Requirements Document – Car Rental System

# 1. Project Purpose

The Car Rental System project aims to create a robust, object-oriented, menu-driven software application that allows users to manage customers, vehicles, leases, and payments. It simulates real-world car rental operations with a strong focus on database integration, user-defined exceptions, and unit testing.

# 2. Problem/Opportunity Statement

Many car rental companies rely on manual or semi-digital systems that are inefficient and prone to errors. This project addresses the need for an integrated system that can:

* Streamline operations like car tracking, customer management, and lease creation.
* Prevent inconsistencies in data entry and improve user experience.
* Enhance data reliability through unit testing and exception handling.

# 3. Goals and SMART Objectives

* Specific: Develop a car rental system with full CRUD capabilities for customers and vehicles.
* Measurable: Achieve 100% test coverage for all service methods.
* Achievable: Implement the system using Python, SQL, and OOP within the semester timeline.
* Realistic: Ensure seamless database interaction via a utility-based architecture.
* Time-bound: Complete development and unit testing within the stipulated project submission deadline.

# 4. Project Scope

**Inclusions:**

* Customer, Vehicle, Lease, and Payment modules.
* SQL database integration.
* CRUD operations and user-defined exceptions.
* Property-based DB connection utilities.
* Unit test cases for major operations.

**Exclusions:**

* Online payment gateway integration.
* Real-time GPS tracking of vehicles.
* Web/Mobile front-end interfaces.

**Constraints:**

* Must use the directory/package structure defined.
* OOP principles and exception handling must be followed.
* Submission must be done through GitHub with proper documentation.

# 5. Stakeholder Requirements

* Developers: Need a well-structured modular codebase.
* Trainers/Reviewers: Expect robust unit testing and SQL interaction.
* End Users (Simulated): Require intuitive operations like add, search, lease, return.

# 6. Functional Requirements

**Customer Management:**

* Add/Update/Delete Customer
* Retrieve customer details
* Find customer by ID

**Car Management:**

* Add/Remove car
* List available/rented cars
* Find car by ID

**Lease Management:**

* Create/End lease
* List active leases and history
* Calculate cost (daily/monthly)

**Payment Handling:**

* Record payments
* Retrieve payment history
* Calculate revenue

# 7. Non-Functional Requirements

* Must use a layered architecture (DAO, Model, Exception, Utility)
* System must throw user-defined exceptions when appropriate
* Should connect to SQL DB using properties file
* Ensure code is reusable, maintainable, and testable

# 8. Prioritized Requirements

1. Establish SQL database schema and model classes.
2. Implement core DAO functions for customer and car management.
3. Handle leases and payments with exception handling.
4. Write unit tests and validate business logic.

# 9. Suggested Methodology

Agile (Iterative) – As development will involve multiple phases (model creation, DB interaction, testing), Agile fits well. It allows continuous integration and testing.

# 10. Required Project Documentation

* SRS Document – Functional/Non-functional requirements
* Project Charter – Roles, scope, and stakeholder list
* Team Charter – Developer responsibilities (if team-based)
* Exception Handling Summary – Overview of all custom exceptions
* Test Plan – Unit test strategy and coverage report

# 11. Team and Feasibility Highlights

* **Team Size**: Solo
* **Feasibility**: High – Python is a beginner-friendly language with extensive libraries for database interaction, testing, and modular development. The technology stack is lightweight, making it highly feasible for academic and small-scale real-world applications.

**Resources Needed:**

* Python 3.x and standard libraries
* **SQLite** or **MySQL** with **sqlite3** or **mysql-connector-python** for database operations
* **Unittest** or **PyTest** for unit testing
* GitHub for version control
* Optional: **SQLAlchemy** for ORM support
* Optional: **ConfigParser** for property file handling (DB credentials)

**12. Break Down Requirements**

Breaking down the high-level requirements into granular tasks ensures that each module of the system is well-defined, manageable, and easier to implement and test. This phase bridges the gap between the system goals and the actual development process.

**Customer Management**

The system should allow management of customer data efficiently.

* **Add Customer**: Capture full name, email, and phone number. Validate uniqueness based on phone number or email. Auto-generate customerID.
* **Update Customer**: Allow changes to phone number or email. Validate that the customer exists before updating.
* **Delete Customer**: Can only be removed if there are no active leases linked. Prompt warning if dependencies exist.
* **Retrieve Customer**: Fetch and display individual customer details based on customerID. Implement search/filter functionality.
* **List All Customers**: Display a complete customer directory with basic filtering options.

**Car Management**

The system should support operations on the fleet of vehicles available for rent.

* **Add Car**: Capture car make, model, year, status (available/not available), engine capacity, passenger capacity, and daily rate.
* **Remove Car**: Ensure the car is not linked to any active lease before deletion.
* **Update Availability**: Automatically set to notAvailable when leased, and available when returned.
* **List Available Cars**: Filter and show only those cars with status available.
* **List Rented Cars**: Display cars that are currently leased.
* **Find Car By ID**: Retrieve full vehicle information by vehicleID. Throw exception if not found.

**Lease Management**

Leasing is the core operation where a customer books a car for a certain duration.

