Car Rental System – Project Report

Programme

Master of Software Engineering

Course

MSE800 Professional Software Engineering

Assignment 1

Object-Oriented Programming Assignment

Author

Yi Peng

270648387

Date

7/9/2025

Introduction

1.1 Background & Problem Context

Traditionally, many car rental operations rely on manual processes such as paper-based reservations, direct phone bookings, or spreadsheets for record-keeping. These approaches are error-prone, inefficient, and prone to data loss. For example, double bookings may occur, and administrators must manually enforce rental rules, leading to inconsistent customer experiences. To address these challenges, the project aims to design an automated car rental system that digitizes booking workflows and provides a reliable platform for managing cars, users, and rentals.

1.2 Objective & Scope of the System

The objective of this project is to deliver a robust, user-friendly rental management system that:

- Automates manual booking processes.
- Uses object-oriented programming (OOP) and proven design patterns for maintainability.
- Supports key operations: car listing, booking, rental tracking, and admin approvals.
- Provides a foundation for scalability (future database support, GUI).

The scope includes:

- A command-line interface (CLI) for customers and administrators.
- In-memory and CSV-based data storage for initial implementation.
- Clear separation of user roles and permissions.

1.3 Assumptions, Constraints, and Stakeholders

- Assumptions: Users interact via CLI; CSV serves as the initial data source; administrators enforce policies manually.

- Constraints: Current version uses in-memory lists (no persistent DB), limited GUI support, and basic input validation.
- Stakeholders: Customers (end-users who rent cars), administrators (who manage bookings, vehicles, and policies), and developers/maintainers.

2. Requirements Analysis

2.1 Functional Requirements

The system provides the following functions:

a. User authentication and login (admin/customer).

b. Add, view, and manage cars.

c. Book cars with rental period validation.

d. Admin approval/rejection of pending requests.

e. Return cars and update availability.

f. Display rental history and reminders.

g. Enforce min/max rental days.

h. Generate unique rental IDs automatically.

2.2 Non-Functional Requirements

- Security: Role-based access (admin vs customer), password verification.
- Performance: Quick response for core CLI operations.
- Usability: Simple menu-driven navigation.
- Maintainability: OOP structure, design patterns, clear coding standards (DRY, SRP).
- Scalability: Future support for databases, GUIs, and APIs.

2.3 User Roles & Permissions

- Customer: Login, view cars, create bookings, return cars, view reminders and history.
- Admin: Approve/reject rentals, manage car records, view pending requests, enforce business policies.

3. System Design & Architecture

3.1 Architectural Overview

Object-Oriented Programming (OOP): The system strictly follows OOP principles, abstracting real-world entities (such as users, vehicles, and rentals) into independent classes. This makes the code structure clear, easy to maintain, and scalable.

Abstraction: In models.py, the User class is defined as an abstract base class (via abc.ABC). It specifies a common interface for all user roles, such as the get_username() and check_password() methods. It also contains an abstract method display_menu(), which forces all subclasses (Admin and Customer) to implement their own menu display logic, thereby achieving abstraction.

Inheritance: Both the Admin and Customer classes inherit from the User class. They inherit common attributes (such as username and password) and methods from User, while also adding their own unique behaviors (for example, the administrator’s menu and the customer’s menu differ).

