Student: Zeibel Antonia

**Group: 30431**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

The cake shop application is implemented to provide a user-friendly platform for customers to view, order and track their orders, while allowing administrators to manage orders, inventory, and other related data. The application will have two types of users: customers and administrators. Both customers and administrators will be required to provide a username and a password to authenticate into the application.

# Functional Requirements

Both The customer and the administrator need to authenticate into the application before proceeding to other operations. The authentication process consists of providing a username and the password to the system.

In case of non-existing credentials, the user needs to registers by providing a non-existing username and a strong password. After registering into the application, a new customer will be saved to the database and an email will be sent with the information of the user provided.

The administrator can perform CRUD operations on the users, cakes and the inventory, but a customer can only view the list of cakes and perform CRUD operations on the orders. For the moment and for the sake of implementing and demonstrating some patterns, Observer and Iterator, the implemented CRUD operations are only on users and cakes.

# Non-functional Requirements

As I have briefly mentioned, the application provides non-functional security features. When the users access their accounts, register or login, the provided details are stored into the database, while the password is encrypted for privacy purposes. Moreover, there are specific operations for each type of user according to their role.

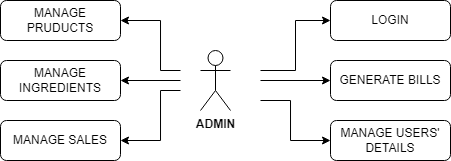
2. Use-Case Model

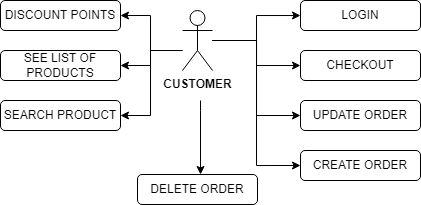
**Use-case:** create cake

**Primary Actor:** admin

**Main success scenario:** The admin authenticates into the application by providing the username and the password already registered into the database. After this step, being authenticated as an admin, you can perform the create cake operation. To perform this kind of operation, you need to provide the necessary information about the name of the cake, the description of the cake and the price. These fields are necessary to be completed for a valid request body. After completing these fields and being validated, a new cake is created and saved into the cake repository.

**Extensions:** The user is correctly authenticated, but does not provide valid data for the fields or the user is not correctly authenticated as an admin, therefore the user cannot complete further operations.





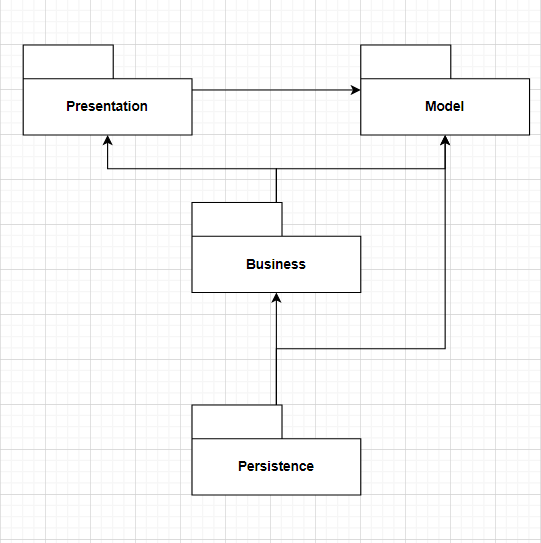
3. System Architectural Design

**3.1 Architectural Pattern Description**

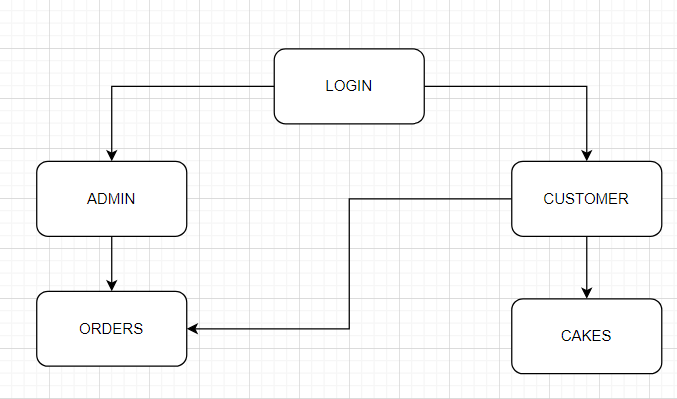
A common architecture for Spring Boot applications is the Model-View-Controller (MVC) pattern. Models represent the data and business logic, views present the data to the user, and controllers process user requests and generate responses. MVC provides a clear separation of concerns and allows for easier maintenance and extensibility.

