School of
Computing Science

Clyde Ryde: An E-Vehicle Share System
LB03–04

Bairui Zhou
Chaoyue Wang
Siddhartha Pratim Dutta
Siwei Chen
Tingyu Zhou
Zhiqi Gao
Zhiying He

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

11th November 2024

Abstract

The Clyde Ryde project is a web application developed to fulfil coursework requirements for Programming & Systems Development, designed as an e-vehicle sharing platform. Built with Django, the application enables customers to rent electric vehicles on demand, prioritizing user-friendliness with features like location tracking and a wallet-based payment system. For operators, Clyde Ryde provides tools to monitor and manage vehicles, while data visualizations offer managers insights into usage patterns of the application. This report details the project's background, requirements analysis, design, implementation, and evaluation, highlighting Clyde Ryde as a functional platform within the shared mobility ecosystem.

Acknowledgements

We would like to express our sincere gratitude to our professors – Dr. Mireilla Bikanga Ada, Dr. Mary Ellen Foster, and Dr. Kevin Bryson – for introducing us to this project, providing foundational knowledge, and guiding us as we explored web application development with the Django technical stack. We extend our appreciation to the graduate teaching assistants, whose support and guidance were invaluable in clarifying our doubts and helping us overcome challenges along the way. Additionally, we are grateful to our classmates, whose discussions and insights helped us better understand project requirements and expectations.

We would also like to acknowledge the assistance of large language models (LLMs) throughout our development process, which supported us in generating ideas, refining design decisions, translating files, and debugging code. Finally, we appreciate the circumstances that brought our team together, allowing us to collaborate and learn from each other throughout this project.

Contents

Chapter 1 Introduction

1.1 Background and Motivation

As urban populations continue to grow, cities face mounting challenges in offering efficient options for short-distance transportation. The rise of the sharing economy has encouraged innovative solutions, with companies like Beryl and nextbike leading urban mobility through shared electric scooters and bikes. These platforms underscore a rising demand for quick, flexible transit options that lessen dependence on private cars, thus helping to reduce urban congestion and pollution.

The Clyde Ryde project was developed to address these urban needs by creating a rental platform specifically for electric vehicles. Unlike traditional public transport, often limited by routes and schedules, Clyde Ryde empowers users to rent nearby vehicles on demand, providing a flexible alternative for city travel. Through a focus on electric vehicles, Clyde Ryde also aligns with ongoing efforts to support cleaner and more sustainable urban transportation solutions.

1.2 Project Goals & Key Features

The Clyde Ryde project centers on three key goals: convenience, accessibility, and operational efficiency.

- **Convenience and Accessibility**: Clyde Ryde allows customers to locate, rent, and return e-vehicles at any available station, bypassing the need for fixed stops or waiting periods. The platform offers a streamlined, minimalistic interface, with an integrated wallet for easy payments and ride management.

- **Operational Efficiency**: For operators, Clyde Ryde simplifies vehicle tracking and maintenance, enabling seamless updates such as charging, repairs, and location changes. Additionally, the application provides management tools that facilitate data-driven decision-making on fleet distribution. Using data visualizations and analytics, managers can pinpoint high-demand areas and adjust vehicle availability, accordingly, thus maximizing operational efficiency.

Key Features: Built with Django and Bootstrap, the Clyde Ryde platform allows customers to check vehicle availability, manage rentals, and handle payments with an embedded payment-system for a seamless experience. Clyde Ryde allows operators to track and manage vehicles. For managers, data visualization tools provide insights various metrics such as – revenue, usage patterns, resource availability etc.

In summary, Clyde Ryde demonstrates a versatile and user-friendly e-vehicle sharing system prototype, designed to explore solutions for urban mobility that prioritize convenience, flexibility, and environmental sustainability.

Chapter 2 Background Survey

2.1 Literature Review

2.1.1 Information Technology Adoption

The shared vehicle rental system benefits from advanced technologies, including GPS positioning, Internet of Things (IoT), mobile payment etc. The adoption of these technologies has greatly improved the convenience and user experience of rental systems. Venkatesh et al. proposed a unified theory of acceptance (UTAUT) to explore the applicability of the technology acceptance model in practical applications [1]. Research on information technology adoption shows that users' acceptance and use of technology are affected by many factors, such as perceived usefulness, perceived ease of use, social influence, and the convenience of technology. Based on the Technology Acceptance Model (TAM) [2], many studies in recent years have focused on how applications can simplify and optimize user experience, especially in the context of the sharing economy, such as shared vehicle rental systems.

2.1.2 Sharing Economy

The concept of sharing economy has been widely accepted and applied in different fields, and shared vehicles are a typical example. Belk (2014) pointed out that the sharing economy is essentially a resource sharing behaviour achieved through a platform, thereby reducing social resource waste and improving resource utilization [3]. Shared e-vehicles not only conform to this concept, but also further promote the green transformation of urban transportation.

2.1.3 People's Sedentary Lifestyle

Contemporary societies are facing widespread health problems, especially rising risks of obesity and cardiovascular disease due to urbanization and sedentary lifestyles. According to the World Health Organization, in 2022, nearly two-thirds of adults and one-third of children in the region were overweight or obese. Physical inactivity and obesity are important risk factors for non-communicable diseases in the region, accounting for nearly 86% of deaths and 77% of the disease burden [4].

2.1.4 Environmental Pollution Issues

Global environmental pollution problems are becoming increasingly serious, especially in big cities. Pollution sources such as vehicle exhaust emissions and industrial waste gas pose a serious threat to air quality and residents' living environment. The United Nations Environment Program (UNEP) points out that carbon emissions from transportation are one of the main factors leading to the increase in greenhouse gases [5]. Fishman et al. (2015) pointed out in their research that promoting shared bicycles can reduce car travel, thereby reducing carbon dioxide emissions by about 200,000 tons per year [6]. Hence, shared e-vehicles, as a green travel method, can significantly reduce carbon emissions.

2.2 Analysis of Existing Systems

2.2.1 Beryl

Beryl was founded in 2012. It was originally a bicycle manufacturer and later transformed into a company that provides shared bicycle services [7]. Beryl adopts a dockless rental model. Users can find available bicycles nearby through Beryl's mobile app without returning the bicycles to specific parking piles. In addition to traditional shared bicycles, Beryl also provides electric scooters and electric bicycles to meet the needs of different users and improve the choice of urban travel.

Beryl's mobile application design provides an intuitive user experience. The main screen displays a city map, and users can quickly view available vehicles nearby, and get recommended cycling routes by setting the starting location and destination. After the user selects a vehicle, it can be unlocked by scanning a QR code or entering the vehicle number.

The "Ride Record" tab displays the user's historical riding data, through which users can analyse their riding habits. Users can manage personal information, including payment methods, account settings, and use coupons. Beryl's app provides user support options in the Help section where users can find answers to common questions or contact customer service via online chat, phone and email.

2.2.2 nextbike

nextbike is an international bike-sharing rental company founded in 2004 and headquartered in Leipzig, Germany [8]. It focuses on providing convenient urban short-distance travel solutions, aiming to encourage people to choose cycling to travel, reduce traffic congestion and environmental pollution. The company operates in many countries and regions around the world, providing flexible bicycle rental services.

nextbike combines the two rental modes of dockless and docked. Users can find available bicycles nearby through the app without returning to fixed parking piles. nextbike offers a variety of bicycle types, including regular bicycles and electric bicycles, to meet the travel needs of different users.

nextbike's mobile application is designed to be simple and easy to use. The application provides real-time bicycle location, rental status and cost information to ensure that users are always aware of their usage. Users can unlock the bicycle by simply scanning the QR code or entering the number, and the entire rental process is efficient and convenient. After the ride, users can park the bicycle in a compliant area and the system will automatically end the rental. And the application provides user riding records and statistics, and users can view riding distance, time and cost. This feature enhances user participation. Users are also encouraged to provide feedback on their riding experience, and they can rate and comment on the app. This feedback mechanism helps nextbike improve its services and increase user satisfaction.

Chapter 3 Requirements Specification

Requirements analysis is a critical stage in planning the development of Clyde Ryde, a shared electric vehicle rental system. This process ensures a thorough understanding of the components necessary for the system's success, guiding subsequent design decisions. In this section, we describe the methods used to identify system requirements, outline the functional requirements, present a prioritized list using the MoSCoW framework, and specify the non-functional requirements expected to be met by the system.

3.1 Elicitation Process

The requirements for Clyde Ryde were primarily established by the course instructors, setting a foundation for the core functionalities expected within the project. Our survey of existing e-vehicle rental systems allowed us to build upon these foundational requirements by identifying common features across established platforms. This exploration enabled us to expand the project scope with additional non-functional requirements, focusing on usability and user experience. These enhancements aim to deliver a cohesive user experience, aligning Clyde Ryde with the usability standards and operational functionalities typical of leading e-vehicle sharing platforms.

3.2 Functional Requirements

Customer Functionalities

1. The system shall allow customers to view available rental locations across the city.
2. The system shall allow customers to rent a vehicle from any location with available, working vehicles.
3. The system shall allow customers to return rented vehicles at any location within the city.
4. The system shall display trip details to customers, including vehicle type and rental duration, upon the vehicle's return.
5. The system shall allow customers to report a vehicle as defective.
6. The system shall charge the customer's account based on trip details when the vehicle is returned.
7. The system shall provide a mechanism for customers to pay any outstanding charges.

Operator Functionalities

8. The system shall allow operators to track the location and view the status of all vehicles.
9. The system shall allow operators to view and update the charging status of any vehicle with depleted battery levels.
10. The system shall allow operators to update the working condition (defective status) of any vehicle.
11. The system shall allow operators to relocate any vehicle as needed.

Manager Functionalities

12. The system shall allow managers to view application metrics, including the total number of rentals, revenue, and vehicle status.
13. The system shall provide managers with data visualizations, such as graphs and charts, based on system data.

3.3 MoSCoW Analysis

The MoSCoW prioritization technique [9] was applied to categorize Clyde Ryde's requirements into four groups: Must have, Should have, Could have, and Won't have. This framework highlights essential features needed for the initial launch while allowing flexibility for enhancements in future phases. The complete prioritized requirements are provided in Table A-1.

3.4 Non-Functional Requirements

To meet user expectations and maintain operational efficiency, several non-functional requirements were defined to ensure Clyde Ryde operates reliably and securely under various conditions:

1. **Performance:** The system shall respond to core actions within 2 seconds, support concurrent users without significant performance degradation, and generate reports within 10 seconds for data spanning at most six months.
2. **Scalability:** The system shall scale to accommodate user growth and allow integration of new vehicle types and locations with minimal code modifications.
3. **Availability:** The system shall maintain 99.9% uptime, ensuring constant availability except during scheduled maintenance.
4. **Security:** The system shall enforce role-based access control and strong password policies to safeguard access.
5. **Usability:** The system shall offer an intuitive, Bootstrap-based interface compatible with modern browsers, including Chrome, Firefox, Safari, and Edge.
6. **Maintainability:** The system shall follow a modular architecture, adhering to Django and Bootstrap best practices to simplify updates and bug fixes.

3.5 Summary

By following this structured approach, Clyde Ryde aims to deliver a user-friendly and scalable electric vehicle sharing system tailored to modern urban commuting needs. The prioritization offers a clear roadmap for both initial development and potential future enhancements, providing a foundation for a reliable, secure, and accessible service. This comprehensive set of requirements is designed to ensure the system meets user expectations and supports an efficient, adaptable shared mobility experience.

Chapter 4 Design

The following chapter presents the high-level design of Clyde Ryde, outlining its key structural components and how they meet the identified requirements. Given the iterative nature of the project's development process, these design components reflect the results of continuous refinement and adjustments made throughout the implementation process.

4.1 System Architecture

Clyde Ryde is a web application that follows a general three-tier architecture [10] (see Figure B-1). The presentation layer handles user interactions through browser-based interfaces using HTML, CSS, and JavaScript templates rendered by Django. The application layer contains the core Django application using an MVC framework, managing requests and responses through controllers, views, and models, while handling authentication and role-based access.

A Redis cache layer optimizes performance for frequently accessed data, like homepage elements, while the database layer uses PostgreSQL to store all persistent data, including users, locations, vehicles, and transactions. This tiered architecture ensures clear separation between user interfaces, business logic, and data management, enabling the system to manage high-level interactions and complex data relations while remaining maintainable and extensible for future enhancements.

4.2 Data Model

Clyde Ryde's data model is built around several key entities and their relationships (see Figure B-2). At its core, there is a user management system where users are assigned specific roles (customer, operator, or manager). Each customer has exactly one wallet (one-to-one), while a single customer can have multiple trips (one-to-many).

The vehicle rental system is based on relationships between vehicles, vehicle types, and locations. Each vehicle belongs to exactly one vehicle type (many-to-one) – storing attributes like model and rental rate, and is associated with one location at a time (many-to-one) – defined by latitude and longitude. The trip management system ties everything together through a trip table, where each trip must reference one user, one vehicle, and two locations (start and end points) through many-to-one relationships. Each trip has exactly one corresponding payment record (one-to-one) that tracks payment amount and status. This structure enables complete tracking of rental operations from rentals to payments, while maintaining data consistency across all operations.

4.3 Use Case Analysis

The use case diagram (see Figure B-3) illustrates three main actor types and their system interactions with Clyde Ryde through connected use cases. Customers have the most direct interaction with the application, following a

clear flow from viewing locations to completing payments. Their use cases form a linear rental journey: view locations → view vehicles → sort/filter → rent → view trip details → return → payment, with parallel paths for wallet management: view wallet → top-up and trip history viewing.

