**REQUIREMENTS ANALYSIS DOCUMENT**

**1. INTRODUCTION**

The purpose of this Requirements Analysis Document is to clearly define and analyze the requirements for developing the **Car Rental System**. It breaks down the system’s requirements into manageable units, categorizes them by priority, and ensures that the development adheres to these structured needs. It also assists stakeholders, developers, and evaluators in understanding the system’s capabilities and limitations

## **2. REQUIREMENT BREAKDOWN**

The requirements of the Car Rental System are divided into **Functional Requirements** and **Non-Functional Requirements** for clarity.

**2.1 FUNCTIONAL REQUIREMENTS**

|  |  |
| --- | --- |
| **ID** | **REQUIREMENT** |
| **FR1** | **Add Customer**  • customerID (auto-generated)  • firstName, lastName  • email, phoneNumber |
| **FR2** | **Update Customer Information**  •Modify existing customer’s email or phoneNumber |
| **FR3** | **Remove Customer**  • Delete by customerID, if no leases exist  • Throw CustomerNotFoundException otherwise |
| **FR4** | **List & Find Customers**  • List all customers  • Find customer by ID or throw CustomerNotFoundException |
| **FR5** | **Add Vehicle**  • make, model, year, dailyRate, status, passengerCapacity, engineCapacity |
| **FR6** | **Update Vehicle Availability**  •Change status between available / notAvailable |
| **FR7** | **List & Find Vehicles**  • List available vehicles  • List currently rented vehicles (join on Lease)  • Find by ID or throw VehicleNotFoundException |
| **FR8** | **Create Lease**  • Daily or Monthly  • Calculate cost = dailyRate × days or monthlyRate × months |
| **FR9** | **Return Car**  • Set lease endDate = CURDATE()  • Update vehicle status to available  • Throw LeaseNotFoundException if leaseID invalid |
| **FR10** | **List Leases**  • Active leases (current date between start/end)• Lease history (endDate < CURDATE()) |
| **FR11** | **Record Payment**  • Insert into Payment with paymentDate = CURDATE()  • Throw exception if leaseID invalid |
| **FR12** | **View Payment History**  • Join Payment → Lease → Customer to list payments by customer |
| **FR13** | **Calculate Total Revenue**  • SELECT SUM(amount) from Payment |

**2.2 NON – FUNCTIONAL REQUIREMENTS**

|  |  |
| --- | --- |
| **ID** | **REQUIREMENT** |
| **NFR1** | **Technology Stack:**  Python 3.x, MySQL database, mysql-connector-python, openpyxl for reporting |
| **NFR2** | **OOP Principles:**  Entities encapsulate data only; use classes, inheritance (ABC), and interfaces |
| **NFR3** | **DAO Pattern:**  Interface ICarLeaseRepository and implementation ICarLeaseRepositoryImpl isolate persistence |
| **NFR4** | **Exception Handling:**  Custom exceptions with clear messages (CustomerNotFoundException, etc.) handled in main |
| **NFR5** | **Unit Testing:**  Comprehensive unittest coverage for positive and negative cases; Excel reporting |
| **NFR6** | **Performance:**  All DAO operations use prepared statements and close connections properly to avoid resource leaks |
| **NFR7** | **Usability:**  Menu-driven CLI, clear prompts, and meaningful error messages |
| **NFR8** | **Maintainability:**  Modular package structure, PEP-8 compliant, clear naming |

**3. REQUIREMENT PRIORITIZATION**

The requirements are prioritized into three levels: **High**, **Medium**, and **Low** based on their impact on system functionality and user expectations.

* + **High Priority**: Essential for the core functionality of the system.
  + **Medium Priority**: Important but not critical; can be improved in later iterations.
  + **Low Priority**: Optional or enhancement features.

|  |  |  |
| --- | --- | --- |
| **Requirement ID** | **Description** | **Priority** |
| **FR1** | Add Customer | High |
| **FR2** | Update Customer Information | High |
| **FR3** | Remove Customer | High |
| **FR4** | List & Find Customers | High |
| **FR5** | Add Vehicle | High |
| **FR6** | Update Vehicle Availability | High |
| **FR7** | List & Find Vehicles | High |
| **FR8** | Create Lease | High |
| **FR9** | Return Car | High |
| **FR10** | List Leases | High |
| **FR11** | Record Payment | Medium |
| **FR12** | View Payment History | Medium |
| **FR13** | Calculate Total Revenue | Medium |
| **NFR1** | Technology Stack (Python 3.x, MySQL, mysql-connector-python, openpyxl) | High |
| **NFR2** | OOP Principles (encapsulation, classes, ABC interfaces) | High |
| **NFR3** | DAO Pattern (ICarLeaseRepository + Impl) | High |
| **NFR4** | Exception Handling (custom exceptions handled in main) | High |
| **NFR5** | Unit Testing (comprehensive unittest coverage + Excel reporting) | High |
| **NFR6** | Performance (prepared statements, proper connection cleanup) | Medium |
| **NFR7** | Usability (menu-driven CLI, clear prompts, meaningful errors) | Medium |
| **NFR8** | Maintainability (modular structure, PEP-8 compliance, clear naming) | Medium |

