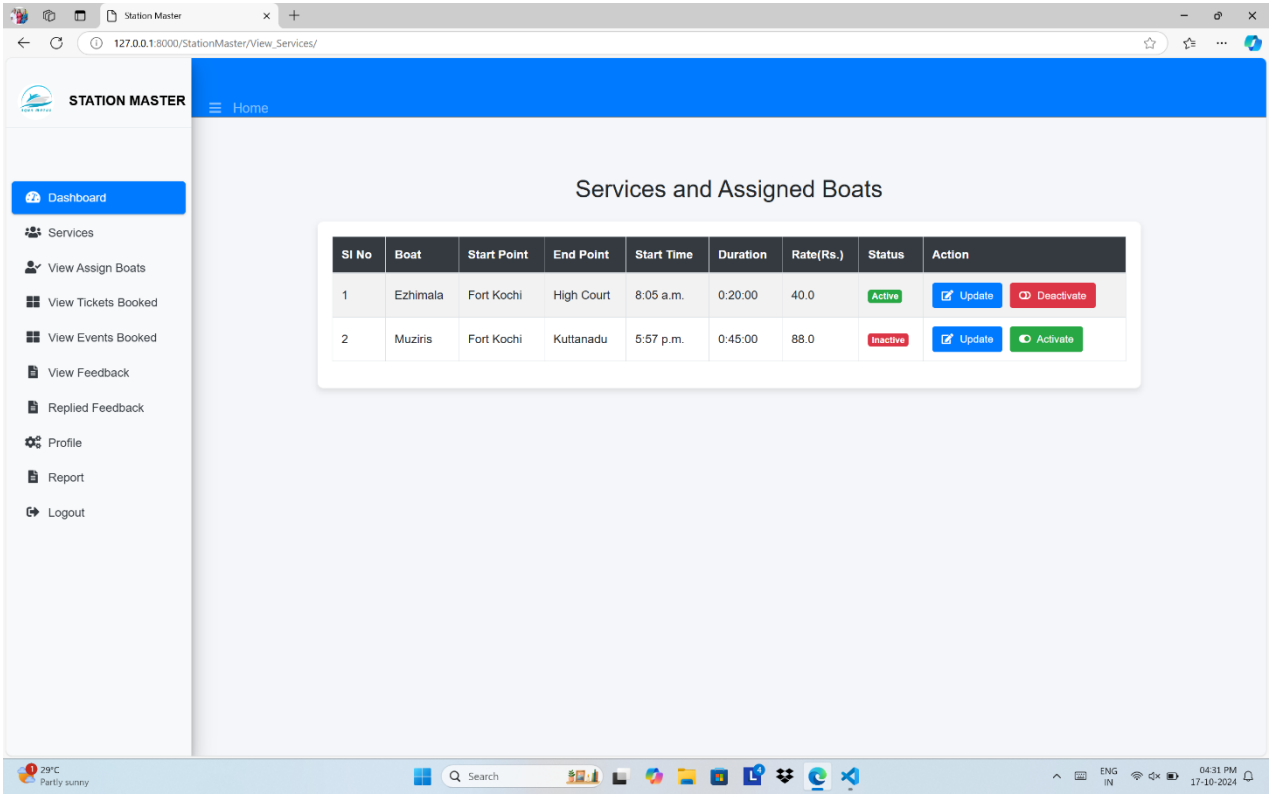

Figure 41: View Servies

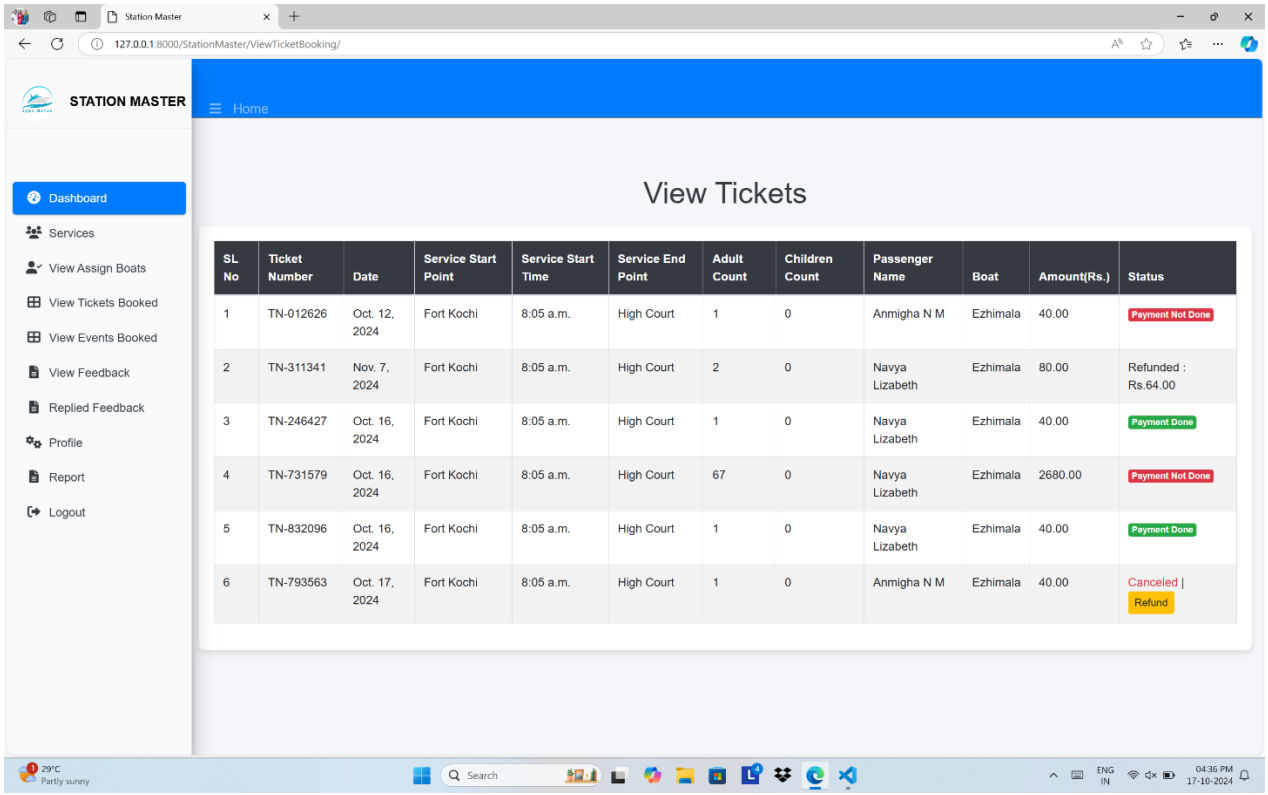

Figure 42: View Tickets

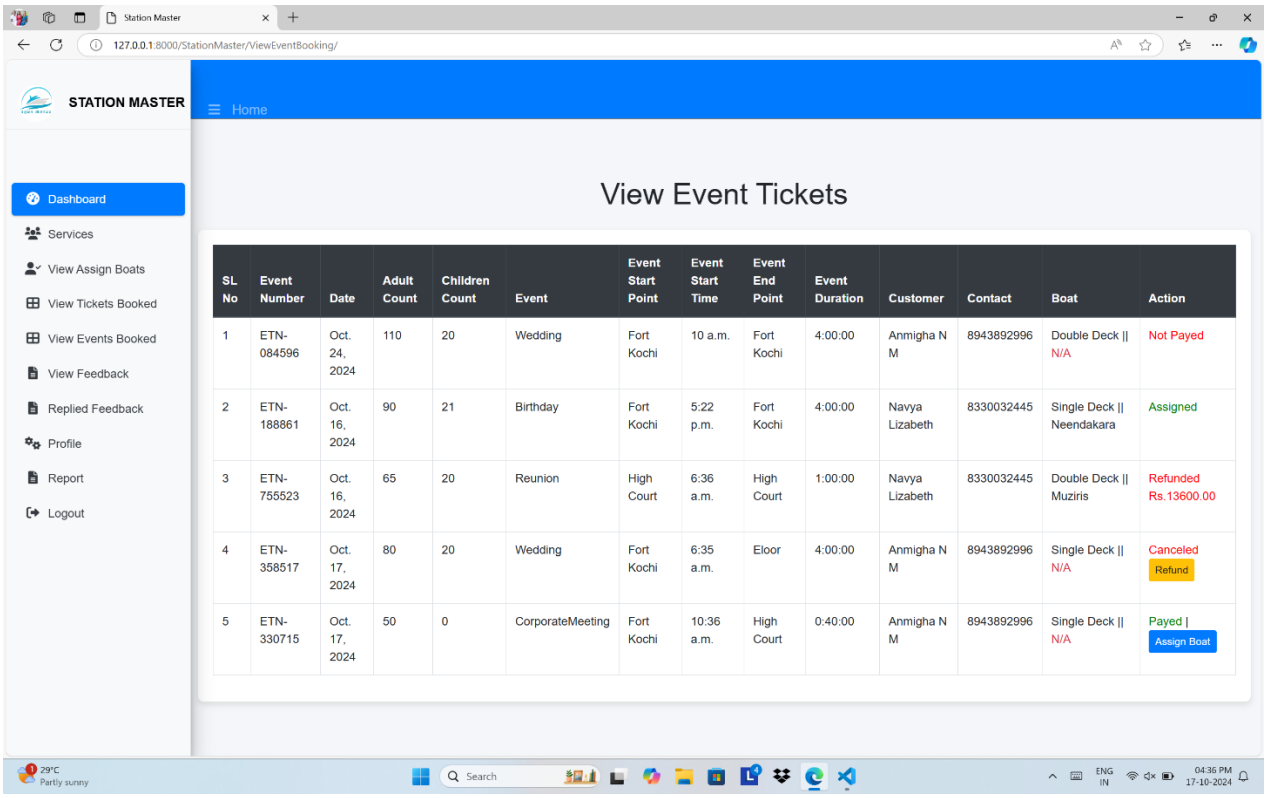

Figure 43: View Event Tickets

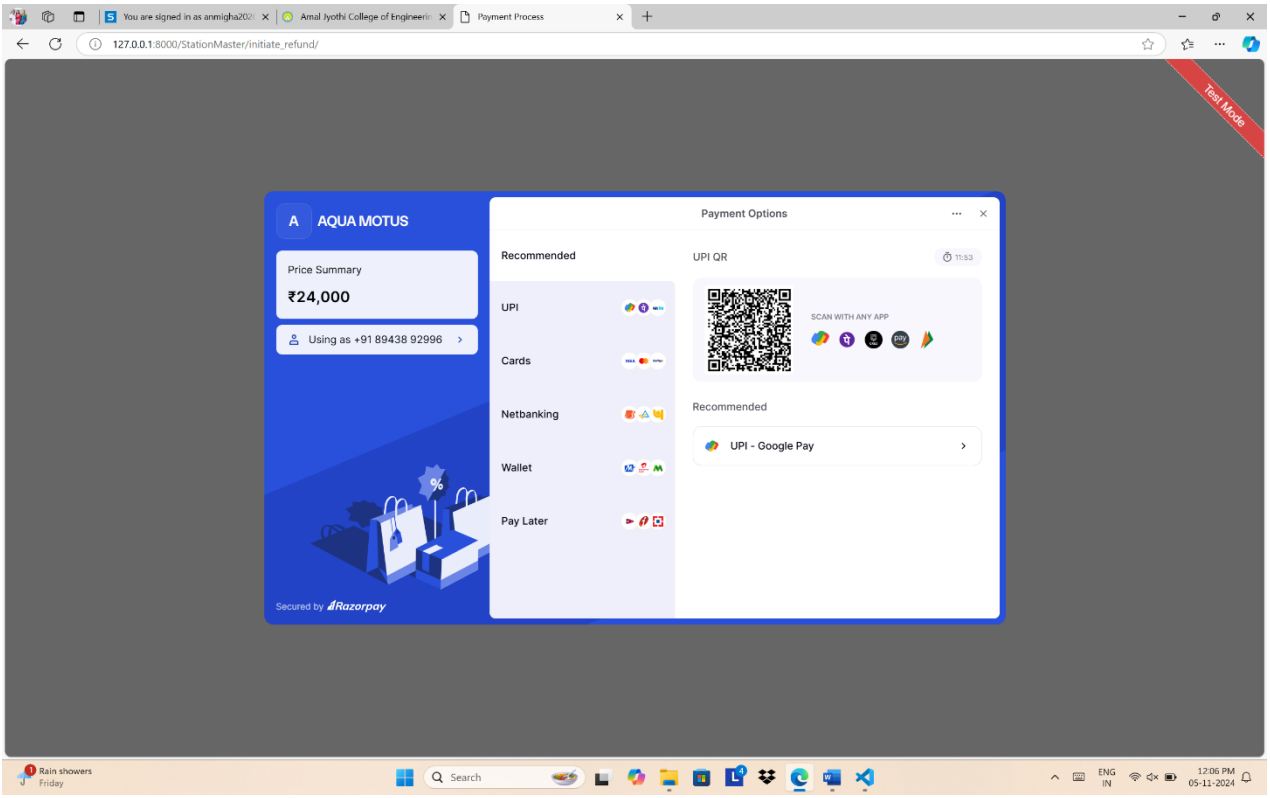

Figure 44: Refund Amount

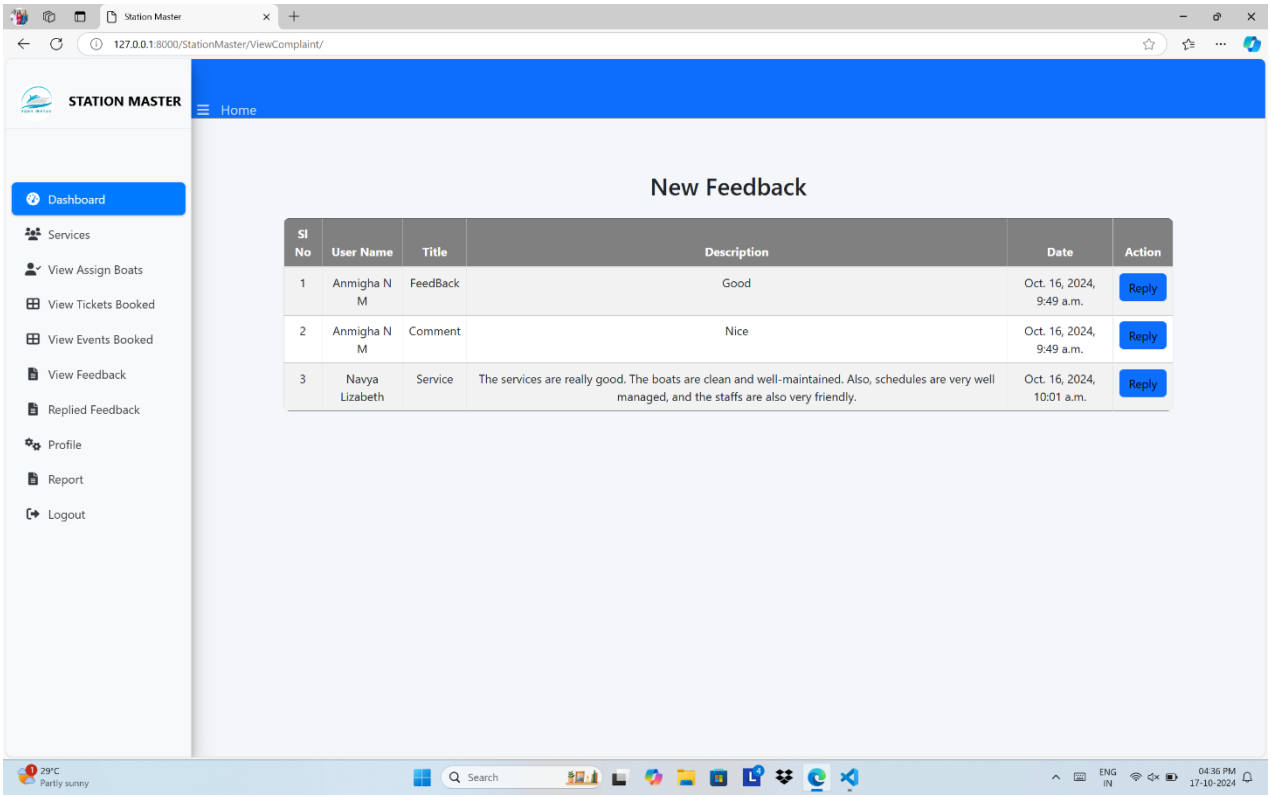

Figure 45: View Feedback