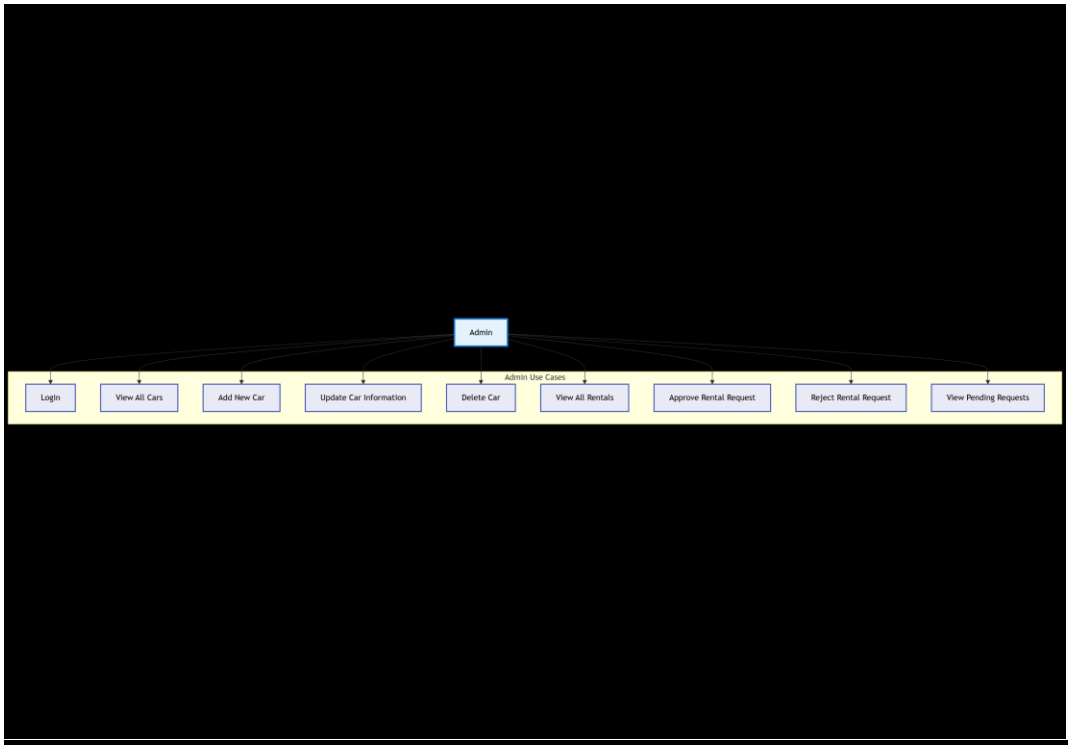
Encapsulation: All classes in models.py (User, Car, and Rental) apply encapsulation well. Their internal data (such as _username, _mileage, _rental_id, etc.) is marked as “protected” (by using a leading underscore), and public getter methods (e.g., Car.get_mileage() and Rental.get_details()) are provided to control access and modification of this data, thus ensuring data integrity.

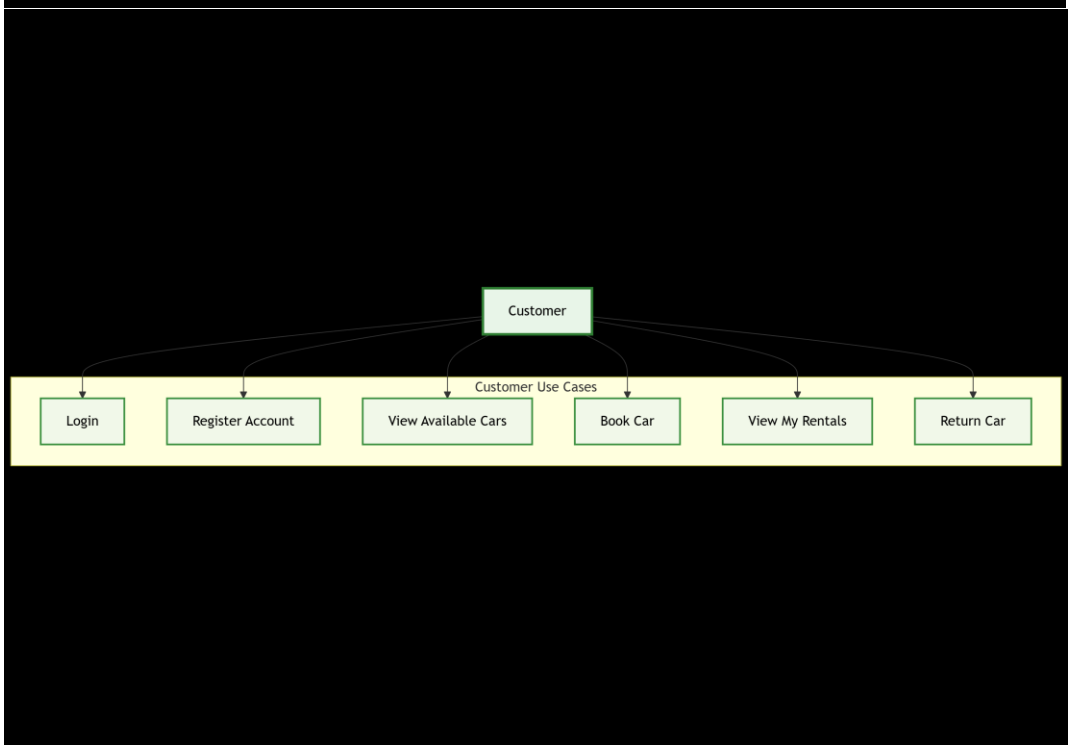
3.2 Design Patterns Selection & Justification

