YOOBEE COLLEGE OF CREATIVE INNOVATION

Car Rental Management System

S M Shawon

270702380

Professional Software Engineering

MSE800

Submitted to: Mohammad Norouzifard

September 21, 2025

Abstract

The Car Rental System is a comprehensive software application designed to streamline vehicle rental operations using Python and MySQL. Leveraging Object-Oriented Programming (OOP) principles such as encapsulation, abstraction, inheritance, and polymorphism and the Model-View-Controller (MVC) architecture, the system promotes modularity, scalability, and maintainability. Core functionalities include secure user registration, role-based authentication, and dynamic car inventory management, automated rental booking with fee calculations, and administrative tools for oversight. This report outlines the project's objectives, system design, implementation details, testing methodologies, and potential future enhancements, demonstrating a robust, efficient solution tailored for contemporary car rental businesses.

Table of Contents

1. Introduction

In the evolving transportation industry, car rental systems require automation to enhance efficiency and user experience. Traditional manual processes often lead to errors and de- lays. This Car Rental System addresses these challenges by providing a digital platform for customers to browse and book vehicles, while administrators manage inventory and bookings. Developed with Python and MySQL, it leverages OOP (encapsulation, abstraction, inheritance, and polymorphism) and MVC architecture for maintainability.

2. Objectives of the Project and Modules

2.1 Objectives

The primary goals of this project are to automate and optimize car rental operations through:

(a) Secure user registration and role-based access control to protect sensitive data.
(b) Dynamic vehicle inventory management for real-time updates.
(c) Automated rental fee calculations and status tracking to minimize manual errors.
(d) Ensuring data integrity and persistence via MySQL database integration.
(e) Delivering scalable and maintainable code using OOP principles and MVC architecture.

2.2 Admin Module

(a) **User Management:** View, modify, or delete user accounts.
(b) **Car Management:** Add, update, or delete vehicles, including details like ID, make, model, year, mileage, availability, minimum/maximum rent periods, and daily rate.
(c) **Rental Management:** Approve or reject booking requests, update vehicle availability, and monitor ongoing rentals.

2.3 Customer Module

(a) **Registration:** Create new accounts with unique usernames, hashed passwords, and assigned roles.
(b) **Login:** Secure authentication using credential verification.
(c) **Browse Vehicles:** View a list of available cars with detailed specifications.
(d) **Profile:** Manage personal information and view booking history (implicitly tied to user ID).

2.4 Rental Module

(a) **Booking:** Select a vehicle, specify rental dates, and submit a booking request.
(b) **Fee Calculation:** Automatically compute total cost based on daily rate and rental duration.
(c) **Status Tracking:** Monitor booking status (pending, approved, rejected) and update vehicle availability accordingly.
(d) **Integration:** Seamlessly connects users, cars, and rentals for cohesive operations.

3. System and Software Specifications

3.1 Hardware Requirements

(a) **Processor:** Intel Core i5 or equivalent (2.5 GHz or higher).

(b) **RAM:** 8 GB or more.

(c) **Storage:** 256 GB SSD or higher.

(d) **Network:** Stable internet connection (required for potential cloud deployment).

(e) **Input/Output Devices:** Keyboard, mouse, and monitor.

3.2 Software Requirements

(a) **Operating System:** Windows 10/11, Linux (Ubuntu 20.04 or later), or macOS.

(b) **Programming Language:** Python 3.8 or higher.

(c) **Database:** MySQL 8.0 or higher.

(d) **Libraries:** mysql-connector-python for database connectivity, hashlib for password hashing, datetime for date handling, and abc for abstract base classes.

(e) **Development Tools:** PyCharm or Visual Studio Code for coding, MySQL Workbench for database management.

4. System Design and Development

The system employs MVC architecture to separate concerns: Models handle data logic and database interactions, Views manage user interfaces, and Controllers orchestrate business logic. OOP principles are applied throughout to promote code reusability and extensibility.

4.1 Data Flow Diagram

Level 0: Depicts high-level interactions where Customers and Admins interface with the Car Rental System. Inputs include registration details, booking requests, and management commands; outputs are confirmations, reports, and updated statuses.