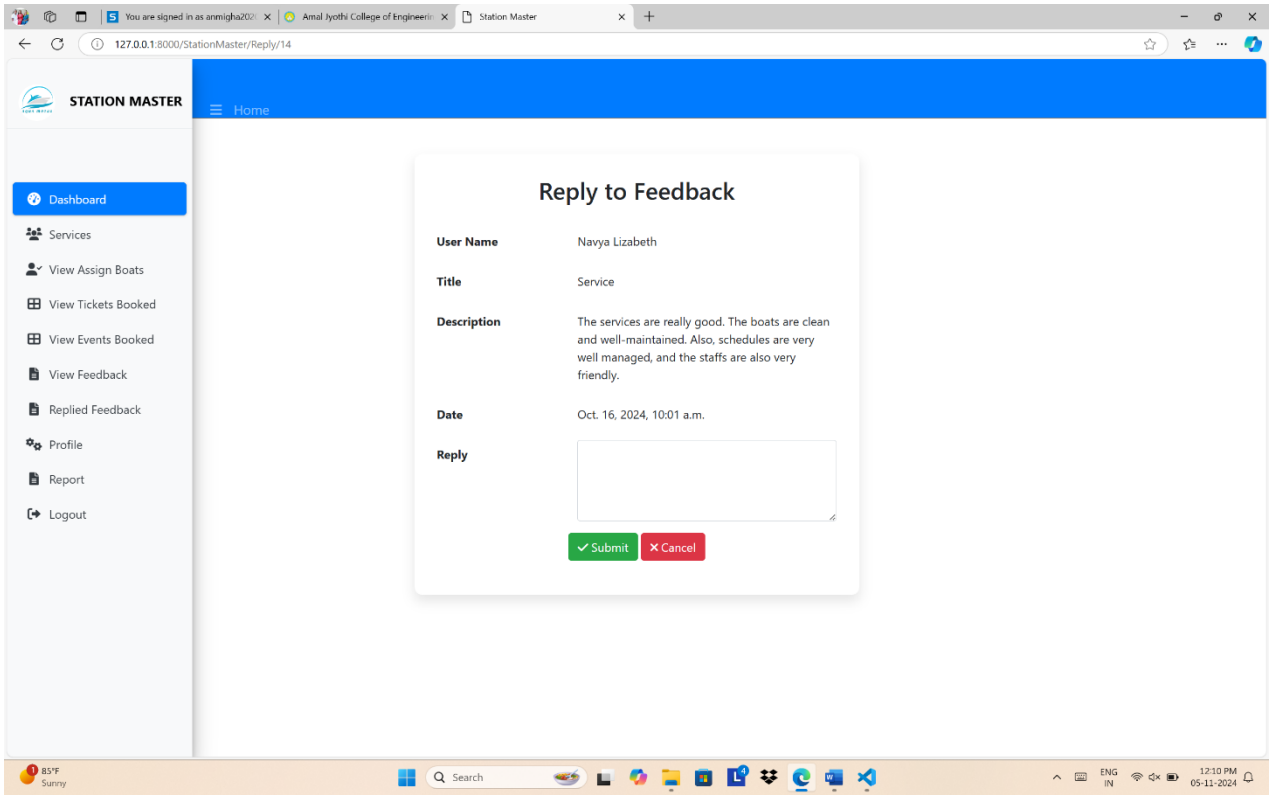

Figure 46: Reply feedback

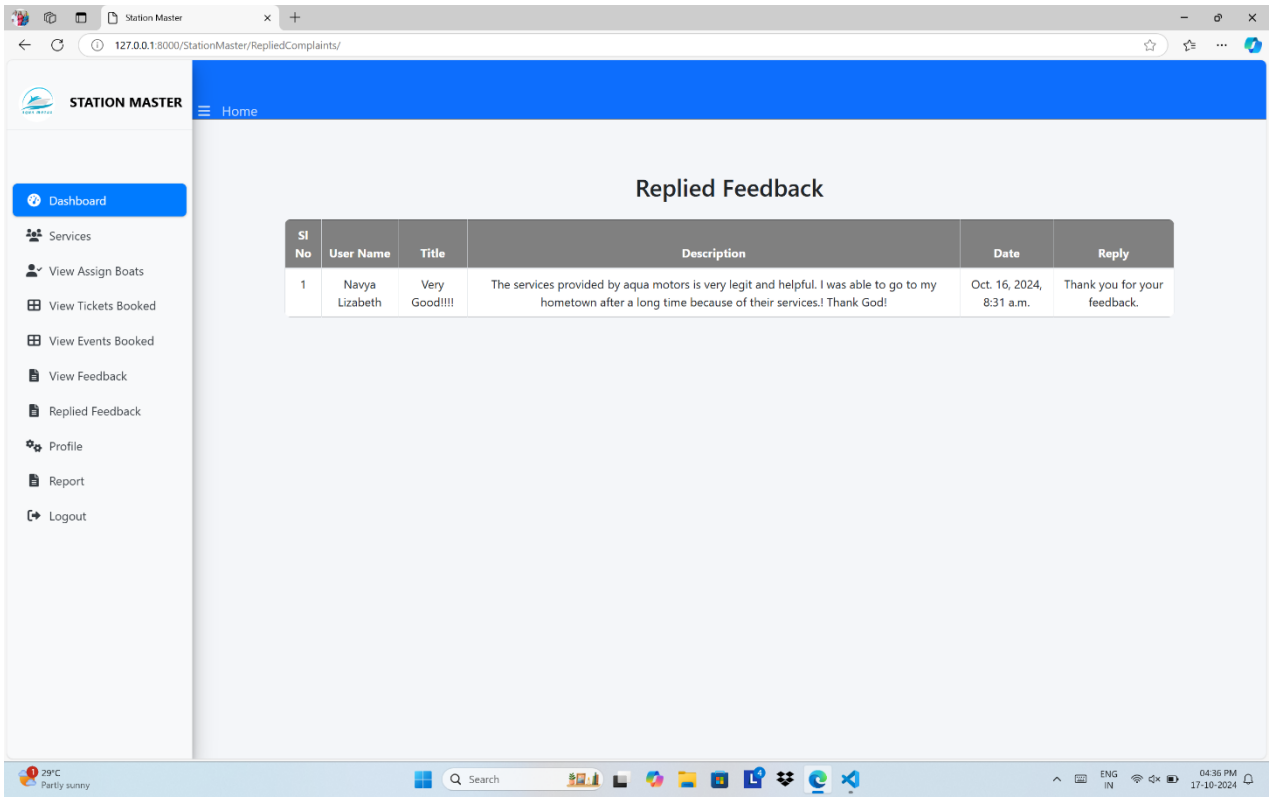

Figure 47: View Replied Feedback

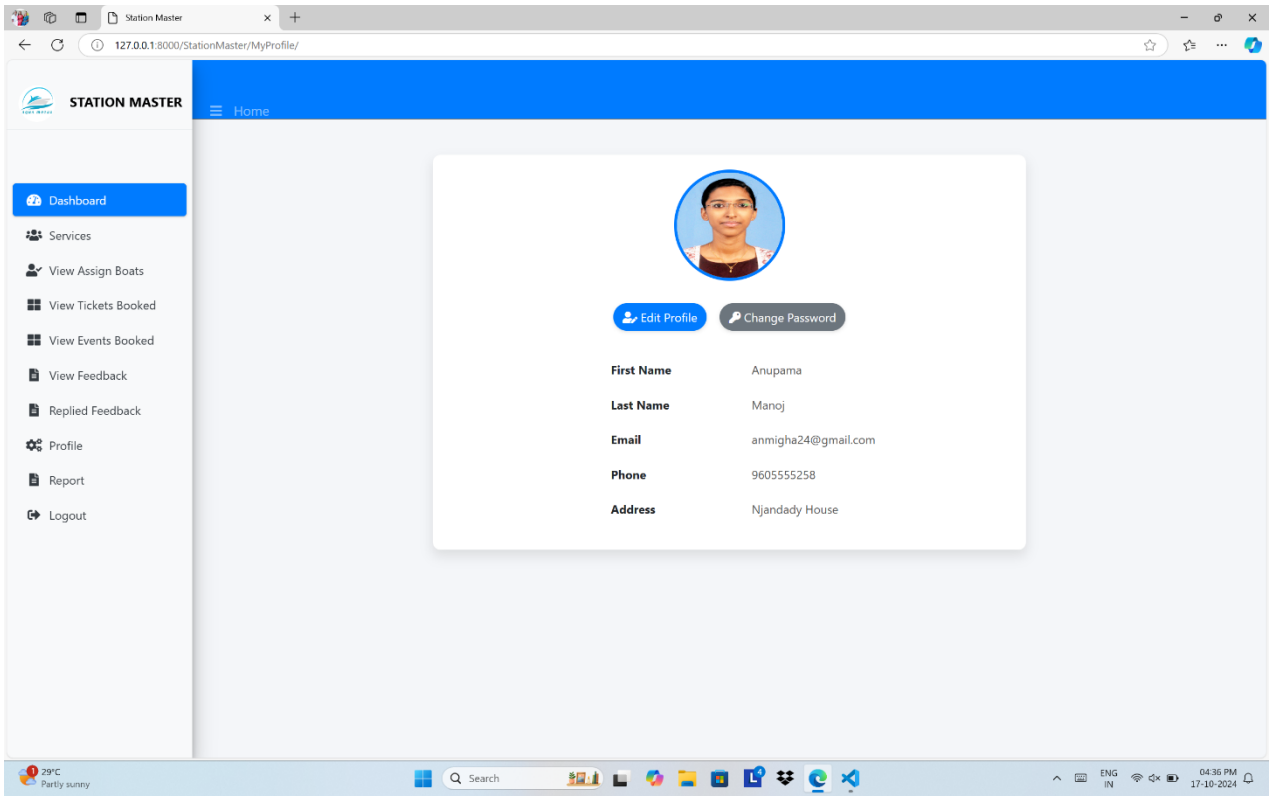

Figure 48: Profile View

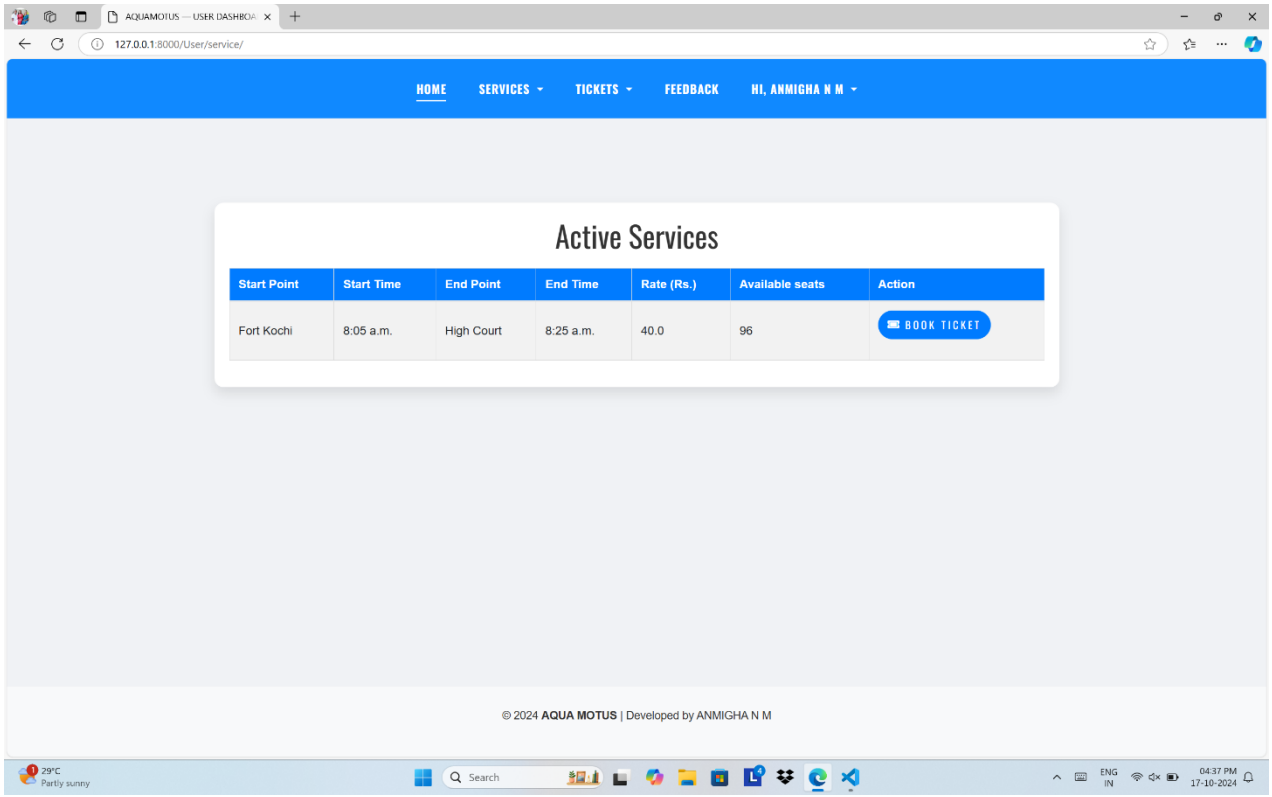

Figure 49: Active Services

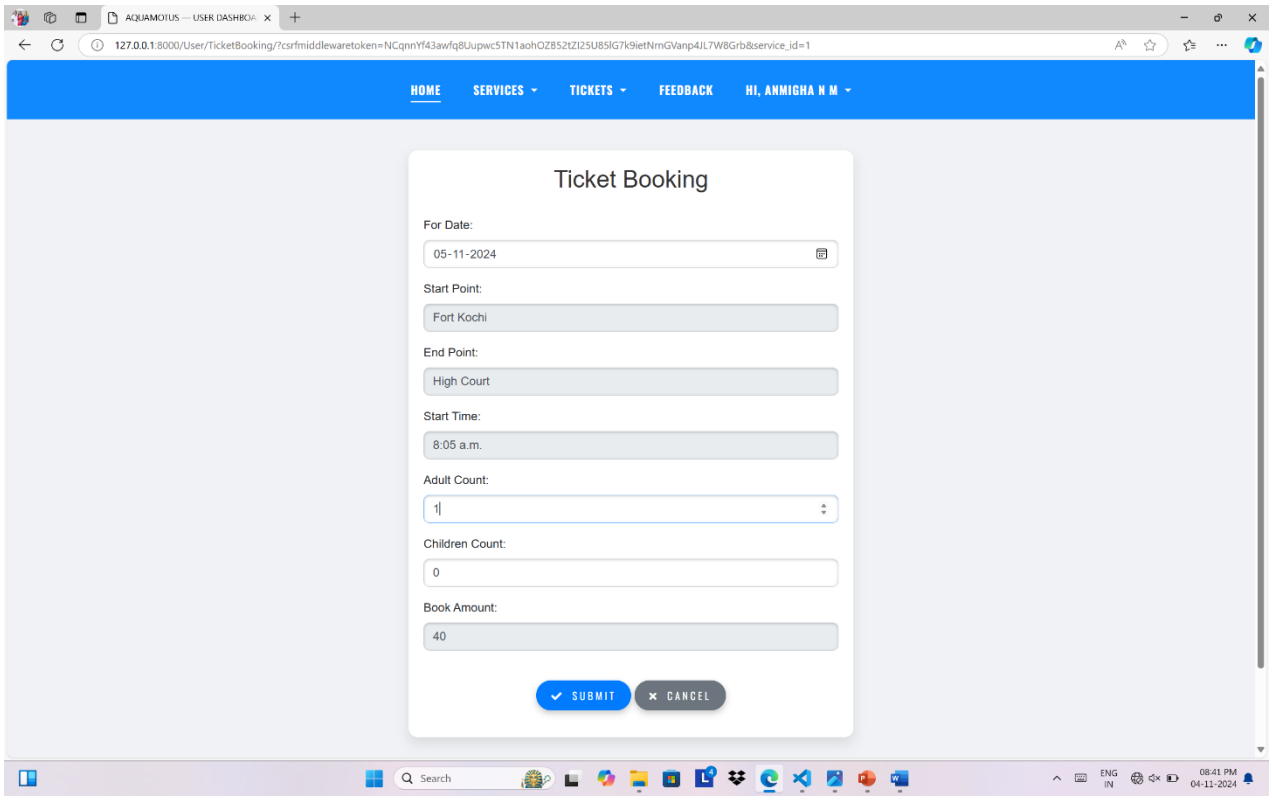