Operators focus on vehicle management and maintenance tasks through interconnected use cases branching from "View All Vehicles" and "View Vehicle Details". From these central points, they can access more specific actions like charging, repairs, and location changes. Managers have a streamlined set of use cases centred around data analysis, with "View Visualizations" connecting to both reporting and filtering capabilities. Their use cases are notably fewer but more data-focused than the other roles.

The interactions create a hierarchical structure where customers handle rental operations, operators manage the fleet, and managers have an overview of the usage of the entire system.

4.4 Interaction Design

The sequence diagram presents the six principal interaction flows within the Clyde Ryde system (see Figure B-4), illustrating communication between actors, the Django application server organized into discrete applications, and the PostgreSQL database. Customer-related flows are the most intricate, beginning with “View Locations,” where a request triggers customer localization, followed by vehicle filtering from the database. This sequence continues through “Rent Vehicle,” where the customer selects a vehicle, updating its status and creating a new trip, and concludes with “Trip Detail,” which encapsulates a complex payment process involving status updates, payment computation, record creation, and updates to both wallet and payment status.

For staff interactions, operators engage in two primary flows: “View Vehicles,” executed via a straightforward query-response mechanism for vehicle listings, and “Update Vehicle,” which handles single update requests for actions like charging, repairs, or location changes. The manager’s “View Reports” flow is distinctively structured, beginning with filtering queries, advancing to aggregation, and culminating in data processing steps prior to final data visualization.

4.5 Summary

This chapter outlined Clyde Ryde’s system architecture, data model, use case analysis, and interaction design. The system architecture described a layered approach with presentation, application, cache, and data layers for efficient user interaction and data processing. The data model highlighted key entities such as users, vehicles, locations, and trips, detailing their relationships and roles in trip management and rentals. The use case analysis examined the interactions of customers, operators, and managers, clarifying their specific actions within the system. Finally, the interaction design was illustrated through a sequence diagram, depicting the flow of interactions for vehicle rental and trip management. Overall, this chapter serves as a foundational reference for understanding Clyde Ryde’s system design and functionality.

Chapter 5 Implementation

This chapter delves into the core of the Clyde Ryde project: the development process. It covers the methodologies and technologies chosen to guide efficient and scalable development, along with insights into key features implemented to meet user needs across customer, operator, and manager functionalities. Additional technical complexities and feature implementations that are broader than the scope of the initial requirements are indicated with a 📌 emoticon.

5.1 Software Development Practices

1. **Project Planning:** In the Clyde Ryde project, effective planning was crucial to track progress across phases and achieve the project objectives. A Gantt chart (see Figure C-1) provided a visual timeline, helping us allocate time efficiently for planning, design, and development tasks, with a clear focus on feature implementation and testing. This tool helped maintain team alignment and ensured timely project completion.

2. **Development Methodology:** For the Clyde Ryde project, we adopted Kanban due to its adaptability and suitability for the constraints of a university project. Unlike the rigid Waterfall approach and the structured sprints of Scrum, Kanban allowed us to prioritize tasks dynamically and manage workflow in real-time. Using a Kanban board (see Figure C-2) with four columns – Backlog, Bugs, In Progress, and Done – we could track task progress visually [11]. This methodology not only fostered individual accountability but also enabled efficient teamwork, and supported continuous delivery, aligning well with the project's goals.

3. **Version Control:** Version control was essential in managing Clyde Ryde's collaborative development, allowing team members to work on features independently and ensuring smooth integration. We adopted a GitHub Flow-inspired strategy [12], using a main branch and creating feature branches for specific issues or functionalities (see Figure C-3). Changes were submitted as pull requests, reviewed for quality, and merged back into the main branch. Our initial release, v0.1, was used for acceptance testing, followed by the final v1.0 release after all validations. This approach maintained an organized and efficient workflow.

4. **Code Standards:** To maintain code quality in the Clyde Ryde project, we established consistent coding practices and employed tools for readability and collaboration. Comments focused on explaining design decisions, while type hints clarified input and output types, making the code easier for team members to understand and extend (see Snippet C-1). SonarLint was integrated into VS Code to catch common Python issues (see Figure C-4), and pre-commit hooks were set up to enforce style checks such as – trailing whitespace, file endings, Python formatting with Black and Flake8 (see Figure C-5) etc. For Django templates, djLint was used to maintain consistency in Jinja syntax. These standards fostered a clean, collaborative codebase that was reliable and easy to maintain.

5.2 Core Technologies

5.2.1 Frontend Client

The frontend client of the Clyde Ryde application delivers a responsive and cohesive user experience using HTML, CSS, Bootstrap for styling and structure and JavaScript for enhanced functionality. Bootstrap's utility classes simplify styling, while custom CSS enhances branding, as seen in the .btn-cr-light (see Snippet D-1) class, which extends Bootstrap's button styling to create a consistent Clyde Ryde colour scheme. Jinja templates provide modular and reusable layouts in Django, enabling consistency and reducing redundancy across pages (see Snippet D-2). Two external JavaScript libraries were integrated to improve functionality:

1. 📌 The **Google Maps JavaScript API** enhances interactivity by enabling customers to view rental locations through map markers, helping them easily identify nearby rental points (see Snippet D-3).
2. 📌 **Chart.js** provides data visualizations in manager reports, offering graphical insights into metrics like revenue and vehicle usage, improving data comprehension (see Snippet D-4).

5.2.2 Backend Application Server

The backend of the Clyde Ryde application is powered by Django, a Python-based web framework widely known for its rapid development capabilities, robust security features, and clear separation of concerns through the Model-View-Template (MVT) architecture. This structure aligns well with Clyde Ryde's needs, providing tools for database management, routing, and user authentication, enabling a focus on core feature implementation. To keep the codebase modular and organized, the project is divided into specific Django apps (see Figure D-1), each addressing a distinct area of the system – users (authentication), core (shared models), customers (trip management), operators (vehicle management) and managers (reports). This app-based structure kept development streamlined, enabling each module to be maintained independently.

Several Django features were implemented to enhance functionality and user experience:

1. 📌 **Internationalization (i18n)**: Leveraging Django's i18n framework, the project provides translations in French and Chinese, allowing users to switch languages seamlessly. This involved the maintenance of translation files (see Snippet D-5) with the help of large language models.
2. 📌 **Django Messages Framework**: This feature was used to deliver real-time feedback (see Snippet D-6), such as confirmation of rental trips or alerts when errors occur which helps enhance user interaction.
3. 📌 **Django Management Commands**: Management commands were developed to populate the database with test data, which simplified the process of creating testing environments. For instance, commands allowed us to add sample users, vehicles, and trips (see Figure D-2), ensuring consistent and readily available data across the development and testing phases.

5.2.3 Cache & Persistence

The Clyde Ryde project uses PostgreSQL as the primary database which is queried using the Django ORM while Redis is used as a cache, collectively ensuring fast and reliable data management.

1. 📌 **PostgreSQL** is an open-source relational database known for robustness and scalability. It provides transactional consistency, complex query support, and data integrity constraints, making it ideal for web applications with relational data. PostgreSQL also supports geospatial data handling with PostGIS, which future-proofs the application for location-based features [13] like finding nearby rental stations or calculating distances between points.
2. 📌 The **Django ORM** (Object-Relational Mapping) seamlessly bridges the application and the PostgreSQL database, enabling efficient data queries without the need for raw SQL. This allows developers to interact with the database using Python code, ensuring consistency, security, and simplified data handling. SQL queries can also be optimized using ORM attributes (see Snippet D-7) which improves system performance. Additionally, indices have been added to frequently queried fields to speed up search and sort operations, further enhancing responsiveness for users.
3. 📌 **Redis** serves as an in-memory cache, speeding up access to frequently used or computationally intensive data, which reduces the load on the main database and improves the user experience. Redis is initialized using a singleton class which also supports a fallback mechanism redirecting to Django's local memory (locmem) cache (see Snippet D-8). For example, data for homepage elements like vehicle types are frequently accessed but do not change often (see Snippet D-9) and hence, can be retrieved faster from Redis.

5.2.4 Containerization

The Clyde Ryde project is containerized using Docker, encapsulating the application and its dependencies for consistent behaviour across all environments. Docker allows each component to run in isolated containers, preventing compatibility issues and streamlining deployment.

📌 The setup includes services for the Django application, PostgreSQL database, and Redis cache, defined in the docker-compose.yml file (see Snippet D-10). The database and cache services are based on official images, while the web service uses a custom Dockerfile to install necessary dependencies and set up the Django application.

📌 The project's Dockerfile (see Snippet D-11) outlines key setup steps – it uses an official Python image as the base and installs required system dependencies. The project files are then copied into the container, following which the Python dependencies are installed and a custom entrypoint script handles database migrations, database seeding and application startup. This approach ensures that ClydeRyde is deployment-ready with minimal setup, offering a stable and efficient environment for both ongoing development and production use.

5.3 Feature Implementations

5.3.1 Customer Functionalities

Dashboard: The customer dashboard provides a table of trip details, including trip date, vehicle type, location, duration, charge, and status (see Figure E-1). 📌 To enhance the user experience, the dashboard is paginated, allowing customers to browse their trips efficiently without long load times.

Viewing Rental Locations: 📌 Customers can view available rental locations on a Google Map interface (see Figure E-2). Additionally, a "Locate Me" feature helps users find nearby rental points based on their current location.

Viewing Available Vehicles: Customers can see a list of available vehicles at each location (see Figure E-3) and sort them by factors such as fare or battery level. 📌 Filtering options by vehicle type improve the selection process, allowing customers to choose based on personal preferences and needs.

Trip Details: The trip detail page displays trip information (see Figure E-4) and provides essential actions, such as returning the vehicle, reporting issues, and completing pending payments. This feature consolidates trip-related tasks, ensuring efficient management within a single view.

Wallet: 📌 The built-in wallet allows customers to handle payments directly within the app. Upon completing a trip, the fare is automatically deducted from the wallet balance through an atomic transaction, maintaining database consistency (see Snippet D-12). If the wallet balance is insufficient, customers can top up and finalize any pending payments later (see Figure E-5).

Internationalization: 📌 The application supports French and Chinese, implemented using Django's internationalization framework. All customer-facing text is marked for translation with *trans* tags or *gettext_lazy* functions, ensuring seamless language-switching. Translation files were managed with the assistance of LLMs, facilitating a smooth multilingual experience for customers (see Figure E-6).

5.3.2 Operator Functionalities

Viewing Vehicles: Operators have access to a comprehensive, paginated list of all vehicles in a table format, enhancing ease of navigation. 📌 This view allows operators to filter vehicles by status and location (see Figure E-7), and to sort by attributes such as vehicle code, model, location, and battery levels in either ascending or descending order. This functionality provides quick access to detailed vehicle information, improving workflow efficiency for managing large datasets.

Managing Vehicles: The vehicle detail page equips operators with essential controls for managing vehicle status and location. Operators can mark vehicles as charged, repair defective units, or update a vehicle's location as needed (see Figure E-8). By centralizing these maintenance tasks, the system enables operators to swiftly ensure vehicle readiness and availability, optimizing the overall vehicle management process.

5.3.3 Manager Functionalities

Dashboard: 📌 The manager dashboard provides an aggregated overview of key application metrics, including the total number of users, total revenue, overall wallet balance, and total trips (see Figure E-9). These metrics are derived from complex aggregation queries, offering a quick snapshot of business performance.

Visualizations: Several visualizations provide deeper insights into operational metrics:

1. **Revenue Visualization**: Displays a dual-axis chart with a bar chart for daily revenue and a line chart for cumulative revenue, plotted against the date on the x-axis (see Figure E-10). This visualization helps in trend analysis by showing daily and cumulative revenue patterns.
2. **Location Popularity Heatmap**: Overlays a heatmap on vehicle location markers, representing the number of trips starting from each location, providing a visual indication of high-demand areas (see Figure E-11).
3. **Trip Duration Histogram**: Shows trip durations in interval-based bins along the x-axis, with the y-axis indicating the number of trips per interval (see Figure E-12). This helps managers analyze popular trip lengths.
4. **Vehicle Count Chart**: A stacked horizontal bar chart displays the counts of available and defective vehicles at each location (see Figure E-13), aiding in location-based inventory and maintenance management.
5. **Usage by Vehicle Type**: A pie chart displays trip counts by vehicle type (see Figure E-14), offering insights into vehicle popularity and usage patterns.

CSV Downloads: 📌 All visualizations include a CSV download option, allowing managers to retrieve and analyze the raw data behind each chart for deeper business insights.

5.4 Summary

This chapter covered the primary implementation components of the Clyde Ryde system, spanning development practices, core technologies, and key functionalities. Practical software development practices were adopted to ensure project efficiency, maintainability, and code quality, while core technologies were strategically chosen to support a robust architecture, including a Django backend, PostgreSQL database, Redis for caching, and Docker for containerization.

The feature implementation section detailed functionalities for customers, operators and managers, emphasizing user experience, streamlined fleet management, and data-driven insights. A video report demonstrating the use of Clyde Ryde is available here, while the source code is available as a GitLab repository. 📌 Table F–1 and Table F–2 highlight the additional features and technical complexities respectively, demonstrating the breadth and depth of the Clyde Ryde application's functionality.