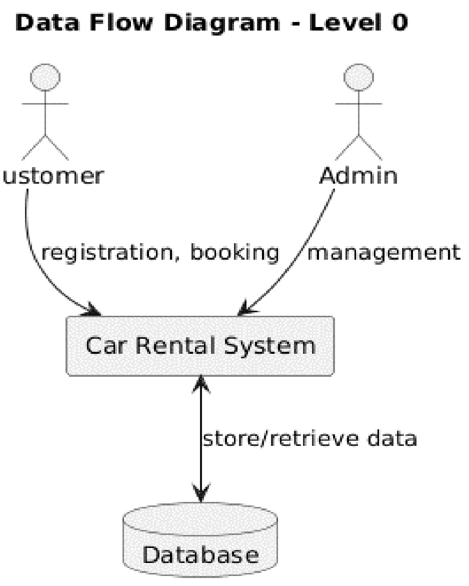


Figure 1

Figure 1: This diagram shows how Customers and Admin interact with the Car Rental System through registration, booking, and management processes.

Level 1: Breaks down into processes like Authentication, Car Management, Booking Processing, and Approval. Data flows to and from stores such as Users, Cars, and Rentals databases.

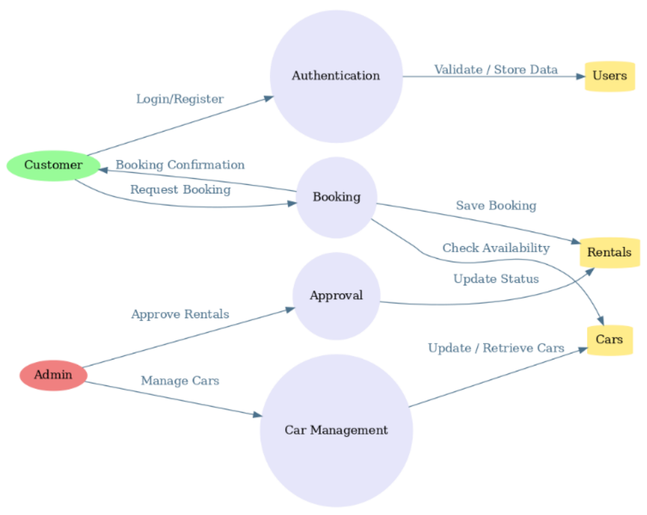


Figure 2

Figure 2: This diagram illustrates the internal processes of the Car Rental System, including Authentication, Car Management, Booking, and Approval, along with data stores (Users, Cars, and Rentals).

4.2 Use-Case Diagram

(a) **Actors**: Customer, Admin, System.

(b) **Use Cases**:

(i) **Customer:** Register, Login, View Available Cars, Book Rental.
(ii) **Admin:** Login, Manage Cars, and Manage Rentals, View Users.
(iii)**System:** Perform Database Operations (extends all use cases).

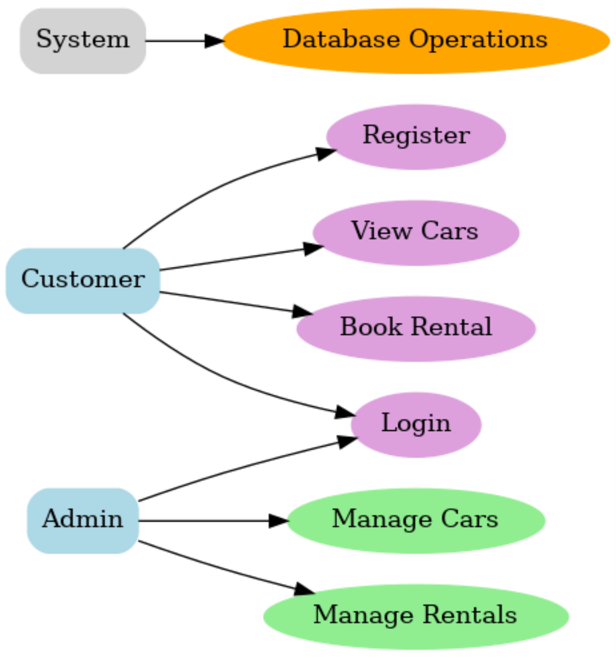


Figure 3

Figure 3: This diagram presents the functional requirements, showing how Customers, Admin, and the System interact with various use cases such as Register, Login, View Cars, Book Rentals, and Manage Cars/Rentals.

4.3 Class Diagram

(a) **Key Classes:**
(i) DatabaseConnection (abstract base class for database operations).
(ii) UserModel, CarModel, RentalModel (inherit from DatabaseConnection; handle CRUD operations).
(iii)UserController, CarController, RentalController (aggregate models for business logic).
(iv) View (handles user interface rendering).
(v) CarRentalSystem (main class orchestrating the application).

(b) **Relationships**:

(i) Inheritance between models and Database Connection; Aggregation between controllers and models.

