ASSESSMENT - 1

Master of Software Engineering Program

(Classification - Individual)

Assignment Title: Design and Implementation of a Car Rental System

Course Code: MSE800

Student ID: 270672899

College: Yoobee College of Creative Innovation

Assignment Title: Design and Implementation of a Car Rental System

Author: Weerakkody Gamage Sachintha Lakshan Weerakkody

Lecturer: Mr. Mohammad Norouzifard

Date: 2025/09/14

Table of Contents

1. Introduction

1.1 Background

Because they can offer both customers and employers various mobility alternatives, car rental services are essential to today's transportation systems. Bookings, automobile management, and customer registration are still handled manually by many conventional enterprises. These traditional procedures take a lot of time, are subject to errors, and are ineffectual; they result in unhappy customers and decreased output.

Software programs that simplify renting operations are highly needed as technological alternatives proliferate. The precision, paperwork, and customer service can all be improved by a sophisticated rental automobile platform. By creating a Python-run system that utilizes object-oriented programming (OOP) methods, this effort meets the need by allowing flexibility, independence, and reliability.

1.2 Objectives and Scope

Creating and implementing a car rental solution that simplifies rental processes while showcasing cutting-edge software development techniques is the main goal of this undertaking. The particular objectives are as follows:

- Offer an intuitive way for customers to see automobiles, register, log in, and make rental reservations.
- To give administrators the ability to add vehicles, log in, and accept or decline reservations.
- To guarantee a modular design by utilizing OOP principles like polymorphism, inheritance, and encapsulation.
- To increase reliability, apply design principles (such as Single for storage handling).
- To create an adequate and error-management system that relies on a requirement.txt file for prerequisites and operates without the need for regular setups.

In this assignment and task, it is limited to a command-line user interface variation of the computer that uses SQLite for organizing data. The technology is just a prototype right now. Even though that it has a good base for future additions like cloud-based setup, mobile application integration, and real-time automobile availability checking.

1.3 Assumptions, Constraints, and Stakeholders

Assumptions

- Users (Admin/Customer) are familiar with the main base of operating a computer terminal.
- User-submitted data, such as vehicle information and rent periods, is accurate and legitimate.
- This system allows both administrators as well as customer to register immediately.

Constraints

- Python as the sole programming language used for developing the entire system.

- It should only use the packaged components specified in requrement.txt and not require additional setups.
- There are no visual elements in the interface; it is limited to a text-driven CUI.

Stakeholders

- Clients: The final users which are going to utilize the method to rent motors.
- Administration: System administrators in charge of bookings and automobile additions.
- Software Designer (Student/Developer): In charge of creating, putting into use, and taking care of the computer system within this course.

Lecturer/Institute: Reviewers who judge the project's caliber and adherence to professional standards.

2. Requirements Analysis

2.1 Functional Requirements

A few essential features of a vehicle rental business must be supported:

Administration of Users

- FR1: Permit prospective clients to register given their role, name, and password (either customers or administrators).
- FR2: Permit users to safely log in with their retained login information.
- FR3: Give consumers the option to exit the entire system.

Automobile Management (Admin only)

- FR4: Include new vehicles together with their ID, make, type, year, miles traveled, daily rate, availability, and best and worst rental durations.
- FR5: Modify current vehicle data if necessary.
- FR6: Cars that are no longer available should be removed
- FR7: View every vehicle in the database with

Rental Booking (Customer only)

- FR8: See a list of accessible vehicles together with vital details.
- FR9: Choose the car and indicate the beginning and ending dates of the hire.
- FR10: The method ought to compute rental costs according to the cost every person and the length of stay.
- FR11: Make a reservation demand with the standard outcome of "In Progress."

Rental Management (Admin only)

- FR12: Check all pending reservations with.
- FR13: Accept or disapprove reservations.
- FR14: The number of cars is updated whenever reservations become official.

2.2 Non-Functional Requirements

- Ease of use: A straightforward and understandable operator interface (CUI) for engagement must be provided by the operating system.
- Reliability - The system must provide insightful alerts when an improper input occurs, such as an incorrect username or password or an inability to pick a car.
- Performance: For an enjoyable user experience, functions including booking, automobile observing, and login must run swiftly.

