CAR RENTAL SYSTEM

|  |  |
| --- | --- |
| SUBMITTED BY | MICHAEL OLEFANE TENORIO |
| STUDENT NUMBER | 270647234 |
| SUBMITTED TO | MOHAMMAD NOROUZIFARD |

# SUMMARY

This CLI-based Car Rental System is a terminal application developed in Python using Object-Oriented Programming principles. Designed for use in a single-branch car rental company, the system features three user roles (Admin, Staff, and Customer) and handles key operations such as car management, customer records, user management, and rental transactions. It employs a modular structure with SQLite for data persistence, robust error handling, and clean navigation using libraries like termcolor and prettytable. Key design patterns used include Singleton (for database connection) and Factory Method (for role-based login).

## FEATURES OVERVIEW

* **Role-based Login**: Login system for Admin, Staff, and Customers.
* **User Management (Admin)**: CRUD operations on staff users.
* **Car Management (Admin/Staff)**: Add, view, update, and delete cars.
* **Customer Management (Admin/Staff)**: Add, view, update, and delete customer records.
* **Rental Operations**: Rent and return cars, and track history logs.
* **Navigation System**: Clear CLI-based menus with back and exit functionality.
* **Design Patterns**: Implements Singleton for DB connection and Factory Method for user creation.
* **Database Persistence**: Uses SQLite for storing users, cars, customers, and rental logs.

## SYSTEM SETUP & REQUIREMENTS

**Prerequisites**

* Python 3.10 or higher
* SQLite3 (built into Python)
* Terminal/Command Line

**Required Python Packages**

* Termcolor – pip install termcolor
* Prettytable – pip install prettytable

## USER ROLE & PERMISSIONS

| **Role** | **Description** | **Permissions** |
| --- | --- | --- |
| Admin | System controller | Manage users, cars, customers, view rental logs, book and return of car |
| Staff | Handles day-to-day rental operations | View cars and customers, perform rentals |
| Customer | End-user of the rental system | Viewing car availability, booking of car |

## DETAILED FEATURE DESCRIPTIONS

**Admin Features:**

* Add, edit, delete, and view staff users
* Add, edit, delete, and view cars
* Add, edit, delete, and view customer details
* View rental and return logs
* Book a car and process return
* Access to full system dashboard

**Staff Features:**

* View cars
* View customer records
* Rent out cars
* View rental history
* No access to user management

**Customer (Future Implementation):**

* View available cars
* Request rentals
* View personal rental history

## USAGE GUIDE

1. Run the exe file
2. Login

* Default admin and staff login. (admin/admin and staff/staff)
* Enter username and password.
* Role is determined from database.

1. Admin Dashboard

* Manage staff
* Manage cars
* Manage customers
* Manage bookings
* View rental history
* Logout or exit

1. Staff Dashboard

* View cars
* View customers
* Rent/
* View logs
* Logout or exit

1. Rental Flow

* Select car and customer
* Confirm availability
* Log rental details with timestamp

1. Return Flow

* Enter car ID and customer ID
* Calculate return date
* Update car availability

## ERROR HANDLING & TROUBLESHOOTING

* Input validation for all user inputs
* Invalid login attempts show an error
* Prevents duplicate car IDs or customer IDs
* Loops back on incorrect menu input
* Try-except blocks around DB transactions
* Built-in login for starting the program and start adding data.

## VERSION CONTROL& PROGRESS LOGS

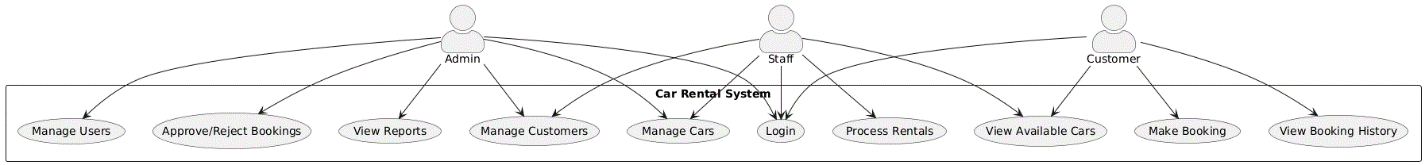
* Git used for versioning (local only)
* Milestones:
  + Week 1: Design and planning
  + Week 2: Core implementation and DB schema
  + Week 3: Completed role-based access and rental module
  + Week 4: Testing, validation, and documentation
* Issues fixed:
* DB singleton not returning connection (resolved)
* PrettyTable alignment issue (fixed with formatting)
* Back navigation added across menus