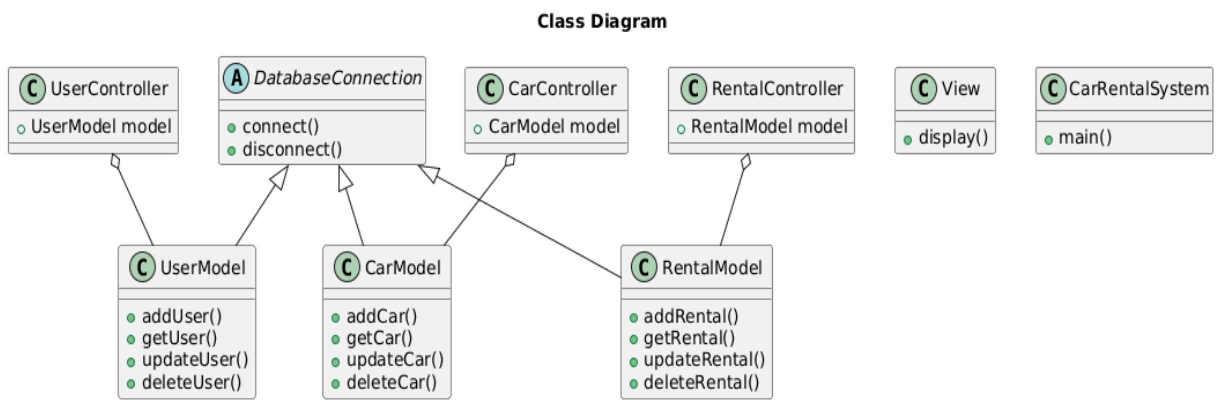


Figure 4

Figure 4: This diagram describes the system's structure using classes, their attributes, and relationships. It demonstrates inheritance from Database Connection and aggregation between Controllers and Models.

4.4 Sequence Diagram (For Booking Process)

(a) Customer logs in via View.
(b) View calls *UserController* to authenticate.
(c) *UserController* queries *UserModel*, which interacts with the database.
(d) Upon success, Customer requests a rental via View.
(e) View passes request to *RentalController*.
(f) *RentalController* verifies availability via *CarModel*, calculates fees, and updates *RentalModel*.
(g) System confirms booking and updates View.

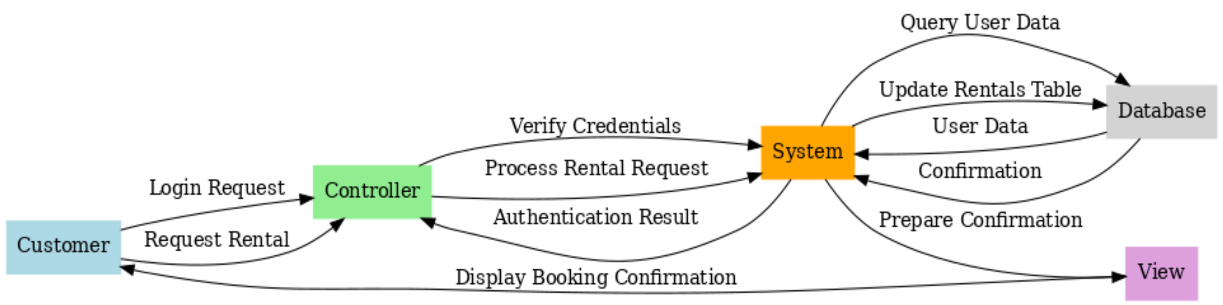


Figure 5

Figure 5: This diagram outlines the booking sequence, where a Customer logs in, requests a rental, and the system verifies details, updates the database, and confirms the booking via the user interface.

4.5 ER Diagram

Entities and relationships can be represented as follows –

(a) Entities:

(i) **User**: id (PK), username (VARCHAR), password (VARCHAR), role (ENUM: 'admin', 'customer').

(ii) **Car**: id (PK), make (VARCHAR), model (VARCHAR), year (INT), mileage (FLOAT), available (BOOLEAN), min_rent_period (INT), max_rent_period (INT), daily_rate (DECIMAL).

(iii)**Rental**: id (PK), user_id (FK to User), car_id (FK to Car), start_date (DATE), end_date (DATE), status (ENUM: 'pending', 'approved', 'rejected'), total_cost (DECIMAL).