Chapter 6 Evaluation

In this chapter, we evaluate the Clyde Ryde application by outlining the testing approach used to validate the system's reliability, efficiency, and alignment with functional and non-functional requirements. Testing is crucial to ensuring that each component functions as expected, contributing to an effective, user-friendly experience for each role: customer, operator, and manager.

Due to time constraints, we conducted manual testing instead of automated unit or integration testing, focusing on acceptance testing to evaluate whether each user action fulfilled its success criteria. This enabled us to prioritize core functionality while ensuring that the system met key performance standards.

6.1 Acceptance Testing

Acceptance testing served as the primary method for assessing the system's behaviour and the success of specific user actions. This method, associated with Black Box Testing, involves verifying outputs based on inputs without analysing the underlying code. Our acceptance test cases (see Table G–1) focused on real-world scenarios that simulated user interactions with key features, such as vehicle rental, payment processing, vehicle management and report generation.

This approach allowed us to verify that each feature met functional requirements and identify any areas for improvement. By using scenarios directly tied to functional criteria, we were able to determine each feature's success based on anticipated outcomes.

6.2 Performance Testing

In addition to verifying functionality, we conducted a light assessment of database performance. Using Kolo [14], we traced Django requests to visualize query patterns and detect inefficiencies. Initial tracing (see Figure G-1) revealed several scopes for improvements primarily arising from the N+1 problem of SQL queries [15], impacting response times. By optimizing the code to use database joins via the ORM, query counts were reduced efficiently (see Figure G-2). This performance testing helped us make enhancements that are critical for maintaining a responsive user experience, particularly in a data-intensive application like Clyde Ryde.

6.3 Summary

The testing process confirmed that the Clyde Ryde application met its functional requirements, with all core features performing as expected for customer, operator, and manager roles. Manual acceptance testing validated each use case, ensuring usability and reliability by comparing actual outcomes to expected results. Additionally, our database query optimizations helped verify that Clyde Ryde delivers a performant and intuitive experience, meeting the non-functional requirements across its key features and interactions.

Chapter 7 Conclusion

7.1 Summary

The Clyde Ryde system was developed to address the unique needs of customers, operators, and managers within an e-vehicle sharing platform. For customers, it provides essential functionality to view, rent, and return electric vehicles with ease, along with a seamless and user-friendly payment system. Operators are equipped with tools to monitor vehicle statuses and locations, enabling effective fleet management and maintenance. Managers benefit from reporting metrics and data visualizations that allow them to analyze vehicle usage patterns and optimize resource allocation.

The system was also built with non-functional requirements in mind to ensure it remains responsive, scalable, and secure as usage grows. With a design centered on user experience, Clyde Ryde offers an intuitive and reliable platform that effectively meets the demands of all roles involved.

7.2 Future Improvements

1. **User Rating and Feedback System**: Introducing a feedback mechanism where customers rate vehicles and share their rental experience could improve customer satisfaction and assist operators in prioritizing vehicle maintenance.
2. **Gamification and Incentive Programs**: Adding rewards for frequent rentals, eco-friendly behavior, and referrals could drive engagement, promote sustainable usage, and foster community-based usage.
3. **Dynamic Pricing Models**: Demand-based pricing that adjusts based on peak times and vehicle availability would optimize resources, improve revenue, and increase vehicle access during busy periods.
4. **Enhanced Reporting Features**: Advanced filtering and data comparison options in reporting would enable managers to analyze trends more deeply and improve resource allocation.
5. **Automated Testing**: Adding automated unit and integration tests would enhance efficiency and ensure consistent performance, reducing maintenance effort over time.
6. **Improved User Interaction Design**: While the current UI meets basic requirements, adding real-time feedback features, such as live vehicle status updates, would further enhance the user experience.
7. **Strengthened Data Security**: Expanding security with data encryption and two-factor authentication would reinforce user privacy and data protection, especially for payment related flows.

These improvements would further refine the Clyde Ryde system, supporting continued growth, enhancing user experience, and maintaining high standards of performance, security, and scalability.

Chapter 8 Contributions

The success of the Clyde Ryde project is a testament to the collaborative efforts and diverse skills of the entire development team. Each member played a vital role in bringing the application to life, contributing unique perspectives and expertise across all phases of development. From the initial requirements analysis and system design to the implementation of the Django and PostgreSQL stack, every contribution was essential in shaping the final product. This collaborative environment fostered a culture of learning, where everyone could develop their skills while contributing to project goals. The table below highlights the main tasks and the team members that were primarily responsible for them.

Task	Team Member(s)
Project Planning	Tingyu Zhou, Chaoyue Wang
Database Modelling	Siwei Chen, Zhiqi Gao
System Design	Siddhartha Pratim Dutta, Zhiying He
UI Design & Frontend	Bairui Zhou, Chaoyue Wang
Customer Features	Siwei Chen, Zhiying He
Operator Features	Tingyu Zhou, Zhiqi Gao
Manager Features	Siddhartha Pratim Dutta
Acceptance Testing	Bairui Zhou
Introduction & Report Compilation	Zhiqi Gao
Background Survey	Chaoyue Wang
Requirements Specification	Tingyu Zhou
Design	Siddhartha Pratim Dutta
Implementation	Siddhartha Pratim Dutta, Siwei Chen, Zhiying He
Evaluation	Siddhartha Pratim Dutta, Bairui Zhou
Conclusion	Bairui Zhou

References

[1] V. Venkatesh, M. G. Morris, G. B. Davis and F. D. Davis, "User Acceptance of Information Technology: Toward a Unified View," *MIS Quarterly,* vol. 27, no. 3, pp. 425-478, 2003.

[2] F. D. Davis, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology," *MIS Quarterly,* vol. 13, no. 3, pp. 319-340, 1989.

[3] R. Belk, "You are what you can access: Sharing and collaborative consumption online," *Journal of Business Research,* vol. 67, no. 8, pp. 1595-1600, 2014.

[4] World Health Organization, "Cycling and walking can help reduce physical inactivity and air pollution, save lives and mitigate climate change," World Health Organization, 07 June 2022. [Online]. Available: https://www.who.int/europe/news/item/07-06-2022-cycling-and-walking-can-help-reduce-physical-inactivity-and-air-pollution--save-lives-and-mitigate-climate-change.

[5] UN Environment, "The Emissions Gap Report 2010," UN Environment Programme, 2010. [Online]. Available: https://www.unep.org/resources/emissions-gap-report-2010.

[6] E. Fishman, S. Washington and N. Haworth, "Bike share's impact on car use: Evidence from the United States, Great Britain, and Australia," *Journal of Transport & Health,* vol. 31, no. 2, pp. 13-20, 2014.

[7] "Beryl," [Online]. Available: https://beryl.cc/our-story.

[8] "nextbike Company Profile," [Online]. Available: https://websites.nextbike.net/media/nextbike_companyprofile_2016_screen.pdf.

[9] P. Singh and R. Kushwaha, "Envisaging Indian farmers' desires from agricultural index insurance integrating rank sum weighting method and MoSCoW technique: an approach to requirements prioritization," *International Journal of Social Economics,* 2024.

[10] Amazon Web Services, "WS Serverless Multi-Tier Architectures with Amazon API Gateway and AWS Lambda," Amazon Web Services, 20 October 2021. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/web-application.html.

[11] W. Zayat and O. Senvar, "Framework Study for Agile Software Development Via Scrum and Kanban," *International Journal of Innovation and Technology Management,* vol. 17, no. 04, 2020.

[12] GitHub, "GitHub Flow," GitHub Docs, [Online]. Available: https://docs.github.com/en/get-started/using-github/github-flow.

[13] P. Ramsey and V.-B. Columbia, "Introduction to postgis.," *Refractions Research Inc,* pp. 34-35, 2005.

[14] Kolo, "How to: Trace Django Requests," [Online]. Available: https://docs.kolo.app/howto/trace-django-requests.

[15] D. Colley and D. C. Stanier, "Identifying New Directions in Database Performance Tuning," *Procedia Computer Science,* vol. 121, pp. 260-265, 2017.

Appendix A Requirements Analysis

MoSCoW Prioritization		
Must Have: critical to the project		
Title	Description	User
View Locations	View a list of all available locations	Customer
View Vehicles	View a list of all available vehicles at a location	Customer
Rent Vehicle	Rent any available vehicle from a location	Customer
Return Vehicle	Return a vehicle to any location	Customer
Report Vehicle	Report any rented vehicle	Customer
Trip Payment	Pay any outstanding trip charges	Customer
View Vehicles	View all vehicles and their status, location	Operator
Update Vehicle	Update battery level or status of any vehicle	Operator
Vehicle Relocation	Change the location of any vehicle	Operator
View Visualizations	View data visualizations over defined time periods	Manager
Should Have: significant value but not vital		
Title	Description	User
Map Integration	View available locations on a map interface	Customer
Sort & Filter	Customized view of available vehicles	Customer
In-App Wallet	App based wallet to handle trip payments	Customer
Search & Filter	Filter vehicles by a search criteria e.g. location	Operator
Export Data	Export selected data in alternative formats	Manager
Could Have: desirable, but small impact if left out		
Title	Description	User
Customer Tiering	Different charges / discounts, based on membership	Customer
Vehicle Reservation	Reserve vehicles for future rentals	Customer
Internationalization	Support for multiple application languages	Customer
Vehicle Addition	Add additional vehicle types	Operator
Vehicle Audit Log	Persist changelog for vehicle updates	Operator
Dynamic Reports	Customize visualization type based on data	Manager
Will not Have: not a priority for the specific time frame		
Title	Description	User
Gamification	Achievement tracking for customer activity	Customer
Social Integration	Share rental activity via social media	Customer
Live Tracking	Live GPS for vehicles that are currently rented	Operator
Consolidated Report	Summarized report indicating business health	Manager