Figure 50: Ticket Booking form

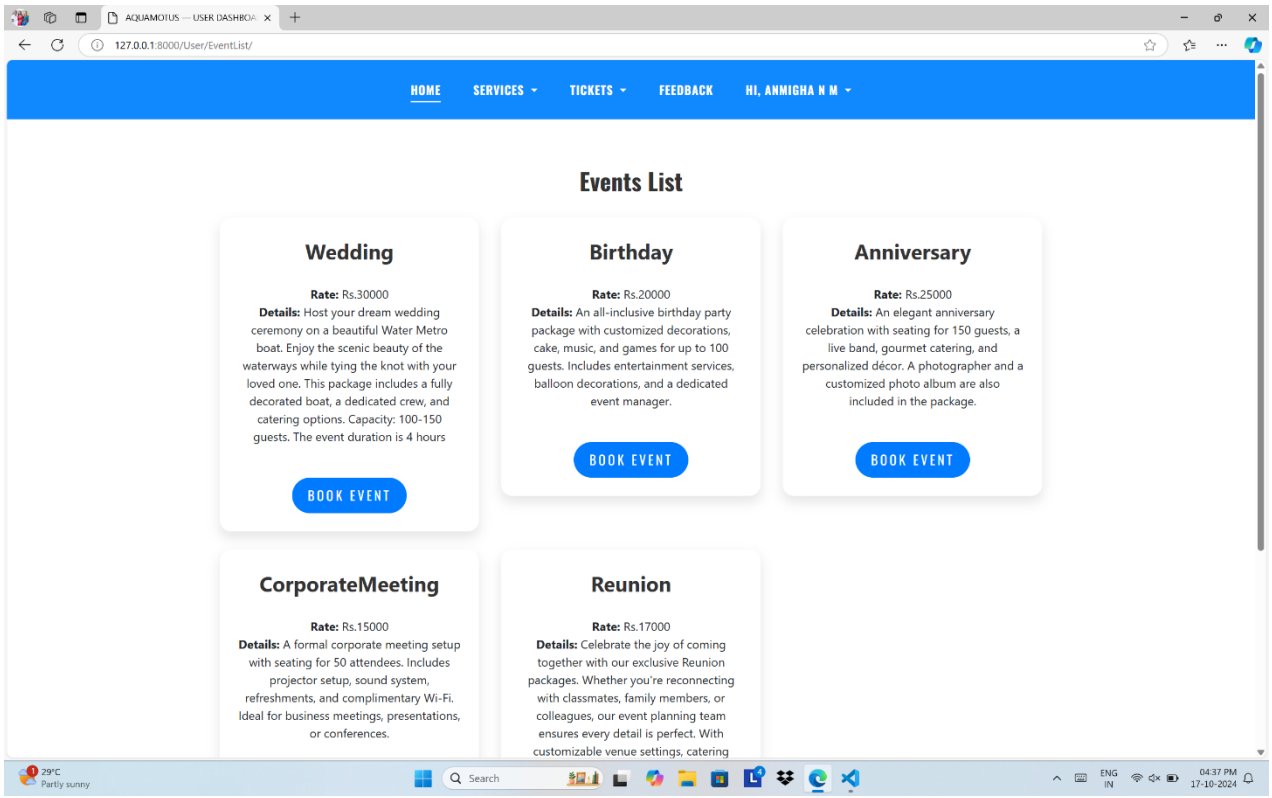

Figure 51: Available Events

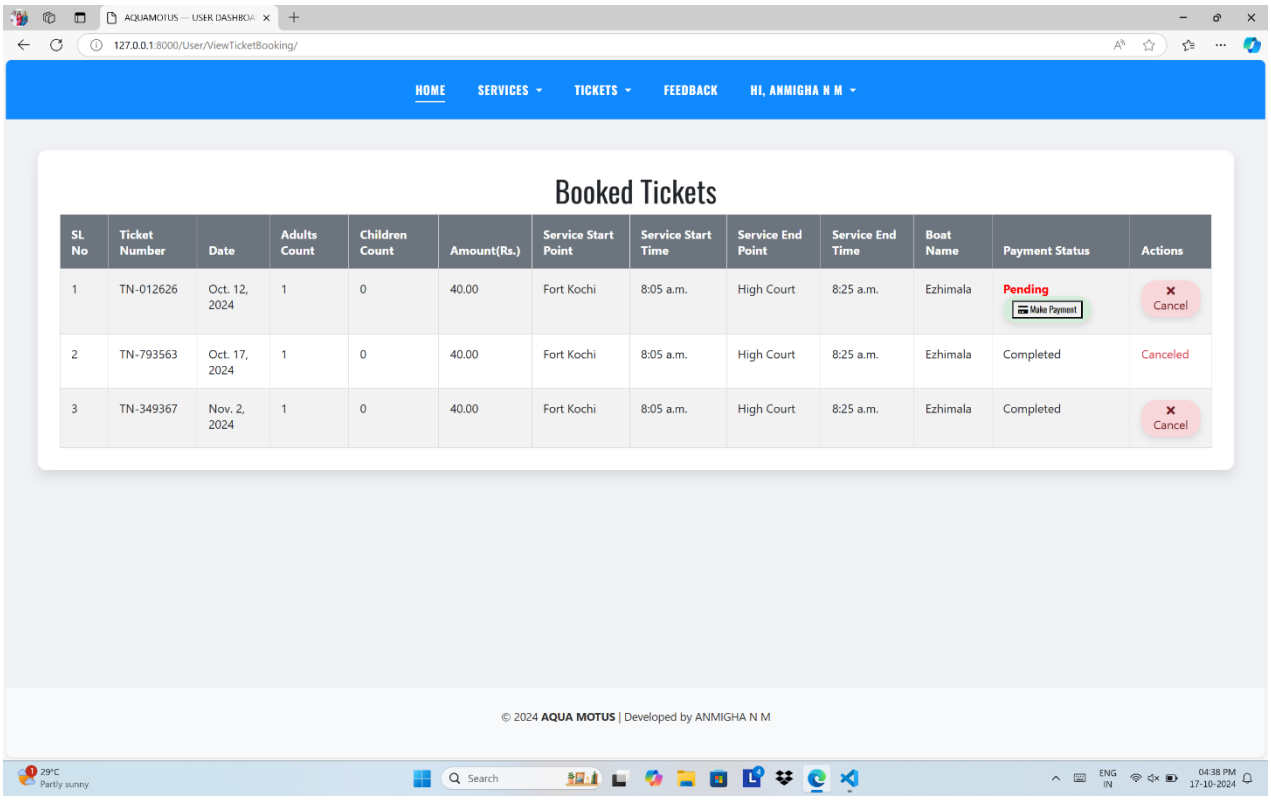

Figure 52: Booked Tickets

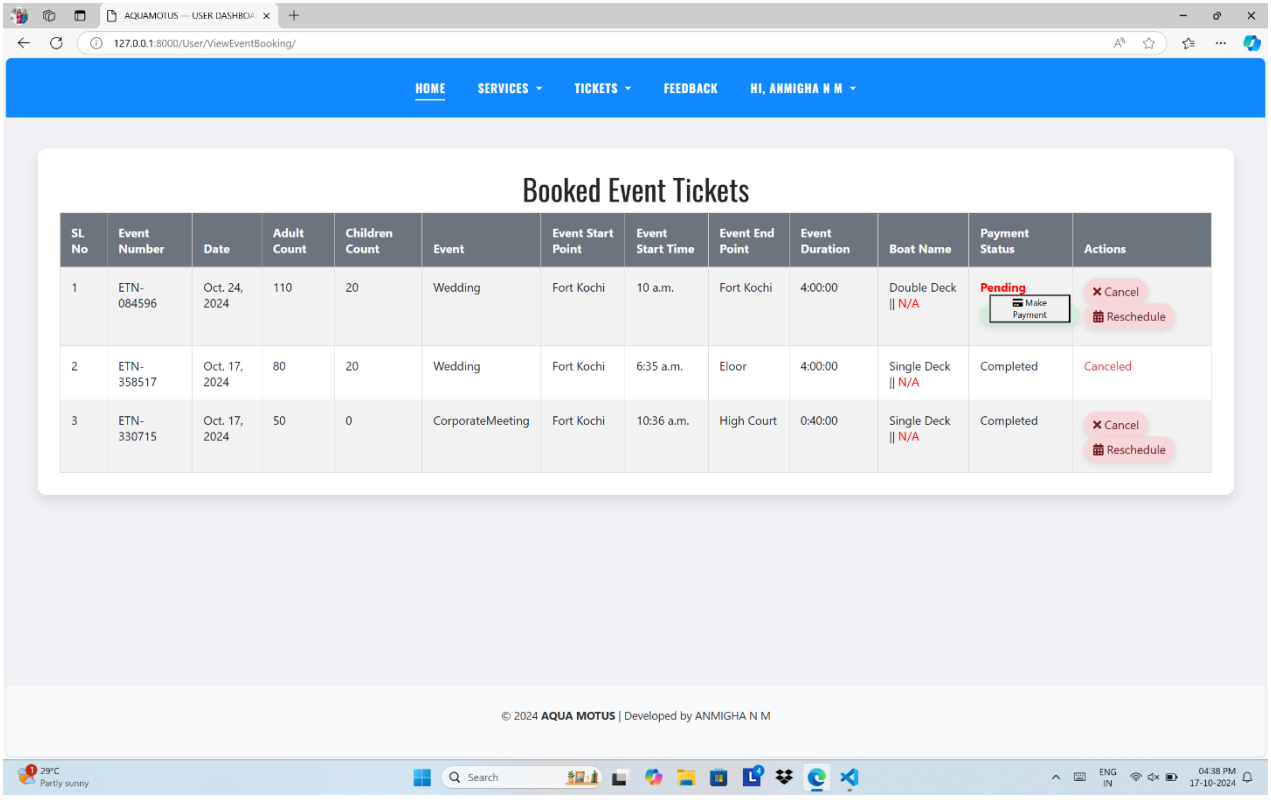

Figure 53: Booked Events

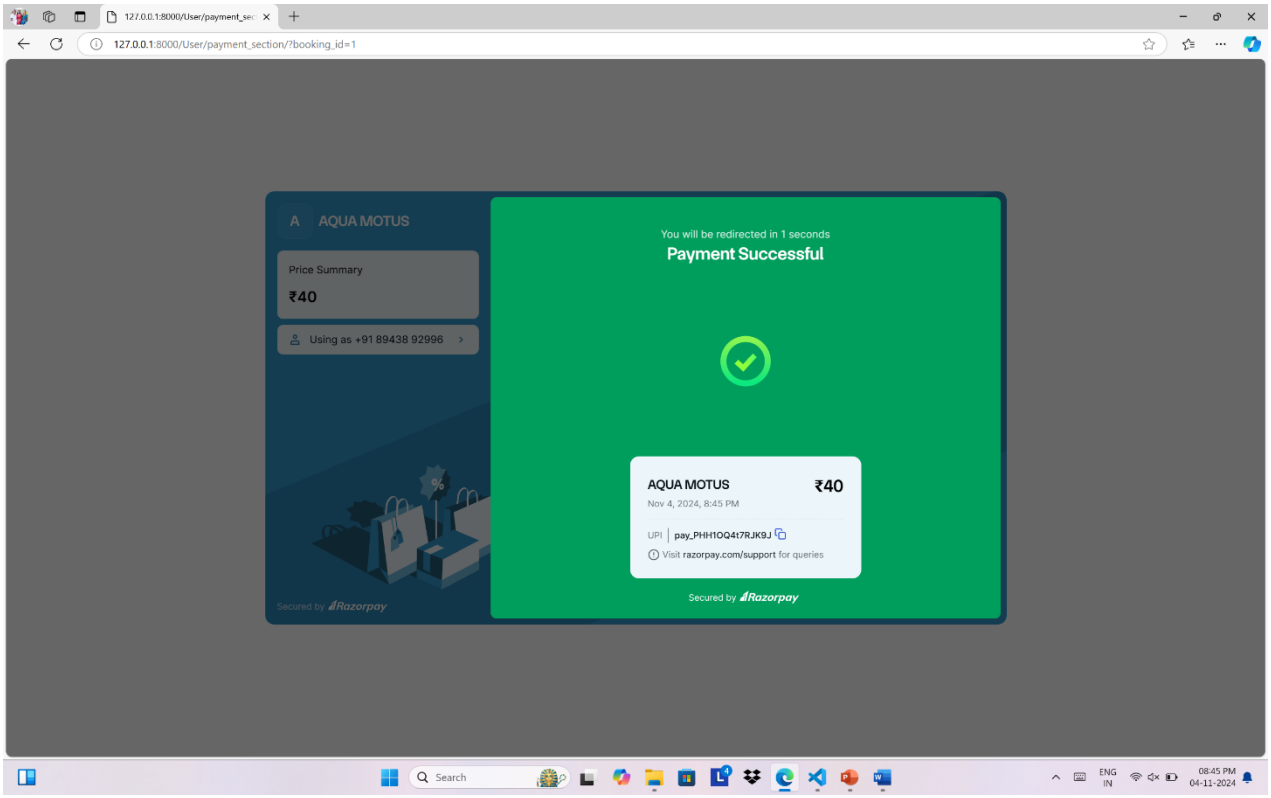

Figure 54: Payment

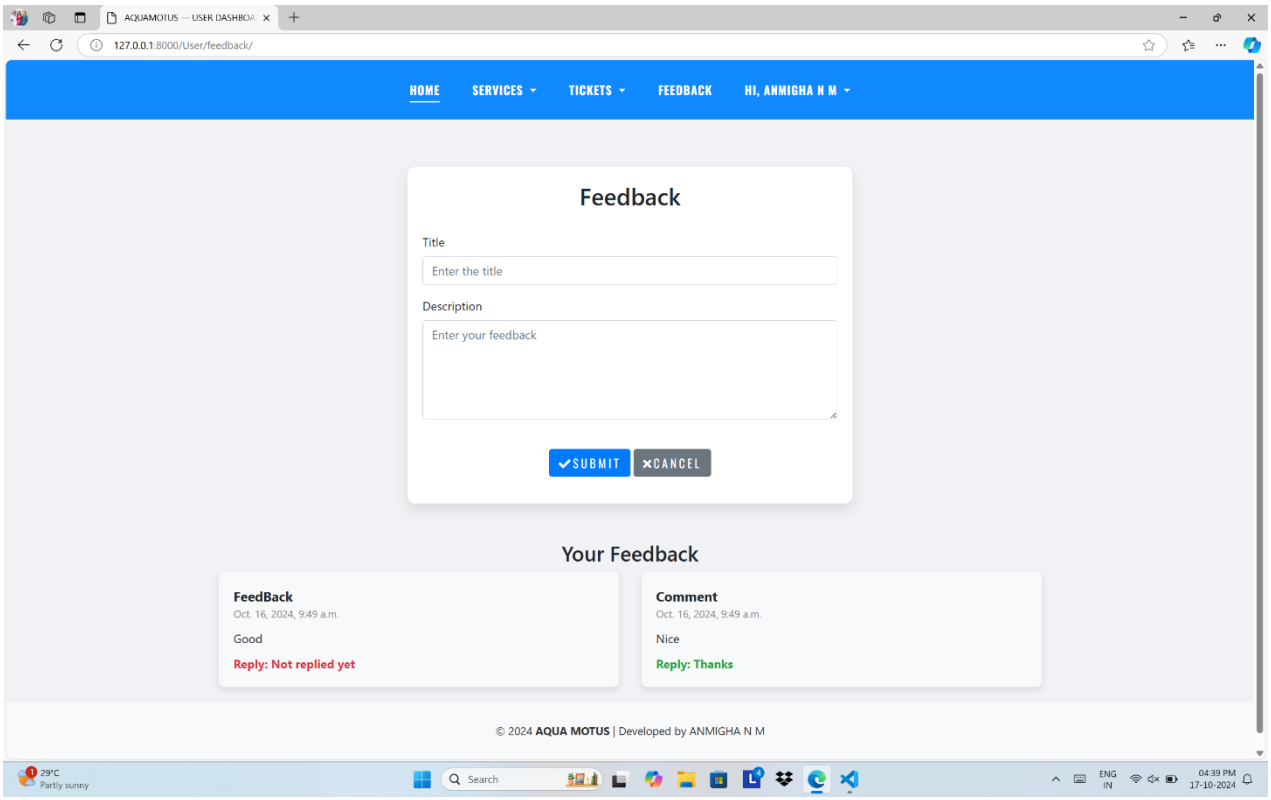