(b) Relationships:

(i) User (1) rents Rental (M)

(ii) Car (1) is rented by Rental (M)

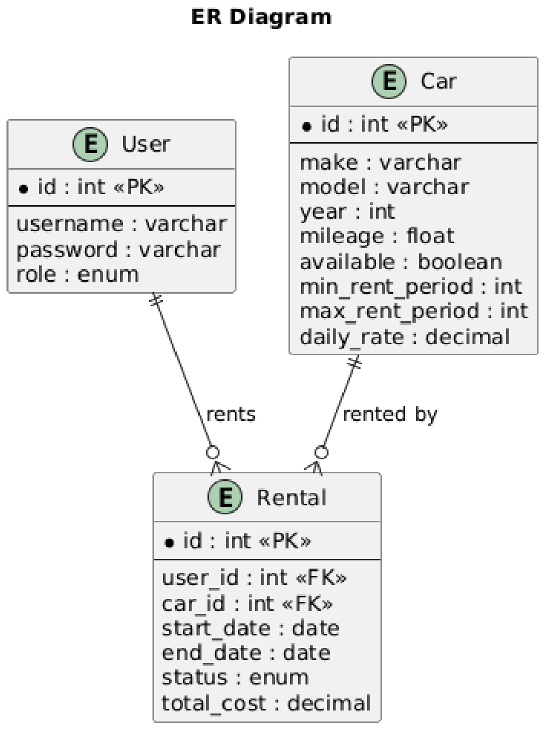
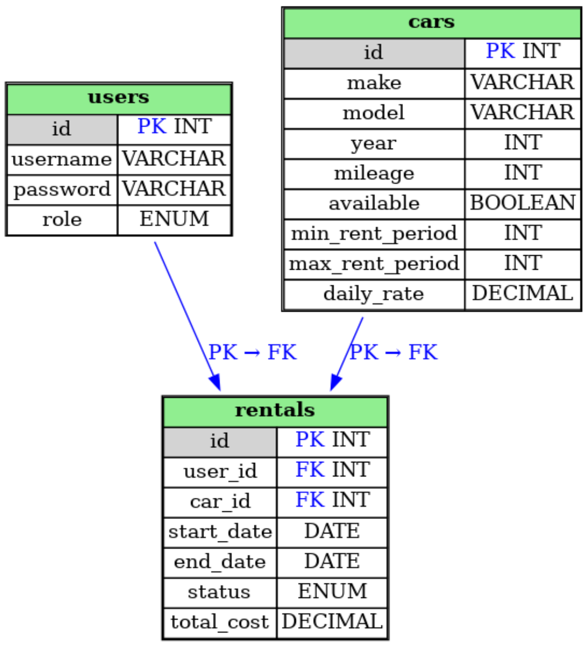


Figure 6

Figure 6: This diagram represents the logical database model with entities (User, Car, and Rental) and relationships: one User can make many Rentals, and one Car can be linked to many Rentals.

4.6 Table Structure



users	
id	PK INT
username	VARCHAR
password	VARCHAR
role	ENUM

cars	
id	PK INT
make	VARCHAR
model	VARCHAR
year	INT
mileage	INT
available	BOOLEAN
min_rent_period	INT
max_rent_period	INT
daily_rate	DECIMAL

rentals	
id	PK INT
user_id	FK INT
car_id	FK INT
start_date	DATE
end_date	DATE
status	ENUM
total_cost	DECIMAL

