ASSESSMENT - 1

Master of Software Engineering

(Classification - Individual)

Professional Software Engineering

Author Name	**:** Parvez Hasan
Student ID	**:** 270680764
Course	**:** MSE800
Lecturer Name	**:** Mohammad Norouzifard

Date: 13/08/2025

Table of Contents

Abstract

This project delivers a comprehensive **Car Rental System** designed to replace error-prone manual workflows with a reliable, auditable, and easy-to-use application. Built in Python with a MySQL backend, it supports a full rental lifecycle for two roles: **Customer** and **Admin,** including user registration and login (with bcrypt hashing), browsing cars, creating bookings, and validating dates and prices. It also contains admin management to approve/reject bookings, and also a unique QR-code-based pickup/return and payment record feature.

The system follows a clean, layered design (**controllers →models → services → utils**) and applies OOP principles. Targeted design patterns improve maintainability: Factory Method (role-aware user creation), Singleton-style session manager and database config access, and Observer hooks prepared around booking state changes (for future notifications/logging). The data model is normalized and indexed (users, cars, bookings, payments, QR tokens) with data integrity and status transitions that reflect real operations (pending → approved → completed/rejected).

A key innovation is the time-limited QR token generated on approval: at pickup, scanning/entering the token activates the booking and locks the car; at return, it completes the rental and fees inventory, reducing desk time and disputes and can be used for mobile scanning. The system is zero-config to run: requirements.txt, bootstrap scripts (schema + seed), and a PyInstaller release ZIP for one-file distribution.

1. Introduction

1.1 Background & problem context

Most car rental businesses rely on manual paperwork and phone calls. Which is time-consuming and prone to errors. The goal is to automate the end-to-end workflow. From browsing cars to booking approval, pickup/return, and payment while remaining simple enough to run on a single system.

1.2 Objective & scope

Build a robust user-friendly, terminal-based system using Python and MySQL and OOP concepts and design patterns that supports

- ❖ User registration/login for admin and customer
- ❖ View all the cars and an admin CRUD operation.
- ❖ Booking with pricing and availability checks
- ❖ Admin approval/rejection and QR-based pickup/return
- ❖ Recording the payment and marking it PAID

Out of Scope: Creating a Website or Mobile App. and hosting

1.3 Assumptions, constraints, stakeholders

- ❖ Assumptions: Single MySQL database, single-process CLI app, one currency, and daily-based rentals.

- ❖ Constraints: it is a terminal-based UI; the internet is not required; it has minimal dependencies and runs on Windows/macOS/Linux.

- ❖ Stakeholders: Customers, Admin, Business owner and Developer/Maker.

2. Requirement Analysis

2.1 Functional Requirements

- ❖ User Management:

  a. Implement user registration and login functionality.

  b. Differentiate between customer and admin roles, each with specific privileges.

- Car Management:

  c. Create a database of available cars, including their details (ID, make, model, year, mileage, available now, minimum rent period, maximum rent period.)

  d. Allow admins to add, update, and delete car records.

- Rental Booking:

  e. Customers view available cars and details.

  f. Implement a booking feature that allows customers to select a car, specify rental dates, and provide necessary details.

  g. Calculate the rental fees based on the selected car, rental duration, and any additional charges.

- Rental Management:

  h. Allow admins to manage rental bookings, including approving or rejecting requests.

2.2 Non-functional Requirements

- Security: Bcrypt is used for password hashing, role-based access, and accurate input validation.

- Performance: Indexed queries and simple joins and O(1) pricings

- Usability: Clear terminal flows and uses minimal steps

- Maintainability: Layered architecture; services and controllers; patterns used (Factory/Singleton/Observer hooks).

- Reliability: transaction is done where needed constraints are added (FKs, CHECK); seeded data is used for a quick start.

- Portability: Python + MySQL and PyInstalled release.

2.3 Roles & Permissions

- Customer: Can browse cars, book a car, ands view their bookings of whether it is accepted by the admin and then the price is calculated. After the booking is approved, a QR code is generated, and the customer can use it for pickup/return.

- Admin: All of the above, plus manage cars, approve/reject bookings, mark the payment scan pickup/return, delete customers, list customers, and manage all the global booking views.

3. System Design And Architecture

3.1 Architectural Overview

Layers & components

- Controllers: main.py, controllers/*—in this folder and file, there are user flows and enforced sessions and roles.

- Services: services/*-- business login (users, cars, bookings workflow, bookings, payments, QR).