This project is mainly designed undet the Singleton Pattern. In system.py, the CarRentalSystem class is designed as a singleton. By checking in the __new__ method whether an instance already exists, it ensures that only one system instance runs throughout the application. This is well-suited for managing shared resources (such as the vehicle list and rental records) and avoids data inconsistency issues.

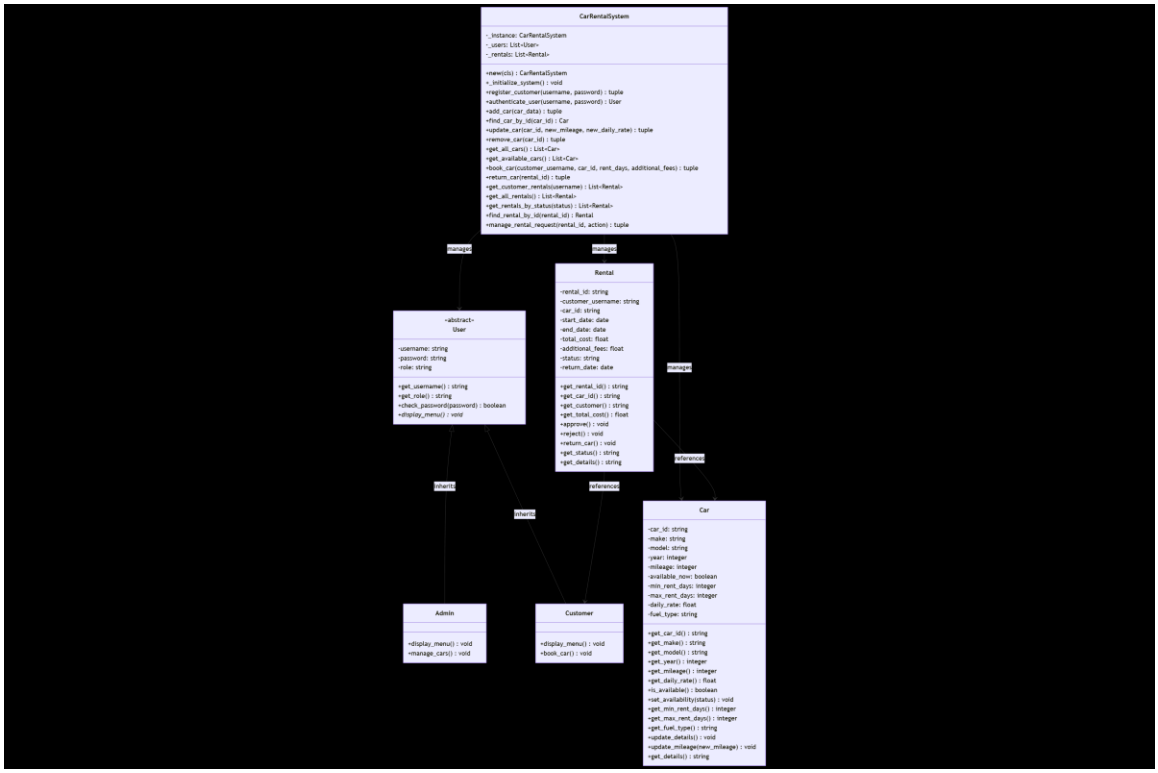
3.3 UML Set

- Use-Case Diagram(Admin and Customer separated to improve readability) :