- Maintainability: To enable future changes, the system must use code that is modular and adhere to OOP principles.
- Portability: Despite requiring any special setup, the software ought to function on any computer with Python installed.
- Security: Passwords need to be safely kept in the database (for example, hashed if they are later expanded).
- Scalability: Additional features like specials, reports, or warnings should be able to be added thanks to the databases and code architecture.

2.3 User Roles & Permissions

Customer

- Able to sign up and get in.
- Can observe the cars that are accessible.
- Able to make reservations for vehicles for preset rental lengths.
- Able to log off.

Admin

- Able to sign up and log in.
- Able to add and remove vehicles.
- Able to see every vehicle.
- Able to see and control reservations (accept or deny).
- Able to log off.

3. System Design & Architecture

3.1 Architectural Overview

- Following the principle of separate problems, the Car Renting System is constructed in Python utilizing a modular object-centered architecture. The main elements are
- The User Manager Section: Manages access according to roles (Admin vs. Customer), logging in and checkout.
- Car Maintenance Module: Gives administrators the ability to view, edit, remove, and add automobiles.
- Booking Administration Module: Enables clients to submit booking inquiries, which administrators can then accept or deny.
- Database Module: Ensures that there is merely one working database association across the whole system by implementing persistent storage using SQLite and the singleton network design technique.
- The CLI Interface Modules facilitate interaction by users through a command-line graphical interface.

Applied Key Design Principles:

- Encapsulation: Material and behaviors are managed independently by every component.
- Inheritance: Users pass on succession to Administration and Consumer.
- Polymorphism: Though certain roles have basic user conduct, they execute distinct activities.

- The Database class uses the Solitary design approach, whereas role-driven user creation can be accomplished by extending the Factory Function.

3.2 Utilizing Design Patterns

- In order to improve speed and avoid conflicts, the databases class uses the singleton pattern, which makes sure that only one particular instance of the connection to the database is utilized.
- Factory Method (An Additional Extension): Based on the role, this approach could be used to dynamically generate Admin or Customers objects upon user creation.
- MVC Inspiration: Although it is not a complete MVC architecture, the architecture's main flow of controls (Controller), User/Car/Booking classes (Model), and CLI (View) components all roughly adhere to the paradigm.

3.3 UML Diagrams

The following UML diagrams illustrate the design:

1. Use Case Diagram: depicts how administrators and consumers communicate with the system.
2. Class Diagram: shows the properties, methods, and connections of the primary classes (Users, Administrative, Buyer, Car, Registration, and Db).
3. A sequence diagram shows how significant situations (such as an individual buying a car or an administrator handling reservation) unfold.