Table A-1: MoSCoW Prioritization of Functional Requirements

Appendix B Design Diagrams

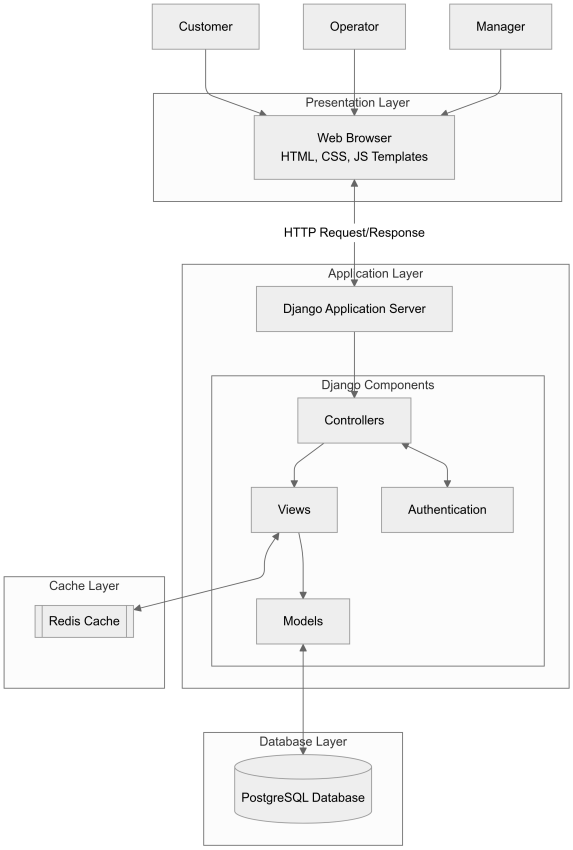

Figure B-1: System Architecture of the Clyde Ryde Application

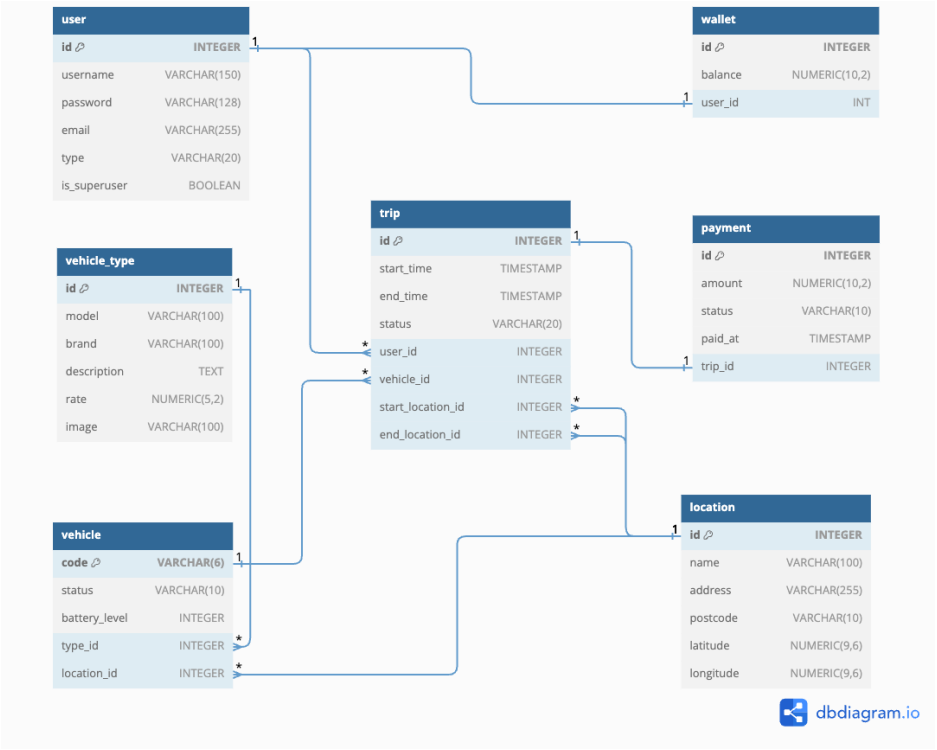

Figure B-2: Entity Relationship Diagram of the Clyde Ryde Database

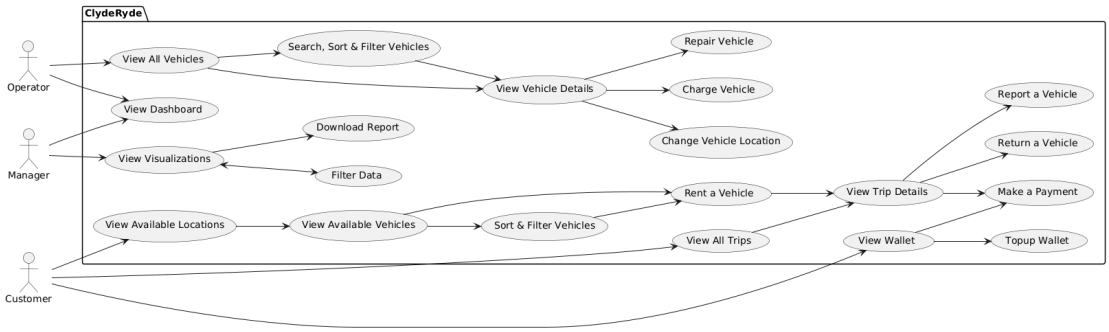

Figure B-3: Use Case Diagram of Clyde Ryde Users

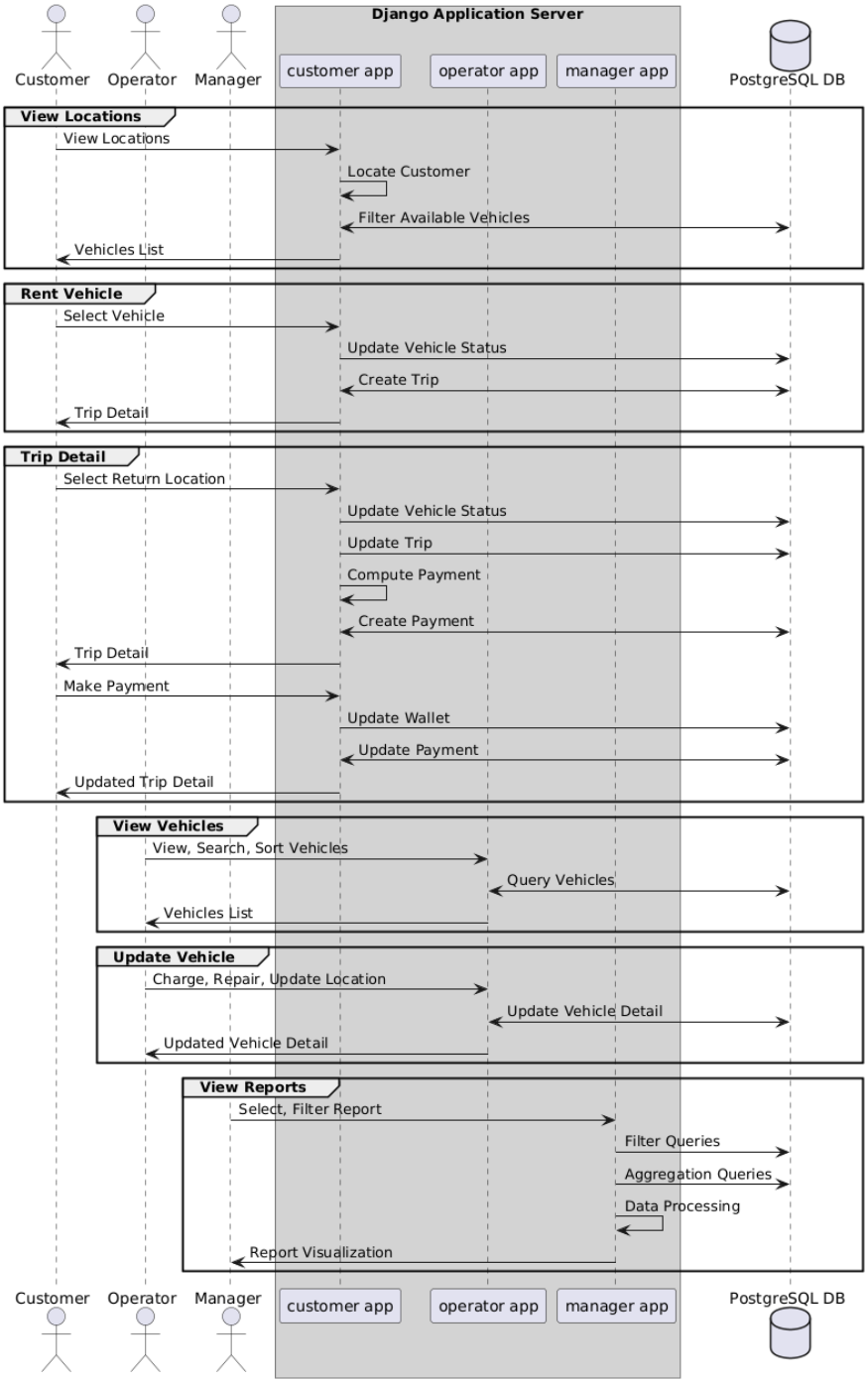

Figure B-4: Sequence Diagram of Clyde Ryde Interactions

Appendix C Project Methodology Resources

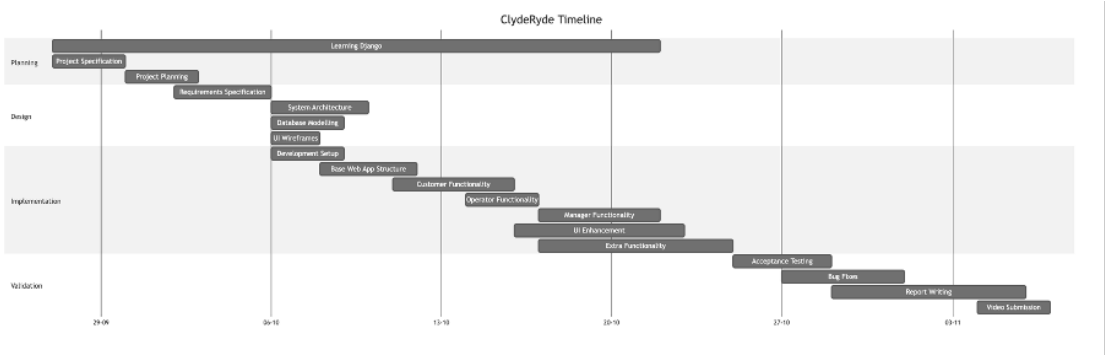

Figure C-1: Gantt Chart for the Project Timeline

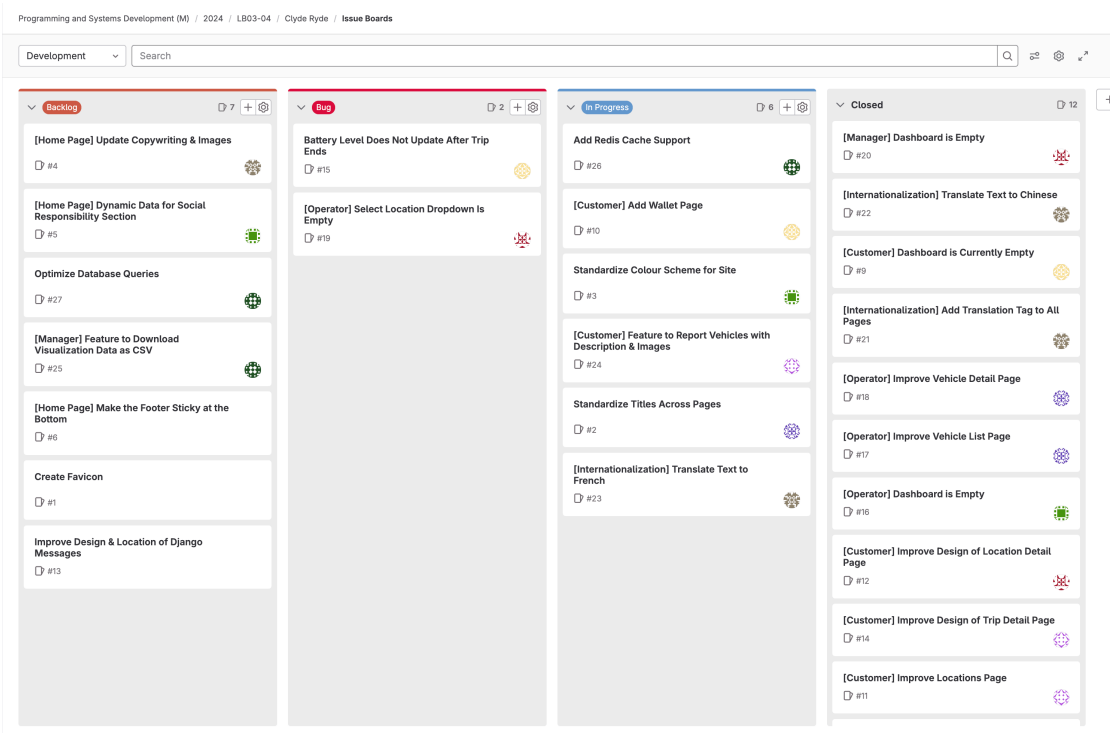

Figure C-2: Project Kanban Board on GitLab

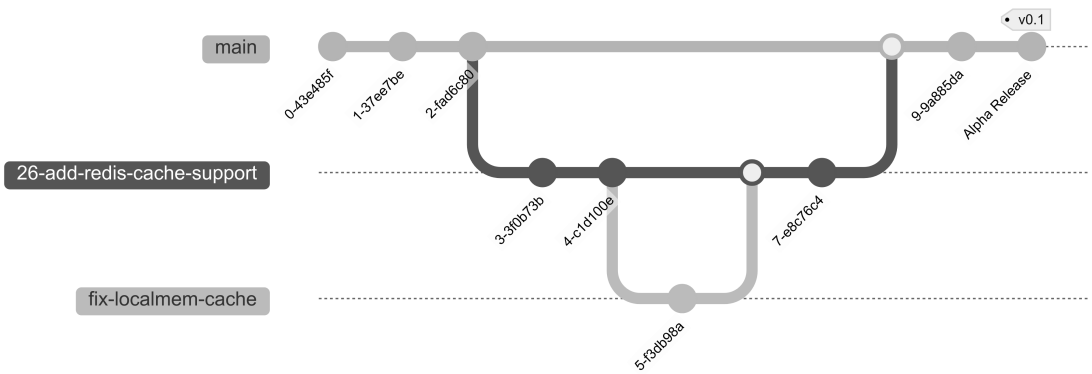

Figure C-3: Project Branching Strategy

```python
def compute_cost(self) -> Optional[Decimal]:
    """Computes cost based on duration and vehicle rate."""
    if self.duration is None:
        return None
    rate = self.vehicle.type.rate
    duration_in_hours = Decimal(self.duration) / Decimal(3600)
    cost = rate * duration_in_hours
    return cost.quantize(Decimal('0.01'))  # 2 decimal places
```

Snippet C-1: Clean Code Example

Figure C-4: SonarLint Checks for Code Quality

```
(venv) siddy@Siddharthas-MacBook-Air clyde-ryde % git commit -m "fix f-string error"
trim trailing whitespace.....................................................Passed
fix end of files.............................................................Passed
check yaml.....................................(no files to check)Skipped
fix double quoted strings....................................................Passed
don't commit to branch.......................................................Passed
black........................................................................Failed
- hook id: black
- files were modified by this hook

reformatted operators/views.py

All done! ✨ 🍰 ✨
1 file reformatted, 1 file left unchanged.

flake8.......................................................................Passed
- hook id: flake8
- duration: 0.21s

core/models.py:8:1: F401 'decimal.Decimal' imported but unused

djLint formatting for Django.........................(no files to check)Skipped
(venv) siddy@Siddharthas-MacBook-Air clyde-ryde %
```

Figure C-5: Pre-Commit Checks for Code Quality

Appendix D Technical Code Fragments

```
.btn-cr-light {
  background-color: #ffd700;
  color: #333333;
  font-weight: 600;
  transition: all 0.3s ease;
}
.btn-cr-light:hover {
  color: #ffffff;
}
```

Snippet D-1: Custom CSS for Consistent Colour Scheme

```
{% block title %}
    {% trans "Dashboard" %}
{% endblock title %}

{% block navbar %}
    {% include "components/navbar.html" %}
{% endblock navbar %}

{% block content %}
    ...
{% endblock content %}
```

Snippet D-2: Reusability with Jinja Templates

```
<script src="https://maps.googleapis.com/maps/api/js?key={{
GOOGLE_MAPS_API_KEY }}&libraries=marker"></script>
<script>
    locations.forEach(function(location) {
            var marker = new
google.maps.marker.AdvancedMarkerElement({
                position: {lat: location.lat, lng: location.lng},
                map: map,
                title: location.name
            });
            bounds.extend(marker.position);
        });
</script>
```

Snippet D-3: Marker Elements with the Google Maps JavaScript API

```
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<script>
    var myChart = new Chart(ctx, {
        type: 'doughnut',
        data: {
            labels: {{ models|safe }};,
            datasets: [{
                data: {{ counts|safe }},
            }]
        },
    });
</script>
```

