Analysis and Design Document

Student: Zeibel Antonia

**Group: 30431**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <dd/mmm/yy> | <x.x> | <details> | <name> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

# Project Specification

The Cake Shop Management System is a web-based application that allows customers to browse and order cakes online. The system includes features for managing inventory, tracking orders, and generating reports for sales. The application will be built using Java Spring Boot, a popular framework for building web applications, and will follow a microservices architecture for scalability and maintainability.

The Cake Shop Management System will have the following user roles: customer (browse and order cakes online, view their order history, and make payments) and admin (manage inventory, track orders, generate reports, and manage customers and orders).

# Elaboration – Iteration 1.1

# Domain Model

The domain model represents the structure and relationships of the main entities or objects that are relevant to the system. It defines the classes, attributes, and methods that represent the core business logic of the application. The domain model in this case would typically include entities such as:

**Cake:** Represents a cake in the catalog, with attributes such as cake ID, name, description, price, and image.

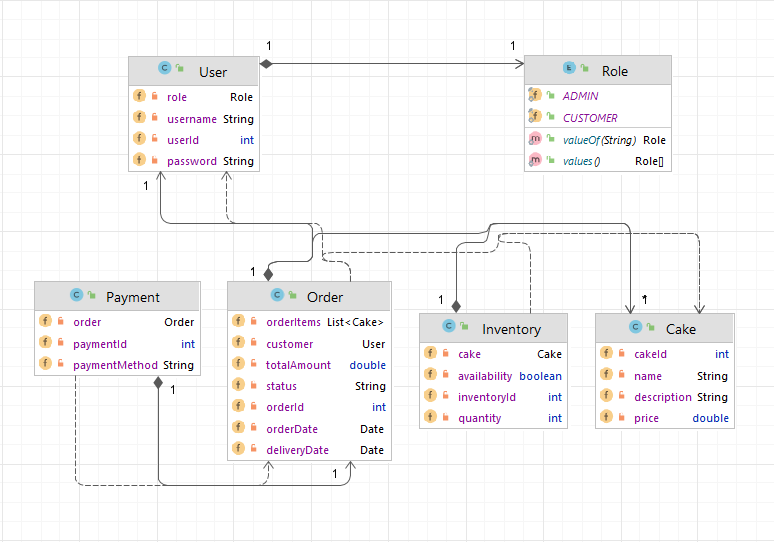
**Order:** Represents an order placed by a customer, with attributes such as order ID, customer details, order items (cakes), total amount, and status.

**User:** Represents a user of the system, with attributes such as user ID, username, password, and role (customer, admin, bakery staff).

**Inventory:** Represents the inventory of cakes, with attributes such as cake ID, quantity, and availability status.

**Payment:** Represents a payment made by a customer, with attributes such as payment ID, payment method, and amount.

The domain model would also include relationships between these entities, such as associations, aggregations, or compositions, depending on the requirements of the system. For example, an order would have a one-to-many relationship with order items (cakes), and a many-to-one relationship with a customer who placed the order. Similarly, the user entity would have a role attribute that represents the role of the user in the system.



# Architectural Design

## Conceptual Architecture

The Cake Shop Management System can be designed using the Model-View-Controller (MVC) architectural pattern, which is a widely used architectural style in web applications, including Java Spring Boot applications.

The MVC pattern separates the application into three main components:

**Model:**

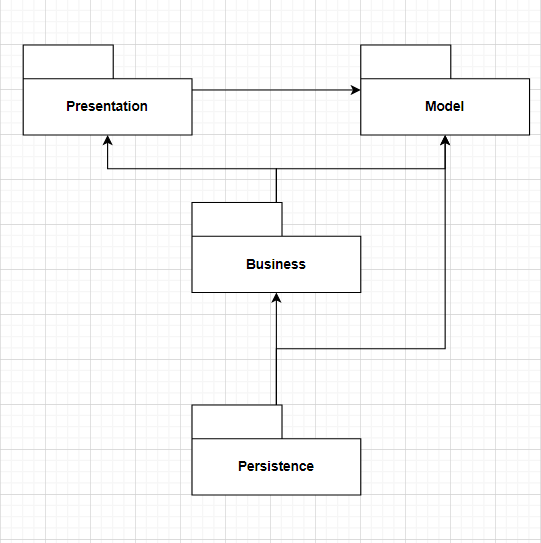
Represents the data and business logic of the application. It includes classes such as Cake, Order, User, Inventory, and Payment, which model the domain objects and their relationships. These classes encapsulate the data and behaviors related to their respective entities.

