Report

Complex Engineering Project

CS-116: Object Oriented Programming

Submitted by:

Muhammad Ahmed Qazi CS-24045

Mujtaba Jawaid Rao CS-24047

Muhammad Zain Rizvi CS-24048

Submitted to:

Ms. Fauzia Yasir

Ms. Tahreem Khan

Practical Group:

G2

Date of submission: Monday, 19/05/2025

DEPARTMENT OF COMPUTER & INFORMATION SYSTEMS ENGINEERING
BACHELORS IN COMPUTER SYSTEMS ENGINEERING
Course Code: CS-116
Course Title: Object Oriented Programming
Complex Engineering Problem
FE Batch 2024, Spring Semester 2025
Grading Rubric
TERM PROJECT

Group Members:

Student No.	Name	Roll No.
S1	Muhammad Ahmed Qazi	CS-24045
S2	Mujtaba Jawaid Rao	CS-24047
S3	Muhammad Zain Rizvi	CS-24048

CRITERIA AND SCALES				Marks Obtained		
				S1	S2	S3
Criterion 1: Does the application meet the desired specifications and produce the desired outputs? (CPA-1, CPA-3)						
1	2	3	4			
The application does not meet the desired specifications and is producing incorrect outputs.	The application partially meets the desired specifications and is producing incorrect or partially correct outputs.	The application meets the desired specifications but is producing incorrect or partially correct outputs.	The application meets all the desired specifications and is producing correct outputs.			
Criterion 2: How well is the code organization?						
1	2	3	4			
The code is poorly organized and very difficult to read.	The code is readable only to someone who knows what it is supposed to be doing.	Some part of the code is well organized, while some part is difficult to follow.	The code is well organized and very easy to follow.			
Criterion 3: How friendly is the application interface? (CPA-1, CPA-3)						
1	2	3	4			
The application interface is difficult to understand and use.	The application interface is easy to understand and but not that comfortable to use.	The application interface is very easy to understand and use.	The application interface is very interesting/ innovative and easy to understand and use.			
Criterion 4: How does the student performed individually and as a team member? (CPA-2, CPA-3)						
1	2	3	4			
The student did not work on the assigned task.	The student worked on the assigned task, and accomplished goals partially.	The student worked on the assigned task, and accomplished goals satisfactorily.	The student worked on the assigned task, and accomplished goals beyond expectations.			
Criterion 5: Does the report adhere to the given format and requirements?						
1	2	3	4			
The report does not contain the required information and is formatted poorly.	The report contains the required information only partially but is formatted well.	The report contains all the required information but is formatted poorly.	The report contains all the required information and completely adheres to the given format.			
			Total Marks:			

Table of Contents

1. Introduction

1.1 Problem Description

The aim of this project is to design and implement an Online Car Rental System named “AutoHire”, which allows users to register, browse available vehicles, make car reservations, and manage their bookings. The system caters to two roles: User and Administrator.

The application satisfies all the minimum required features:

- ☑ Account registration and authentication with user details
- ☑ Car catalogue management
- ☑ Availability-based car display
- ☑ Implementation of single reservation policy
- ☑ Automatic payment processing
- ☑ Rental history tracking
- ☑ User dashboard and balance management
- ☑ Administrative reporting portal
- ☑ Administrative fleet management

The application extends the essential feature set by providing the following features:

- ☑ Graphical User Interface (GUI) (Web version)
- ☑ Return date verification and automated adjustments
- ☑ Modular and scalable class architecture

1.2 Project Scope

This project demonstrates the application of programming concepts learned during the course. It emphasises:

- ➔ **Object-Oriented Design and Modelling**
  Implementation of classes, inheritance, polymorphism, abstract base classes, and encapsulation to represent system entities and behaviours.
- ➔ **User Interface and Interaction**
- ➔ **Error Handling and Validation**
  Robust use of exception handling (`try-except` blocks) and input validation to ensure system reliability and security.
- ➔ **Data Management and Persistence**
  Efficient file handling for persistent storage of users, cars, and reservations, along with data serialisation/deserialization.

2. Distinguishing Features of AutoHire

AutoHire is not just a basic implementation of a car rental system — it includes several thoughtful features and design choices that enhance its professionalism, usability, and extensibility. Below are the key distinguishing features:

- ➔ **Realistic Object-Oriented Design**
  - Use of inheritance and abstraction to model different car types (`EconomyCar`, `LuxuryCar`, `CommercialCar`) using a base `Car` abstract class.
  - Separation of concerns through dedicated classes: `User`, `Admin`, `Reservation`, `Fleet`, and `RentalSystem`.
- ➔ **Role-Based System Functionality**
  - Admin and user roles have completely separate access scopes, forms, and dashboards.
  - Admins can manage the fleet and generate reports, but cannot book cars.
- ➔ **Dynamic Car Availability Filtering**
  - Cars shown to users are filtered in real-time based on selected rental dates and existing reservations, preventing double bookings.
- ➔ **Reservation Logic with Smart Validation**
  - Users can have only one active reservation at a time.
  - Early return triggers refund; late return applies a penalty — both reflected in the account balance.
- ➔ **Professional Receipt Generation**
  - PDF receipts are generated automatically upon successful reservation using `pdfkit`, styled with a logo and structured layout.
- ➔ **Clean and Responsive UI**
  - Fully responsive front-end design using Bootstrap 5.
  - Modal-based forms and toast alerts for seamless user experience.
- ➔ **Session Management and Flash Messaging**
  - Flask session tracking for login/logout.
  - Flash-based toasts for dynamic alerts such as errors, warnings, and confirmations.
- ➔ **Admin Reporting Tools**
  - Admin dashboard provides:
    - Customer reservation overview.
    - Active reservations on the fleet.
    - CSV report export functionality.
- ➔ **User Profile Management**
  - Users can view and edit profile information, including license details and profile image.
  - Users can return cars, view current/past reservations, and delete their account.
- ➔ **Data-Driven Architecture**
  - The system operates on a JSON-based storage model.
  - All state changes are reflected persistently via the `FileHandler` utility class.