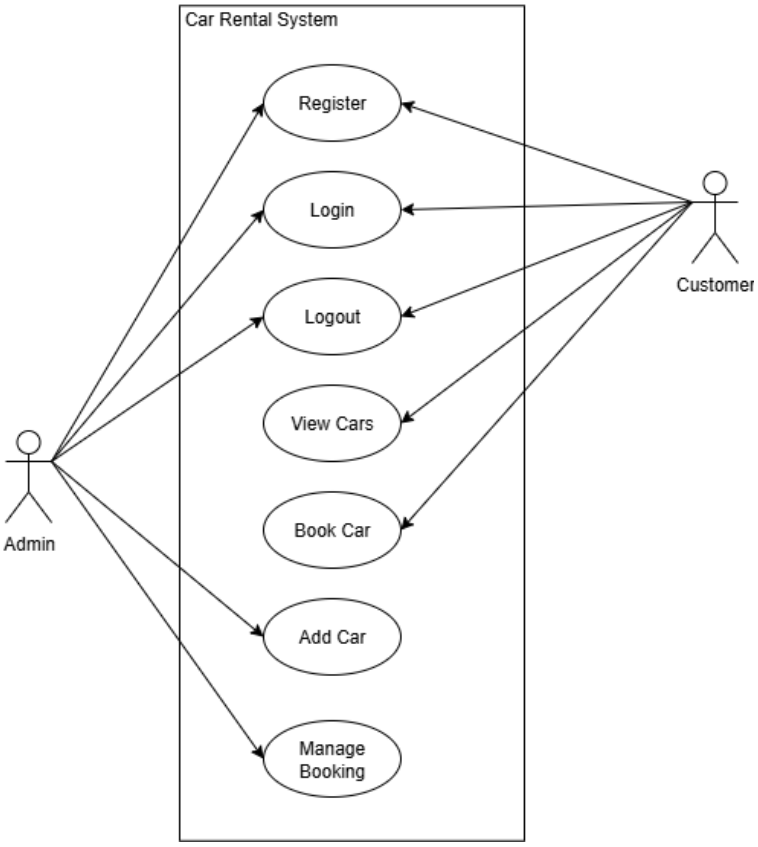

Figure 1: Use Case Diagram

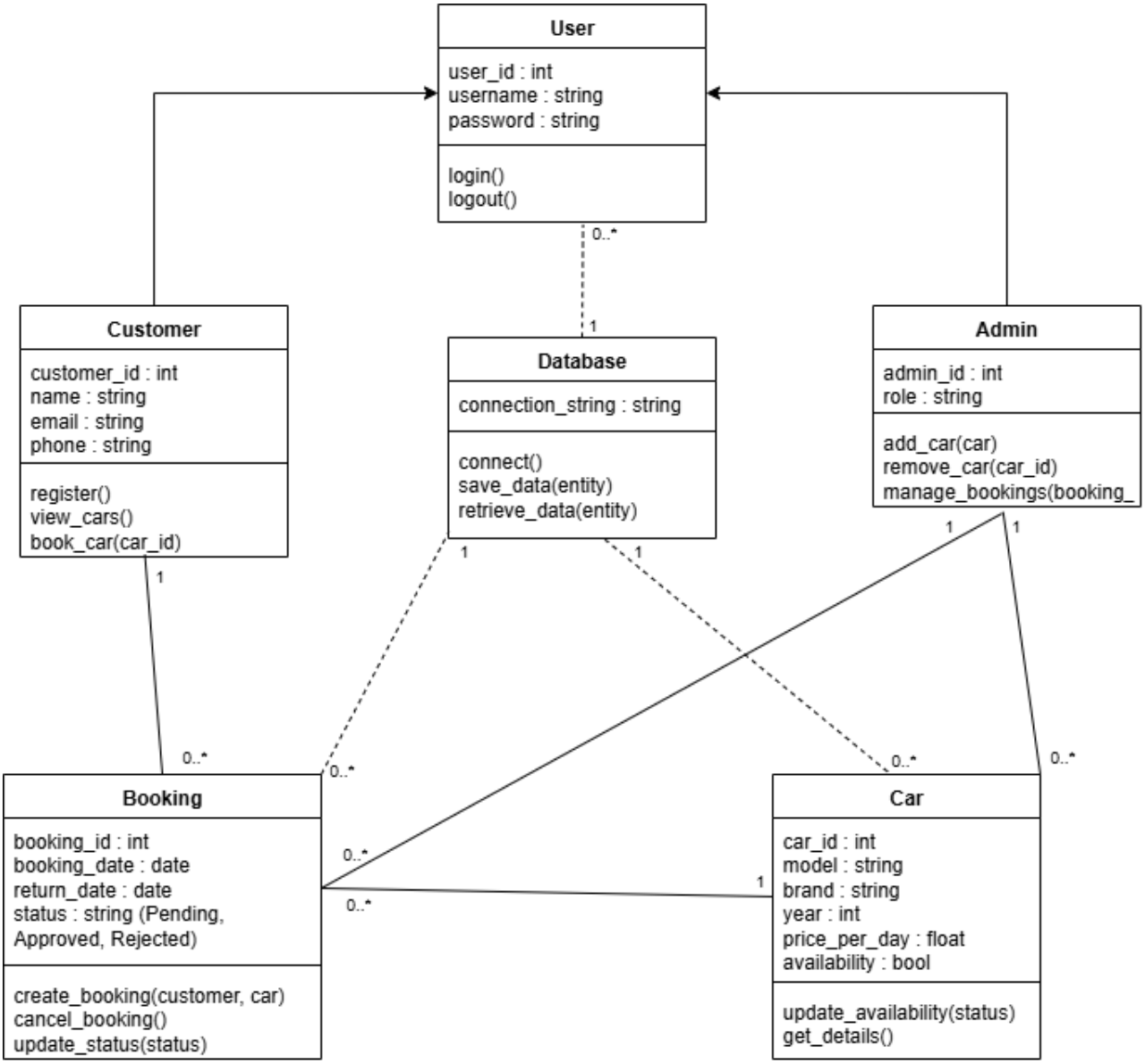

Figure 2: Class Diagram

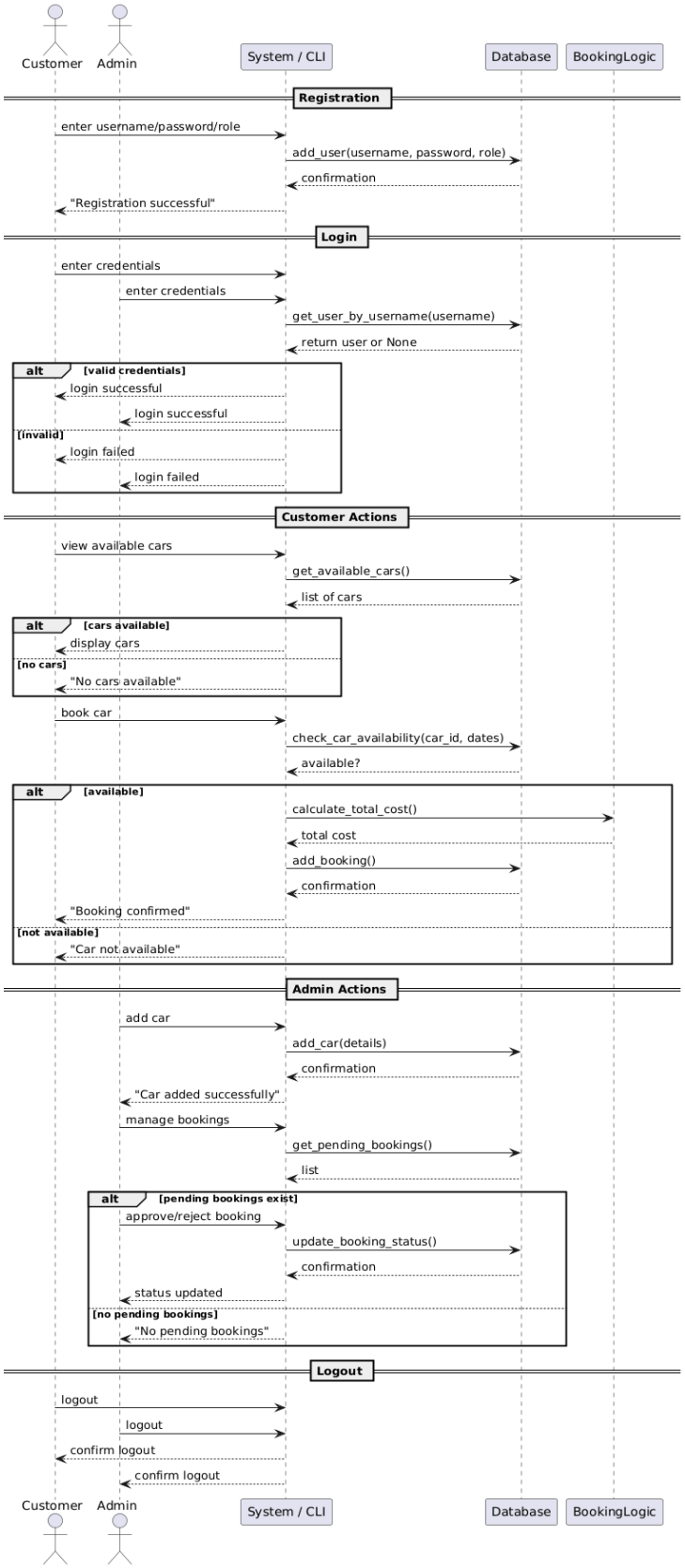

Figure 3: Sequence Diagram

4. Coding Standards & Conventions

4.1 Modularity and Encapsulation

The entire system is separated into several parts. Each of which is in charge of a certain issue (users.py, car.py, booking.py, database.py, and main.py). Such a modular strategy facilitates future additions, enhances comprehension, and issue diagnostics. Where feasible, class characteristics are kept confidential, and procedures are used to manage accessibility.