**3.2 Diagrams**

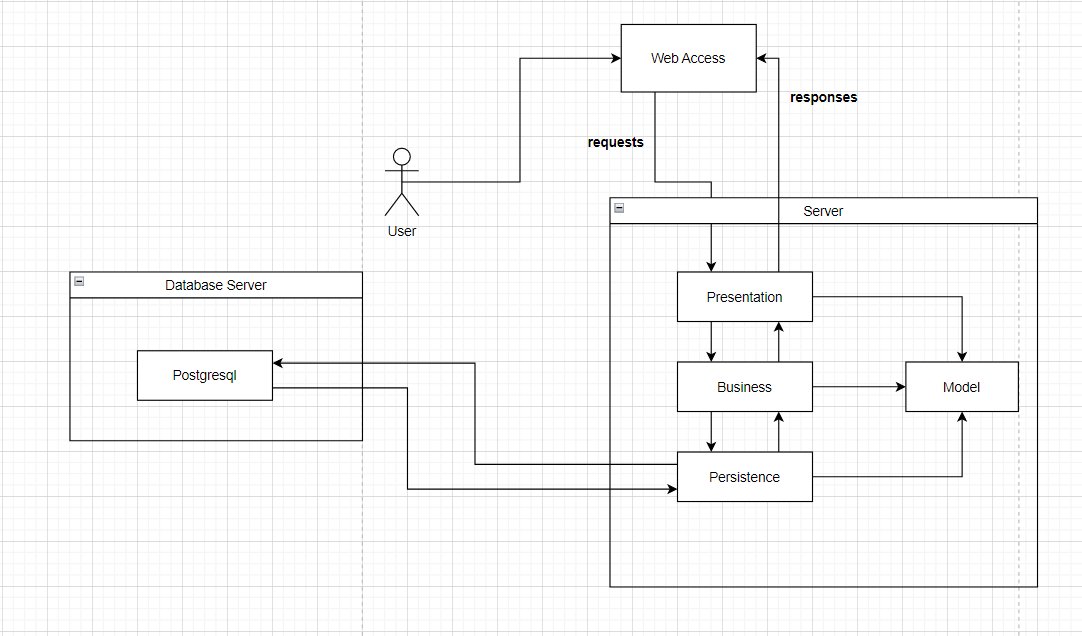
**Package diagram:**



**Component diagram:**

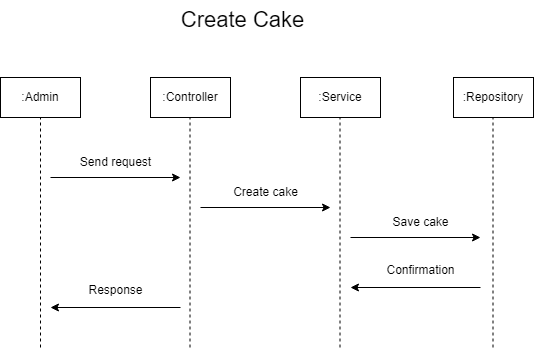


**Deployment diagram:**



4. UML Sequence Diagrams

Sequence diagram for creating a cake.