## FUTURE IMPROVEMENTS

* GUI implementation using Tkinter or PyQt
* Multi-branch support with cloud DB (Firebase or PostgreSQL)
* Mobile version with Flutter or Kivy
* Email/SMS notification on rentals and returns
* Dashboard with charts and reporting tools
* OTP-based login or secure hash for passwords

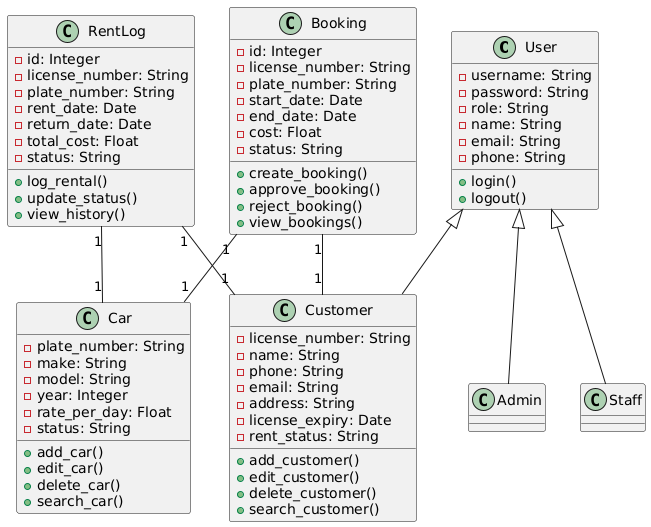
## DIAGRAMS

### Use Case Diagram



The Admin has the most extensive access, with capabilities to manage users, approve or reject bookings, view reports, manage customers, and manage cars. The Staff role focuses on operational tasks such as logging into the system, viewing cars and customers, and processing rentals. Meanwhile, the Customer can interact with the system by logging in, viewing available cars, making bookings, and viewing their booking history. Each actor is linked to specific system functions, illustrating their respective permissions and responsibilities within the system. This diagram provides a clear overview of how different users interact with the system and helps guide the system’s functional design.

### Class Diagram



This defines six main classes: User, Customer, Car, Booking, RentLog, and two specialized subclasses — Admin and Staff.

The **User** class contains attributes like username, password, role, name, email, and phone, with methods for login() and logout(). Both **Admin** and **Staff** inherit from this class, suggesting they share common functionalities like authentication.

The **Customer** class includes personal and license details such as license\_number, name, phone, email, address, license\_expiry, and rent\_status, with methods to manage customer records.

The **Car** class defines each car by plate\_number, make, model, year, rate\_per\_day, and status. It includes methods to add, edit, delete, and search for cars.

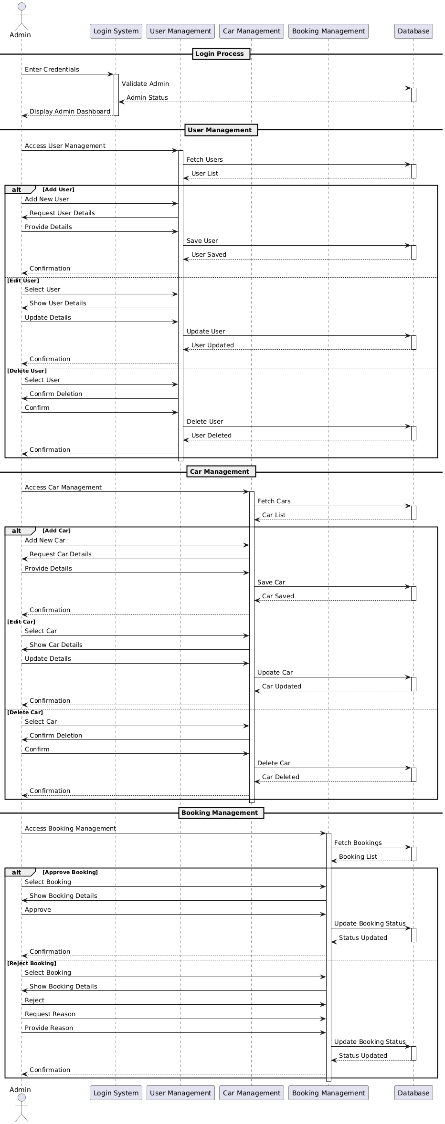
The **Booking** class handles booking records, identified by id, and includes license\_number, plate\_number, start\_date, end\_date, cost, and status. It supports methods for creating, approving, rejecting, and viewing bookings.