3. Object-Oriented Features and Project Flow

This section outlines the object-oriented programming (OOP) principles implemented in the system and presents the logical flow of the application's architecture.

3.1 Object-Oriented Programming Features Implemented

The project demonstrates the following core OOP features:

1. **Encapsulation**
   Data and behaviours are encapsulated within relevant classes such as `User`, `Admin`, `Car`, and `Reservation`, with controlled access through getter and setter methods.
2. **Inheritance**
   Common attributes and methods are defined in abstract base classes (`Person`, `Car`) and inherited by specialised classes (`User`, `Admin`, `EconomyCar`, `LuxuryCar`, `CommercialCar`).
3. **Abstraction**
   The abstract base classes (`ABC`) define interface methods, such as `calculate_rental_cost()` in the `Car` class, which must be implemented by each subclass.
4. **Polymorphism**
   The system handles car objects through the base `Car` type, but specific implementations like `EconomyCar` or `LuxuryCar` override methods and attributes dynamically.
5. **Composition**
   Complex classes like `RentalSystem` utilise other classes like `Fleet`, `User`, and `Reservation` to delegate responsibilities, promoting modularity.
6. **Class-Level and Static Mapping**
   The use of class maps (`CAR_CLASS_MAP`, `USER_CLASS_MAP`) allows dynamic instantiation and loading of class types from persistent storage (JSON).

3.2 Project Flow Overview

The high-level flow of the application is as follows:

1. **System Initialization**
   - Loads user and car data from JSON files via the `FileHandler` helper class.
   - Initialises the `RentalSystem` object, which manages all users, cars, and reservations.

2. **User Interaction**
    - Users or admins register/login through a Flask-based web interface.
    - User roles are determined via email domain logic and validated on registration.
3. **Admin Module**
    - Admins can manage the car fleet (add, edit, and delete cars) and generate reports on customer activity and reserved cars.
    - Access to these features is restricted via role-based authentication.
4. **Customer Module**
    - Users can browse and filter available cars based on booking dates and type.
    - Users can make reservations, return cars, and download receipts in PDF format.
5. **Reservation Processing**
    - Ensures a single active reservation per user.
    - Checks for license expiry, sufficient balance, and updates the reservation and car objects accordingly.
    - Refunds or late fees are calculated upon car return and the user's balance is updated.
6. **Reporting and Logging**
    - Reports for administrators are available in both tabular and downloadable CSV formats.
    - All key actions are accompanied by Bootstrap alerts using Flask's `flash()` messages.
7. **Session and Error Management**
    - Sessions track logged-in users.
    - Appropriate flash messages are displayed for errors, warnings, or confirmations across the app.