Snippet D-4: Chart.js Usage for Data Visualization

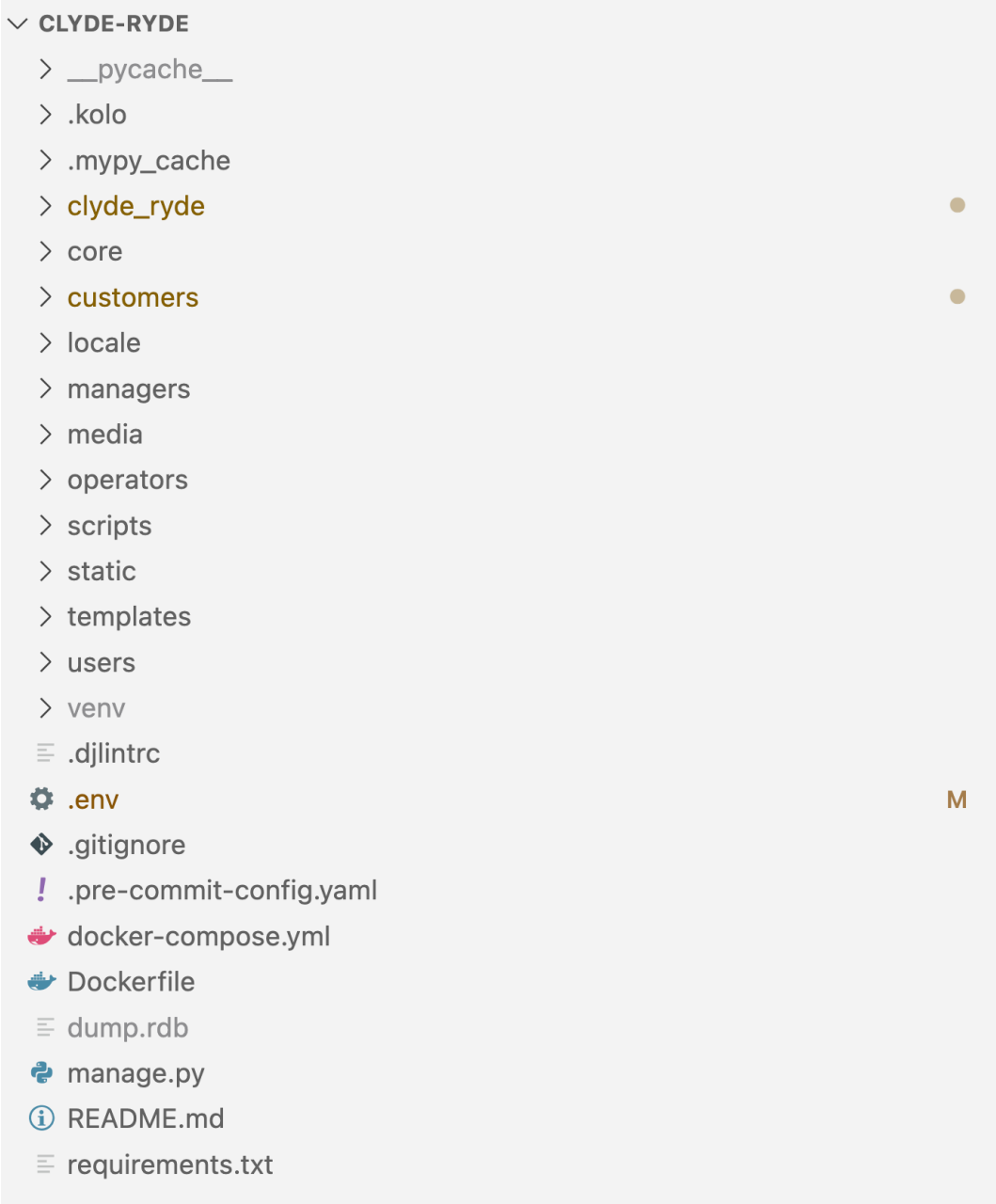

Figure D-1: Django Project Structure

```
#: customers/views.py:197
msgid "The vehicle has been reported as defective."
msgstr "Le véhicule a été signalé comme défectueux."

#: customers/views.py:210
#, python-format
msgid "You have been charged £%(trip_cost).2f."
msgstr "Vous avez été facturé de £%(trip_cost).2f."

#: operators/views.py:93
msgid "Vehicle %(code)s is charged to 100%%!"
msgstr "Le véhicule %(code)s est chargé à 100% !"
```

Snippet D-5: Sample Translation Content for French

```
if trip.vehicle.status == Vehicle.Status.DEFECTIVE:
    messages.error(request, _('The vehicle is already reported!'))
else:
    trip.vehicle.status = Vehicle.Status.DEFECTIVE
    trip.vehicle.save(update_fields=['status'])
    messages.success(request, _('The vehicle has been reported as
defective.'))
```

Snippet D-6: Providing Real-Time Feedback via Django Messages

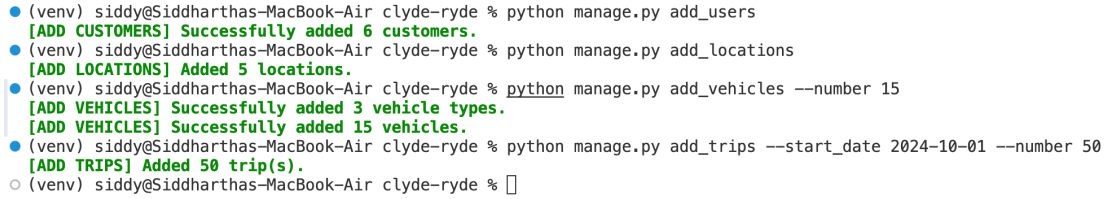

Figure D-2: Django Management Commands Usage to add Dummy Data

```
queryset = super().get_queryset()
vehicle_qs = (
    Vehicle.objects.only(
        'code',
        'battery_level',
        'location_id',
        'type__model',
        'type__brand',
        'type__rate',
        'type__image',
    )
    .select_related('type')
    .filter(status=Vehicle.Status.AVAILABLE)
)
sort, filter = self.request.GET.get('sort'),
self.request.GET.get('filter')
if sort == 'rate':
    vehicle_qs = vehicle_qs.order_by('type__rate')
elif sort == 'battery_level':
    vehicle_qs = vehicle_qs.order_by('-battery_level')
if filter:
    vehicle_qs = vehicle_qs.filter(type_id=filter)
queryset = queryset.prefetch_related(
    Prefetch('vehicles', queryset=vehicle_qs,
to_attr='available_vehicles')
)
```

Snippet D-7: Django ORM Usage for Efficient SQL Queries

```
class CacheUtil:
    _instance = None

    def __new__(cls):
        """Singleton pattern"""
        if cls._instance is None:
            cls._instance = super().__new__(cls)
            cls._instance.cache = cls.__initialize_cache()
        return cls._instance

    @staticmethod
    def __initialize_cache():
        """Attempts RedisCache, defaults to LocMemCache"""
        try:
            cache = caches['redis']
            cache.get('key')
            return cache
        except Exception as e:
            logger.debug(f'Error using Redis cache: {e}')
            logger.debug('Using LocMem cache instead.')
        return caches['default']
```

Snippet D-8: Singleton Class for Cache Initialization

```
vehicle_types = cache.get(VEHICLE_TYPES_CACHE_KEY)
if not vehicle_types:
    logger.debug(f'Cache miss: {VEHICLE_TYPES_CACHE_KEY}')
    vehicle_types = VehicleType.objects.all()[:3]
    cache.set(VEHICLE_TYPES_CACHE_KEY, vehicle_types, ONE_DAY)
```

Snippet D-9: Cache Usage for Frequently Accessed Data

```
services:
  db:
    image: postgres:14
    volumes:
      - postgres_data:/var/lib/postgresql/data
  cache:
    image: redis:alpine
    volumes:
      - redis_data:/data
  web:
    build: .
    volumes:
      - .:/app
    depends_on:
      - db
      - cache
```

Snippet D-10: Configuration of Services in docker-compose.yml

```
# Use the official Python image from the Docker Hub
FROM python:3.12-slim

# Set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# Set working directory
WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    libpq-dev \
    gcc \
    && rm -rf /var/lib/apt/lists/*

# Copy project
COPY . /app/

# Install Python dependencies
RUN pip install --upgrade pip
RUN pip install -r requirements.txt

# Prepare entrypoint script
COPY scripts/entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

# Expose the port
EXPOSE 8000

# Run entrypoint script
CMD ["/entrypoint.sh"]
```

Snippet D-11: Dockerfile to Create the Image for Clyde Ryde

```
with transaction.atomic():
    if request.user.wallet.debit(trip_cost):
        payment.complete_payment()
        messages.success(
            request,
            _('Trip completed! You have been charged
£%(trip_cost).2f.')
            % {'trip_cost': trip_cost},
        )
    else:
        messages.error(
            request,
            _('Insufficient balance. Please top-up your wallet.'),
        )
```