5. Class Design

**5.1 Design Patterns Description**

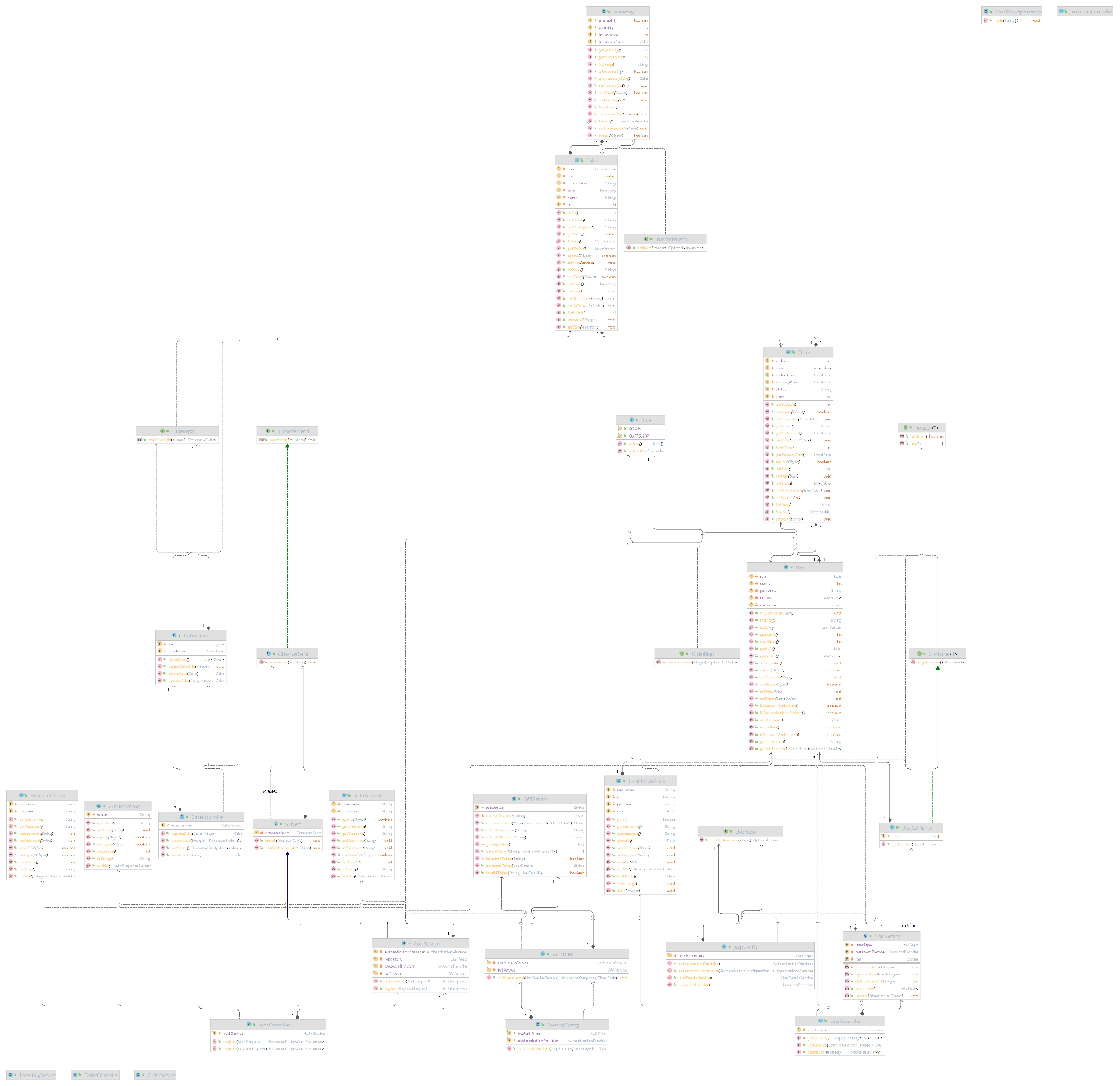
*[Describe briefly the used design patterns.]*

For this project, I have implemented the Observer pattern and the Iterator pattern.

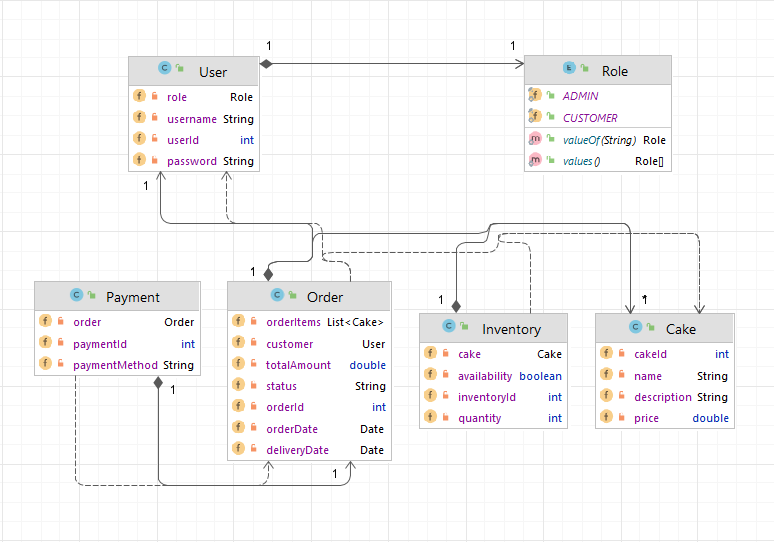
Observer is a behavioral design pattern. It specifies communication between objects: observable and observers. An observable is an object which notifies observers about the changes in its state. I have used three actor classes. Subject, abstract class, Observer, concrete class, and an Observer interface. Subject is an object having methods to attach and detach observers to an observer object. In this project, when a user registered successfully, an email was sent.

An Iterator is one of many ways we can traverse a collection. we need to obtain an Iterator from a Collection and this is done by calling the iterator() method. The hasNext() method can be used for checking if there's at least one element left to iterate over. The next() method can be used for stepping over the next element and obtaining it.

**5.2 UML Class Diagram**



6. Data Model



7. System Testing

As for the testing part, the creation and deletion of the cakes are tested using Junit, Faker. Basically, the data used for the testing was mocked using faker, then using different methos annotated with @Test to assert if the expected result is equal to the actual result.

8. Bibliography

* <https://youtu.be/KxqlJblhzfI>
* <https://www.youtube.com/watch?v=9SGDpanrc8U&t=2187s>
* <https://www.baeldung.com/java-iterator>
* <https://www.baeldung.com/java-observer-pattern>