The **RentLog** class logs the rental history, tracking attributes such as id, license\_number, plate\_number, rent\_date, return\_date, total\_cost, and status, along with methods to log rentals, update statuses, and view history.

The diagram also shows **1-to-1 relationships** between the entities like Booking, Customer, Car, and RentLog, emphasizing how each booking and rental is associated with a specific customer and car. This diagram is essential for understanding the object-oriented design and data flow in the car rental application.

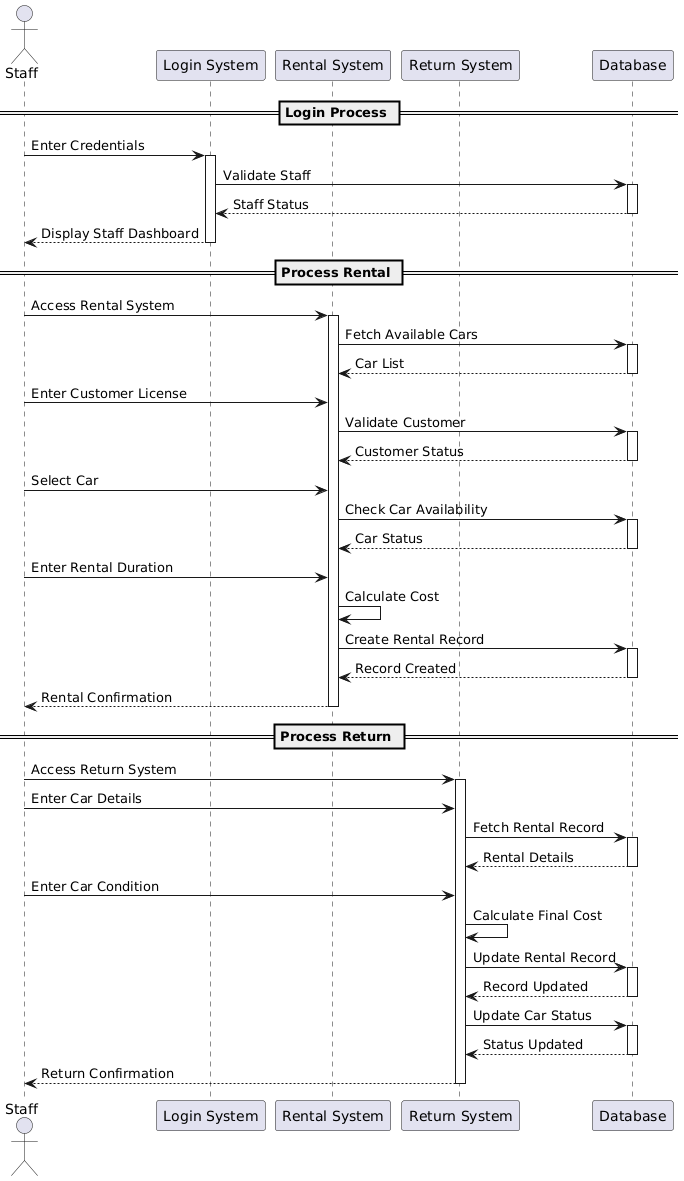
### Sequence Diagram

#### Admin Sequence Diagram



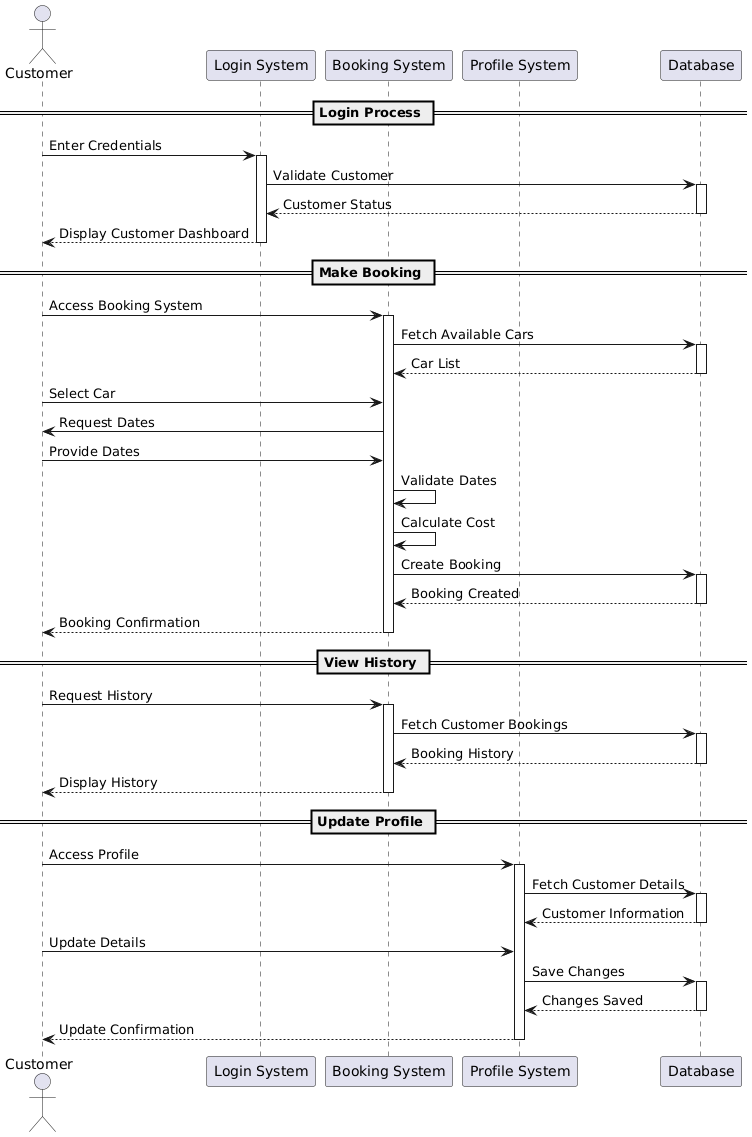
The sequence diagram for the Admin user captures the step-by-step interaction between the Admin and the system components during various administrative tasks. After the Admin logs in with valid credentials, the system authenticates the user and directs them to the Admin Dashboard. From there, the Admin can initiate user management, car management, or booking management operations. In user management, the Admin sends a request to add, edit, or delete a user. The system responds by validating the input, updating the database, and confirming the action. Similarly, in car management, the Admin requests to add or modify car details, and the system processes the