**Controller:**

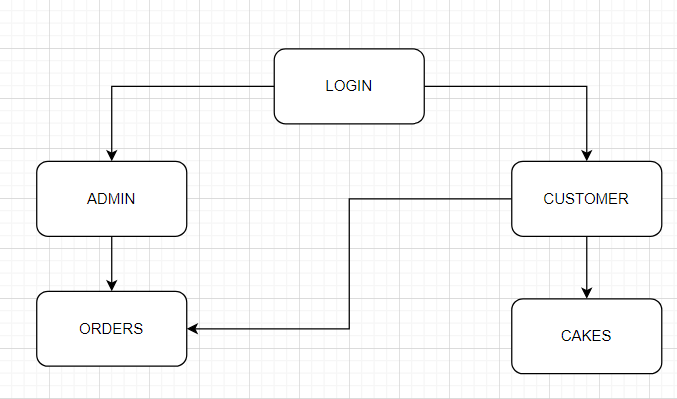
It receives user inputs from the View, processes them, and updates the Model. It handles user interactions, processes business logic, and coordinates the flow of data. In a Spring Boot application, the Controller is typically implemented using Spring MVC framework.

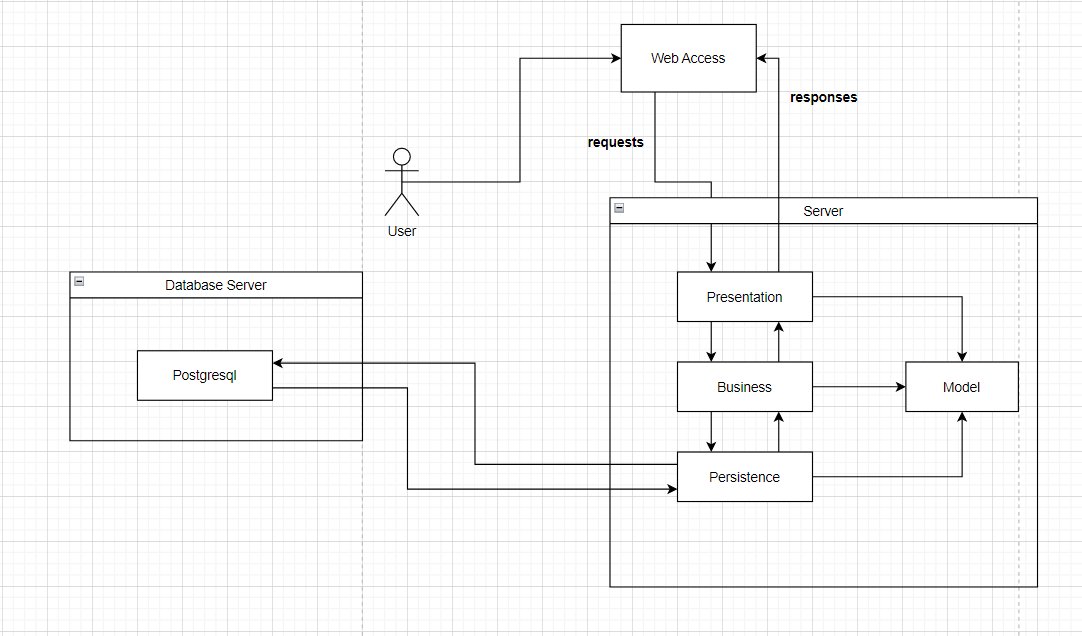
MVC promotes a modular approach, where changes in one component do not affect the others. It also allows for reusability of components. MVC provides flexibility in choosing different technologies for each component. For example, in a Spring Boot application, the Model can be implemented using JPA for data persistence and the Controller can be implemented using Spring MVC for handling user input. Scalability is also a motivation for using a MVC architectural pattern because it allows for easy scalability by adding or modifying components as needed, without disrupting the entire system.