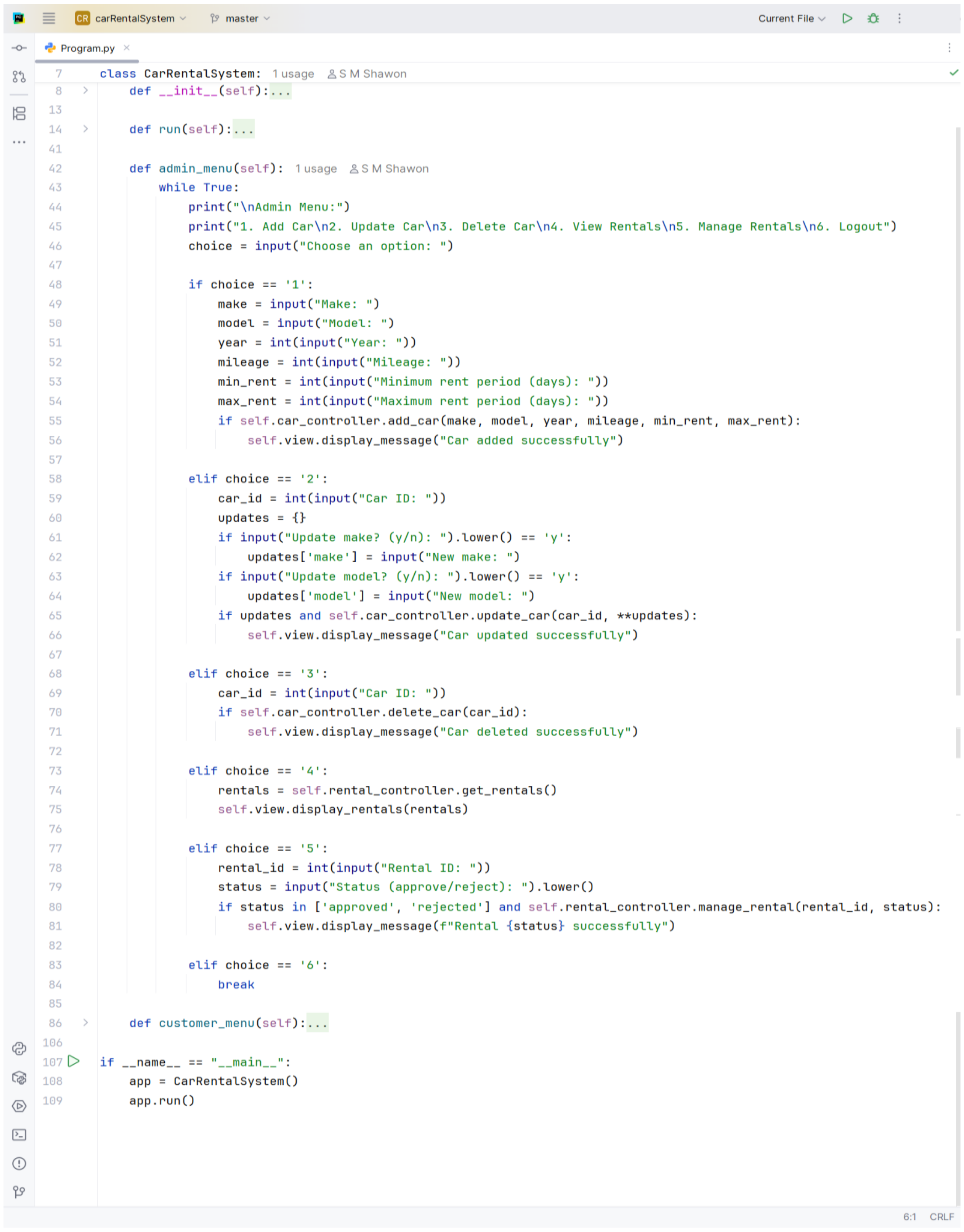
<u>Figure 7</u>

Figure 7: This diagram shows the physical database schema, including the structure of tables (users, cars, and rentals), primary keys, foreign keys, and data types.

