Car Rental System Report

When I first designed this car rental system, my goal was to create something simple yet practical - something that would actually work in real-world situations. It might not be the biggest or most complex system out there, but it has all the essential features a basic car rental service needs.

After checking out real car rental websites, I noticed two main ways people rent cars:

1. Book online and pay when you pick up the car
2. Book and pay completely online  
   I went with the first option because implementing real payment systems would require bank integrations, so I used mock data for this part instead.

The system has six key components:

 **User Manager** - The brain behind user accounts. It handles:

Registration and login

Profile updates (like adding driver's license info)

Order management

Personal details storage

 **Car Manager** - Keeps track of our vehicle fleet. It's responsible for:

Adding/removing cars from the system

Updating car details (pricing, availability, specs)

Tracking which cars are currently rented

 **Admin Manager** - The control panel for staff. It enables:

Viewing all user accounts

Managing car listings

Moderating reviews

Processing special requests

 **Order Manager** - Handles the rental process end-to-end:

Creating new reservations

Calculating rental durations and costs

Managing pickup/return times

 **Review Manager** - Deals with customer feedback:

Posting new reviews

Calculating average ratings

Moderating inappropriate content

Showing/hiding reviews based on admin settings

 **Database Handler** - The foundation that keeps everything running:

Safely stores all system data

Handles connections between components

Uses singleton pattern to prevent multiple connections

Manages transactions and data integrity

I broke the project into three phases (with maybe an extra one if time allowed). Here's how it went:

**Phase 1:**

* Built the core database
* Created test data in main.py
* Made sure all five main components could access the database
* Implemented basic create/update functions

Then came a tough choice - should I use a framework for the frontend or stick with command line? After researching frameworks (which seemed too complex for this project), I decided on simple HTML/CSS/JS with Flask handling the backend. Frontend work definitely took some getting used to!

**Phase 2:**

* Added login, registration, and car search
* Started building actual web pages (not my strongest skill, I'll admit)
* Implemented ordering, user profiles, order details, etc.
* As features piled up, the workload grew much bigger than expected - some planned features had to be cut

**Final Phase:**

* Implemented singleton pattern for the database class
* Added admin functions
* Shifted focus to testing, bug fixes, and frontend tweaks rather than new features

**How to run it:**

1. Install Flask (just run pip install flask in VS Code terminal)
2. Start the server with python app.py
3. Open your browser to: <http://localhost:5000>

Test account:

* Email: [test13@example.com](https://mailto:test13@example.com)
* Password: test123  
  (Or register a new account)

Admin login:

* Email: [admin@premiumrentals.com](https://mailto:admin@premiumrentals.com)
* Password: securepassword123

Below are screenshots of the main features' interfaces



