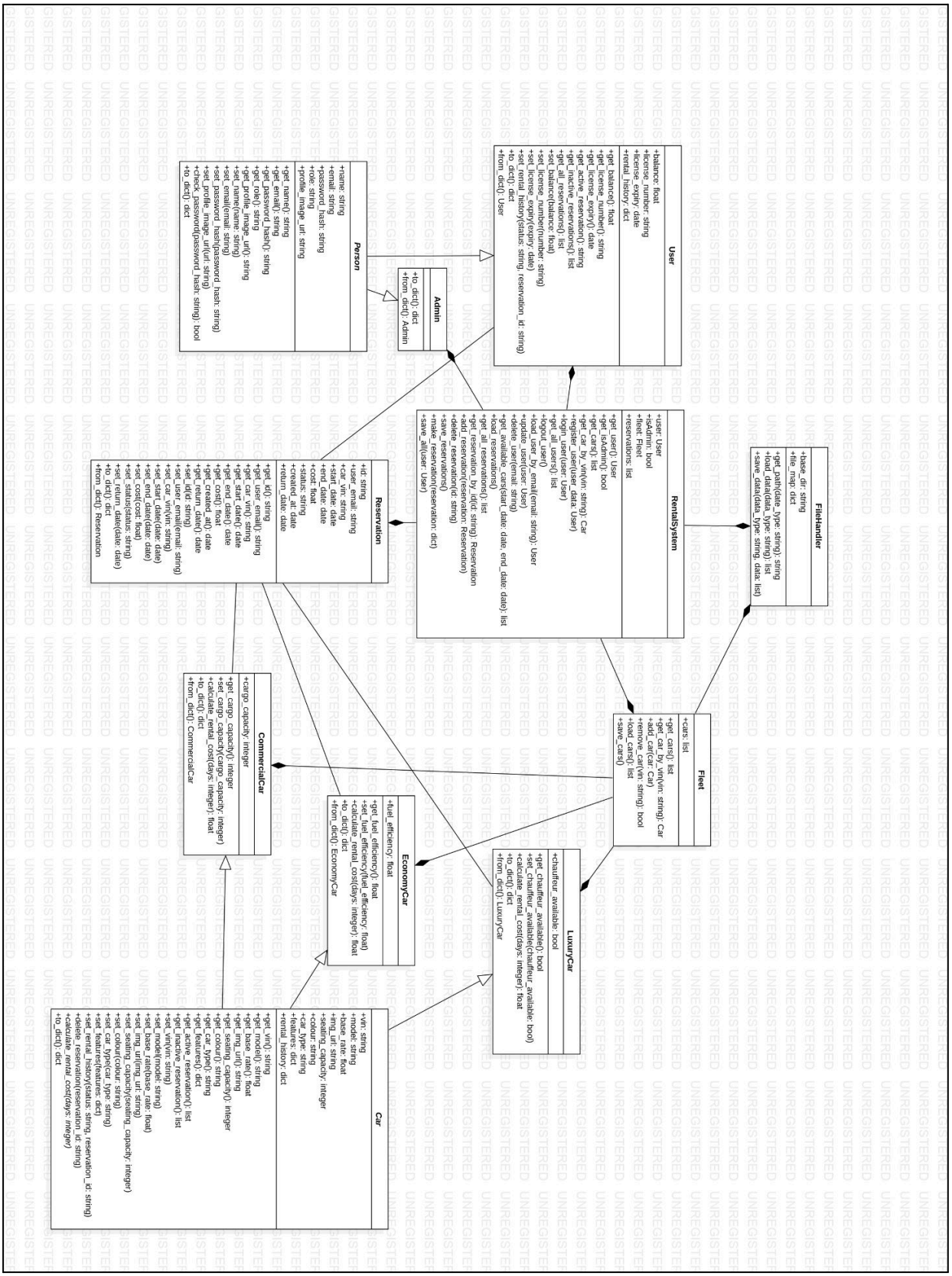
3.3 Data Storage and Persistence

This project uses **JSON files** for persistent data storage instead of a traditional database system. This allows structured yet lightweight data management, suitable for a prototype-level web application.

➔ **Directory structure:**
- `cars.json` – Stores a list of car objects with details such as VIN, model, base rate, category, rental history, etc.
- `users.json` – Contains user/admin profiles, including credentials (hashed), balance, license details, and rental history (for users).
- `reservations.json` – Tracks all reservations with metadata like start/end dates, cost, return status, and associated user/car.

➔ **Dictionary-based serialisation:** All classes implement `to_dict()` and `from_dict()` methods for seamless conversion between Python objects and JSON-compatible dictionaries.

3.4 Class Diagram



4. Technical Implementation

This section outlines the technical setup of the application, highlighting the core engine, the web integration, and the libraries and tools that power the system.

4.1 RentalSystem Setup

At the core of the application lies the `RentalSystem` class, which orchestrates the key functionalities:

- ➔ **Central Coordinator:** It acts as the controller, managing the interaction between users, cars, and reservations.
- ➔ **Encapsulation of Logic:** All login/logout operations, user registration, reservation handling, and business logic are encapsulated in this class.
- ➔ **Data Delegation:** It delegates file operations to the `FileHandler` class and fleet operations to the `Fleet` class for modularity.
- ➔ **Session Awareness:** It stores the current logged-in user and ensures consistent data flow across requests.

4.2 Flask Web Integration

The entire system is wrapped in a Flask web server to provide an interactive and user-friendly GUI:

- ➔ **Routing:** Flask routes (`@app.route`) handle URLs for login, registration, reservations, admin reports, and more.
- ➔ **Templates:** HTML templates with Bootstrap styling are rendered using Jinja2 templating engine.
- ➔ **AJAX Support:** JavaScript and fetch API are used for client-server interactions like login and registration without page reload.
- ➔ **Session Management:** Flask's `session` object tracks user authentication and flash messages for alert handling.

4.3 Technologies and Packages Used

- ➔ **Flask**: lightweight web framework used to build the web server and handle routing
- ➔ **Bootstrap:** frontend CSS framework for responsive UI styling
- ➔ **Jinja2**: templating engine used with Falsk for HTML rendering
- ➔ **pdfkit**: used to convert reservation receipts into downloadable PDF format
- ➔ **wkhtmltopdf:** backend engine for pdfkit to generate PDFs from HTML templates
- ➔ **Uuid:** generates unique reservation IDs
- ➔ **json:** for reading/writing structured application data
- ➔ **csv:** for generating admin reports in downloadable formats
- ➔ **werkzeug:** implements password hashing for authentication security

5. Challenges and New Learnings

5.1 Challenges Faced

- **Modular Project Architecture**
  Structuring the project into cleanly separated classes and packages (e.g. `RentalSystem`, `Fleet`, `FileHandler`) while ensuring proper communication between components was a challenge initially.
- **Dynamic Object Mapping from JSON**
  Loading polymorphic class instances (`EconomyCar`, `LuxuryCar`, `CommercialCar`, `User`, `Admin`) from a common `.json` file required implementing class mapping dictionaries and custom constructors.
- **Duplicate Data Prevention**
  Identifying and solving the duplication issue in reservations demanded deep inspection of recursive loading and save methods and safeguarding them with proper checks.
- **PDF Generation Cross-Platform Issues**
  Integrating `pdfkit` with `wkhtmltopdf` proved difficult especially on Windows environments, requiring environment-specific setup and exception handling.
- **Session Persistence and Flash Alerts**
  Handling session validity after abnormal app termination and ensuring flash alerts were timely and visible required tuning session-clearing logic and implementing Bootstrap-based alert overlays.