Snippet D-12: Payment Updates Performed as an Atomic Transaction

Appendix E Application Screenshots

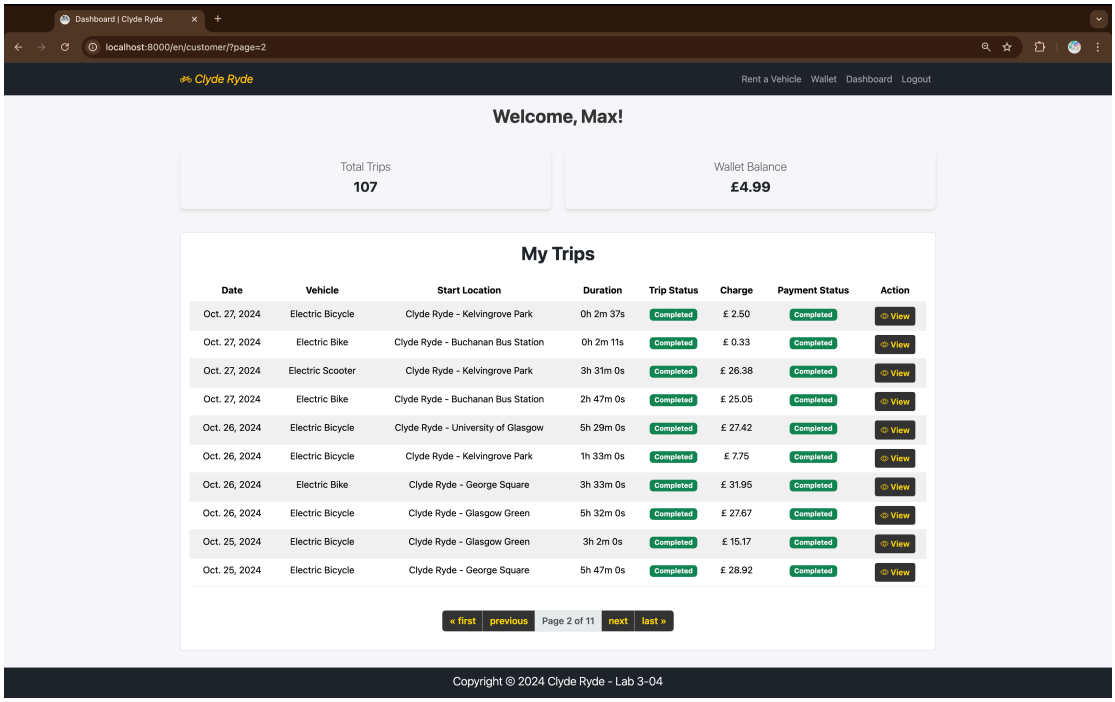

Figure E-1: Customer Dashboard

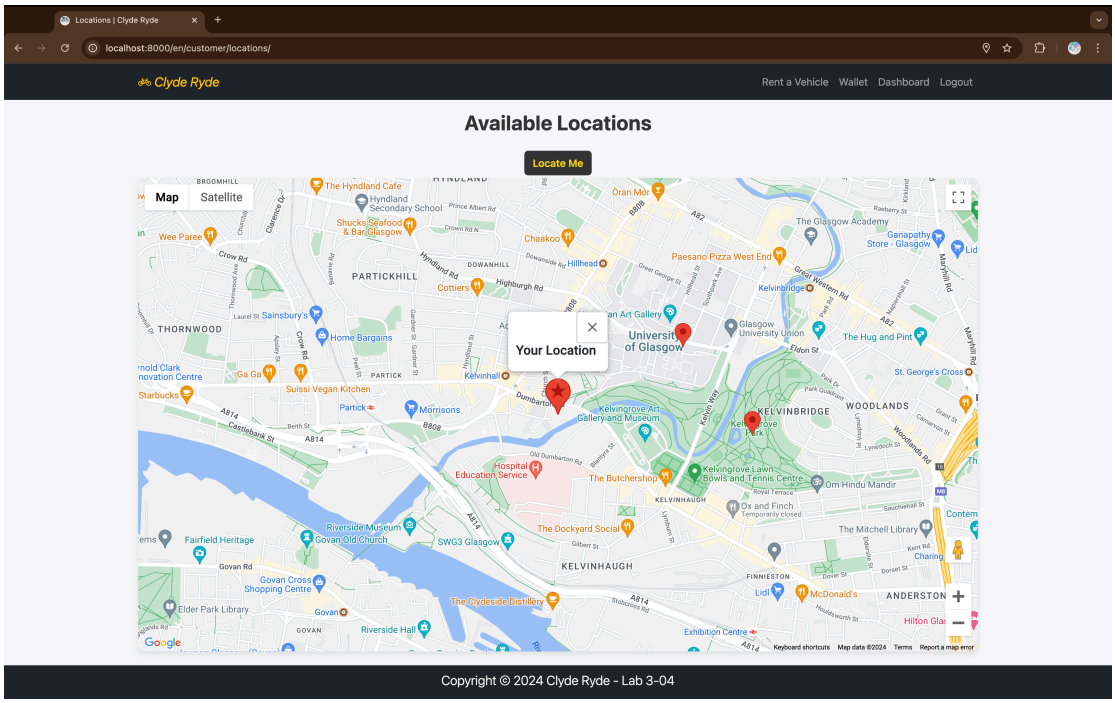

Figure E-2: Viewing Rental Locations

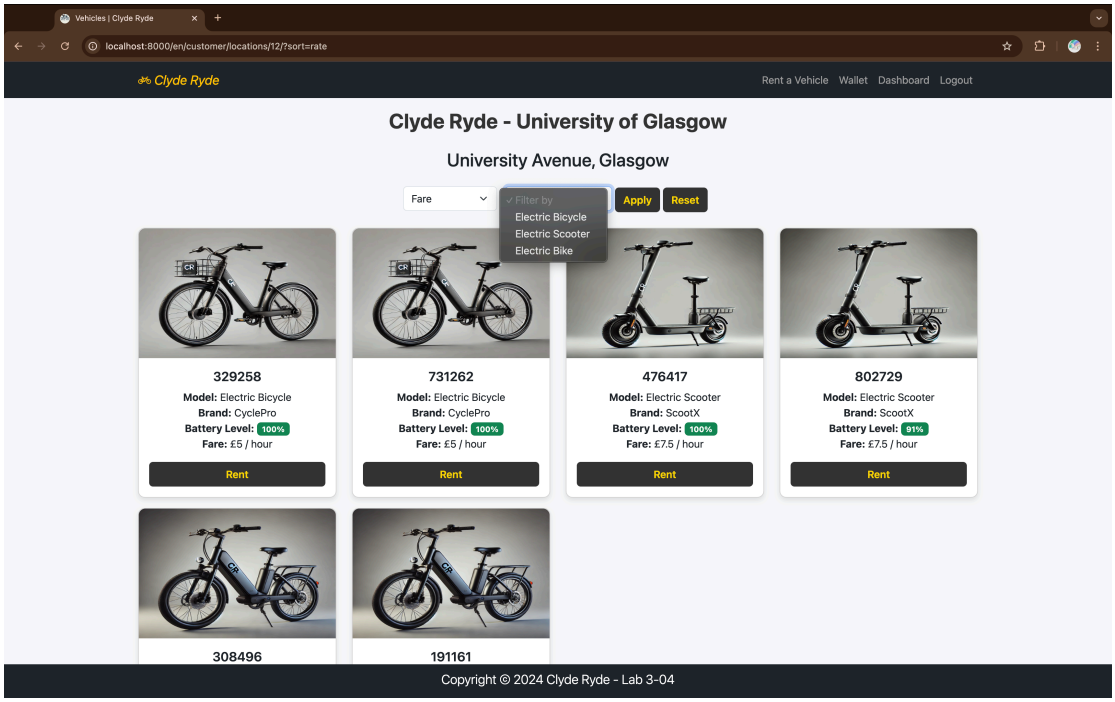

Figure E-3: Viewing Available Vehicles

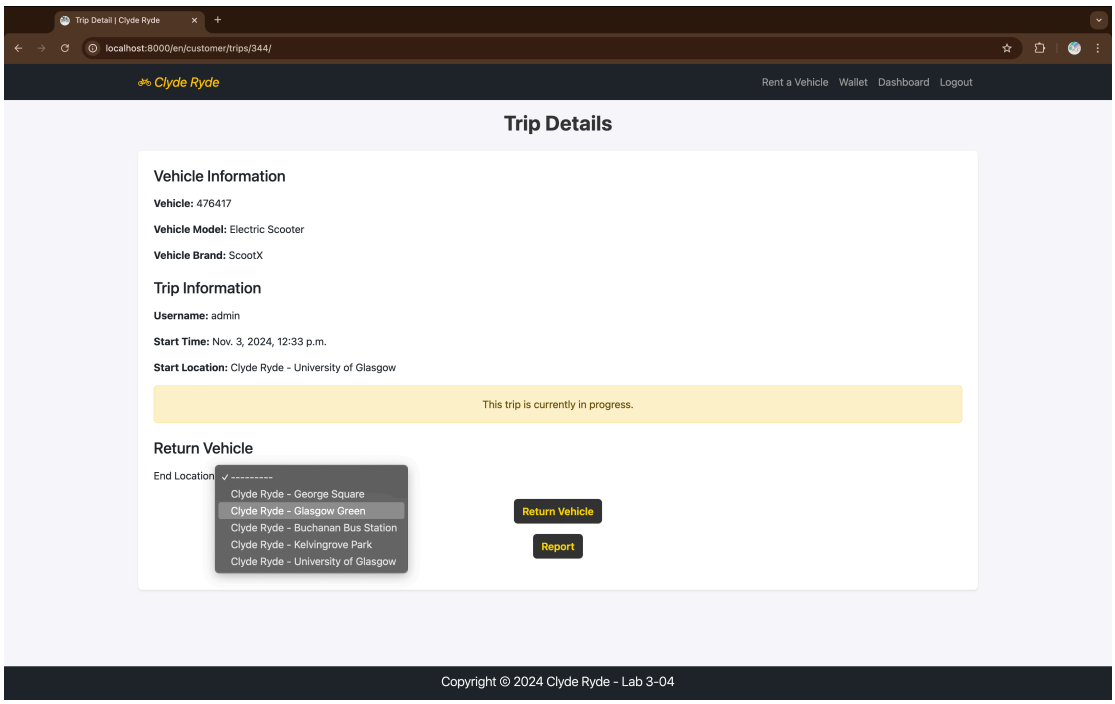

Figure E-4: Trip Details

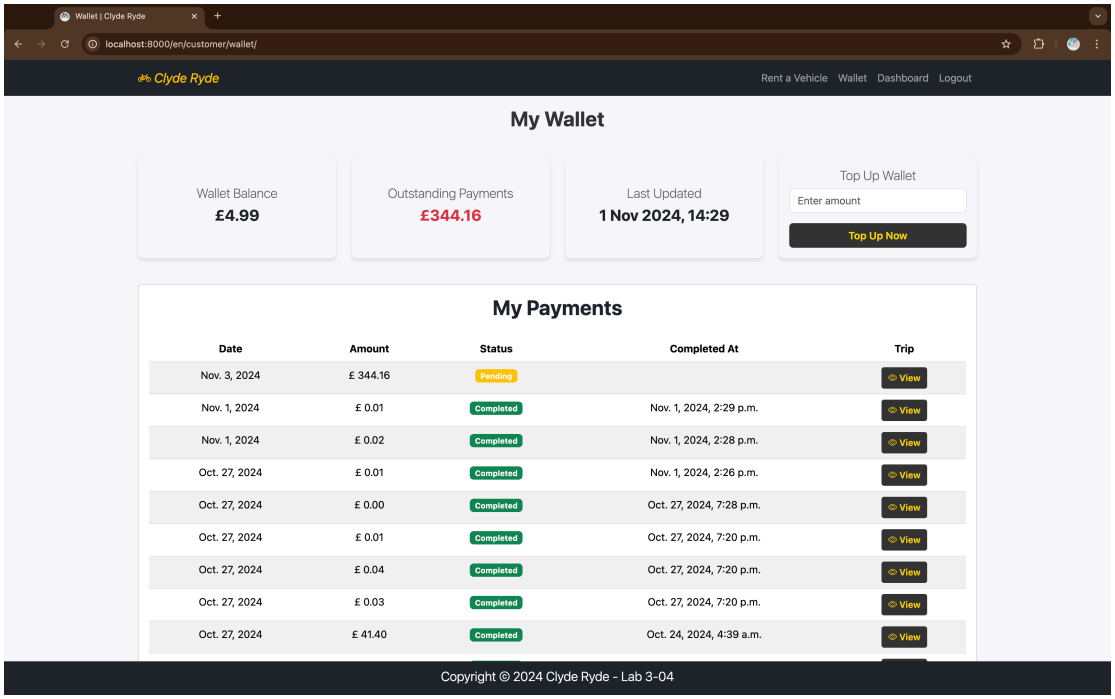

Figure E-5: Customer Wallet

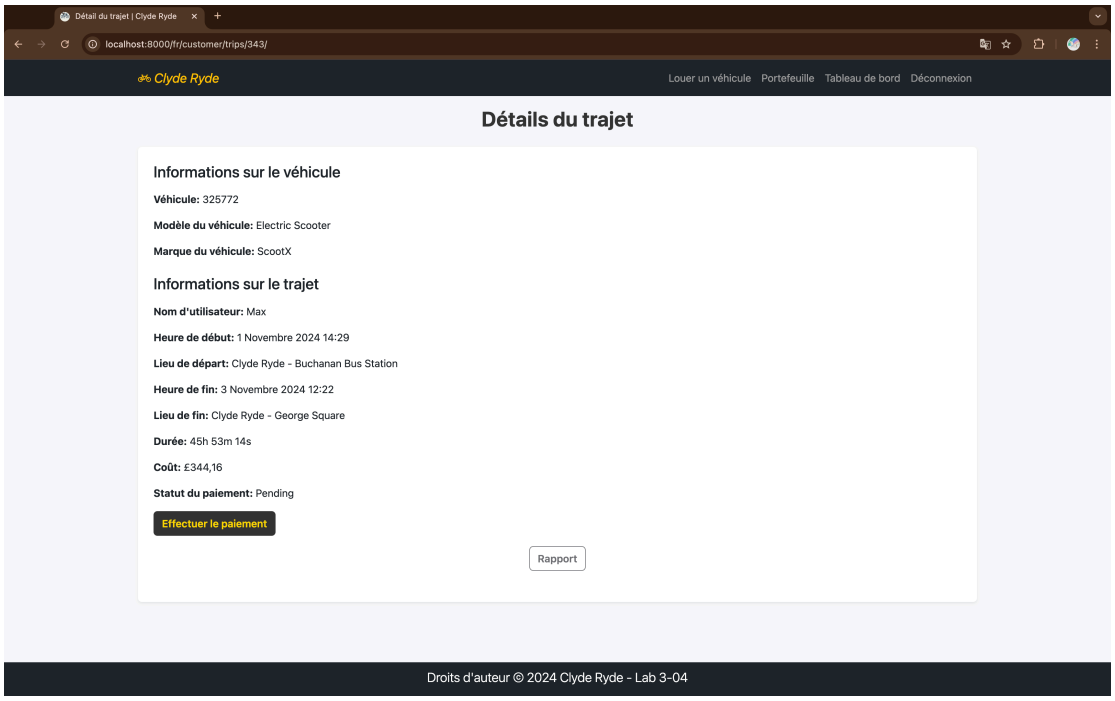

Figure E-6: Trip Detail Page in French

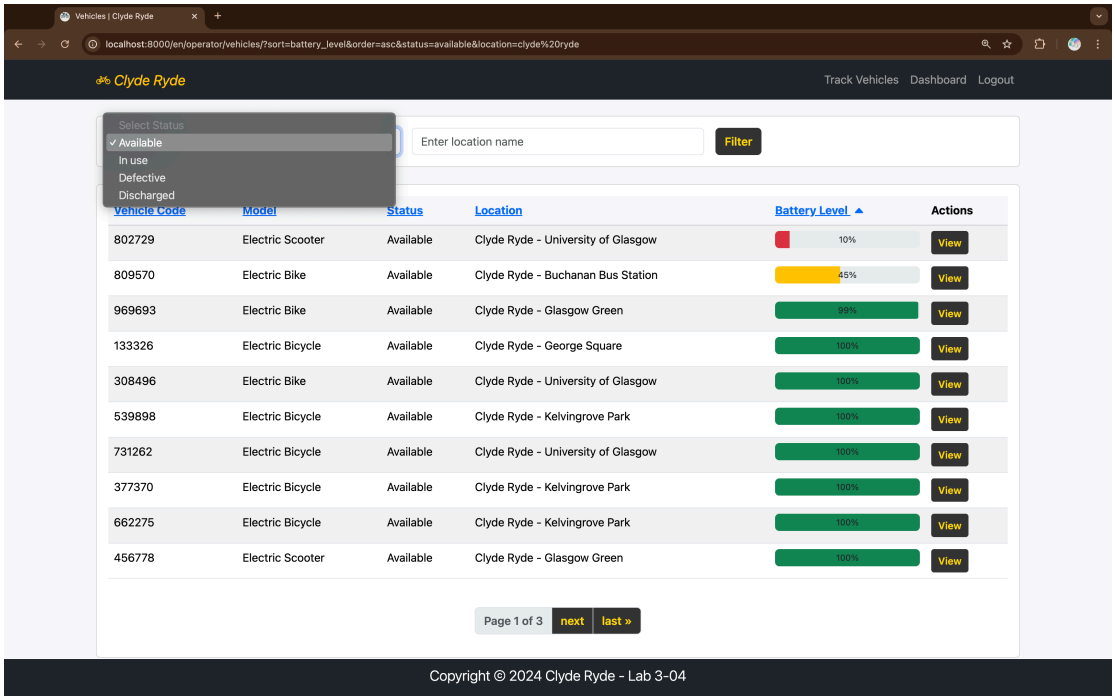

Figure E-7: Vehicle Listing Page

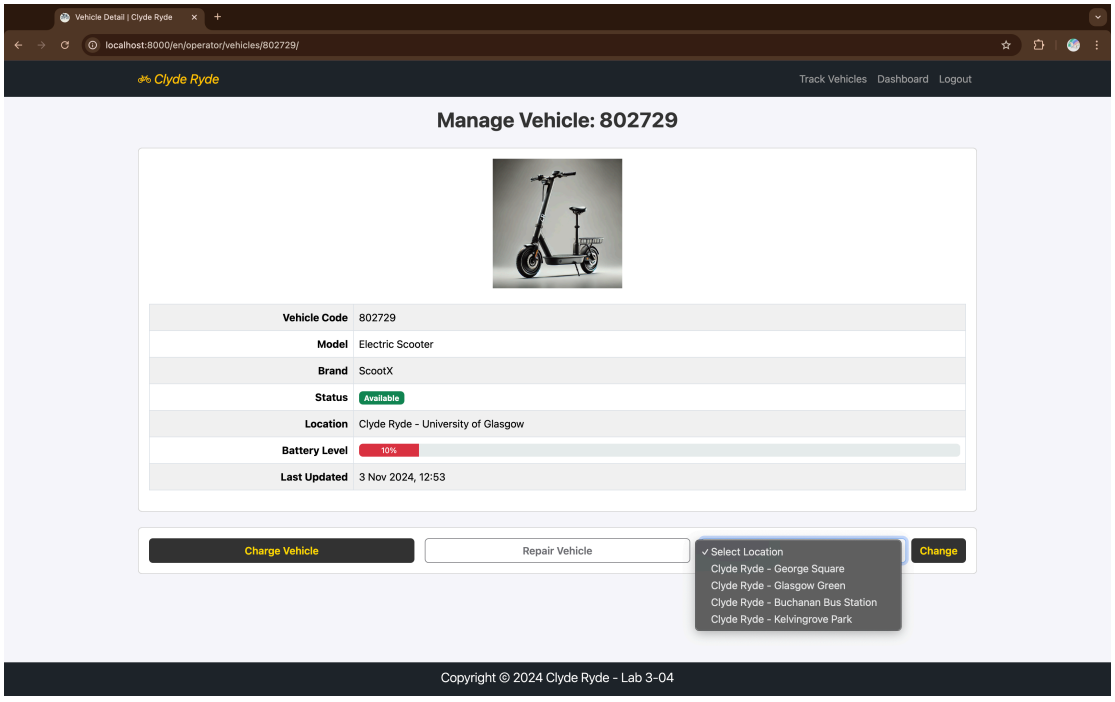

Figure E-8: Vehicle Detail Page

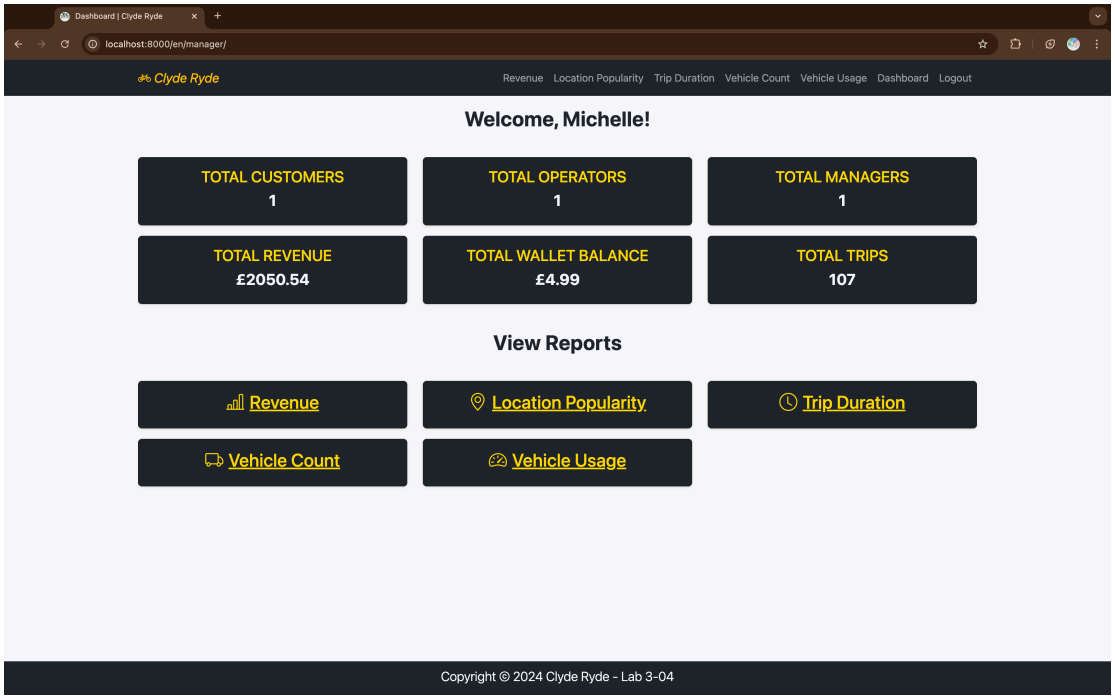

Figure E-9: Manager Dashboard Metrics

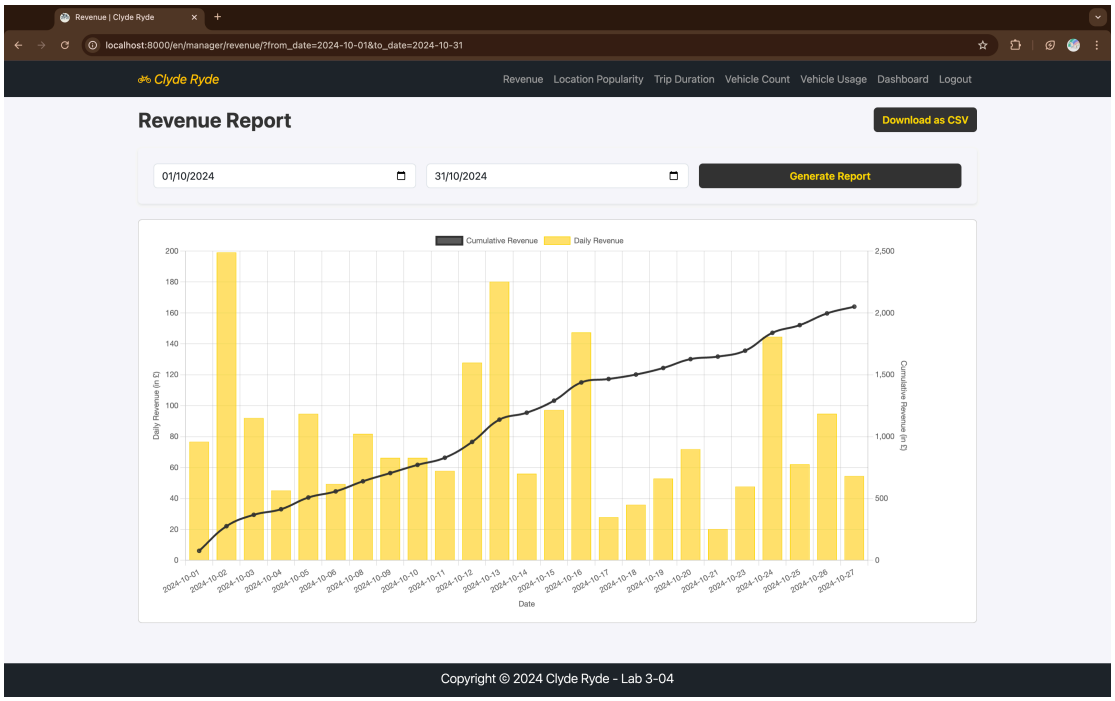

Figure E-10: Revenue Report Visualization

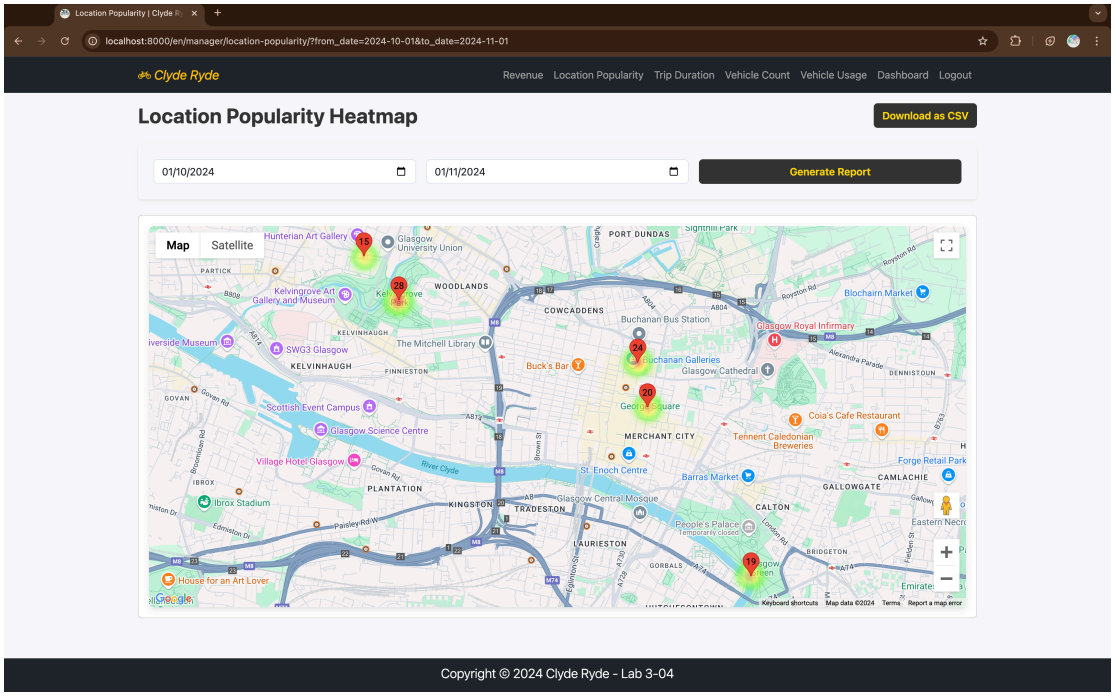

Figure E-11: Location Popularity Visualization

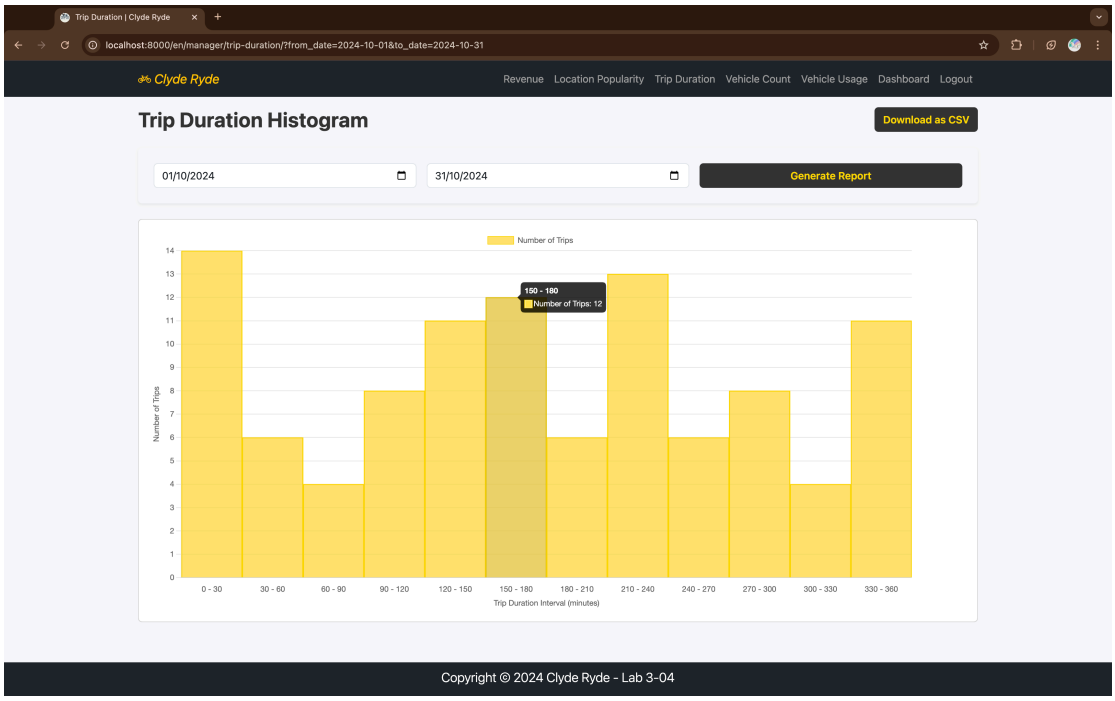

Figure E-12: Trip Duration Visualization

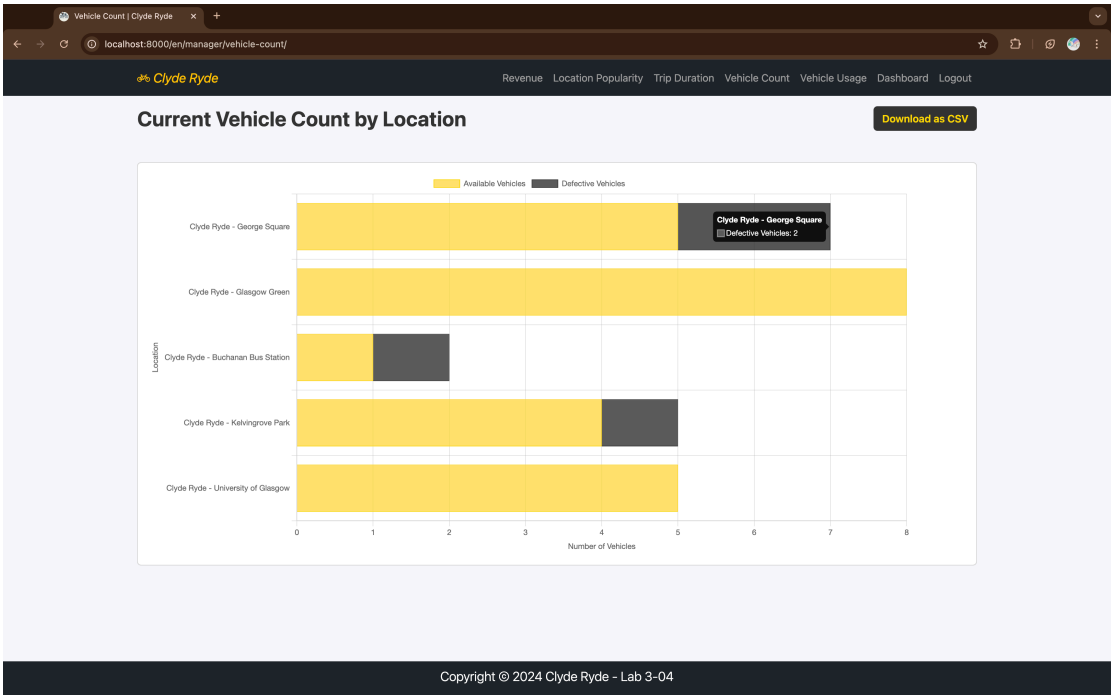

Figure E-13: Vehicle Count Visualization

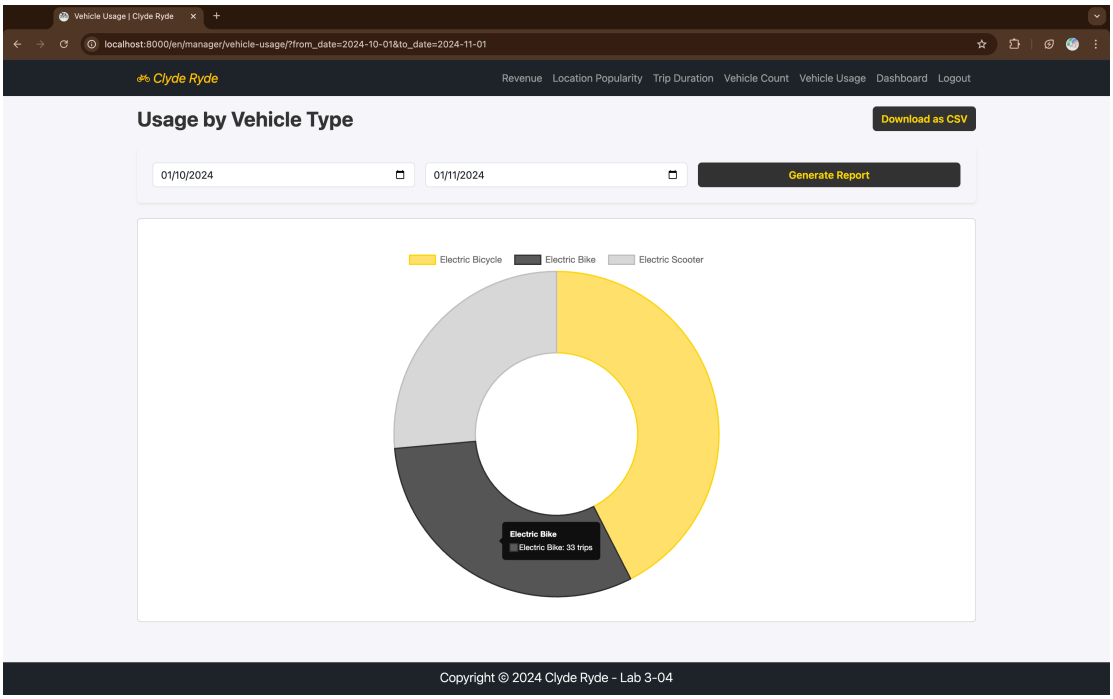

Figure E-14: Vehicle Type Usage Visualization

Appendix F Implementation Highlights

Title	Description
Customer Dashboard	Implemented a tabulated summary view for customers to track their trips, complete with pagination to support ease of navigation through historical trip data.
Location Detection	Enabled real-time location detection for customers, providing rental locations nearest to their current position, enhancing convenience in vehicle selection.
Customer – Sorting / Filtering Vehicles	Added sorting options by fare and battery level, along with filtering by vehicle type, allowing customers to tailor vehicle selection based on their preferences.
In-App Wallet	Integrated an in-app wallet system to streamline payments and top-ups, offering customers a seamless, secure experience for handling transactions.