4.2 Naming Conventions

- Variables and Functions: Snake_case used to write (add_user, get_available_cars).
- Classes: Pascal Case used to write (User, Customer, Admin, Car, Booking).
- Constants: uppercase used to write (like DB_NAME = "car_rental.db").

This keeps uniformity.

4.3 Indentation & Formatting

- The entire code is indented according to PEP 8 (4 characters per column).
- For accessibility, lines of text are limited to 80-100 letters.
- Functional code segments are separated by lines that are empty.

4.4 Commenting & Documentation

- Document strings are present in each course and method to explain their use and purposes.
- Inline notes clarify intricate reasoning or crucial process phases.
- A README file with setup instructions, use guidelines, license regulations, known difficulties, and acknowledgment is included with the entirety of the project.

4.5 Performance Considerations

- The Solitary structure minimizes memory utilization by guaranteeing that there is only one active database connection.
- Limitations (e.g., WHERE available=1) are used to improve query performance and prevent needless data acquisition.
- To reduce the number of duplicate calculations, lists and looping are employed effectively.

4.6 Error Handling Conventions

- To identify mistakes linked to typing and databases, try & except sequences are utilized.
- Rather than mysterious technical failures, the user receives clear issue notifications.
- Edge cases are resolved politely, such as when there are no cars accessible, no outstanding reservations, or a wrong account.

4.7 Version Control

repositories hosted on GitHub can be used with this endeavor's structure. To guarantee reproducibility, the original code is kept in a special subdirectory along a requirement.txt file. This promotes accountability and teamwork.

5. Maintenance & Support Plan

5.1 Maintenance Strategy

The rental system for cars has a regular upkeep approach to provide sustained dependability and flexibility. Efficiency enhancement, improvements to the product, and frequent bug tracking are all part of this. Preventive (bug-fix patches), adaptable (environment modifications), and effective

(improvements caused by user's feedback) upkeep tasks are divided into three categories. To keep track of all modifications and assure accuracy, a version-driven repo is kept up to date.

- The following changes are planned:
- Adding support for consecutive rentals to the reservation code
- Improving edge case correction processes
- Modules reworking for greater scaling

5.2 Versioning

Logical versions are used by the algorithm to explicitly express the purpose of alterations:

• Important version: Adds noteworthy improvements or fundamental modifications
• Minor version: includes improvements that are reversible.
• Patch version: Resolves minor problems or glitches

For instance, the first release, v1.0.0 Version 1.1.0: Included an adaptive price module; Version 1.1.1: Resolved an issue in the booking status updates

5.3 Backward Compatibility

The system has the following features to maintain confidentiality of information across updates: Security tests to guarantee modular consistency; stable API endpoints to avoid interfering with CLI operations; and migrate scripts for changing structure of databases

In order to accommodate upcoming improvements without compromising present capabilities or user data, retrogression is given priority.

6. Project Management

6.1 SDLC Choice and Rationale

Following the Agile Software Development Life Cycle approach, the Car Renting System was developed. Agile was chosen because of its incremental structure, which facilitates initial evaluation of key features, modular adoption, and ongoing improvement. This method gave flexibility to integrate feedback and improvements while enabling the gradual release of services including registration of users, vehicle control, and reservation routines.

The modules that were the focus of each sprint were authorization for users, automobile supply, reservation logic and management functions. During the development cycle, agile concepts including continual integration, list sorting, and time-bound iterations been used.