5.2 New Learnings

- **Practical Use of Object-Oriented Design**
  Applied core OOP concepts (inheritance, abstraction, polymorphism) in a real-world scenario and understood their benefits in code organisation and scalability.
- **Client-Server Interaction Using AJAX**
  Understood how to decouple form submission from page reloads using JavaScript's Fetch API and handle JSON data asynchronously.
- **Robust File Handling with JSON**
  Learned how to structure, parse, and persist application data using JSON and how to model complex objects from file-based storage.
- **UI/UX Principles Using Bootstrap**
  Gained hands-on experience designing responsive and professional UIs using Bootstrap grid system, cards, forms, and modals.
- **Security with Password Hashing**
  Learned to secure user credentials using hashing (`hashlib`) and the importance of not storing passwords in plaintext.
- **Report Generation (CSV, PDF)**
  Implemented dynamic report generation using Python's `csv` module and PDF rendering using HTML templates and external tools like `wkhtmltopdf`.

6. Individual Contributions

Team Lead: Muhammad Ahmed Qazi (CS-24045)

- ➔ **Overall system design and architecture**, including file structure, modularisation, and flow between Flask routes and Python classes.
- ➔ **Development of RentalSystem core logic**, session management, data coordination, and exception handling.
- ➔ **Integration of PDF and CSV generation**, flash alerts, and full-stack reservation flow.
- ➔ **Developed and styled all major frontend pages** using Bootstrap (e.g. hero, registration, login, dashboards, car listings).
- ➔ **Implemented final integration** between car classes, file handlers, Flask, and Jinja templates.
- ➔ **Created final testing scripts, README, and documentation**.

Class Models and Testing: Mujtaba Jawaid Rao (CS-24047)

- ➔ **Designed and implemented all major Python class models**, including `Car`, `EconomyCar`, `LuxuryCar`, `CommercialCar`, `User`, `Admin`, `Person`, and `Reservation`.
- ➔ **Helped define and implement inheritance and abstract classes**, mapping functions and data serialisation.
- ➔ **Designed test cases** for class methods, including rental calculations, car filtering, and to/from dictionary conversion.
- ➔ **Coordinated edge case validations** in user and car behaviour during development.

Fleet Model and Testing: Muhammad Zain Rizvi (CS-24048)

- ➔ **Implemented the `Fleet` class** to manage the car inventory: add, remove, search, and store car records.
- ➔ **Connected the fleet logic** to the admin dashboard and ensured seamless integration with car modals (add/edit/delete).
- ➔ **Tested car-related logic thoroughly**, including VIN management, feature mapping, and availability status handling.
- ➔ **Contributed to the report generation logic** for the admin report section (reserved cars and customers).

7. Potential Improvements for the Future

While the current system is feature-rich and fully functional, there are several directions in which it can be expanded and improved in future iterations:

- ➔ **Online Payment Gateway Integration**
  Integration with actual payment providers (e.g., Stripe or PayPal) to simulate real-world transactions with credit/debit card validation.

- ➔ **Email Notifications**
  Automated email confirmations for reservations, account creation, password resets, and car return summaries.

- ➔ **Role-Based Access and Permissions**
  Fine-grained user role management allowing multiple admin levels, auditors, or support staff access.

- ➔ **Car Pickup/Drop-off Scheduling**
  Incorporate user preferences for pickup/drop-off points and scheduling with route mapping.

- ➔ **Feedback and Rating System**
  Allow users to submit reviews and ratings for vehicles and rental experience.

- ➔ **Mobile Responsiveness and PWA Support**
  Enhance mobile usability and consider converting the system into a Progressive Web App (PWA) for offline access.

- ➔ **Enhanced Analytics and Reporting**
  Visual reports with charts and graphs for revenue, car utilisation, and user behaviour analytics.

- ➔ **Security Improvements**
  Implement JWT tokens, CSRF protection, and robust input sanitisation for enhanced application security.

- ➔ **Hourly Rentals and Dynamic Pricing**
  Add support for flexible billing models such as hourly rentals, promotional pricing, or seasonal rates.

- ➔ **Database Integration**
  Migrate from JSON file storage to SQL/NoSQL databases (like SQLite or MongoDB) for better performance, scalability, and queryability.

8. Test Cases

Test Case 1: Successful Car Reservation

☑ **Purpose:** Verifies that a user with a valid license, sufficient balance, and no active reservations can successfully book a car.

