CAR CONNECT – A CAR RENTAL PLATFORM

**Introduction**

Through a safe, effective, and user-friendly platform, Car-Connect is an end-to-end car rental management system that aims to simplify the rental car procedure. Customers can browse and book available cars using Car-Connect, a Python-based car rental management system that also lets administrators manage fleet information and keep an eye on reservations. Through layered design and exception handling, it guarantees that business rules are upheld and connects to a relational SQL database for persistent storage.

**Scope of the Project**

The goal of the Car-Connect project is to offer a complete, scalable, and modular vehicle rental system that accommodates admin and consumer features. Small to medium-sized automobile rental companies looking to automate and digitize key processes including vehicle management, reservations, and reporting can use this solution.It has a secure, role-based login for customers and admin.

**Functional Requirements**

**Customer Functions:**

1. Register a new customer account.
2. Log in using username and password.
3. View list of available vehicles.
4. Create a reservation by selecting vehicle and dates.
5. Update or cancel existing reservations.
6. View personal reservation history.

**Admin Functions:**

1. Register and log in as admin.
2. Add new vehicles to the system.
3. Update or delete existing vehicle records.
4. View and manage customer accounts.
5. Monitor all reservations (active, pending, completed).
6. Generate reports (revenue, vehicle usage, customer trends).

**Reservation Functions:**

1. Check vehicle availability based on date/time.
2. Prevent double booking with conflict resolution.
3. Automatically calculate total cost based on rental duration and daily rate.
4. Cancel or modify active reservations.

**Authentication and Security:**

1. Secure login system for both customer and admin.
2. Password hashing for safe credential storage.
3. Role-based access control to restrict features based on user type.

**Authentication and Security:**

1. Secure login system for both customer and admin.
2. Password hashing for safe credential storage.
3. Role-based access control to restrict features based on user type.

**Non-Functional Requirements**

**Security:**

* All user passwords are securely hashed before storage.
* Role-based access control is implemented for customers and admins.
* Input validation is enforced to prevent SQL injection and unauthorized access.
* Custom exception handling ensures robust and secure execution.

**Performance:**

* Real-time vehicle availability and reservation operations are optimized for speed.
* Efficient SQL queries support smooth handling of concurrent users.
* System performance remains consistent under typical load conditions.

**Maintainability & Modularity:**

* Layered architecture (entity, dao, service, util, exception, main) promotes clean separation of concerns.
* Code should be modular and follow OOP principles for easy updates and feature extension.

**Reliability:**

* Unit testing ensures key components such as authentication, reservation, and vehicle management are reliable.
* Graceful handling of system and database-level errors using custom exceptions.
* Transactional integrity is maintained during reservation and update operations.

**Data Integrity:**

* Primary and foreign keys enforce data relationships in the database.
* Atomic operations prevent partial data updates and maintain consistency.

**Portability:**

* Compatible with any environment that supports Python and MySQL.
* Minimal dependencies for easy deployment and cross-platform support.

**Database Schema**

CUSTOMER TABLE:

VEHICLE TABLE:



ADMIN TABLE:



RESERVATION TABLE:



**Technologies Used**

|  |  |  |
| --- | --- | --- |
| **Category** | **Technology** | **Purpose/Usage** |
| Programming Language | |  | | --- | |  |  |  | | --- | | Python | | Core application logic using OOP principles |
| Database | |  | | --- | |  |  |  | | --- | | MySQL / SQLite | | Relational database for persistent data storage |
| SQL | |  | | --- | |  |  |  | | --- | | Structured Query Language (SQL) | | For schema creation and CRUD operations |
| Testing Framework | unittest (Python Standard Library) | Unit testing of business logic and database functions |
| Architecture | |  | | --- | |  |  |  | | --- | | Layered Architecture | | Modular code structure (entity, dao, service, etc.) |
| Password Security | hashlib | Secure hashing of passwords |
| Database Connectivity | sqlite3 / mysql-connector-python | Database interaction layer |
| IDE/Text Editor | |  | | --- | |  |  |  | | --- | | VS Code / PyCharm | | |  | | --- | |  |  |  | | --- | | Application development and debugging | |
| Version Control | Git, GitHub | Source code management and collaboration |

**PROJECT STRUCTURE:**

The Car-Connect application is organized using a modular, layered architecture to promote separation of concerns, maintainability, and scalability. The following is the structure of the project:

* **entity/**: Contains entity classes like Customer, Vehicle, Reservation, and Admin. These classes only contain data attributes with constructors and getters/setters.
* **dao/**: This package includes service interfaces and their database-interacting implementation classes, such as CustomerServiceImpl, VehicleServiceImpl, etc.
* **exception/**: Contains custom exception classes (e.g., AuthenticationException, ReservationException, VehicleNotFoundException) to handle business logic errors gracefully.
* **util/**: Provides utility classes for handling database connections , reading configuration files , and helper methods like password hashing.
* **service/**: Implements core business logic like user authentication (AuthenticationService) and reporting (ReportGenerator).
* **test/**: Includes unit test cases using Python's unittest framework to verify correctness of functionalities.
* **main/**: Entry point of the application (MainModule.py) featuring a menu-driven interface for customer and admin functionalities.
* **db/**: SQL schema files and any database setup scripts.
* **README.md**: Instructions for setting up and running the project.