Car Rental System
Design and Architecture Document

Student Name: Yanqian Chen

Student ID: 270595725

Supervisor: Mohammad Norouzifard

Yoobee College of Creative Innovation
31.01.2025

1. Introduction

This document is written to clearly state the design and architecture of the system, Car rental System. A good design and architecture is basic of a stable, extendable and maintainable system. This document will describe the architecture and actions of the system in detail, using UML.

2. Innovation

A key highlight of our system's internationalization (i18n) language support is its simplicity and extensibility. I have implemented i18n without relying on any third-party libraries, opting for a lightweight and self-contained approach. This design makes adding new languages exceptionally easy: no code modification is required. To add a new language, simply copy an existing language file, such as *language_en.py*, rename it with the new language code, for example, *language_fr.py*, and then translate the text content within the file. The system will automatically detect and load the new language file without requiring code recompilation or redeployment. This design significantly streamlines the language expansion process, reduces maintenance costs, and lays the groundwork for the system to quickly support more languages in the future.

3. UML Diagrams

a) Class Diagrams

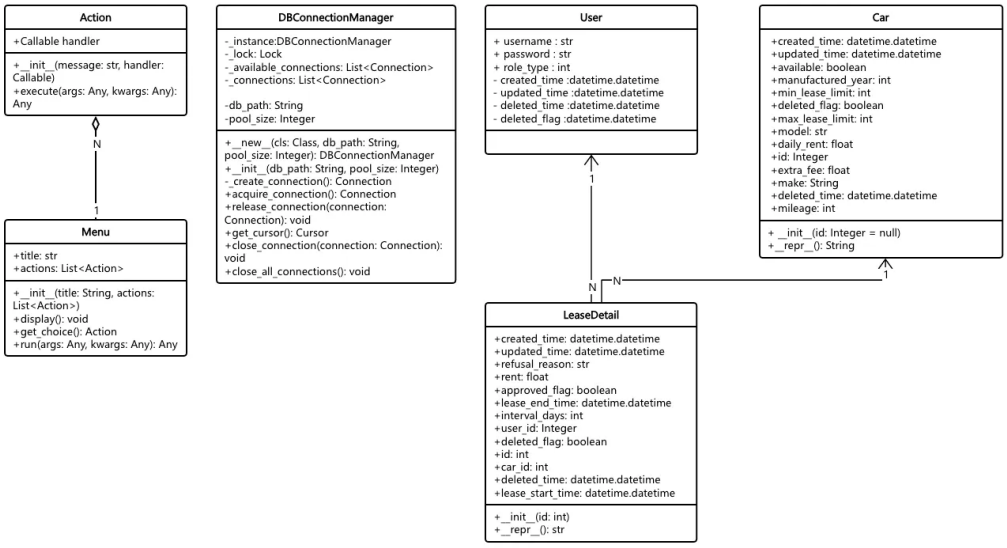

Figure 1 : Class Diagram

b) Sequence Charts

i. Users Login

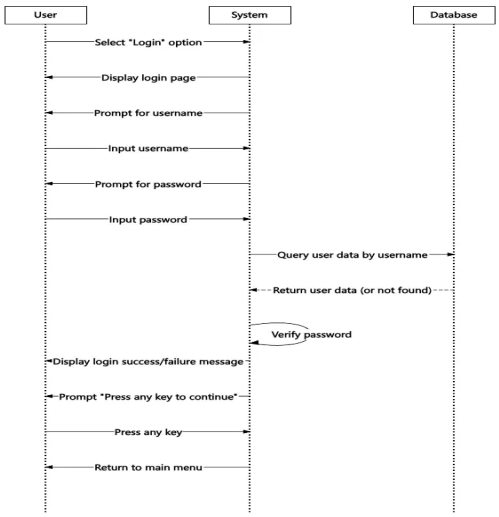

Figure 2 : The Sequence Chart of Users Login

ii. Car management

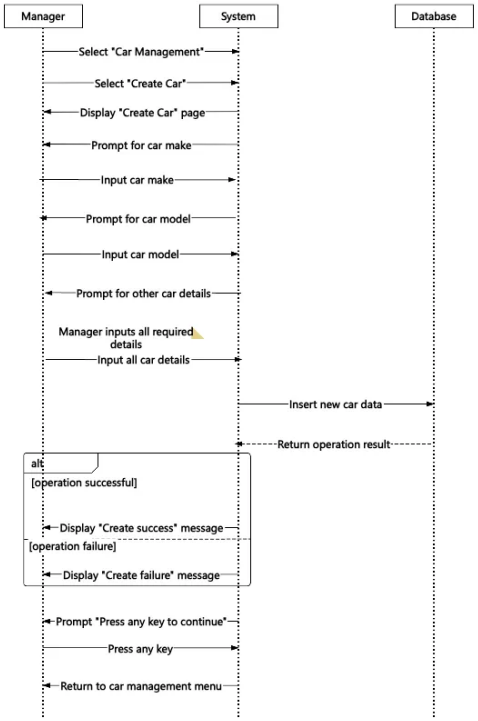

Figure 3 : The Sequence Chart of Car Management

iii. Booking cars

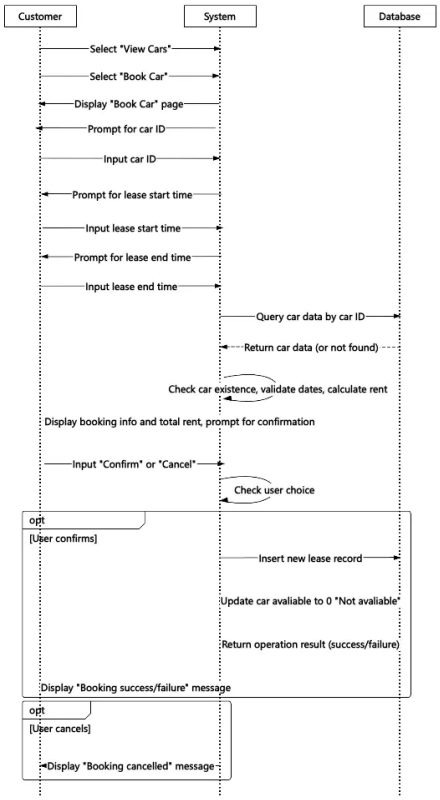

Figure 4 : The Sequence Chart of Booking Cars

iv. Auditing applications

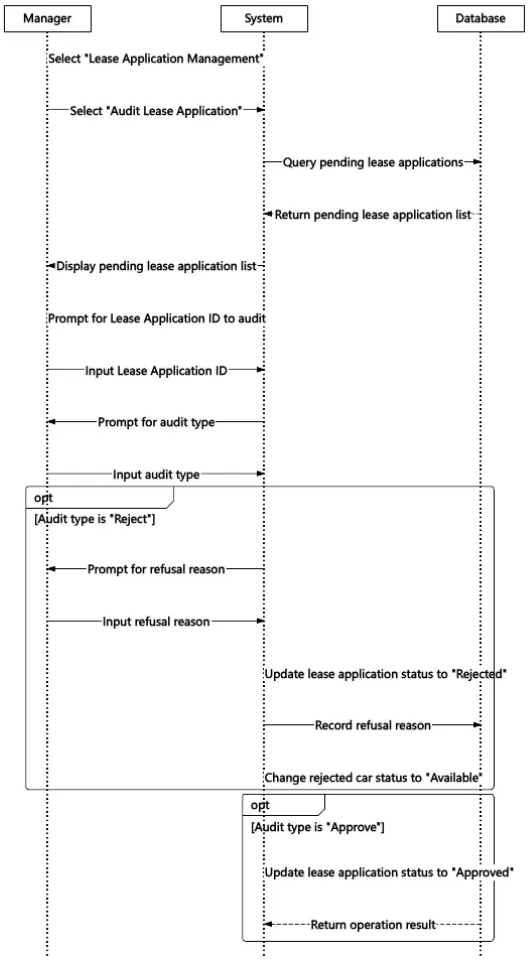

Figure 5 : The Sequence Chart of Auditing Applications

c) Use Case Diagram

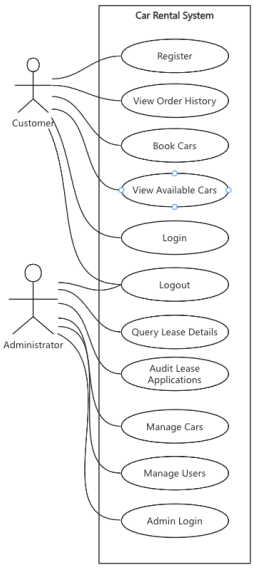

Figure 6 The Use Case Diagram

4. Design Pattern

a) Singleton Pattern

The primary intent of utilizing the Singleton pattern here is to control access to the database connection pool. By enforcing a single instance of DBConnectionManager, the system effectively manages and centralizes database connection resources. This prevents the uncontrolled proliferation of database connections, which could lead to resource exhaustion and performance degradation.
Implementation: The DBConnectionManager class achieves the Singleton behavior through:

