**CASE STUDY ON PROJECT MANAGEMENT SYSTEM**

**1. INTRODUCTION**

**1.1 Purpose**

The purpose of this document is to define the functional and non-functional requirements for the Car Rental System, a console-based Python application. It will guide developers, trainers, and evaluators in building a menu-driven system for managing customers, vehicles, leases, and payments with robust error handling, database persistence, and unit testing.

**1.2 Scope**

The Car Rental System will provide the following capabilities:

* Customer Management: Add, update, remove, list and find customers.
* Vehicle Management: Add vehicles, change availability, list and find vehicles.
* Lease Management: Create daily or monthly leases, calculate costs, return cars, and view active/history leases.
* Payment Handling: Record payments, view payment history, and calculate total revenue.
* Reporting: Generate an Excel test report summarizing unit-test results.
* Database Persistence: Store all data in MySQL with proper schema and utilities.
* Exception Handling: Use custom exceptions for “not found” scenarios.

The system will be developed using Python (backend), MySQL (database), and unit testing frameworks for reliability assurance.

**1.3 Stakeholders**

|  |  |
| --- | --- |
| **Stakeholder** | **Role** |
| Trainer / Evaluator | Reviews and validates functionality and code quality |
| Developer (You) | Implements and tests the system |
| End-User (Clerk) | Uses the CLI to manage rentals |

**1.4 Out of Scope**

* No web or graphical user interface (console only)
* No user authentication or multi-user concurrency
* No integration with third-party payment gateways

**2. FUNCTIONAL REQUIREMENTS**

|  |  |
| --- | --- |
| **ID** | **Requirement Description** |
| FR1 | The system must allow adding a new customer. |
| FR2 | The system must allow updating an existing customer’s email or phone number. |
| FR3 | The system must allow removing a customer (only if they have no leases) and throw CustomerNotFoundException. |
| FR4 | The system must allow listing all customers and finding a customer by ID. |
| FR5 | The system must allow adding a new vehicle with make, model, year, rate, status, capacity, engine specs. |
| FR6 | The system must allow updating vehicle availability (available/notAvailable). |
| FR7 | The system must allow listing available vehicles, rented vehicles, and finding a vehicle by ID. |
| FR8 | The system must allow creating a daily or monthly lease and calculate its total cost. |
| FR9 | The system must allow returning a car (update lease end date and vehicle status) and throw LeaseNotFoundException. |
| FR10 | The system must allow listing active leases (current) and lease history (completed). |
| FR11 | The system must allow recording a payment for a lease and throw an exception if the lease ID is invalid. |
| FR12 | The system must allow viewing payment history for a customer. |
| FR13 | The system must allow calculating total revenue (SUM(amount) from payments). |

**3. NON-FUNCTIONAL REQUIREMENTS**

|  |  |
| --- | --- |
| **ID** | **Requirement Description** |
| NFR1 | Implemented in Python 3.x with MySQL for data storage. |
| NFR2 | Follow OOP principles: entity classes contain data only; use DAO interfaces and implementations. |
| NFR3 | Use custom exceptions (CustomerNotFoundException, etc.) and handle them gracefully in the CLI. |
| NFR4 | Provide comprehensive unit tests using unittest (PyUnit) for both positive and negative scenarios. |
| NFR5 | Generate an Excel report of test results using openpyxl. |
| NFR6 | All database operations must use prepared statements and properly close connections in finally blocks. |
| NFR7 | CLI must be menu-driven with clear prompts and user-friendly error messages. |
| NFR8 | Code must follow PEP-8, use logical package structure, and be easy to maintain. |

**4. FEASIBILITY STUDY**

**4.1 Technical Feasibility**

The Car Rental System uses only free, widely available tools—Python 3, MySQL, and two libraries (mysql-connector-python and openpyxl). There’s no special hardware or paid software needed. Its modular design (entities, DAO, utilities, CLI) means it installs and runs easily on any machine with Python and MySQL.

**4.2 Operational Feasibility**

This is a simple, menu-driven console app for a single clerk. Clear prompts and straightforward options (customers, vehicles, leases, payments) make it easy to learn and use. Because it runs locally, it works even without internet access and fits smoothly into daily workflows.

**4.3 Economic Feasibility**

All components are open source, so there’s no licensing cost. The project reuses standard patterns and libraries, which keeps development time—and therefore cost—low. It’s an affordable solution ideal for training or small operations.

**5. ASSUMPTIONS AND CONSTRAINTS**

**5.1 Assumptions**

* Python 3.x and MySQL Server are installed and configured.
* The user has basic familiarity with console applications.
* The database runs locally; no network deployment is required.

**5.2 Constraints**

* No GUI or web interface—console only.
* No built-in user authentication or role management.
* Designed for single-user use without concurrency controls.6

**6. METHODOLOGY**

We followed an **Agile, iterative** approach, delivering the system in small increments: first defining the database schema and utilities, then building entity and exception classes, implementing the DAO layer, integrating a menu-driven CLI, and finally adding comprehensive unit tests and an automated test-reporting script. Continuous code reviews and demos ensured that each increment met the requirements and maintained high quality throughout development.

### **7. EXCEPTIONS**

The Car Rental System employs a robust, formalized exception handling mechanism to ensure clear feedback and graceful recovery from error conditions. Three domain-specific exceptions are defined:

* **CustomerNotFoundException**  
  Raised when an operation attempts to access or modify a Customer record that does not exist in the database.
* **VehicleNotFoundException**  
  Raised when an operation attempts to access or modify a Vehicle record that does not exist in the database.
* **LeaseNotFoundException**  
  Raised when an operation attempts to access or modify a Lease record that does not exist in the database.

This separation of concerns preserves system stability and provides the end user with immediate, actionable information regarding invalid input or state.