6.2 Planning, Estimation, and Tracking

An abridged Kanban sheet was used for organizing the project and classifying jobs into "To Do," "Ongoing Development," and "Finalized." Each module was scheduled for execution within a one-week sprint, and estimates were calculated according to relationships and complexities.

A task checklists and software versioning changes were used to personally maintain progress, guaranteeing openness and reliability throughout the development process.

6.3 Risk Management and Mitigation

Few issues were notified during the planning phase, along with regarding mitigation strategies:

Table 1

Risk	Impact	Mitigation
Incomplete feature implementation	Medium	Key improvements were given priority initially throughout the rush period.
Dependency conflicts	High	using simulated settings and requirements.txt
Data loss / corruption	Medium	Implemented validation and backup routines
CLI usability issues	Low	tested efficiency and improved instructions.
Time constraints	High	Effectively identified functionality and prevented perspective creep

7. Limitations and Future Work

7.1 Current Limitations

Even though the Car Renting System satisfies all essential functional as well as non-functional needs, testing and creation revealed a number of obstacles:

- No user interfaces (GUI): the computer only uses the scripting screen, thus may make it less accessible to beginners.
- Single-User Identity: Neither parallel multiple users nor handling sessions are supported by the present solution.
- Static Cost Format: Regardless of demand, length of stay, or kind of auto, rental costs are set but never change.
- Streamlined Booking Process: The framework does not manage partial-day reservations or time-based disputes because it expects distinct rental dates.
- Limited Error Recording: Although error prevention is in place, system occurrences and failures are not tracked via a central monitoring method.

In order to guarantee prompt shipment and conformity with the evaluation scope, certain limitations were agreed upon.

7.2 Proposed Future Enhancements

Some improvements are suggested to increase the system's capacity to grow its functionality:

- Dynamic Pricing Mechanism: To maximize income and scheduling conduct, implement price-based on demand through a strategy structure.
- Website-Based Interface: To increase usability and ease of use, create a responsive web homepage.
- Support for Multiple Users: Put concurrent control and session control into practice for in-the-moment communication.
- Sophisticated Booking Logic: Expand booking validity to include day-only reservations, overlapping rentals, and cancellation guidelines.
- Centralized Recording and Evaluation: Include a tracking framework to record system effectiveness metrics, user activities, and running faults.

Cloud Implementation: To facilitate expansion, remote use, and compatibility with external services, host the whole thing on a platform that is cloud-based.

8. Conclusion

The planning, creation, and assessment of a Python-based desktop rental automobile system are provided in this report. Key rental functions like registration for users, vehicle control of inventory, and scheduling procedures are all effectively automated by the system. It uses design guidelines to improve flexibility and maintenance, as well as important object-based coding theories like encapsulation, inheritance, abstraction, and polymorphism.

All of the specific as well as non-functional demands listed in the MSE800 assessment brief are satisfied by the app thanks to its permanent information handling, role-driven control via roles, and organized system layout. The system is guaranteed to be both economically viable and educationally rigorous by its incorporation of UML models.

Additionally, the research investigates possible future updates, including the use of the cloud, smartphone connection, and price flexibility, showcasing its practical scalability. All things considered, the car renting system offers a strong basis for further improvement and creativity and exhibits a methodical software creation methodology.

9. References

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Martin, R. C. (2008). *Clean code: A handbook of agile software craftsmanship*. Prentice Hall.

Larman, C. (2004). *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development* (3rd ed.). Prentice Hall.

Fowler, M. (2003). *Patterns of enterprise application architecture*. Addison-Wesley.

Pressman, R. S., & Maxim, B. R. (2014). *Software engineering: A practitioner's approach* (8th ed.). McGraw-Hill Education.

Python Software Foundation. (2023). *Python documentation*. https://docs.python.org/3/

SQLite. (2023). *SQLite documentation*. https://www.sqlite.org/docs.html

IEEE Computer Society. (2014). *Guide to the software engineering body of knowledge (SWEBOK V3.0)*. https://www.computer.org/education/bodies-of-knowledge/software-engineering

Open Web Application Security Project. (2023). *OWASP secure coding practices*. https://owasp.org/www-project-secure-coding-practices/