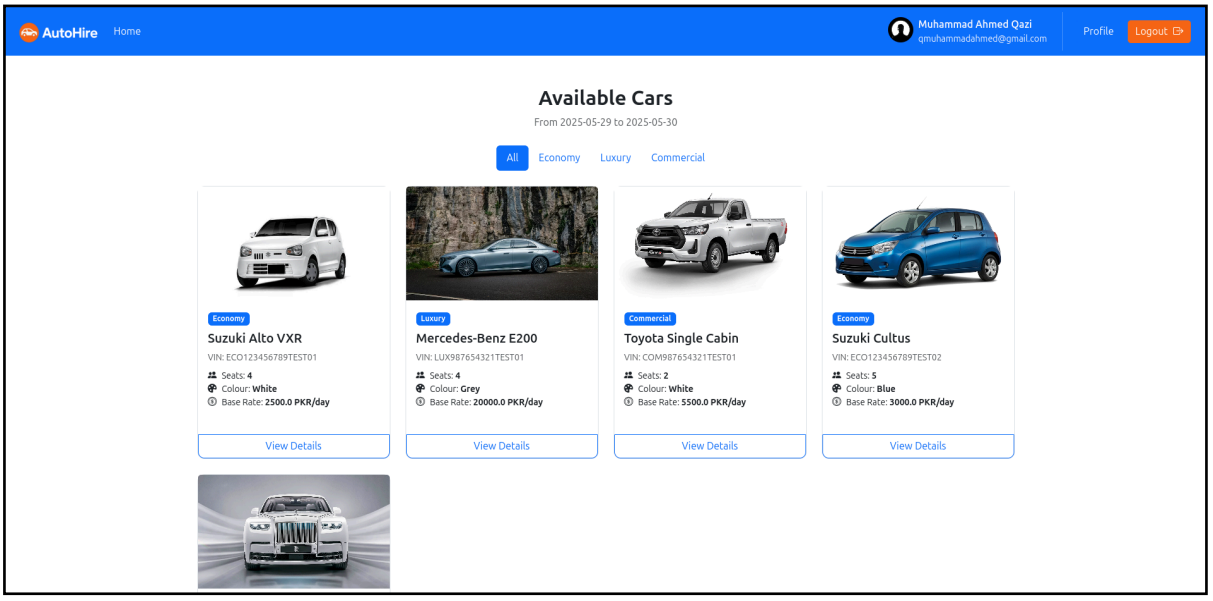


Fig 1. Car listing before reservation

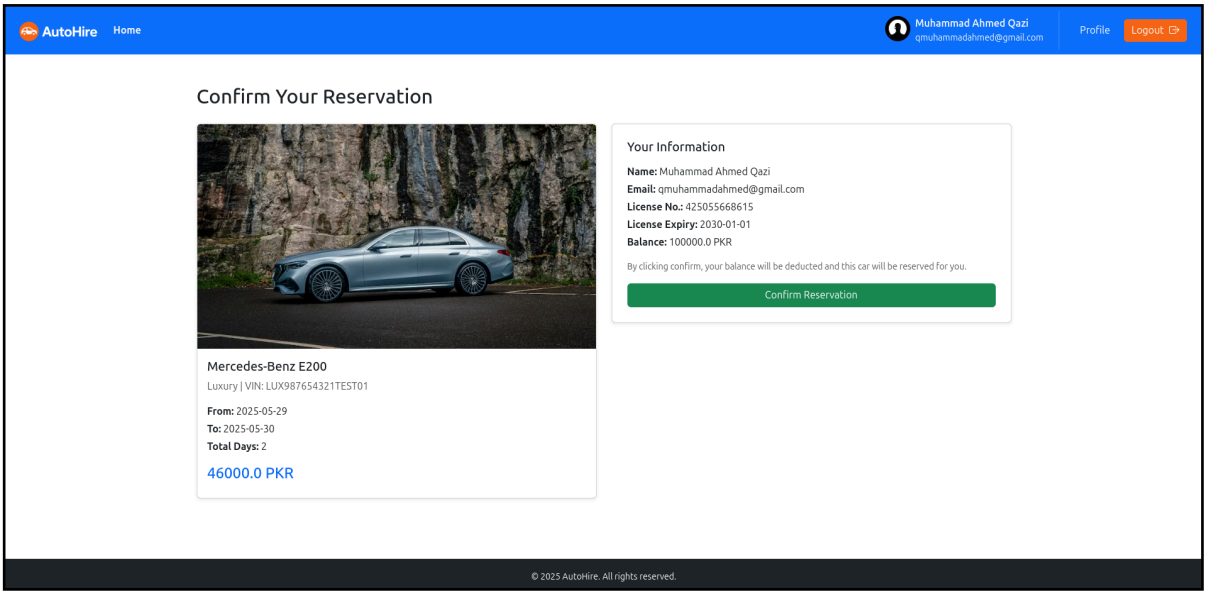
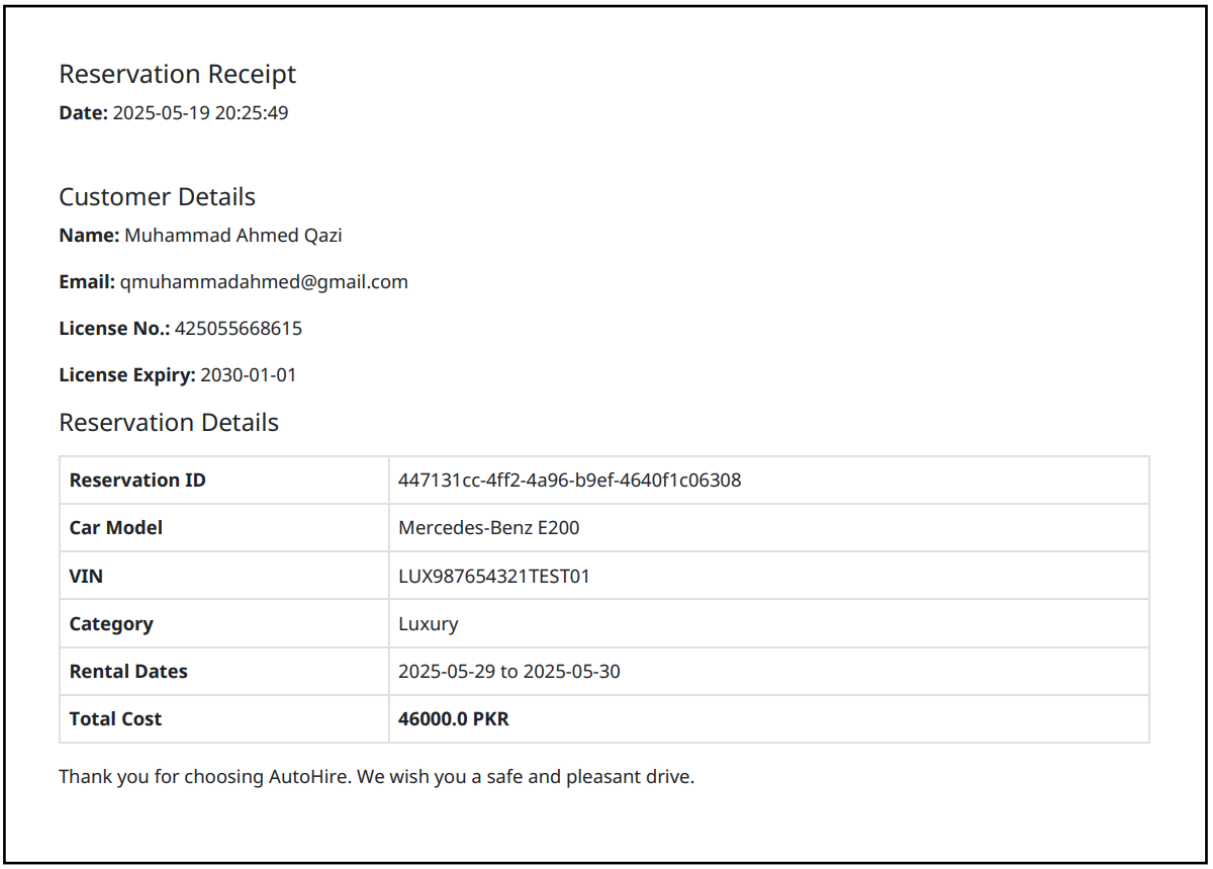