## Package Design



## Component and Deployment Diagrams



****

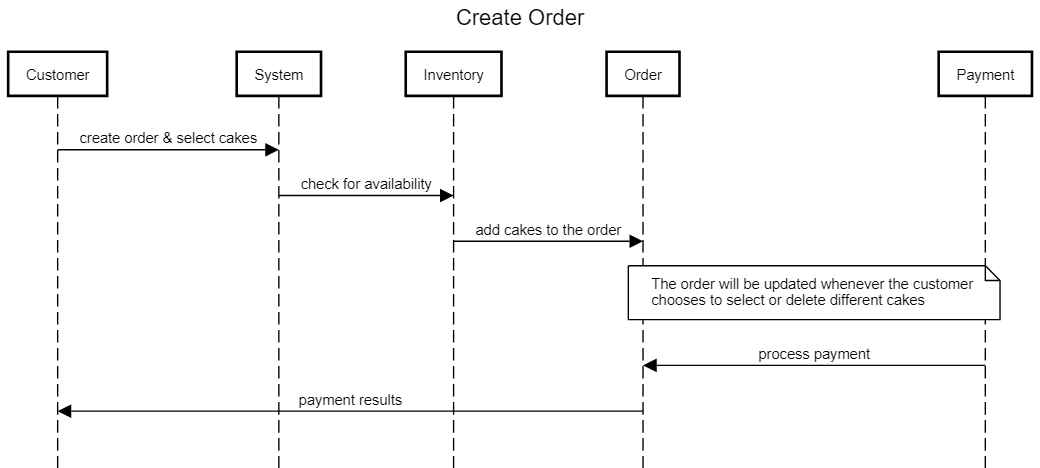
# Elaboration – Iteration 1.2

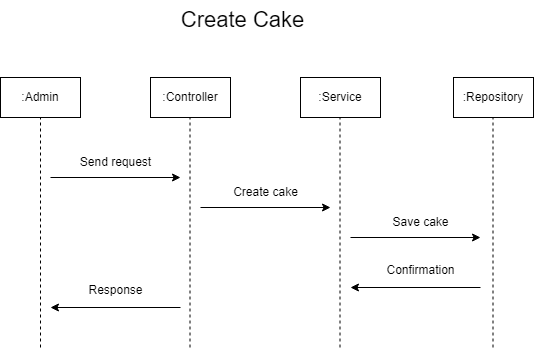
# Design Model

## Dynamic Behavior

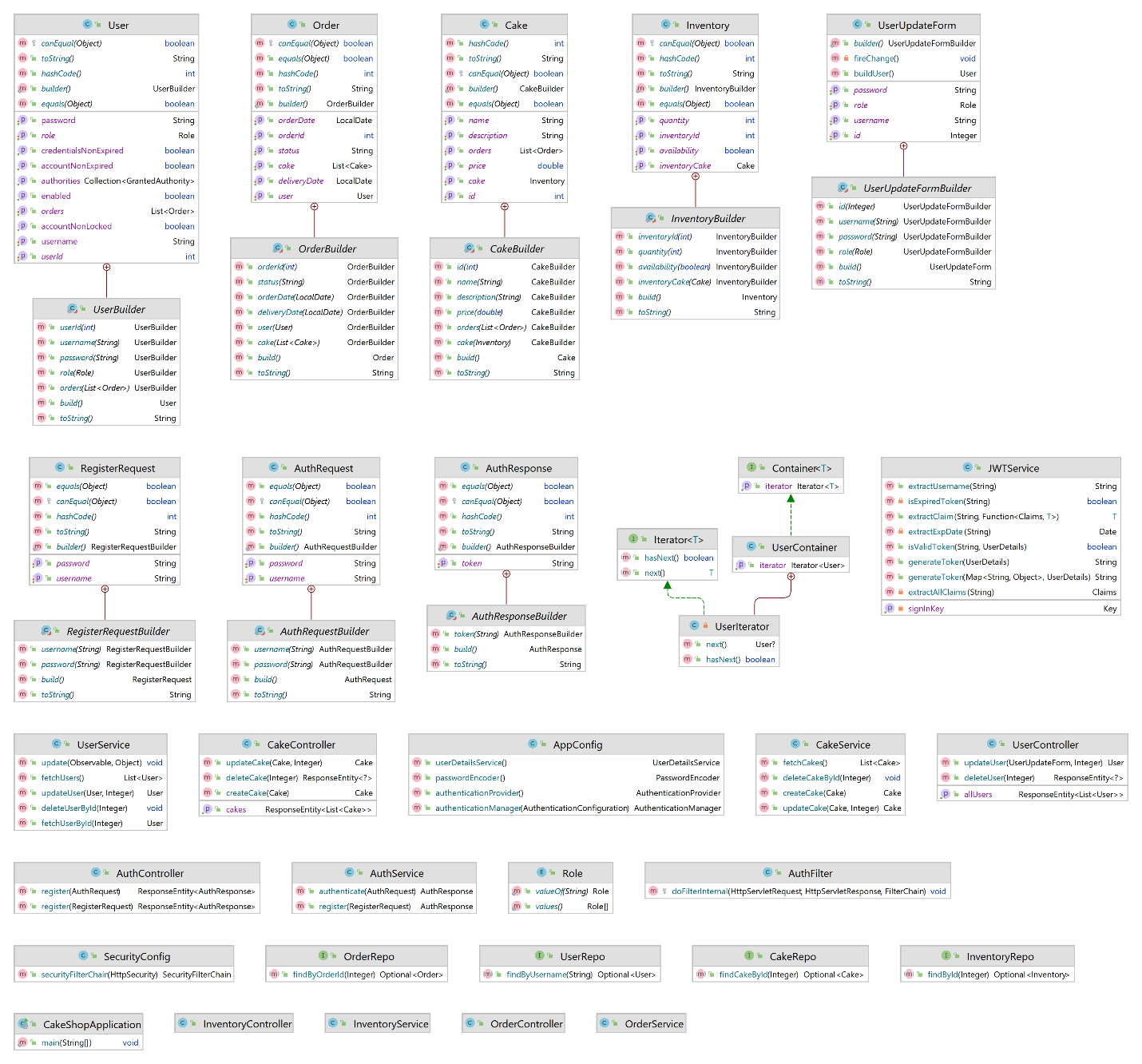
A sequence diagram is a type of interaction diagram that shows the interactions between objects or components in a system, typically in the order in which they occur. A communication diagram, on the other hand, shows the interactions between objects or components in a system in a more visual and intuitive way, using arrows to indicate the flow of messages between them.

Sequence diagram for creating an order and communication diagram for creating a cake:





## Class Design

**

# Data Model

The data model describes the structure and relationships of the data that the application uses. It is a representation of the various entities in the application and the attributes that define them. In this case, the data model consists of five entities: Cake, Order, User, and Payment.

Each entity has a set of attributes that defines it. For example, the Cake entity has attributes such as cake ID, name, description and price. The Order entity has attributes such as order ID, customer name and customer ID, order items (cakes), total amount, and status. Also, there are delivery date and order date. The User entity has attributes such as user ID, username, password, and role (customer and admin). The Payment entity has attributes such as payment ID, payment method, and amount.

The data model also includes the relationships between these entities, such as associations and aggregations. For instance, an order would have a many-to-many relationship with order items (cakes), and a many-to-one relationship with a customer who placed the order. Similarly, the user entity would have a role attribute that represents the role of the user in the system.

# Unit Testing

The unit testing focused on the cakes’ operations. The create and delete methods were tested by creating a new class with the @SpringBootTest, indicating that this main role of this class is testing, and @Transactional, which indicates that the results are rolled back at the end of the test, annotations. Another important annotation is @InjectMocks and @Mock, which creates an instance of the interface CakeRepo, creates a mock and inject the CakeService mock into the test class.

The first implemented method, initTest(), creates an instance of the Cake to be used further in the tests. This method used the JavaFaker library to generate a real-looking data from addresses to popular cultural references.

The second method implemented, testCreateCake(), tests the cakeService.createCake() method. Firstly, it saves a fake cake, the one returned in the initTest() method, then it creates an actual cake by using the createCake() method defined in the CakeService class. The results of these operations are then compared by using assertions. The assertion used in this case is assertEquals(cake, result).