* **Create Lease**:
  + Validate both vehicleID and customerID.
  + Allow the user to choose lease type (daily/monthly).
  + Calculate lease duration and set start and end dates.
  + Store lease details in the Lease table.
* **Return Car**:
  + Mark lease as completed and change car status to available.
  + Optionally, allow notes on return condition.
* **Calculate Lease Cost**:
  + Daily Lease = days × daily rate
  + Monthly Lease = months × monthly rate (you can estimate monthly rate or use a predefined value)
* **List Active Leases**:
  + Show ongoing leases with customer name, car details, and expected return date.
* **List Lease History**:
  + Show completed leases with payment and duration details.

**Payment Handling**

All financial transactions must be accurately recorded and traceable.

* **Record Payment**:
  + Link payment to a lease using leaseID.
  + Store payment date and amount.
* **Retrieve Payment History**:
  + Filter by customerID or leaseID.
* **Calculate Total Revenue**:
  + Sum of all recorded payments. Can be used to show business performance.

**Utility Modules**

Python utility classes enhance the architecture:

* **Database Connection Utility**:
  + Use ConfigParser to read database connection properties from .ini or .properties file.
  + Provide a static method to return DB connection object using sqlite3 or mysql.connector.
* **Property Utility**:
  + Abstract all DB configuration into a property file to maintain clean separation of concerns.

**Custom Exceptions**

Create and handle domain-specific exceptions:

* **CarNotFoundException**
* **CustomerNotFoundException**
* **LeaseNotFoundException**

These improve error handling and debugging.

**Unit Testing**

Using Python's unittest or pytest frameworks:

* Test customer and car creation functionality.
* Test lease creation logic and cost calculations.
* Validate payment recording.
* Ensure exceptions are thrown when invalid IDs are used.

**13. Prioritize Requirements**

To manage the project efficiently, requirements are prioritized based on their **importance**, **dependency**, and **contribution to the MVP (Minimum Viable Product)**.

**High Priority (Must Have)**

These requirements are essential for core functionality and project success. The system cannot function without them.

* Implement customer and car creation, retrieval, and listing.
* Enable lease creation and return with correct cost calculations.
* Ensure payment recording works and connects with lease data.
* Basic exception handling to manage invalid input scenarios.
* Connect the application to the database and ensure persistence.

**Medium Priority (Should Have)**

These add robustness and completeness to the system but are not critical for MVP.

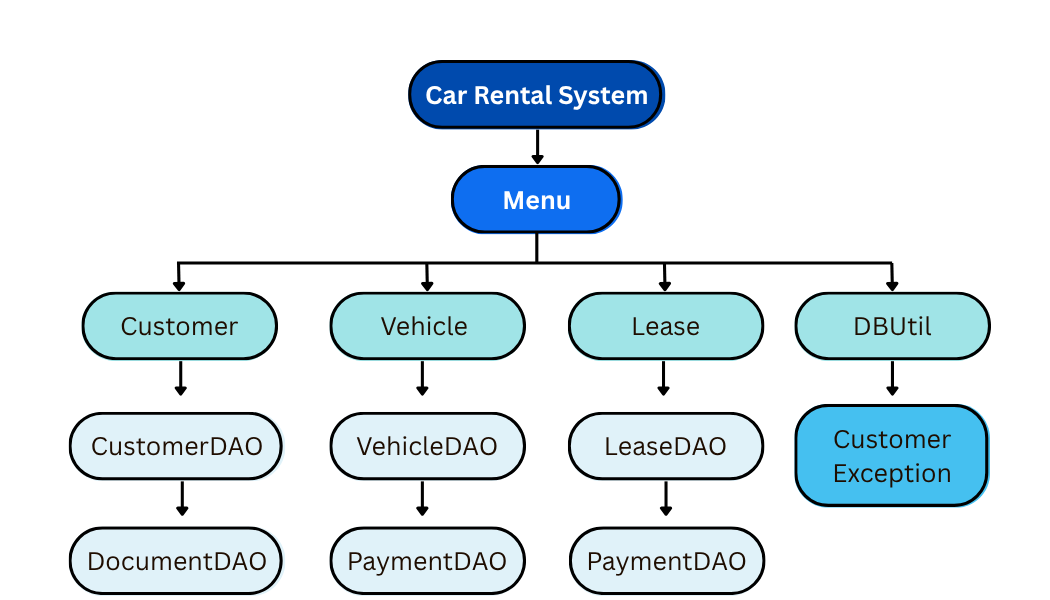
* Updating and deleting customer and vehicle data.
* Lease history display and lease filtering.
* Retrieving payment history by customer.
* Applying layered architecture (DAO, utils) to ensure maintainability.
* Unit testing for all main modules.

**Low Priority (Nice to Have)**

Useful for extended functionality and future improvements. Not required for the initial academic submission.

* ORM integration using SQLAlchemy for object-relational mapping.
* Logging system for tracking operations.
* Admin dashboard-style summaries (revenue reports, active leases).
* SMS/email notifications on lease creation or return.
* Export reports to CSV or PDF.

**REQUIREMENTS BLOCK DIAGRAM**

****