Assignment 1 part 2

CSE2115: Software Engineering Methods

Group 06b

TU Delft

Assignment 1 part 2

by

Group 06b

Instructor:	Dr. A. Panichella & F. Mulder
Teaching Assistant:	M. Segers
Institution:	Delft University of Technology
Place:	Faculty of Electrical Engineering, Mathematics and Computer Science
Date:	23-12-2022

TU Delft

Summary

This report provides an overview of the proposed architecture for Anni's Pizza online portal, which was explained in Part 1 of the report, and also highlights two design patterns that were used when building the portal, which is elaborated in this report.

Part 2

Design patterns are conventional responses to frequent issues in software development. Each pattern functions as a blank canvas that we may use to address a specific design issue in our system. In this report, we will provide a description of why and how the design patterns are implemented in our system and their corresponding class diagrams. Both design patterns are implemented in the code.

Contents

1

Facade design pattern

The facade pattern conceals the system's complexity and offers the client an interface via which they can access the system. Each of our microservice contains several complex structured service classes and a controller class. As the number of service classes grows, it will become more complicated for the controller class to access each of them. Given that the facade pattern adds an interface to an existing system to conceal its complexity, we apply it to our microservices.

We have implemented the facade pattern to three of our microservices to consolidate different service classes into a single facade service class such that the controller class only needs to interact with the facade service class.

1.1 Store Microservice

The store microservice is constituted by the store feature and email feature. The store feature has three service classes: ProcessStoreService, QueryStoreService, ValidateStoreService. We aggregate those into a facade service class called StoreService. The StoreController only interacts with the StoreService class that has all functionalities of the classes mentioned above.

Additionally, the email feature embraces ProcessEmailService, QueryEmailService and ValidateManagerService. We encapsulate those into the EmailService class such that the EmailController only calls this facade service class.

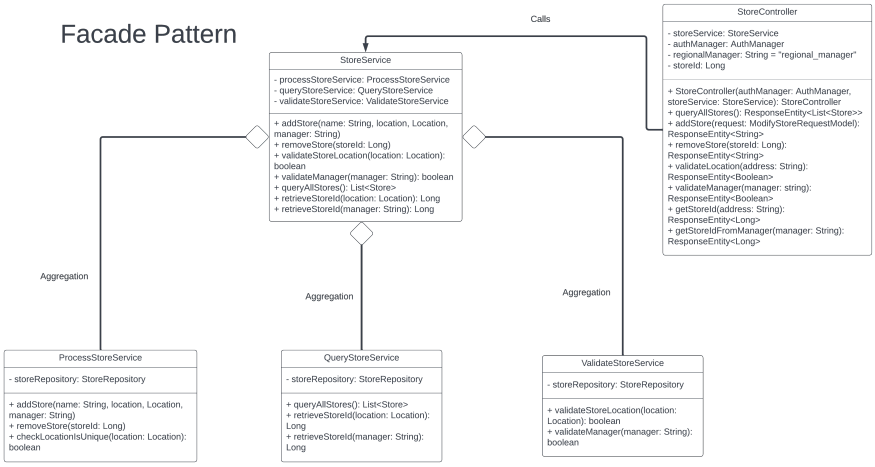

Figure 1.1: StoreService class diagram

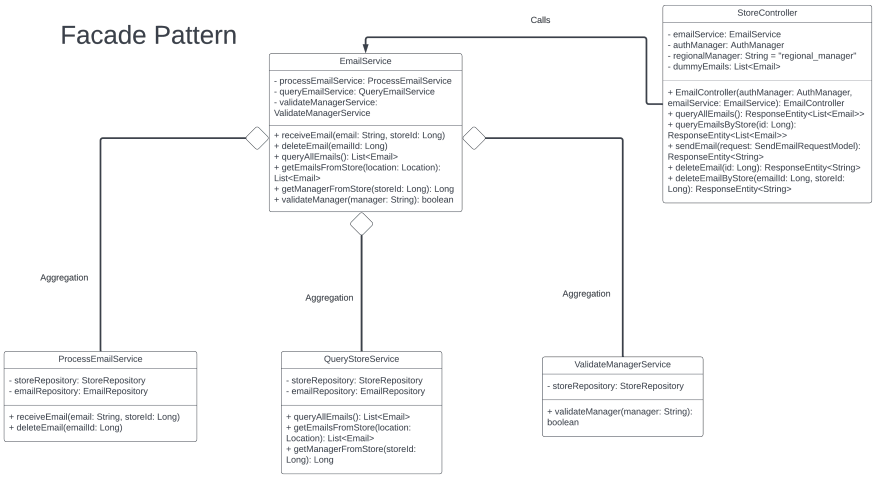

Figure 1.2: Email class diagram

1.2 Authentication Microservice

The authentication microservice consists of four service classes: RoleCreationServiceImpl, JwtTokenGeneratorImpl, RegistrationServiceImpl, JwtUserDetailsService. A facade service class called AuthenticationServiceImpl is created by combining the service classes. Only the AuthenticationServiceImpl will be contacted by the AuthenticationController.

