Rental Car Management System – Waterfall Model Design

Requirements: Gathered and documented all functional and non-functional requirements of the Rental Car Management System.

Functional Requirements:

1. User registration, login with role (Admin, Regional Admin, Customer)
2. Car reservation and cancellation
3. Interstate travel support
4. Document upload (ID, License)
5. Admin approval workflow
6. Payment processing and fee calculations
7. Vehicle return and inspection handling
8. Car servicing and maintenance
9. After-hours return handling

Non-Functional Requirements:

1. Secure password encryption
2. High availability and performance
3. Data integrity and audit trail
4. Modular design with maintainability

- Use case diagrams and descriptions
- Sequence and Activity Diagrams

Analysis: Analysed the functional and non-functional requirements. Decided and prioritised the functions that would be made available.

Functional Requirements: Detailed actions and processes the system must support.

- **User Management**
  - Register customers and admins
  - Differentiate roles: Admin, Regional Admin, Customer
  - Secure login and logout
- **Car Inventory Management**
  - Add, update, remove cars
  - Track car status (available, reserved, under service)
  - Record mileage, service frequency, and location
- **Reservation Management**
  - Customers can:
    - Search available cars
    - Make, cancel, and view reservations
    - Book interstate and one-way trips

        - Upload required documents (license, ID)
    - Admins can:
        - Approve/reject reservations
        - Manage vehicle assignments
- **Returns and Inspections**
    - Log return time and location
    - Damage and fuel level assessment
    - Fee calculations (overstay, relocation, damage)
- **Payment Handling**
    - Rental fee + deposit + additional fees
    - Refunds and final settlement
    - Track payment status (paid, refunded)
- **Notifications**
    - Email or system alerts for approvals, bookings, and returns

Non-Functional Requirements: System-level qualities and constraints:

- **Security:** Password hashing, role-based access control
- **Performance:** Quick search for available cars
- **Scalability:** Support multiple locations and users
- **Reliability:** Prevent double-booking and data conflicts
- **Maintainability:** Modular code structure for future upgrades
- **Usability:** Command-line UI with clear prompts and validations

Business Rules: Specific rules and policies that must be enforced:

- Customer must be greater than 21 years old
- Must upload valid ID, passport, and driver's license
- Interstate trips must be greater than or equal to 6 days and should be booked at least 72 hours in advance
- Different pickup/drop-off locations trigger relocation fees
- After-hours returns are inspected the next working day
- Cancellation fees depend on timing relative to pickup

Stakeholder Identification: Identify people or groups who will use or be affected by the system:

- **Primary users:** Customers
- **Admins:** Approve reservations, inspect returns
- **Regional Admins:** Manage Admin tasks for specific locations
- **System Developers:** Build and maintain the system
- **Management:** Monitor overall operation

Assumptions and Constraints

- Assumes SQLite will be used as a database
- Users have internet access and a basic CLI interface
- No integration with external payment APIs (initial version)
- Only supports bookings within New Zealand

Output/Deliverables

- Use case diagrams and descriptions
- Activity and Sequence Diagrams
- Class Diagrams

Design: Finalize the system components, define data flow, system architecture, database schema, and interface structures.

- Architecture: Layered MVC-style modular structure
- Presentation Layer: CLI/GUI
- Business Logic Layer: Services
- Data Access Layer: SQLite via Singleton pattern
- Database Design:
  Tables: USERS, CUSTOMER, CAR, RESERVATION, PAYMENTS, INTERSTATE_TRAVEL, LOCATIONS, RETURNS, DOCUMENTS, etc.
  Relationships with foreign keys and constraints.
- Class Diagram (UML):
  Entities like Customer, Reservation, Car, Return, Payment, etc.
  Services: UserService, CarService, ReservationService, ReturnService, etc.
- Security Design: Password hashing
- Role-based access control

Coding and Implementation: Translated design documents into fully functional code.

Tools & Technologies:

- Language: Python
- Database: SQLite
- Frameworks: Standard Library + CLI-based UI
- Patterns: Singleton (DB), Factory (Entities), Observer (Notifications)

Tasks:

- Develop modules: user_service.py, reservation_service.py, car_service.py, etc.
- Implement input validation utilities
- Integrate password hashing and ID generators
- Build the Command-line Interface (CUI)
- Seed the database with admin, locations, and document types

---

Testing: Ensure the system is bug-free and meets all specifications.

Types of Testing: (Not all of these were carried out)

- Unit Testing: Individual functions for user registration, reservation logic, etc.
- Integration Testing: Payment with reservation, admin approval chain.
- System Testing: Complete booking, return, and cancellation flow
- User Acceptance Testing (UAT): Admin, Customer, and Regional Admin test cases.
- Tools: unittest or pytest (Python)
- Test script simulating each role and scenario

Operation/Deployment: Release the system to the production or end-user environment. (This was not done.)

Steps:

- Deploy the SQLite database and Python scripts
- Package with CLI instructions
- Provide documentation for:
    - Installation
    - Admin setup
    - Usage flows (reservation, return, etc.)

Optional:

- Prepare a GUI (Flask/Django) in future sprints
- Integrate with payment gateway API if expanding beyond the prototype

---

Maintenance: Handle bug fixes, updates, and potential enhancements based on user feedback. (This was not done)

- Monitor error logs and user feedback
- Enhance features (e.g., support for modifying reservations, online payment)
- Update modules for new locations, vehicle types, or pricing logic
- Patch security vulnerabilities