4.7 Code Sample

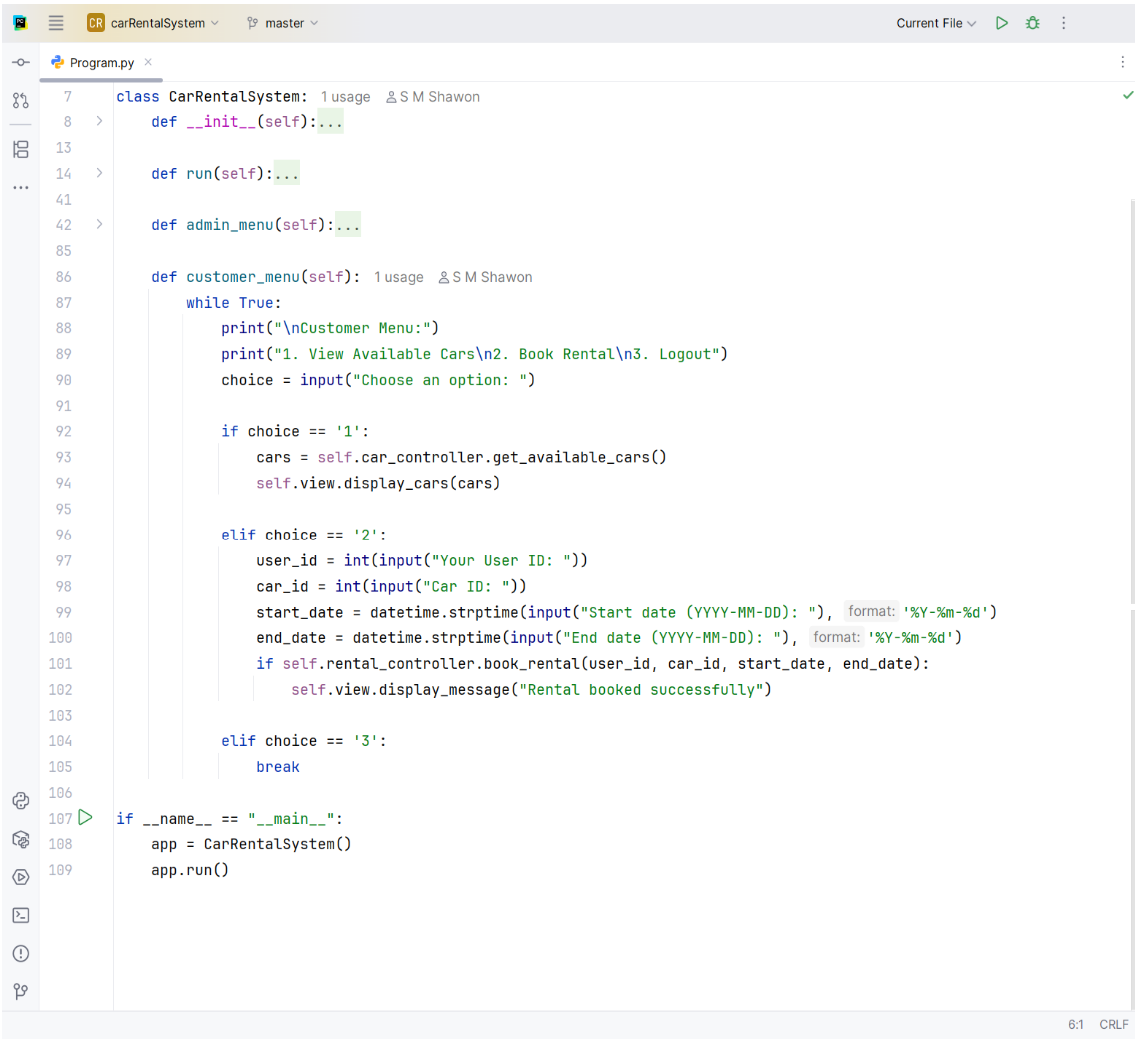
This subsection has sample code snippets that demonstrate the integration of crucial functionalities within the system modules (Customer, Admin, and Rental). The examples hold crucial logic and interaction with the database, showing how the design of the system is translated into functional code. The samples are categorized by module to depict their role in enabling the operations of the system.

4.7.1 Admin Module Code



```python
class CarRentalSystem:
    def __init__(self):...

    def run(self):...

    def admin_menu(self):
        while True:
            print("\nAdmin Menu:")
            print("1. Add Car\n2. Update Car\n3. Delete Car\n4. View Rentals\n5. Manage Rentals\n6. Logout")
            choice = input("Choose an option: ")

            if choice == '1':
                make = input("Make: ")
                model = input("Model: ")
                year = int(input("Year: "))
                mileage = int(input("Mileage: "))
                min_rent = int(input("Minimum rent period (days): "))
                max_rent = int(input("Maximum rent period (days): "))
                if self.car_controller.add_car(make, model, year, mileage, min_rent, max_rent):
                    self.view.display_message("Car added successfully")

            elif choice == '2':
                car_id = int(input("Car ID: "))
                updates = {}
                if input("Update make? (y/n): ").lower() == 'y':
                    updates['make'] = input("New make: ")
                if input("Update model? (y/n): ").lower() == 'y':
                    updates['model'] = input("New model: ")
                if updates and self.car_controller.update_car(car_id, **updates):
                    self.view.display_message("Car updated successfully")

            elif choice == '3':
                car_id = int(input("Car ID: "))
                if self.car_controller.delete_car(car_id):
                    self.view.display_message("Car deleted successfully")

            elif choice == '4':
                rentals = self.rental_controller.get_rentals()
                self.view.display_rentals(rentals)

            elif choice == '5':
                rental_id = int(input("Rental ID: "))
                status = input("Status (approve/reject): ").lower()
                if status in ['approved', 'rejected'] and self.rental_controller.manage_rental(rental_id, status):
                    self.view.display_message(f"Rental {status} successfully")

            elif choice == '6':
                break

    def customer_menu(self):...

if __name__ == "__main__":
    app = CarRentalSystem()
    app.run()
```