- Models: models/*-- in this folder all the data objects are stored.

- Utils: utils/*—in this folder are all the validation, security , sessions , pricing, and QR helpers.

- Persistence: config/database.py plus ge_connection() – MySQL access information.

- Database: config/car_rental.sql schema and config/seed.sql for demo data.

Key Interactions

1. Customers log in → view cars → create booking (pending status is generated).

2. Admin reviews pending→ approve (creates/updates pending payment and QR token) or reject.

3. Pickup: scan QR → booking becomes active and the cars is locked

4. Return: scan QR → booking completed; car available.

5. Admin can mark the payment as PAID

3.2 Design patterns selection & justification

- ❖ Factory Method (User creation): Construct User instances by role (Customers/Admin); it keeps the role behavior clean and testable.

- ❖ Singleton (for Session Manager / Config): SessionManager to centralize the active sessions, and the database is centralized through one access module to avoid duplication.

- ❖ Observer (hook points): to manage the booking status change Simple hook methods are provided for future extension.

3.3 UML Diagrams

- ❖ Use Case Diagram—
  Actors: Customer and Admin
  Cases: register, login, browse cars, book, approve/reject, view bookings, scan pickup/return, manage cars, record payment.
  Description: The diagram shows what the system should do from the perspective of its users. It includes checks such as validating email format, password strength, duplicate accounts, rental dates, and payments. A customer can register, log in, see available cars, book a car, and make a payment. An administrator can add, update, or delete cars, and the system can either approve or reject bookings. Common checks are reused across different actions, while special cases like not having enough money or payment failure are shown as exceptions. Overall, the diagram gives a simple view of the roles, responsibilities, and actions in the system.

- ❖ Activity Diagram—
  Description: This diagram shows the step-by-step flow of the car rental process. A user signs up or logs in, searches cars, checks availability, picks dates, books, and pays. At key steps the system asks simple yes/no questions like “Is the user valid?”, “Is the car available?”, and “Did the payment work?” Based on the answer, the flow continues to confirmation or stops with a failure message. Admin actions—adding, editing, or deleting cars and approving or rejecting bookings—happen alongside and affect what customers see. It’s a clear picture of how the process starts and ends.

- ❖ Sequence Diagram—

Description: This diagram shows what talks to what, in order. For a customer: log in → search cars → check availability → reserve → pay through the payment service → get confirmation. If the car isn't available or the payment fails, the system returns a clear error. For an admin: log in → update the car list and availability → manage users and bookings. The diagram makes the order of messages and responses easy to follow across the UI, the core system, the car inventory, the payment service, and notifications.

- Class Diagram—
  Description: This class diagram for the Car Rental System shows how the system is organized into different parts that work together. It includes domain classes like User, Customer, Admin, Car, Booking, Payment, and BookingQR, each with their own details such as IDs, dates, amounts, and statuses. It also shows services like BookingService, CarService, UserService, PaymentService, and QRService, which handle the main business logic, such as creating bookings, managing cars, processing payments, and generating QR codes. Utilities like Security, Validators, and Pricing provide support functions for password handling, input checking, and rental fee calculations. Persistence is handled through DBConfig to connect with the database, and session handling is managed with a SessionManager. Overall, the diagram explains how data, services, and helpers are linked, showing both the system's structure and the roles of each part in keeping the car rental process running smoothly.

  [**Note: Diagrams included under System Documentation in Github Readme.md file.**]

3.4 Data model (ERD & schema)

- **Tables:** users, cars, bookings, payments, booking_qr_codes
- **Relationships:**
  - bookings.user_id → users.user_id (CASCADE)
  - bookings.car_id → cars.car_id (RESTRICT)
  - payments.booking_id → bookings.booking_id (CASCADE)
  - booking_qr_codes.booking_id → bookings.booking_id (CASCADE)

[**Note: for full SQL in config/car_rental.sql and demo data in seed.sql find it in Github under Database Schema**]

3.5 API & module interfaces (CLI/services)

- **Controllers**
  - UserController.register_user() / login_user()

        - ➢ CarController.view_available_cars() / book_car() / view_my_bookings() / customer_view_qr()
        - ➢ Admin-only: add_car() / update_car() / delete_car() / list_all_cars()
    - ❖ **Services**
        - ➢ UserService.register_user() / login_user() / list_customers() / delete_customer()
        - ➢ CarService.list_available_cars() / add_car() / update_car() / delete_car() / list_cars()
        - ➢ BookingService.create_booking() / approve_booking() / list_user_bookings() / list_all_bookings()
        - ➢ PaymentService.create_or_update_pending() / mark_paid()
        - ➢ QRService.generate_for_booking() / scan_pickup() / scan_return()

3.6 Security model

- ❖ **AuthN**: password is hashed using bcrypt
- ❖ **AuthZ**: server-side checks in controllers/services and session tokens are bound to user_id.
- ❖ **Validation**: input validators for email/password data parsing min/max days, role checks, SQL constraints & FKs.