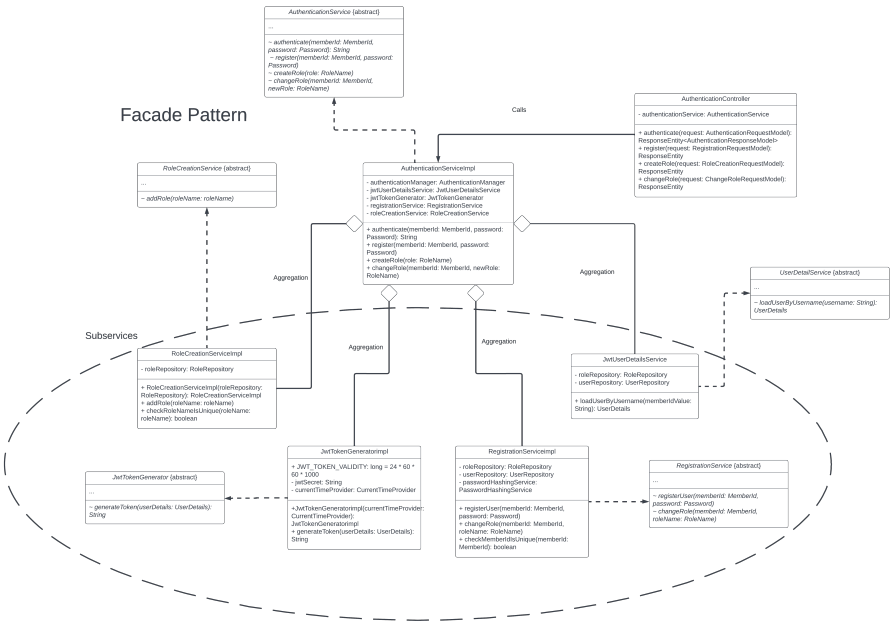

Figure 1.3: Authentication class diagram

1.3 Order Microservice

The order microservice has three service classes: OrderEditorImpl, OrderProcessorImpl, OrderCouponImpl. Besides, it is built by three communication classes to support the communication between the order microservice and other microservices. We aggregate the service classes into a facade service class called OrderServiceImpl. The OrderController will only interact with the OrderServiceImpl.

Furthermore, given the complexity for the OrderController to access each communication class separately, we aggregate those into the OrderProcessorImpl class to improve the usability. The facade pattern is applied here as well.