Fig 2. Reservation confirmation stage



Reservation Receipt

Date: 2025-05-19 20:25:49

Customer Details

Name: Muhammad Ahmed Qazi

Email: qmuhammadahmed@gmail.com

License No.: 425055668615

License Expiry: 2030-01-01

Reservation Details

Reservation ID	447131cc-4ff2-4a96-b9ef-4640f1c06308
Car Model	Mercedes-Benz E200
VIN	LUX987654321TEST01
Category	Luxury
Rental Dates	2025-05-29 to 2025-05-30
Total Cost	**46000.0 PKR**

Thank you for choosing AutoHire. We wish you a safe and pleasant drive.

Fig 3. Downloaded receipt preview

Test Case 2: Adding Car to Rental Fleet

☑ **Purpose:** Demonstrates admin's ability to manage the rental fleet through the fleet management dashboard.

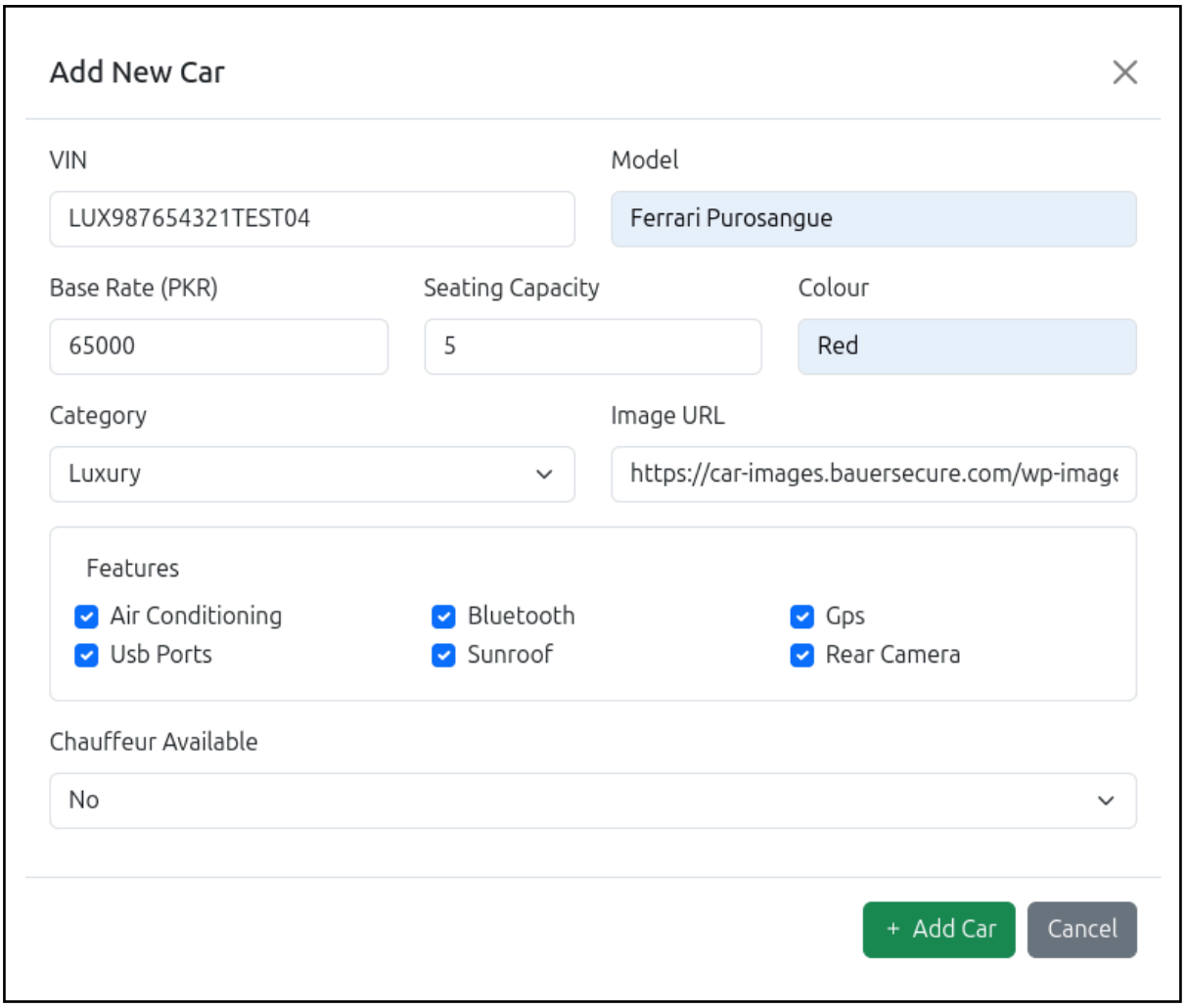


Fig 4. Form to add a new car

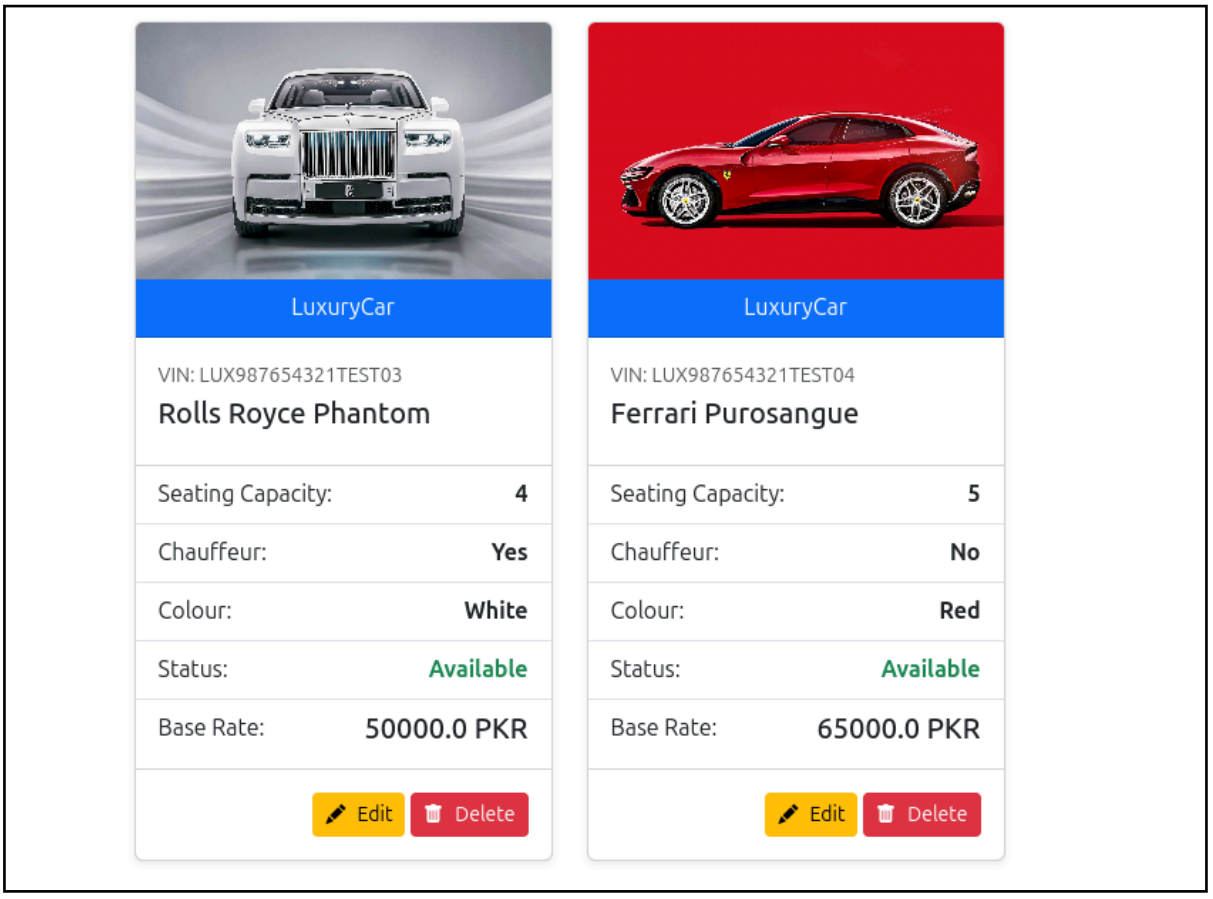


Fig 5. New car added to list of cars in the rental fleet

Test Case 3: Admin Report Generation

☑ **Purpose:** Validates the admin's ability to vew system-wide data and generate downloaded reports.