4. Implementation

4.1 Tech stack & project structure

- ❖ Python 3.11 plus MySQL , mysql-connector-python, bcrypt, qrcode/qrcode-terminal,
- ❖ The project tree is documented in README; requirements.txt is provided.

4.2 Key classes & modules (OOP)

- User, Car, Booking, and Payment model the domain.
- Services encapsulate business rules; controllers are where the flow starts.
- SessionManager encapsulates session lifecycle.
- pricing.py encapsulates date, math & pricing

4.3 Notable algorithms & data structures

- Pricing: Calculate the duration in days (inclusive of start, exclusive of end) and use fixed fees hooks. Also, there are constraints for min/max periods.
- QR flow: time-limited tokens, which are stored on the server side and in the database, are joined and used to retrieve the booking.

4.4 Configuration & environment

- Configurations are written in requirement.txt on README.

5. Coding Standard and Conventions

- Modularity & Encapsulation: layered architecture only contains service in service methods; there are no databases in controllers. The database file is kept in a config file.
- Performance: Indexed columns (status, dates, availability) only required columns selected when transactions are needed.
- Commenting & Docs: inline docstrings for README and this report
- Indentation & Formatting: Use consistent names for functions or variables and use camel case for classes.
- Naming: clear domain-driven names like approve_booking.

6. Testing Strategy

7. Deployment and Release Build

8. User Documentation (ReadMe Summary)

- ❖ Install: pip install -r requirements.txt

[Note: all details are given in README on GitHub.]

9. Innovative Solution

9.1 Feature Concept

QR-based pickup/return: each approved booking gets a time-limited, opaque QR token. At pickup, admin scans/enters token → booking becomes active and car is locked; at return/pickup, scanning completes the rental and frees the car.

9.2 Architecture & Implementation

To implement, first you have to create a unique constraints inside database like booking_qr_codes(booking_id, qr_token, expires_at,...). Then create a service for QR codes, which will be generated when booking, and then make 2 more services for picking up and returning the car. Then in the UX, a token is generated and displayed as an ASCII QR in the terminal and saved as PNG in qr code folder.

9.3 Industry need & competitive advantage

The main advantage is that it reduces desk time and errors; it also shows a clear status change at physical events, and later it can be used as a mobile scanner and also be easily integrated with email/SMS or displayed on a website. It can also be displayed for a digital invoice.

10. Maintenance and Support Plan

11. Project management and Process (GPO4)

11.1 SDLC

- ❖ Agile: used for small, testable increments and has short feedback cycles.

11.2 Planning & tracking

- For tracking, Github is used by timely committing to the project to see if any issues arise, and if any issue arises, resolve it in the next commit.

11.3 Risk management

- Database corruption → backups & SQL dumps; password security → bcrypt; environmental → requirements.txt

12. Evaluation and Results

12.1 Demo Scenario

- Customer: login → registered user → View cars → Book 3 days → “pending.”
- Admin: View pending → Approve → QR generated → Pickup (active) → Return (completed) → Mark paid

12.2 Performance/Usebility

- Client response instant for typical data sets indexed keep the list fast pand prompts will be shorter and clear.

13. Limitations and Future Works

- No web/mobile interface (CLI only)
- Manual payment marking (no real payment gateway)
- Availability login can be tightened with full overlay checks across dates.
- Notifications (email/SMS) not implemented.
- Multi-branch & dynamic pricing not implemented
- QR code is oly generated when the status is not updated to mark it paid
- Pickup and Return is not created

14. Conclusion

The system delivers a complete and maintainable car-rental workflow aligned with the rubrics for the assessment, OOP plus design patterns , providing a clear layering, QR-based pickup/return, terminal UX and automatic setup (schema/seed) together providing a strong submission-ready project that is easy to run and extend.

[Github link: Car Rental System]

[Project Demo:]