Readably
Architecture Notebook

1. Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation.
It explains the design of the system, its architectural mechanisms, its architecturally significant requirements and its constraints.

2. Architectural goals and philosophy

Readably aims to provide a well-designed architecture that can be used easily by everyone.
Readably is a web application project.
Readably's interface language is English.
Secure and proper user functionalities (login, logout, registration, etc.) will be provided as security should be the major concern of the project as the user data must remain confidential and since the user engagement is the main deal of an online store, user functionalities must be implemented correctly.
Readably's goal is to achieve high functionality through simplicity, this philosophy is also applied to the project's architecture.
Another major, maybe the most important, architectural goal of the project is the speed. High response time is not acceptable for every stakeholder in this project.

3. Assumptions and dependencies

System should have a high capacity since a high web traffic is expected.
Readably must be up 24/7.
Database must store all of the data and storing must be done in a secure way.
System must have a secure and fast payment method.
Readably must have agreements with delivery companies and system must have a reliable and secure delivery tracking.
There are no hardware-related assumptions or dependencies.

4. Architecturally significant requirements

The system must record every modification to customer information.
The system must record every modification to products.
The system must respond within 3 seconds.
The system must accept book orders.
The system must provide secure payment.
The system must provide password protection for all accounts.
The system must provide the ability to search books, authors, publishers, ISBNs.
The system must provide electronic links between the online bookstore system, the database, payment system and the shipping fulfillment system.
The system must maintain reviews of books, and rating of books.

5. Decisions, constraints, and justifications

Database should be MySQL since it is:

- is globally renowned for being the most secure and reliable database management system,
- a database with high-performance,
- scalable and
- open source.

Codes should be written in a clean and readable manner, and this decision is crucial since Readably is a team project and all the contributors must be able to read the code.
A theme can be used and the theme should be tweaked to the Readably's needs as a theme is easy to configure and it is Bootstrap-based so the tweaking can be done easily.
Spring Boot should be used because:

- Spring Boot makes it easy to create Spring-powered, production-grade applications and services with absolute minimum fuss.
- Spring Boot provides a radically faster and widely accessible getting started experience for all Spring development.

Spring Security is used for security because:

- It has configuration support to Java.
- It is portable.
- It has comprehensive support to tasks like authorization and authentication.
- It provides CSRF protection.

Readably should use Hibernate to connect the web app to the database because:

- It is database independent,
- It is open source,
- It is scalable,
- Easy to learn, and
- Hibernate's transparent persistence ensures the automatic connection between the application's objects with the database tables.

Spring Data JPA is chosen to provide an API to perform CRUD operation because:

- It provides CRUD capabilities to any domain object without the need of any boilerplate code.
- It minimizes the amount of source code needed to write custom queries.
- It offers simple abstractions for performing common tasks like sorting an pagination.

6. Architectural Mechanisms

Code Cleanliness: Ensures the readability of the code.

- **Modularity:** Code must utilize functions to avoid duplicate code.
- **Semantic:** Variable names should represent the use of itself.
- **Indentation:** Spacing should be clean and correct.
- **Implementation:** Algorithms should be implemented correctly.

Documentation: Ensures that there is no misunderstanding between stakeholders.

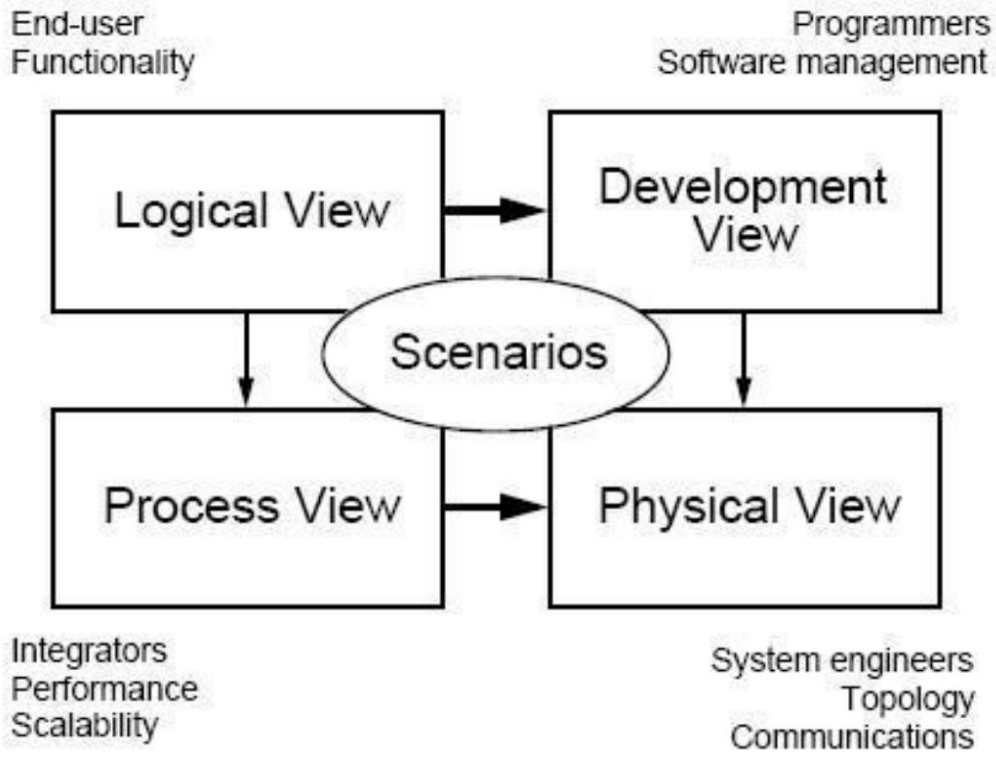
- **Purpose:** Specify the purpose of the class.
- **Function Definition:** Specify the purpose of the function.
- **Definition:** Special requirements and cases.