Multiple Language Support	Extended language support to include Chinese and French, broadening accessibility and catering to a diverse customer base.
Operator – Sorting / Searching Vehicles	Developed sorting and search capabilities for operators, allowing them to efficiently locate vehicles by criteria such as code, model, status, location, and battery level for effective fleet management.
Manager Dashboard	Provided a comprehensive dashboard for managers to monitor key metrics, including user count, trip volume, and total revenue, offering insights into overall business performance.
CSV Downloads	Enabled data export functionality for managers to download visualization data as CSV files, facilitating further analysis and offline processing as needed.

Table F–1: Summary of Extra Features

Title	Description
Google Maps JavaScript API	Configured a Google Cloud project to obtain necessary credentials, enabling custom marker placement on the Google Maps interface through the JavaScript API.
Chart.js Library	Leveraged the Chart.js library to create modern, animated visualizations with mixed chart types, customized to match the application's color palette for a cohesive user interface.
Django Internationalization	Implemented Django's internationalization support to provide multilingual options, allowing the application to be accessed in Chinese and French for improved accessibility.
Django Messages Framework	Incorporated Django's messages framework to display "flash" notifications for users, enhancing feedback on actions performed within the application.
Django Management Commands	Automated database setup by adding custom management commands to populate dummy data. Commands include parameters for customization, such as date range or entry count, allowing flexible data generation for testing and development.