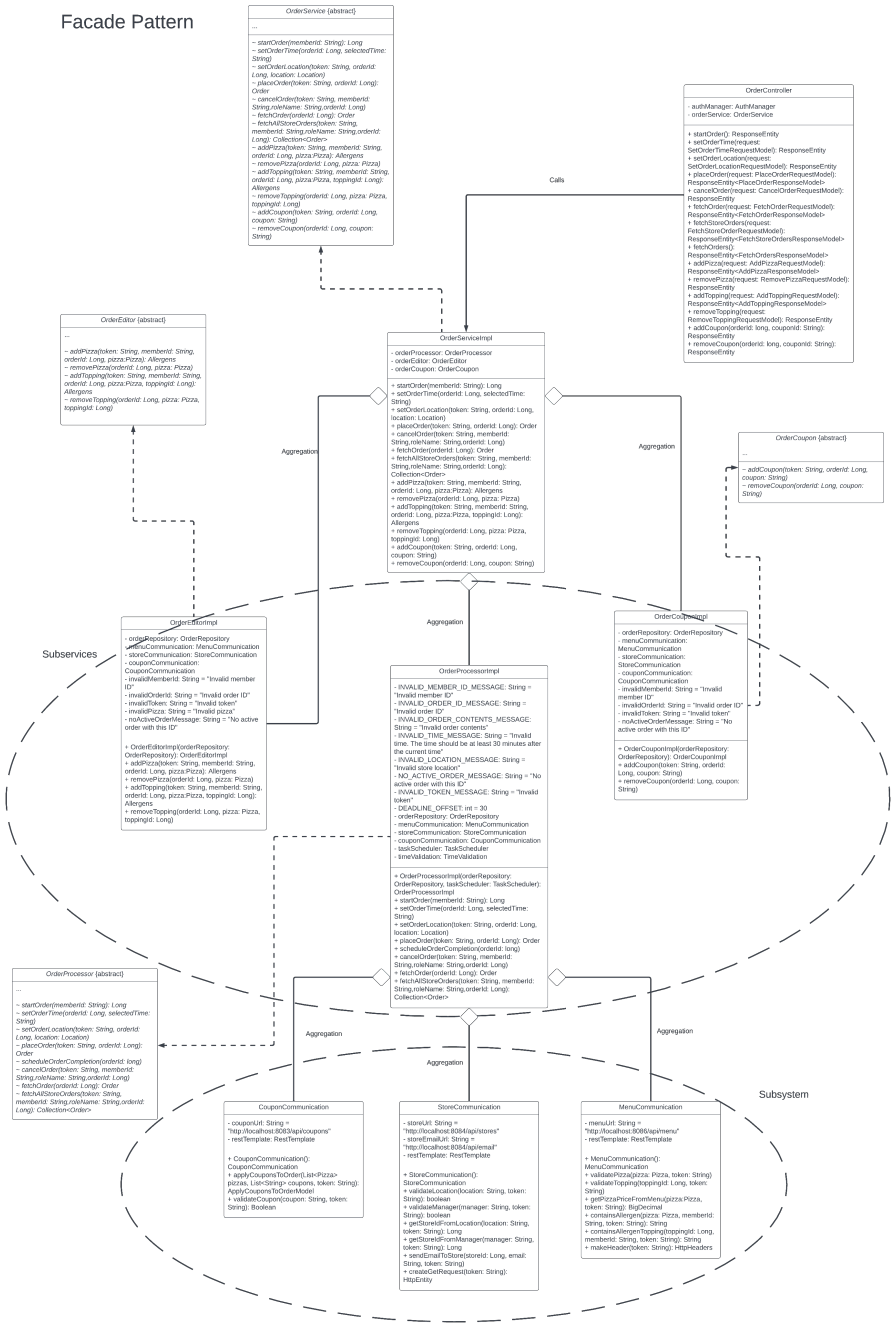

Figure 1.4: Order class diagram

2
Builder design pattern

The builder design pattern enables us to build complicated objects step by step. The ability to generate several kinds and instances of an object while still utilizing the same building code is another benefit. As the number of the attributes of the order class increases, it becomes complicated for our system to construct different order objects. Besides, creating the order object directly within the class becomes error-prone because it might be difficult for the client side to keep the order of the arguments when there are too many arguments being passed to the constructor. Given that the builder pattern separates the creation of a complex object's construction from its representation, we apply it to our system.

We have implemented the builder pattern into the order microservice to imporve the creation and maintainbility of different orders.

Order Microservice

To create an order, we create a separate Builder interface that contains the creation and assembly of the components of an order. We divide the building process into 17 distinct building methods that each function on their own A properly constructed order object will be returned by the build method. Instead of constructing the orders directly, the director classes delegates their creation to an OrderBuilder object that builds an order object without much complex logic in the building process.

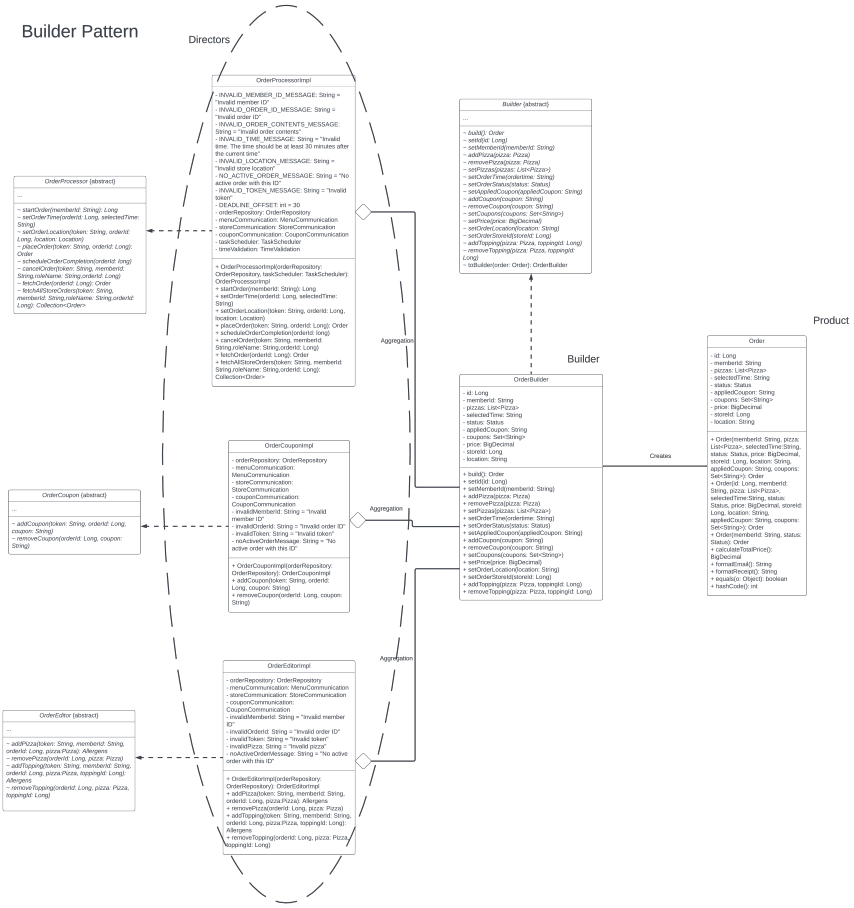

Figure 2.1: OrderBuilder class diagram

List of Figures