input and updates the car records. For booking management, the Admin retrieves a list of pending bookings, selects a booking, and either approves or rejects it. Upon approval, the system updates the booking status and sends a confirmation to the customer; upon rejection, it prompts the Admin to enter a reason, updates the booking as rejected, and notifies the customer. The sequence ends when the Admin logs out, and the session is terminated.

#### Staff Sequence Diagram



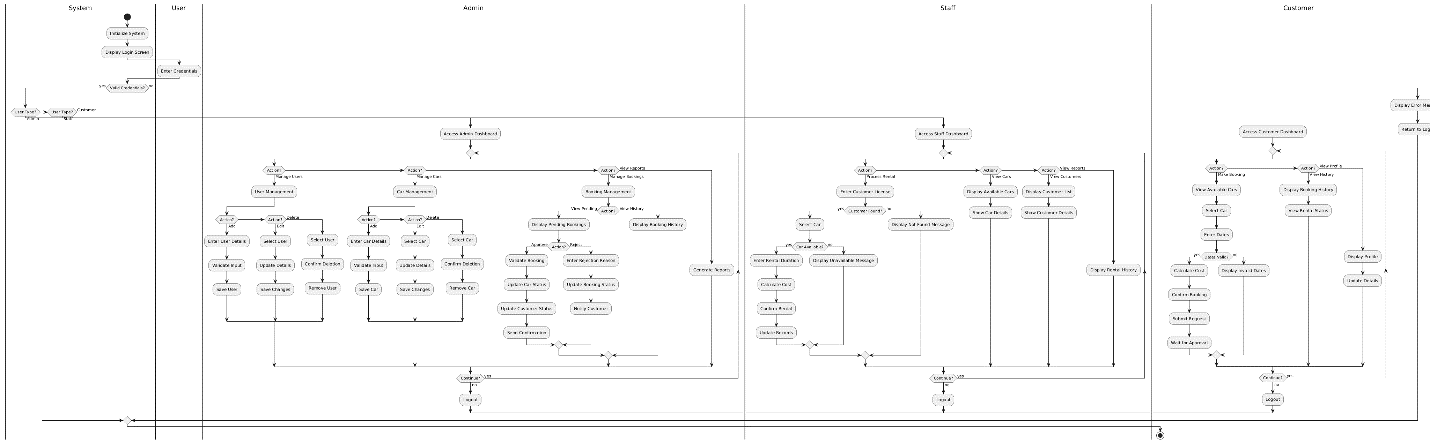
The sequence diagram for Staff shows how staff members interact with the system primarily to process rentals and manage basic car and customer data. After successful login and authentication, the staff member accesses the Staff Dashboard. To process a rental, the staff inputs the customer’s license number, prompting the system to search for the customer in the database. If the customer is found, the staff selects an available car and inputs the rental duration. The system checks car availability, calculates rental costs, and displays it. The staff then confirms the rental, and the system updates the rental records accordingly. The staff can also request to view car lists or customer profiles, to which the system responds with the relevant data. As with the other roles, the interaction ends when the staff member logs out.