- A private static attribute, _instance, to hold the sole instance of the class.
- A thread lock, _lock, to guarantee thread-safe instantiation in concurrent environments.
- Overriding the __new__ method to intercept instance creation requests and ensure that only one instance is ever created. Subsequent requests for an instance will return the already existing one.

Benefits in the Car Rental System:

- **Resource Management:** The Singleton pattern optimizes database resource utilization by limiting the number of DBConnectionManager instances, thereby preventing excessive connection overhead.
- **Centralized Access:** It provides a single, well-known point of access to the database connection pool, simplifying connection management and ensuring consistent database interactions across the system.
- **Thread Safety:** The inclusion of a thread lock during instance creation ensures that the Singleton remains thread-safe, crucial in multi-threaded application environments.

b) Decorator Pattern

The @db_connection decorator is designed to encapsulate database connection management logic and apply it uniformly to service functions that require database interaction. This separation of concerns enhances code modularity and readability by isolating database handling from the core business logic of the service functions.
Implementation: The @db_connection decorator functions as follows:

- It is a higher-order function that takes a function (func) as input and returns a wrapped function (wrapper).
- The wrapper function encapsulates the database connection lifecycle: acquiring a database cursor and connection from the DBConnectionManager, executing the original function (func), committing database transactions, and finally releasing the connection back to the pool.

Benefits in the Car Rental System:

- **Separation of Concerns:** The Decorator pattern cleanly separates database connection management from the business logic of service functions, making the code more organized and easier to maintain.

- **Code Reusability:** The @db_connection decorator promotes code reuse by providing a consistent and reusable mechanism for handling database connections across multiple service functions.
- **Reduced Boilerplate Code:** It eliminates the need to write repetitive database connection and transaction handling code in each service function, leading to cleaner and more concise service implementations.
- **Improved Readability:** Service functions become more focused on their core business logic, enhancing code readability and understanding.