The last method implemented tests the cakeService.deleteCakeById(cake). The mock behavior is set by returning an optional by the findById method containing the provided cake object, then it deletes the cake. The assertion used in this case is assertDoesNotThrow(), meaning that no exception is thrown during the deletion process.

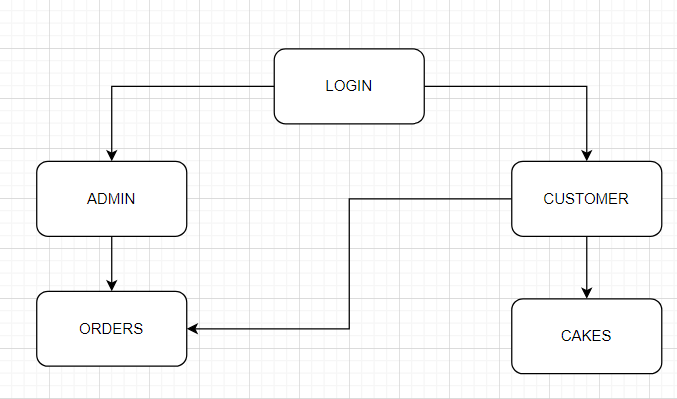
To be noted that for all these methods, the annotation used is @Test which will help the Junit testing framework to recognize them as test cases and execute them separately.