Figure 55: Feedback Page

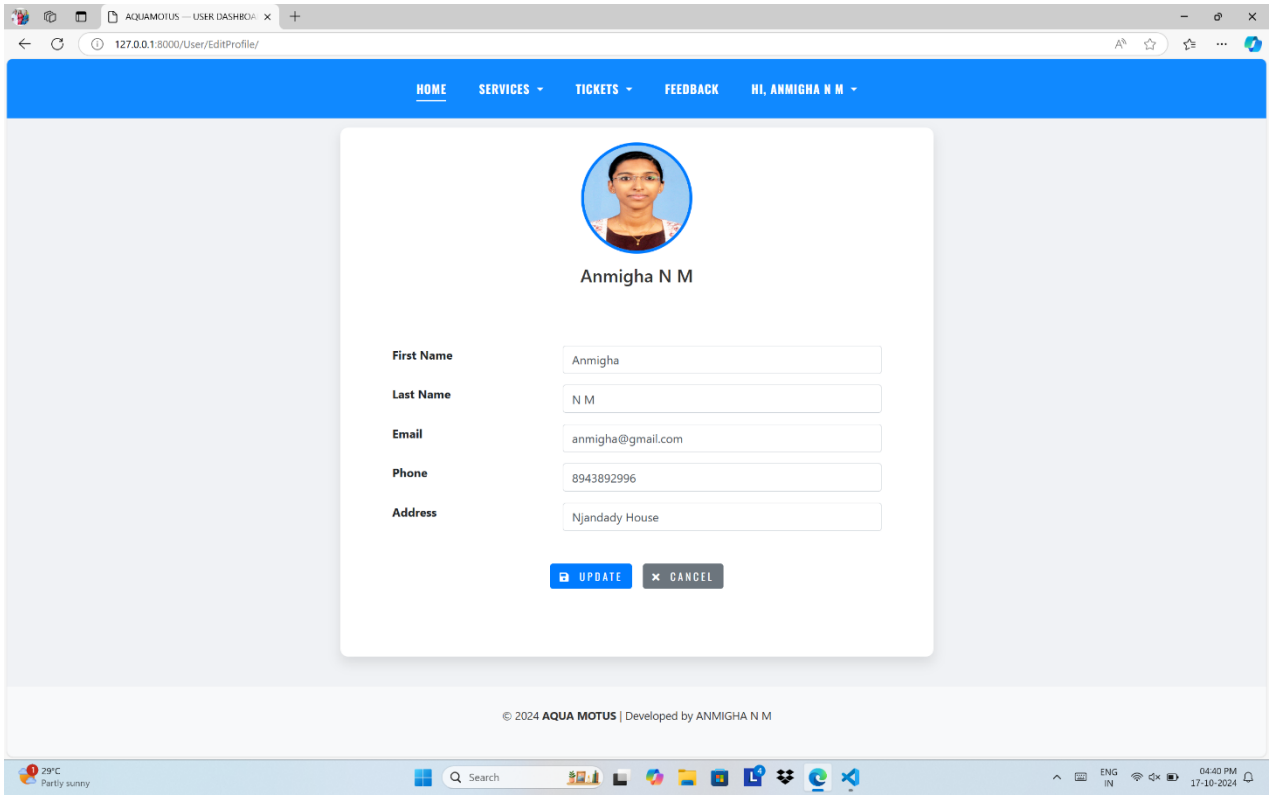

Figure 56: Profile Updation

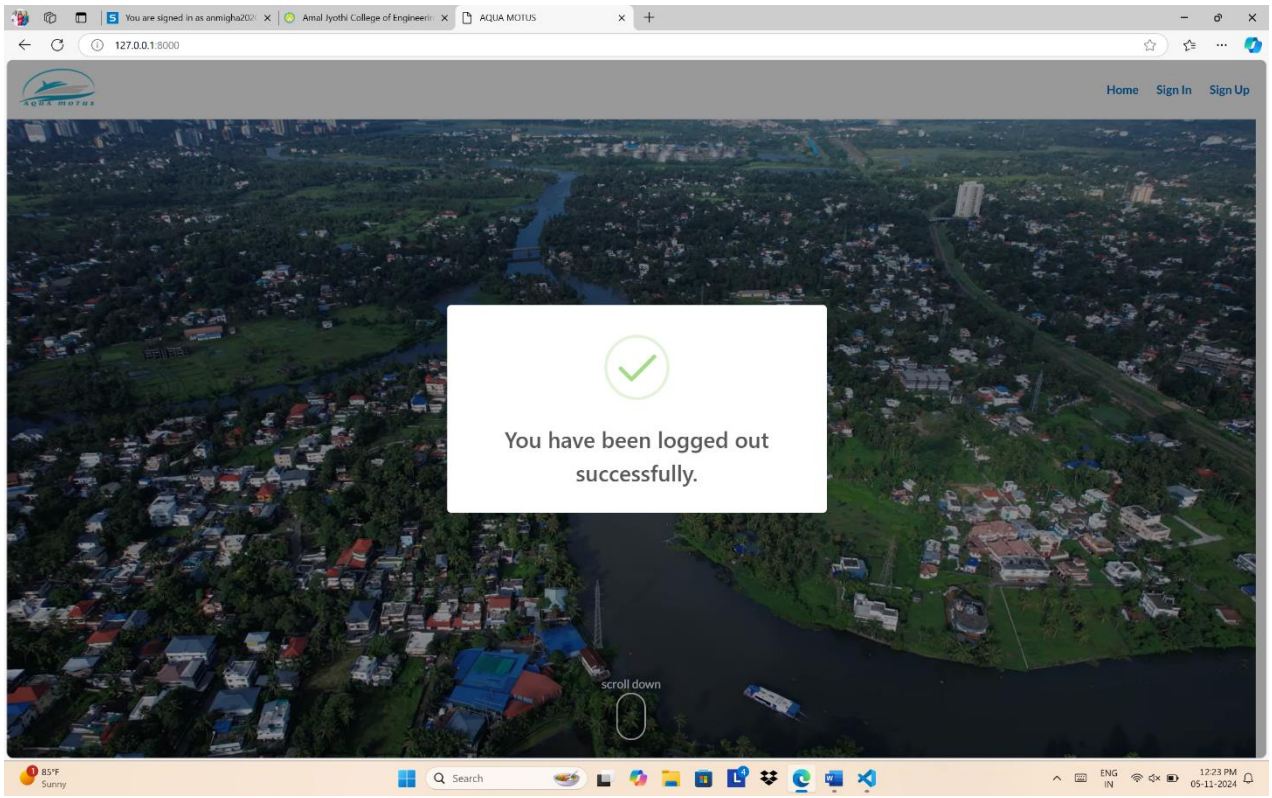

Figure 57: Logout

9.3 GIT LOG

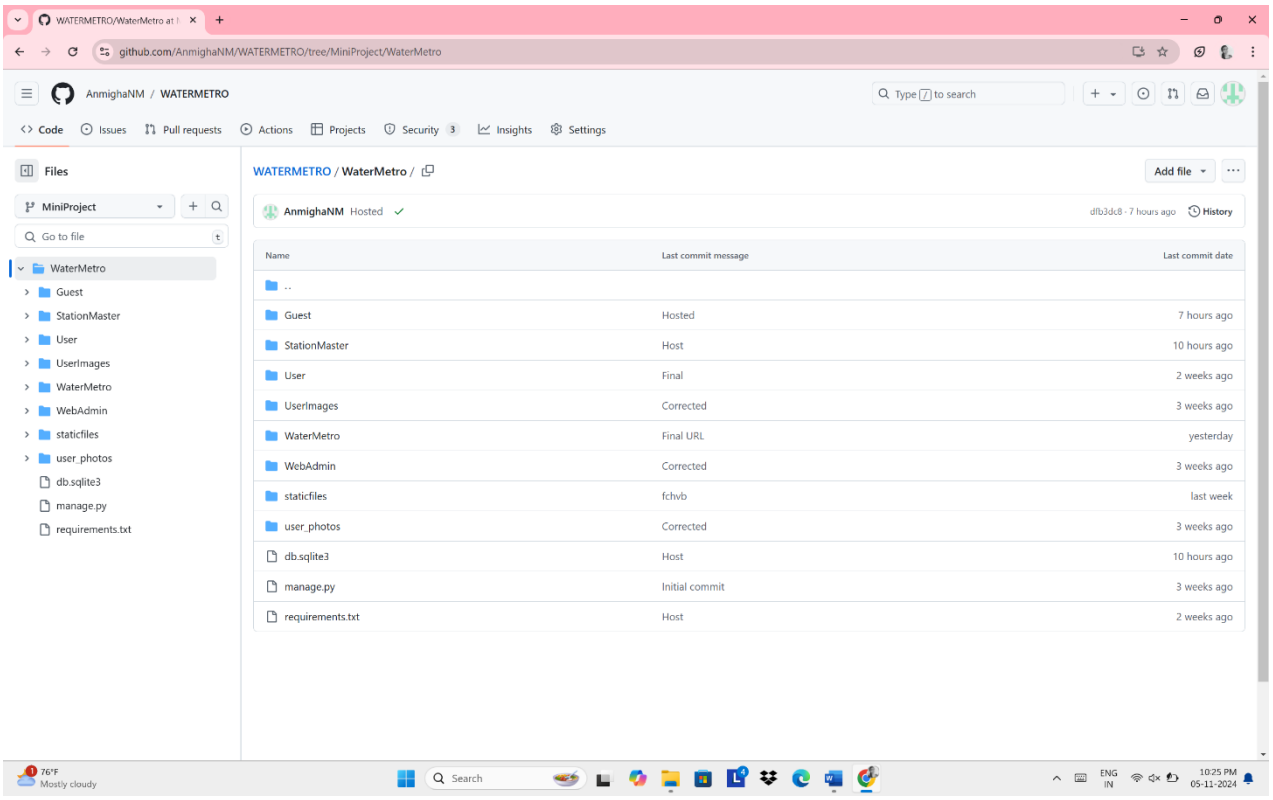

Figure 58: GitHub

AnmighaNM / WATERMETRO

Code Issues Pull requests Actions Projects Security 3 Insigh