Django ORM	Utilized Django's ORM for complex database operations, including advanced SQL queries with aggregations and index creation, simplifying interaction with the database while optimizing performance.
Redis	Integrated Redis as a caching solution to improve performance by storing frequently accessed data, reducing load on the primary database and enhancing response times.
Docker	Created a Dockerized multi-container environment, bundling all necessary dependencies and services (including web, database, and cache) into isolated containers, ensuring consistent deployment and streamlined setup.

Table F–2: Summary of Technical Complexities

Appendix G Evaluation Results

Test	Description / Action	Expected Result	Actual Result
View Locations	Customer retrieves available locations	Available locations are displayed on a map	Passed
Rent Vehicle	Customer selects a vehicle to rent	Vehicle status is updated, a new trip is created	Passed
Return Vehicle	Customer returns a rented vehicle at a location	Trip is updated, and a new payment is created	Passed
Report a Vehicle	Customer reports a vehicle as defective	Vehicle status is updated	Passed
Pay Charges	Customer pays a pending charge	Payment record is updated	Passed
Track Vehicles	Operator views locations of all vehicles	List of vehicles is displayed with locations	Passed
Charge a Vehicle	Operator charges a vehicle	Vehicle battery level is updated	Passed
Repair a Vehicle	Operator repairs a vehicle	Vehicle status is updated	Passed
Move a Vehicle	Operator changes the location of a vehicle	Vehicle location is updated	Passed
Generate Reports	Manager selects a date filter and report type	Filtered data is displayed as a visualization	Passed

Table G–1: Acceptance Test Table

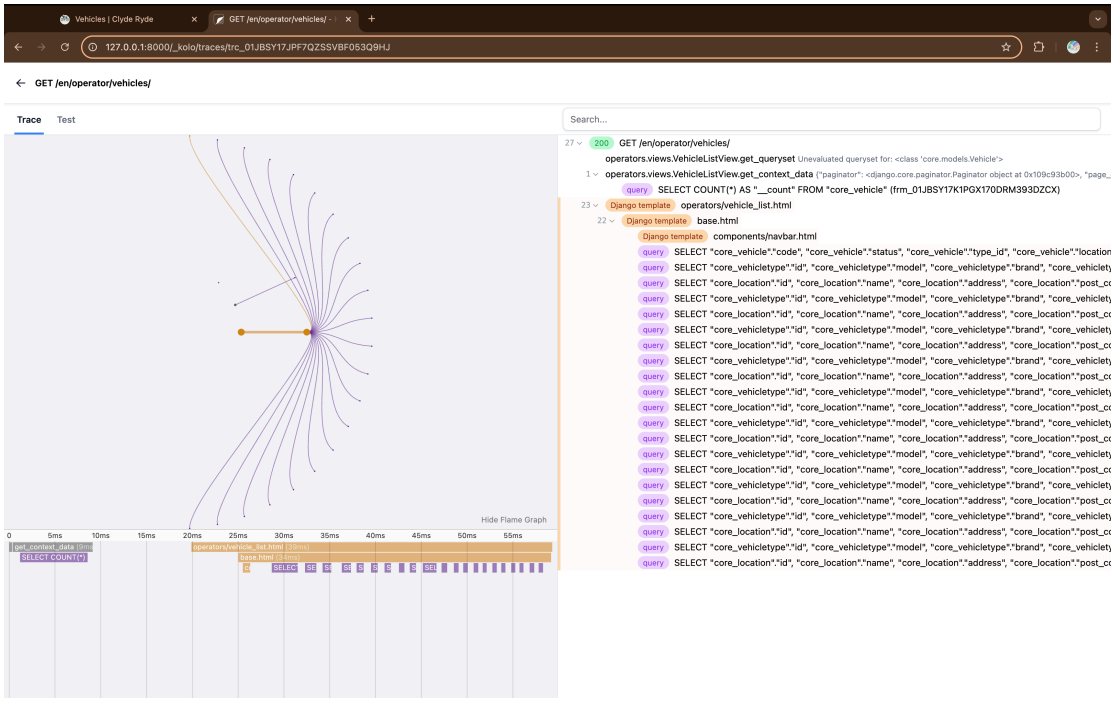

Figure G-1: Database Queries before Optimizations for Vehicle Listing

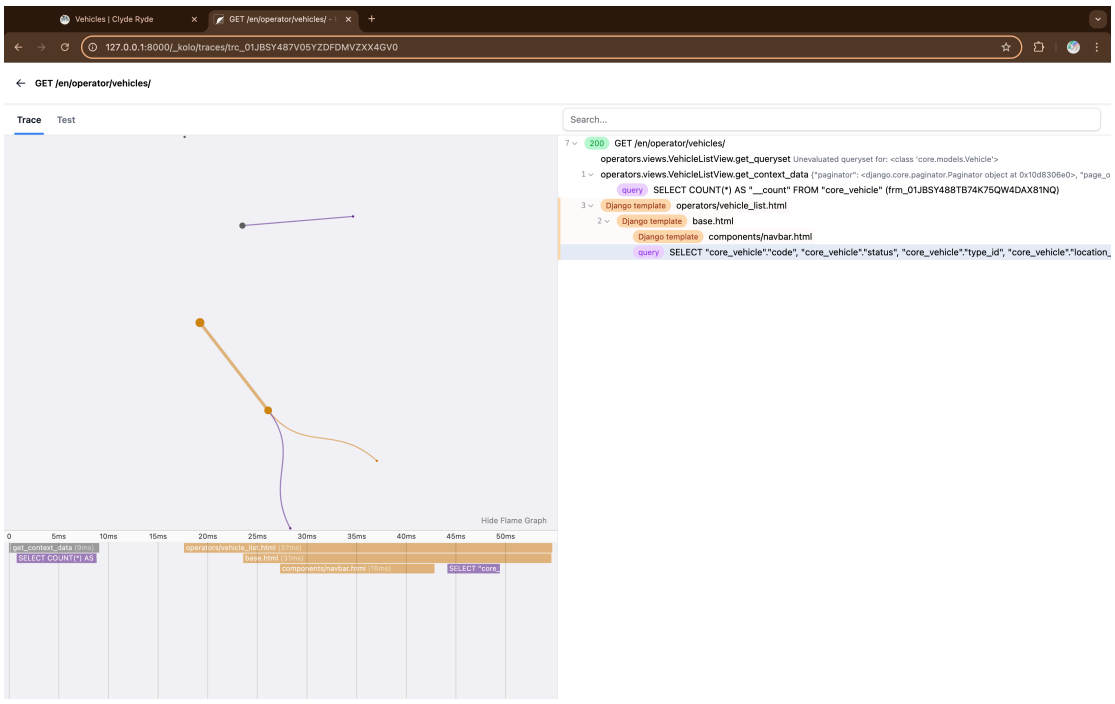

Figure G-2: Database Queries after Optimizations for Vehicle Listing