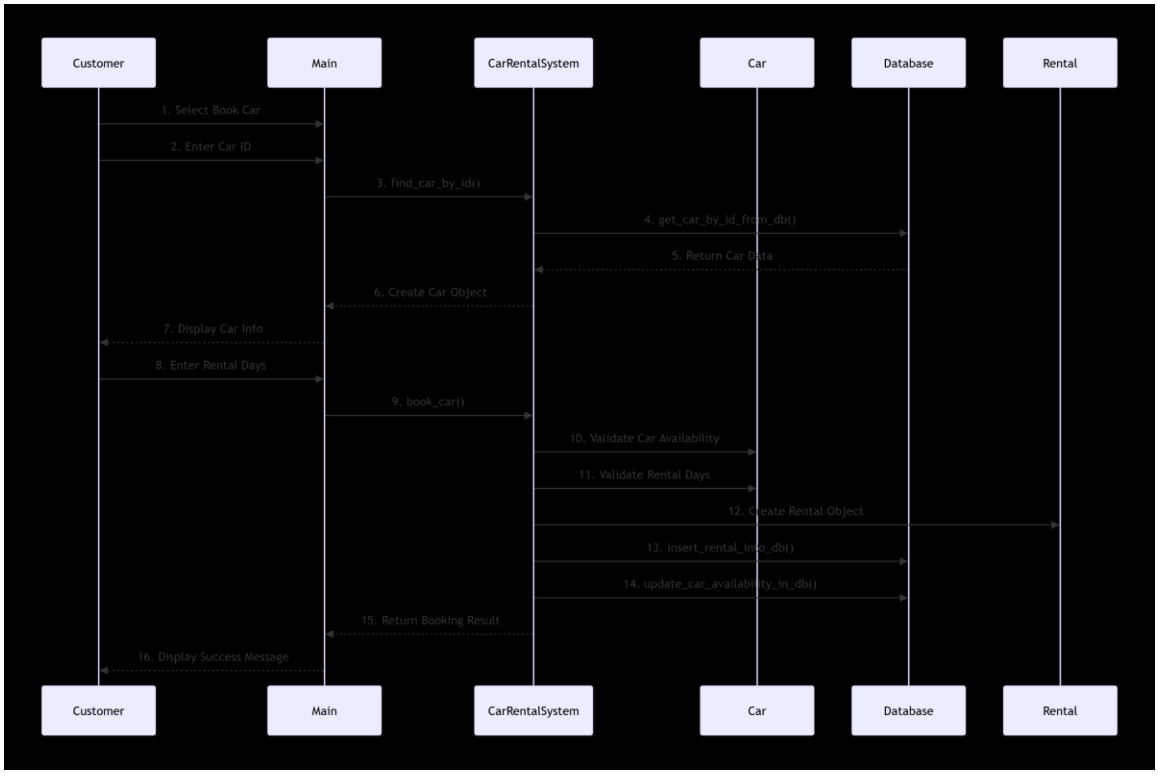




- Class Diagram:



- Sequence Diagram:



3.4 Data Model

- Entities: User, Car, Rental.

- Schema (CSV initial): car_id, make, model, year, mileage, available_now, min_rent_days, max_rent_days, daily_rate, fuel_type.

3.5 API & Module Interfaces

- CLI Services:
  - Login menus (display_menu()).
  - Booking functions.
  - Approval and return flows.
- Modules:
  - models.py – User, Car, Rental classes.
  - system.py – CarRentalSystem singleton logic.
  - utilities.py – Validation helpers.
  - main.py – CLI interaction.
  - database.py (planned future).

3.6 Security Model

- Authentication (authn): User login with username/password.

- Authorization (authz/RBAC): Admin vs Customer menus.

- Input Validation: Centralized in utilities.py.

- Future: Password hashing and DB-backed authentication.

4. User Documentation (ReadMe Summary)

Please refer to the ReadMe.md file in the source code folder.

5. Innovative Feature

The project introduces an admin-in-the-loop approval workflow and contextual reminders, setting it apart from traditional CRUD rental apps:

- Human-in-the-loop: Admin reviews each booking and approves/rejects it. Car availability updates accordingly. This prevents fraud and enforces policy.
- Reminders: Customers receive reminders of rental status; admins see pending requests.
- Constraints enforcement: Min/max rental days enforced per car, reducing invalid bookings.

Together, these enhancements improve trust, control, and policy compliance, showing innovation beyond basic student projects.