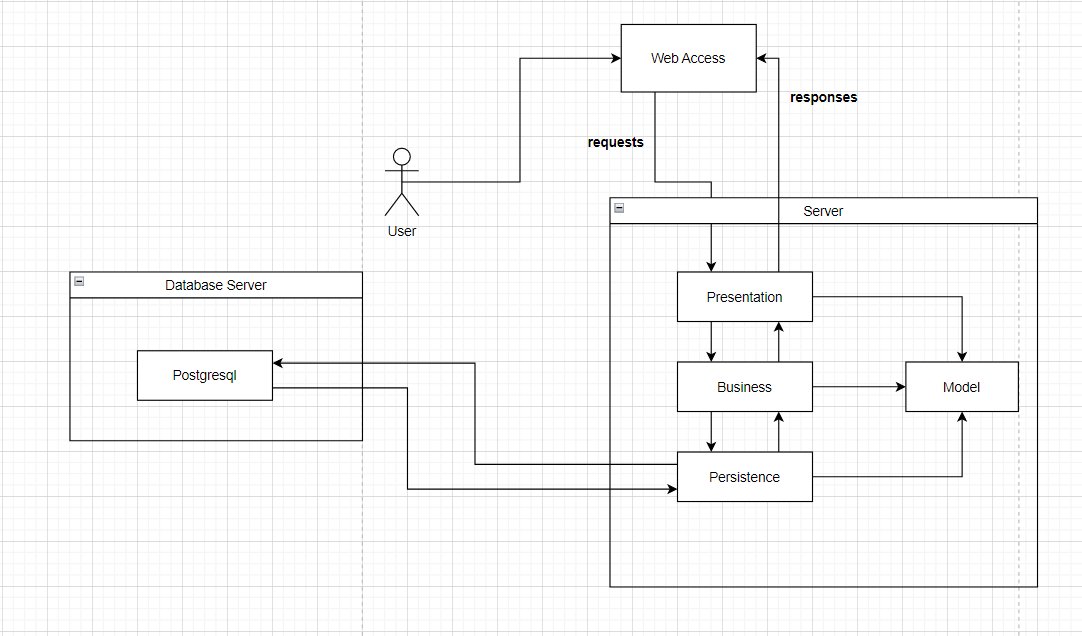
# Elaboration – Iteration 2

# Architectural Design Refinement

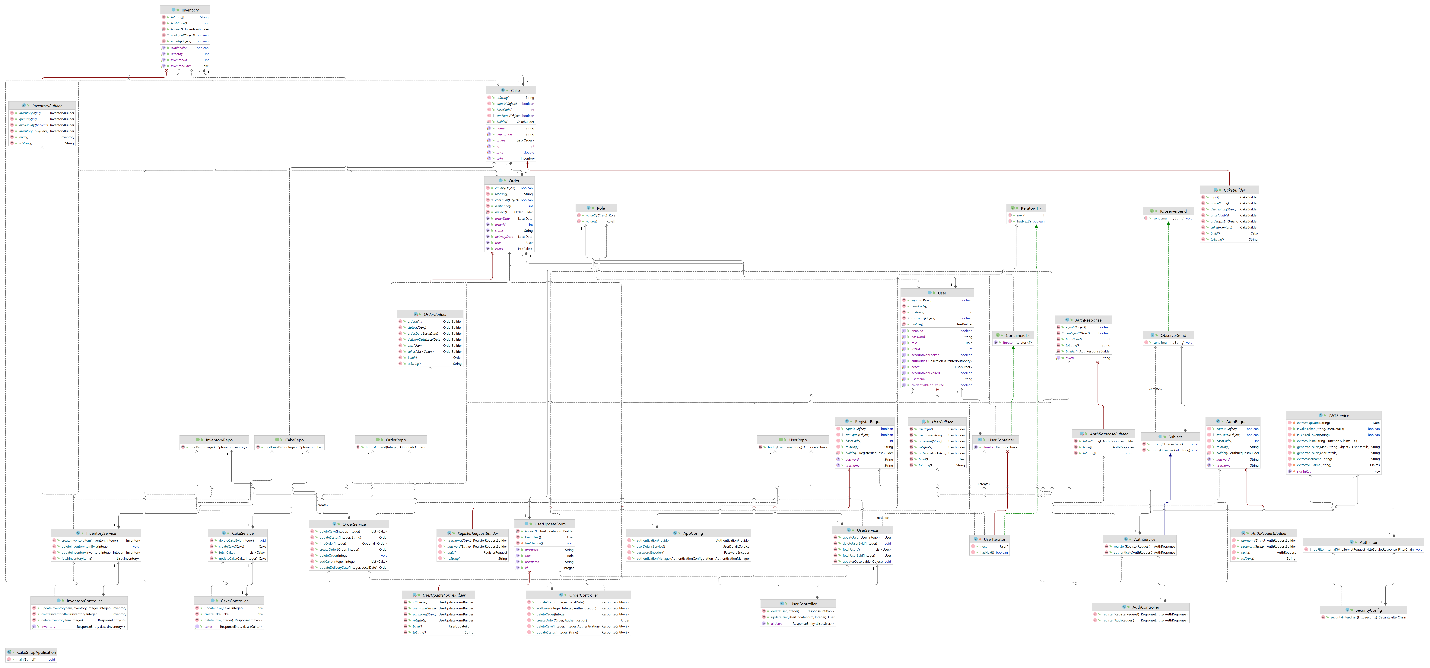
The packages used to design the application are:

* entity – contains all the classes which models the data
* service – contains all the classes responsible to the operations performed on the data stored in the database
* controller – contains all the classes which are responsible for processing incoming requests and returning the responses based on the methods called from the services
* repo – contains all the interfaces responsible for providing the mechanism for storage, retrieval and search
* config – contains all the classes responsible for the security of the application
* auth – contains all the classes responsible for authentication and register process
* iter – contains all the classes responsible for implementing the iterator pattern
* mail – contains all the classes responsible for implementing a mail service



****

# Design Model Refinement

****

# Construction and Transition

# System Testing

The integration testing in the Cake Shop application is described by the @SpringBootTest annotation which indicates that this class is responsible for testing and allows the testing of the application.

**Testing the creation of a cake:**

**Scenario:** Verify that a cake can be created and saved correctly in the database.

**Steps:** Provide mock behavior, create a cake and verify if the fake cake and the actual result are equal or not.

**Testing the deletion of a cake:**

**Scenario:** Verify that a cake can be deleted correctly from the database.

**Steps:** Provide a mock behavior, delete the cake and verify if there is an exception thrown or not.

# Future improvements

Some future improvements for the cake shop application would be:

* creating a user-interface,
* implementing a payment method,
* generating a bill based on the order,
* notify the user via mail/SMS over the order changes

# Bibliography

* <https://www.baeldung.com/java-faker>
* <https://www.baeldung.com/java-iterator>
* <https://www.tutorialspoint.com/design_pattern/observer_pattern.htm>
* Lecture slides & laboratory presentations