**4. USE CASE DIAGRAM**

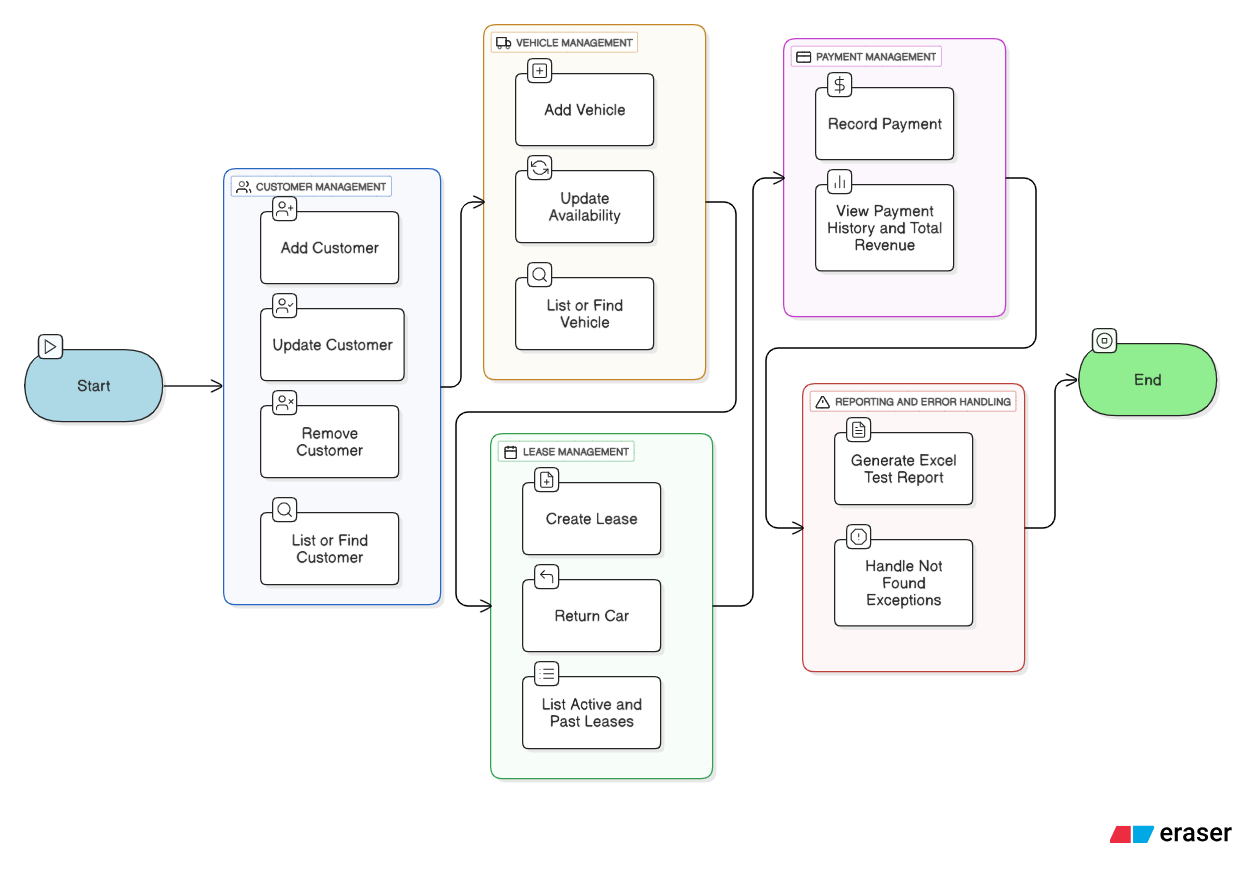
**4.1 INTRODUCTION**

The Use Case Diagram provides a high-level view of interactions between the system and its actors.