4.7.2 Customer Module Code



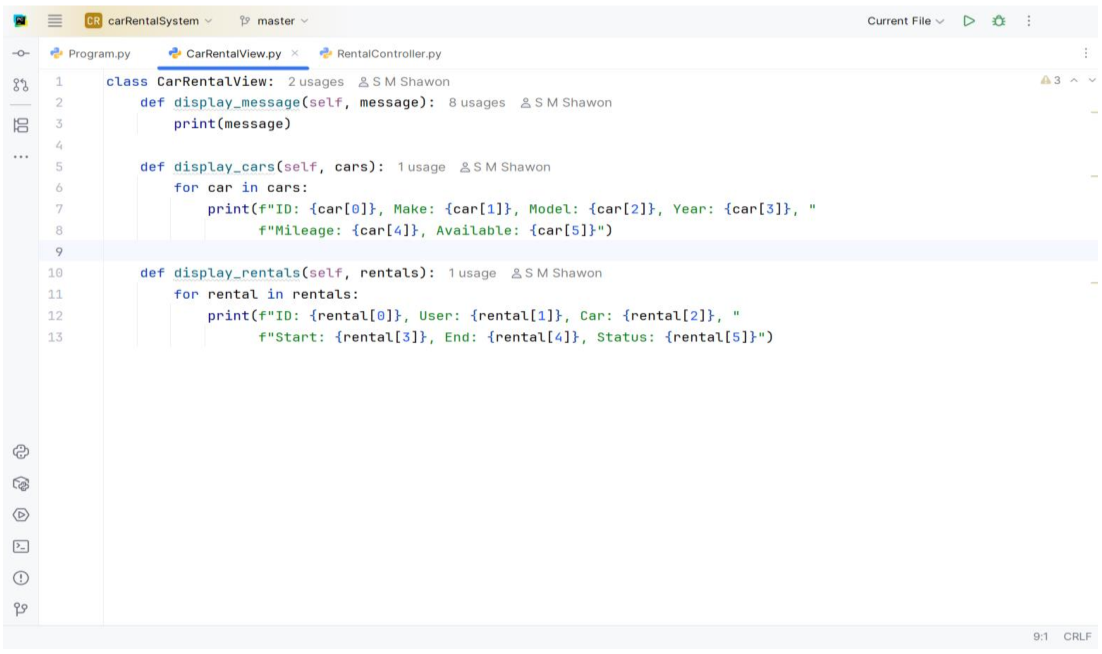
carRentalSystem master Current File

Program.py

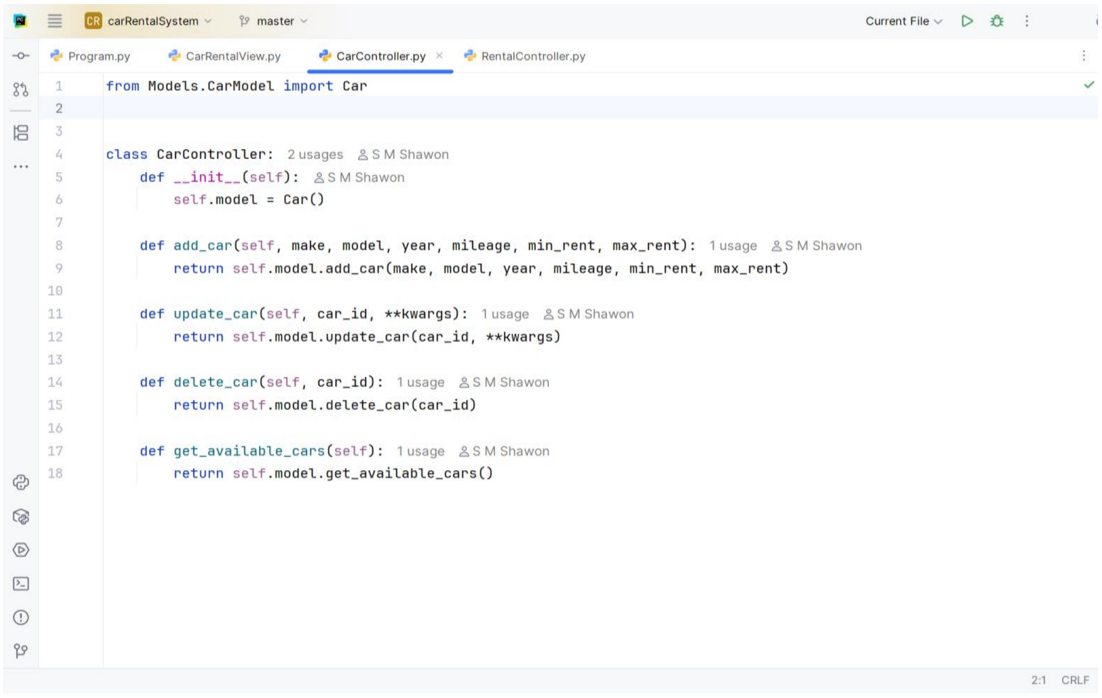
```
class CarRentalSystem:  1 usage  S M Shawon
    def __init__(self):...

    def run(self):...

    def admin_menu(self):...

    def customer_menu(self):  1 usage  S M Shawon
        while True:
            print("\nCustomer Menu:")
            print("1. View Available Cars\n2. Book Rental\n3. Logout")
            choice = input("Choose an option: ")

            if choice == '1':
                cars = self.car_controller.get_available_cars()
                self.view.display_cars(cars)

            elif choice == '2':
                user_id = int(input("Your User ID: "))
                car_id = int(input("Car ID: "))
                start_date = datetime.strptime(input("Start date (YYYY-MM-DD): "),  format: '%Y-%m-%d')
                end_date = datetime.strptime(input("End date (YYYY-MM-DD): "),  format: '%Y-%m-%d')
                if self.rental_controller.book_rental(user_id, car_id, start_date, end_date):
                    self.view.display_message("Rental booked successfully")

            elif choice == '3':
                break

if __name__ == "__main__":
    app = CarRentalSystem()
    app.run()
```