Commits

MiniProject All users All time

Commits on Nov 5, 2024

Hosted

dfb3dc8

AnmighaNM committed yesterday · ✓ 1 / 1

Host

22cbdd2

AnmighaNM committed yesterday · ✓ 1 / 1

Commits on Nov 4, 2024

last

4a4e9b4

AnmighaNM committed 2 days ago · ✓ 1 / 1

Final URL

28a2d4c

AnmighaNM committed 2 days ago · ✓ 1 / 1

Commits on Oct 29, 2024

Final

b7145dd

AnmighaNM committed last week · ✓ 1 / 1

aaa

68e0cb8

AnmighaNM committed last week · ✓ 1 / 1

fchvb

52635a7

AnmighaNM committed last week

change

`2efc060`

AnmighaNM committed last week

aaaaa

`a1a1650`

AnmighaNM committed last week

host settings

`0a70357`

AnmighaNM committed last week

Commits on Oct 25, 2024

Host update

`c841b8f`

AnmighaNM committed 2 weeks ago

Hosted

`c7b2ee7`

AnmighaNM committed 2 weeks ago

Hosting

`27f5436`

AnmighaNM committed 2 weeks ago

Hosting

`4c5d44c`

AnmighaNM committed 2 weeks ago · ✕ 0 / 1

Host

`d198003`

AnmighaNM committed 2 weeks ago

Commits on Oct 19, 2024

Final

`63255aa`

AnmighaNM committed 3 weeks ago

Commits on Oct 17, 2024

Corrected

0c32124

AnmighaNM committed 3 weeks ago

Commits on Oct 15, 2024

validation

ac989e3

AnmighaNM committed 3 weeks ago

Initial commit

eb36b7f

AnmighaNM committed 3 weeks ago

Commits on Oct 12, 2024

Initial commit

ca38f81

AnmighaNM committed last month