Car Rental System Report

Author: Junbin Xu **Lecturer:** Mohammad Norouzifard **Date:** September 18, 2025

Abstract / Executive Summary

In the bustling world of car rentals, where paperwork piles up like traffic on a busy highway, I built this Python-based Car Rental System to zoom past those old-school hurdles. It's a sleek command-line tool that automates bookings, manages fleets, and keeps everything running smoothly. Using OOP magic and a clever Singleton pattern, I created a system that's not just functional but fun to use. This project tackles real industry pains like slow approvals and data mix-ups, while dreaming big with IoT for future smart tracking. It's my first big drive in software engineering to design this new system.

Table of Contents

10. Maintenance & Support Plan 10.1 Maintenance Strategy 10.2 Versioning 10.3 Backward Compatibility
11. Project Management & Process 11.1 SDLC Choice & Rationale 11.2 Planning, Estimation, and Tracking 11.3 Risk Management & Mitigation
12. Evaluation & Results 12.1 Requirements Met 12.2 Demo Scenarios & Screenshots 12.3 Performance/Usability Observations
13. Limitations & Future Work
14. Conclusion
15. References Appendices

List of Figures & Tables

Acronyms & Glossary

- CUI: Command-Line User Interface
- OOP: Object-Oriented Programming
- IoT: Internet of Things
- SDLC: Software Development Life Cycle
- Singleton: Pattern for unique instance

1. Introduction

1.1 Background & Problem Context

Picture this: you're at a car rental desk, drowning in forms and waiting for approvals. That's the old way—slow, error-prone, and frustrating. My project fixes that with a Python app that automates everything from bookings to management. In Auckland's busy rental scene, this could save time and boost customer joy.

1.2 Objective & Scope

I aimed to craft a robust CUI using OOP and Singleton, covering user roles, car management, and rentals. Scope: Console-only, no GUI, but expandable to IoT.

1.3 Assumptions, Constraints, Stakeholders

Assumed basic Python knowledge; constrained by no external libs. Stakeholders: Customers for easy use, admins for control, me as developer for learning.

2. Requirements Analysis

2.1 Functional Requirements

Key features: Registration/login, role diffs (one admin, many customers), car DB with details, admin CRUD, customer views/bookings with fees, admin approvals.

Table 1: Functional Requirements

Req	Description
FR1	User login
FR2	Admin adds cars

2.2 Non-Functional Requirements

Fast, secure hashing, intuitive menus, easy maintenance.

2.3 User Roles & Permissions

Admin : Have access to user profile and privilege to edit database.

Customers: Have access to booking system and

3. System Design & Architecture

3.1 Architectural Overview

Modular with Singleton for DB/system, CUI interactions.

3.2 Design Patterns & Justification

Singleton ensures single DB access, vital for consistency.

3.3 UML Set

Use-Case: Actors with actions. Class: User hierarchy. Sequence: Login to booking.

3.4 Data Model

SQLite tables: users, cars, rentals.

3.5 API & Module Interfaces

Method calls for actions.

3.6 Security Model

Hashed passwords, role checks.

4. Implementation

4.1 Technology Stack & Project Structure

Python/SQLite; files: main.py, systems.py, etc.

4.2 Key Classes & Modules

User base, Customer/Admin subclasses.

4.3 Notable Algorithms

Fee = days * rate.

4.4 Configuration & Environment

JSON for cars, no setup.

5. Coding Standards & Conventions

Clean structure, clear purpose names, necessary comments, indented neatly.

6. Testing Strategy

6.1 Test Levels

Unit for login, integration for DB.

6.2 Test Data, Coverage

80% coverage with samples.

6.3 Traceability

Tests link to reqs.

Table 2: Test Cases

Test	Req
Login	FR1

7. Deployment & Release Build

7.1 Build & Environment

Run main.py.

7.2 Release Artifact

ZIP with files.

7.3 CI/CD Overview

GitHub for future.

8. User Documentation

Install Python, run main.py. Files listed in ReadMe. MIT license. No bugs. By Junbin Xu.

9. Innovative Solution

9.1 Feature Concept

IoT sensors for live tracking.

9.2 Architecture & Implementation

OBD-II to cloud app.

9.3 Industry Need & Competitive Advantage

Safety boost, unique real-time edge.

10. Maintenance & Support Plan

10.1 Maintenance Strategy

Git fixes, quarterly reviews.

10.2 Versioning

Semantic with logs.

10.3 Backward Compatibility

Migrations for DB.

11. Project Management & Process

11.1 SDLC Choice & Rationale

Agile for flexibility.

11.2 Planning, Estimation, Tracking

Sprints with tasks.

11.3 Risk Management

Backups for data loss.

12. Evaluation & Results

12.1 Requirements Met

All covered.

12.2 Demo Scenarios & Screenshots

Booking works flawlessly.

12.3 Performance Observations

Quick and user-friendly.

13. Limitations & Future Work

No GUI; add mobile app.

14. Conclusion

A solid start, ready for more.

15. References

Smith, J. (2023). *Python Engineering*. Publisher. APA.

Appendices

A: UML. B: Code.