6:1 CRLF

4.7.3 Rental Module Code



```python
class CarRentalView:
    def display_message(self, message):
        print(message)

    def display_cars(self, cars):
        for car in cars:
            print(f"ID: {car[0]}, Make: {car[1]}, Model: {car[2]}, Year: {car[3]}, "
                  f"Mileage: {car[4]}, Available: {car[5]}")

    def display_rentals(self, rentals):
        for rental in rentals:
            print(f"ID: {rental[0]}, User: {rental[1]}, Car: {rental[2]}, "
                  f"Start: {rental[3]}, End: {rental[4]}, Status: {rental[5]}")
```



```python
from Models.CarModel import Car


class CarController:
    def __init__(self):
        self.model = Car()

    def add_car(self, make, model, year, mileage, min_rent, max_rent):
        return self.model.add_car(make, model, year, mileage, min_rent, max_rent)

    def update_car(self, car_id, **kwargs):
        return self.model.update_car(car_id, **kwargs)

    def delete_car(self, car_id):
        return self.model.delete_car(car_id)

    def get_available_cars(self):
        return self.model.get_available_cars()
```

4.8 GitHub Repository

The complete source code for the project, including all modules and their implementations, is hosted on GitHub.

Access the repository at: https://github.com/shawon2jg/CarRentalSystem for a comprehensive view of the system’s codebase, configuration files, and documentation.

5. System Testing and Implementation

5.1 Implementation

The system is implemented as a Command-Line Interface (CLI) application in Python (file: car_rental_system.py). It automatically creates database tables upon initialization, uses SHA-256 hashed passwords for security, and provides role-based menus for intuitive navigation.

5.2 Testing

(a) **Unit Testing:** Individual methods like login(), book_rental(), and approve_rental() were tested using Python's unittest framework.

(b) **Integration Testing:** End-to-end workflows, such as Register → Login → Book Rental → Approve, were validated.

(c) **Functional Testing:** Verified core features with sample data, ensuring customers can book and admins can approve without issues.

(d) **Non-Functional Testing:** Assessed performance with over 100 records, security through input validation and authentication checks.

(e) **Edge Cases:** Handled scenarios like invalid dates, overlapping bookings, and unavailable vehicles with appropriate error messages.

6. Scope of Future Enhancements

(a) **Graphical User Interface (GUI):** Transition to a web-based interface using Flask or Django for broader accessibility.

(b) **Advanced Features:** Integrate payment gateways (e.g., Stripe), email/SMS notifications, and GPS tracking for vehicles.

(c) **Security Upgrades:** Implement JSON Web Tokens (JWT), Two-Factor Authentication (2FA), and robust input sanitization.

(d) **Analytics:** Add dashboards for revenue tracking, usage statistics, and predictive insights.

(e) **Mobile Integration:** Develop Android and iOS applications for on-the-go access.

(f) **Scalability:** Deploy on cloud platforms like AWS or Azure with caching mechanisms for high traffic.

7. Conclusion

The Car Rental System provides a secure, efficient, and user-friendly platform built on Python, MySQL, OOP, and MVC principles. It fulfills all outlined objectives, with rigorous testing confirming its reliability and performance. By addressing key industry challenges and incorporating modern trends, this system serves as a solid foundation for rental services. Future enhancements, such as GUI integration and analytics, will further enhance its scalability, positioning it as a competitive model in the growing car rental market.

References

[1] Internal project code and MySQL documentation.