**4.2 ACTORS IDENTIFIED**

* **User (Clerk):** Manages customers, vehicles, leases, and payments.
* **System:** Executes business logic, persistence, and reporting.

**4.3 USE CASE DIAGRAM**

****

**4.4 DESCRIPTION**

|  |  |  |
| --- | --- | --- |
| **USE CASE** | **ACTOR** | **DESCRIPTION** |
| Manage Customers | Clerk | Add, update, remove or view customer records |
| Manage Vehicles | Clerk | Add vehicles, change availability, or view |
| Manage Leases | Clerk | Create/return leases and list active/history |
| Manage Payments | Clerk | Record payments and view history/revenue |

**5. ENTITY-RELATIONSHIP DIAGRAM (ERD)**

### **5.1 INTRODUCTION**

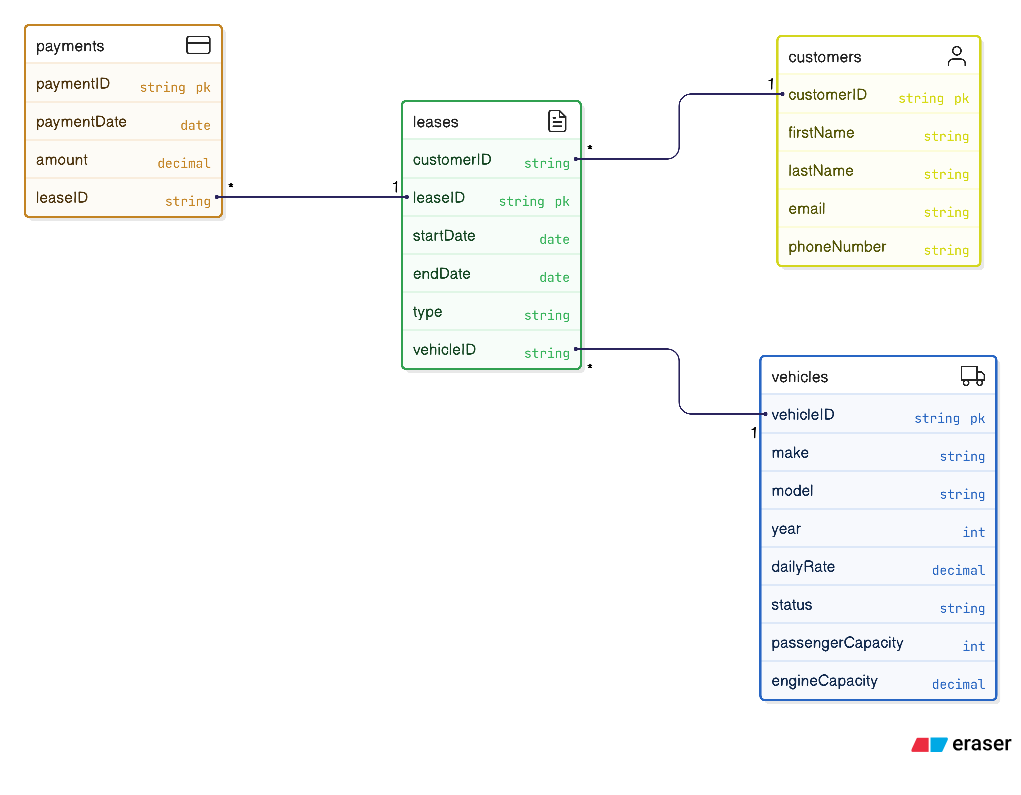
The Entity-Relationship Diagram (ERD) models the logical structure of the Car Rental System’s database. It shows each table (entity), its columns (attributes), primary keys (PK), foreign keys (FK), and the relationships between them. This ensures data integrity and guides both the SQL schema design and the application’s DAO layer.

### **5.2 SCHEMA OVERVIEW**

The system comprises four core entities:

* Customer
* Vehicle
* Lease
* Payment

**5.3 ENTITY-RELATIONSHIP DIAGRAM**

****

### **5.4 DESCRIPTION**

* **Customer → Lease (1 to many):**  
  A single customer may have multiple leases over time.
* **Vehicle → Lease (1 to many):**  
  Each vehicle can be leased many times (but only once per active lease).
* **Lease → Payment (1 to many):**  
  A lease can have multiple payments (e.g., deposit, balance).

Foreign key constraints enforce that every Lease row refers to a valid Customer and Vehicle, and every Payment refers to a valid Lease.