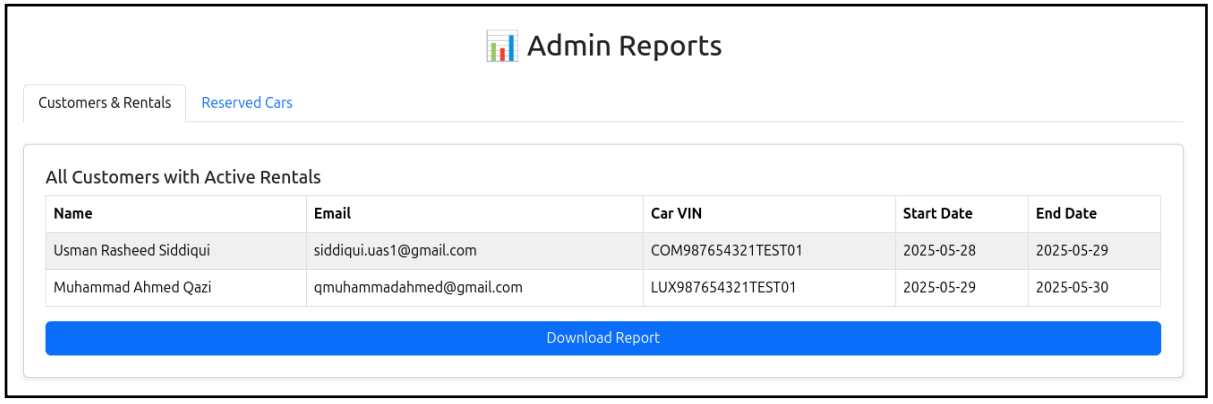


Fig 6. Admin report dashboard showing customers with active rentals

	A	B	C	D	E	F
1	Name	Email	Car VIN	Start Date	End Date	
2	Usman Rasheed Siddiqui	siddiqui.uas1@gmail.com	COM987654321TEST01	2025-05-28	2025-05-29	
3	Muhammad Ahmed Qazi	qmuhammadahmed@gmail.com	LUX987654321TEST01	2025-05-29	2025-05-30	
4						
5						

Fig 7. Report downloaded in .csv format

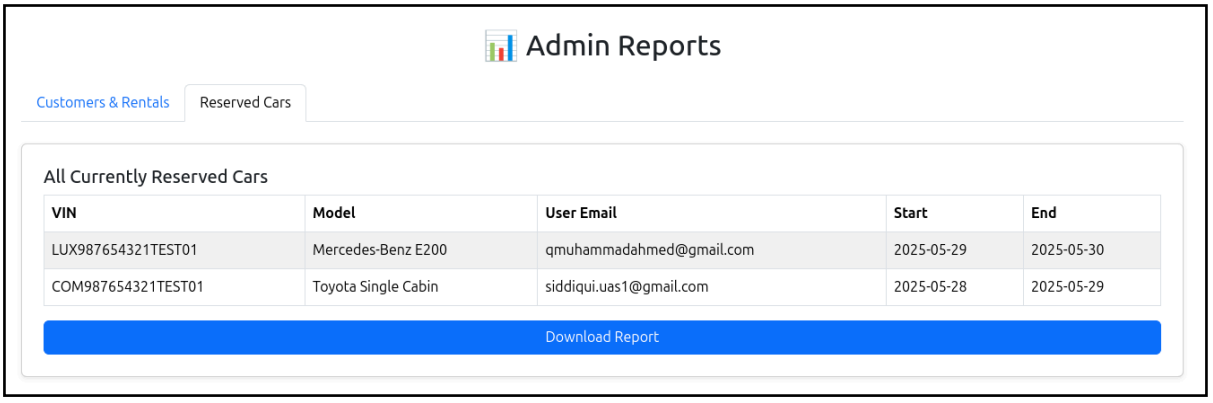


Fig 8. Admin reports dashboard showing cars reserved

	A	B	C	D	E	F
1	VIN	Model	User Email	Start Date	End Date	
2	LUX987654321TEST01	Mercedes-Benz E200	qmuhammadahmed@gmail.com	2025-05-29	2025-05-30	
3	COM987654321TEST01	Toyota Single Cabin	siddiqui.uas1@gmail.com	2025-05-28	2025-05-29	
4						
5						

Fig 9. Report download in .csv format

9. References

Python Official Documentation
– https://docs.python.org/3/

Flask Documentation
– https://flask.palletsprojects.com/

Bootstrap Documentation
– https://getbootstrap.com/docs/5.3/

Jinja Templating
– https://jinja.palletsprojects.com/

Wkhtmltopdf for PDF generation
– https://wkhtmltopdf.org/

Stack Overflow
– https://stackoverflow.com/

Scan to access repository containing all code, files, and documentation