#### Customer Sequence Diagram



The Customer sequence diagram illustrates the flow of interactions when a customer accesses and uses the car rental system. The sequence starts with the customer entering their login credentials. Once authenticated, the system grants access to the Customer Dashboard. The customer may initiate a booking by requesting to view available cars. The system responds with a list of cars, and the customer selects a car and inputs rental dates. The system then validates the dates, calculates the cost, and if all inputs are valid, it allows the customer to confirm the booking and submit the request. The system stores the booking as pending and waits for Admin approval. Additionally, the customer can view their booking history, rental status, or update their profile information. All actions involve a series of requests and responses between the customer and the system, ending with the customer choosing to log out.

#### Activity Diagram



This activity diagram illustrates the dynamic workflow of the Car Rental System, outlining the sequence of activities performed by different user roles—Admin, Staff, and Customer—starting from system initialization to the completion of various tasks. The system begins by initializing and displaying a login screen, where users enter their credentials. Once credentials are verified, the user is directed to the appropriate dashboard based on their role: Admin, Staff, or Customer.

For Admin users, the diagram shows access to an admin dashboard with several key functions. These include managing users (adding, editing, or deleting user records with input validation), managing cars (adding, editing, or removing cars), and handling bookings. In booking management, Admins can view pending requests, approve or reject bookings, update booking and customer statuses, and notify customers accordingly. Additionally, Admins have access to reports and booking histories to monitor system activity.

Staff users access their own dashboard, where they can process car rentals by entering a customer's license. If the customer is found in the system, the staff member selects an available car, enters rental duration, calculates the cost, confirms the rental, and updates the records. If the customer is not found or the car is unavailable, appropriate messages are displayed. Staff can also view cars, display their details, and access customer information and rental reports.

Customers interact with the system through a customer dashboard. They can make bookings by browsing available cars, selecting one, entering rental dates, and submitting a booking request. If the dates are valid, the system calculates the cost and awaits admin approval. Customers can also view their rental history and booking status, as well as access and update their profile information.

Each user role concludes their session by choosing to log out, which returns them to the login screen. In cases where invalid login credentials are entered, the system displays an error message. Overall, the activity diagram provides a comprehensive and structured overview of the system’s behavior and the flow of actions for each user role.

## MAINTENANCE AND SUPPORT

#### Maintenance Strategy

To ensure smooth operation and longevity of the Car Rental System, the following maintenance plan is recommended:

* **Database Backup**  
  Regular backups of the SQLite database (car\_rental.db) should be taken, ideally daily or weekly, to prevent data loss.
* **Codebase Updates**  
  Periodic reviews should be done to:
  + Fix bugs
  + Improve performance
  + Update dependencies listed in requirements.txt
* **Data Integrity Checks**  
  Run validation scripts or reports weekly to check for:
  + Orphaned rental logs (missing car/customer links)
  + Duplicate entries
  + Incorrect availability flags
* **User Cleanup**  
  Admin should periodically review inactive staff or customer accounts for possible archiving or deletion.

#### Support Model

Support can be structured into the following tiers:

| **Tier** | **Description** |
| --- | --- |
| **Level 1** | Help with login issues, menu navigation, forgotten credentials |
| **Level 2** | Database troubleshooting, corrupted records, incorrect logic |
| **Level 3** | Core code issues, logic errors, feature malfunctions |

Support can be provided by:

* An assigned **system administrator** for local issues.
* A **developer or maintenance team** for system upgrades and bug fixes.
* **Documentation and logs**, which include:
  + Error messages
  + Logs from database transactions
  + Sample data for testing

#### Testing and QA

Before pushing updates:

* Use test branches with Git
* Run unit tests (to be added in the future)
* Use a sample database to avoid damaging production data

#### Contact Info

For any issues, bugs, or questions about the Car Rental System, you can reach out through email at **270647234@yoobeestudent.ac.nz**.