7. Key abstractions

Administration: Manages and maintains the online book store.
Customer: Interacts with the online book store.
Payment: Third-party application which is used for making secure payment.
Shipping Fulfillment: Third-party application which is used for procurement.

8. Layers or architectural framework

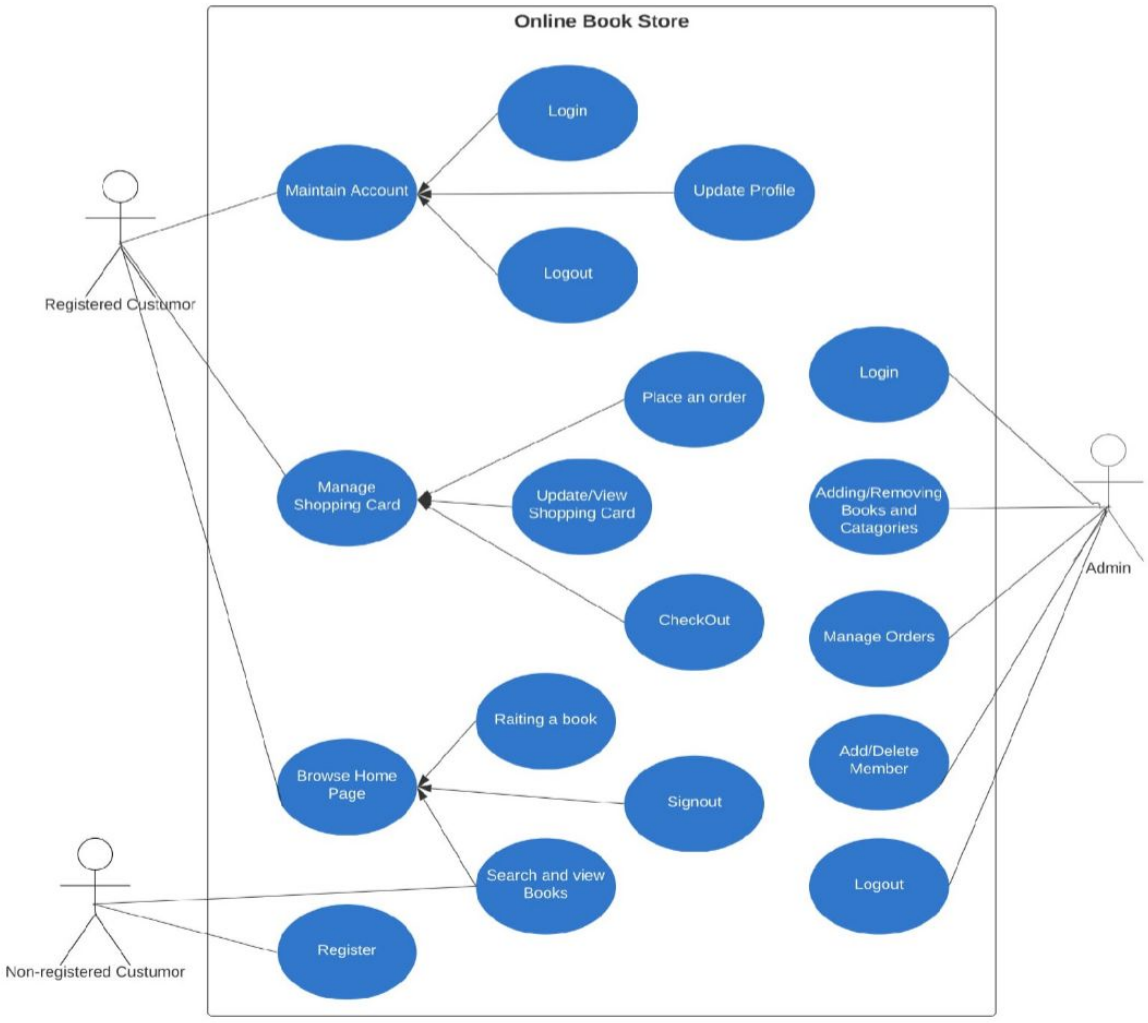
This project's architectural framework is Kruchten's 4+1 Architectural View Model since this model describes the system from the viewpoint of different stakeholders.



Logical View: The functionality. The service.
Process View: Communication between processes and/or services.
Physical View: Deployment of your services.
Development View: File/Folder Structure of your codebase. What you're looking at in your IDE/Editor.

9. Architectural views

- **Logical:** Describes the structure and behavior of architecturally significant portions of the system. This might include the package structure, critical interfaces, important classes and subsystems, and the relationships between these elements. It also includes physical and logical views of persistent data, if persistence will be built into the system. This is a documented subset of the design.
  This contains information about the various parts of the system. In UML the logical view is modelled using Class, Object, State machine and Interaction diagrams (e.g Sequence diagrams). It's relevance is really to developers.
- **Use case:** A list or diagram of the use cases that contain architecturally significant requirements.
  This view describes the functionality of the system from the perspective from outside world. It contains diagrams describing what the system is supposed to do from a black box perspective. This view typically contains Use Case diagrams. All other views use this view to guide them.



- **Process:** This describes the concurrent processes within the system. It encompasses some non-functional requirements such as performance and availability. In UML, Activity diagrams - which can be used to model concurrent behaviour - are used to model the process view.
- **Development:** The development view focusses on software modules and subsystems. In UML, Package and Component diagrams are used to model the development view.
- **Physical:** The physical view describes the physical deployment of the system. For example, how many nodes are used and what is deployed on what node. Thus, the physical view concerns some non-functional requirements such as scalability and availability. In UML, Deployment diagrams are used to model the physical view.