Restaurant Management System

Student: Ratoi Razvan Valeriu

**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 purpose of this assignment was to design and implement a client-server application used to record orders, handle payments and manage the employees’ activity in a restaurant.

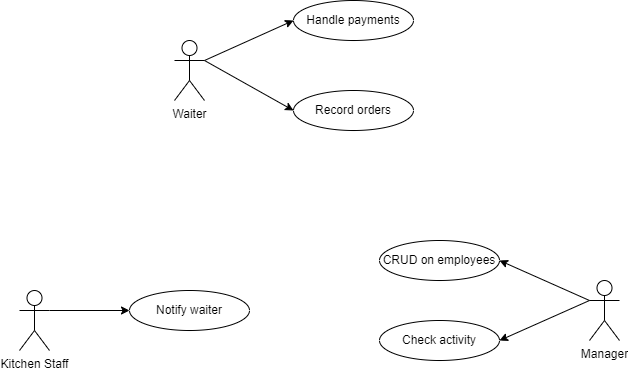
# Functional Requirements

The system allows (*i*) waiters to record orders in the system and to handle payments by considering discounts depending on the loyalty of the client, (*ii*) kitchen employees to send notifications to waiters when the ordered food is ready, and (*iii*) a manager to perform CRUD operations on employees’ information and to check their activity. An order has the following attributes: table number, client ID, waiter ID, the list of ordered products (drinks, food, desert, etc.) as well as their quantity. The loyalty of a client is evaluated according to the number of times the client ordered from the restaurant as well as the cost of each order. This application also has to contain the Observer and Command Design Patterns.

# Non-functional Requirements

Some non-functional requirements would be:

* The data should be stored in a relation database
* The application should be organized in the layered architecture pattern
* Database access should be done using ORM

2. Use-Case Model

Use case: Handle payment

Level: User-goal

Primary actor: Waiter

Main success scenario:

1. The Staff triggers the Execute method by providing a Staff ID and an Order ID as input parameters.
2. The system retrieves the Order object using the provided Order ID from the Order repository.
3. The system creates an instance of EmailSender and attaches it to the Order object.
4. The system retrieves the Client object using the Client ID associated with the Order object from the Client repository.
5. The system calculates the total price of the Order using the GetOrderPrice method with the Order object and the Client object as input parameters.
6. The system updates the loyalty points of the Client using the UpdateLoyaltyPoints method with the Client object as an input parameter.
7. The system sets the total price of the Order and changes its status to "Paid".
8. The system notifies all the subscribers of the Order object by calling the Notify method with the value -1 as an input parameter.
9. The system updates the Order object in the Order repository.

Extensions:

* If the Order object cannot be retrieved from the Order repository, the system logs an error and terminates the use case.
* If the Client object cannot be retrieved from the Client repository, the system logs an error and terminates the use case.
* If any of the methods (GetOrderPrice, UpdateLoyaltyPoints, Notify, or Update) fails, the system logs an error and terminates the use case.

3. System Architectural Design

**3.1 Architectural Pattern Description**

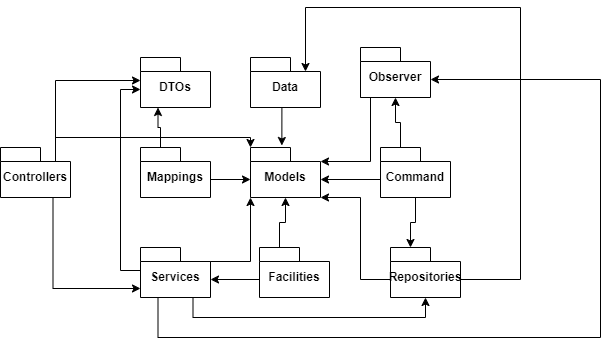
Without having a front-end side for this application, the best way to reproduce the MVC architectural pattern with a layered architecture. Hence, I will be using 3 layers for the back-end of the application: Repository (the one accessing the database), Service (the one processing the data) and Controller (that connects the logic with the requests).

*Diagram

Description automatically generated***3.2 Diagrams**

Text

Description automatically generated

**

*Diagram

Description automatically generated with low confidence*4. UML Sequence Diagrams

5. Class Design

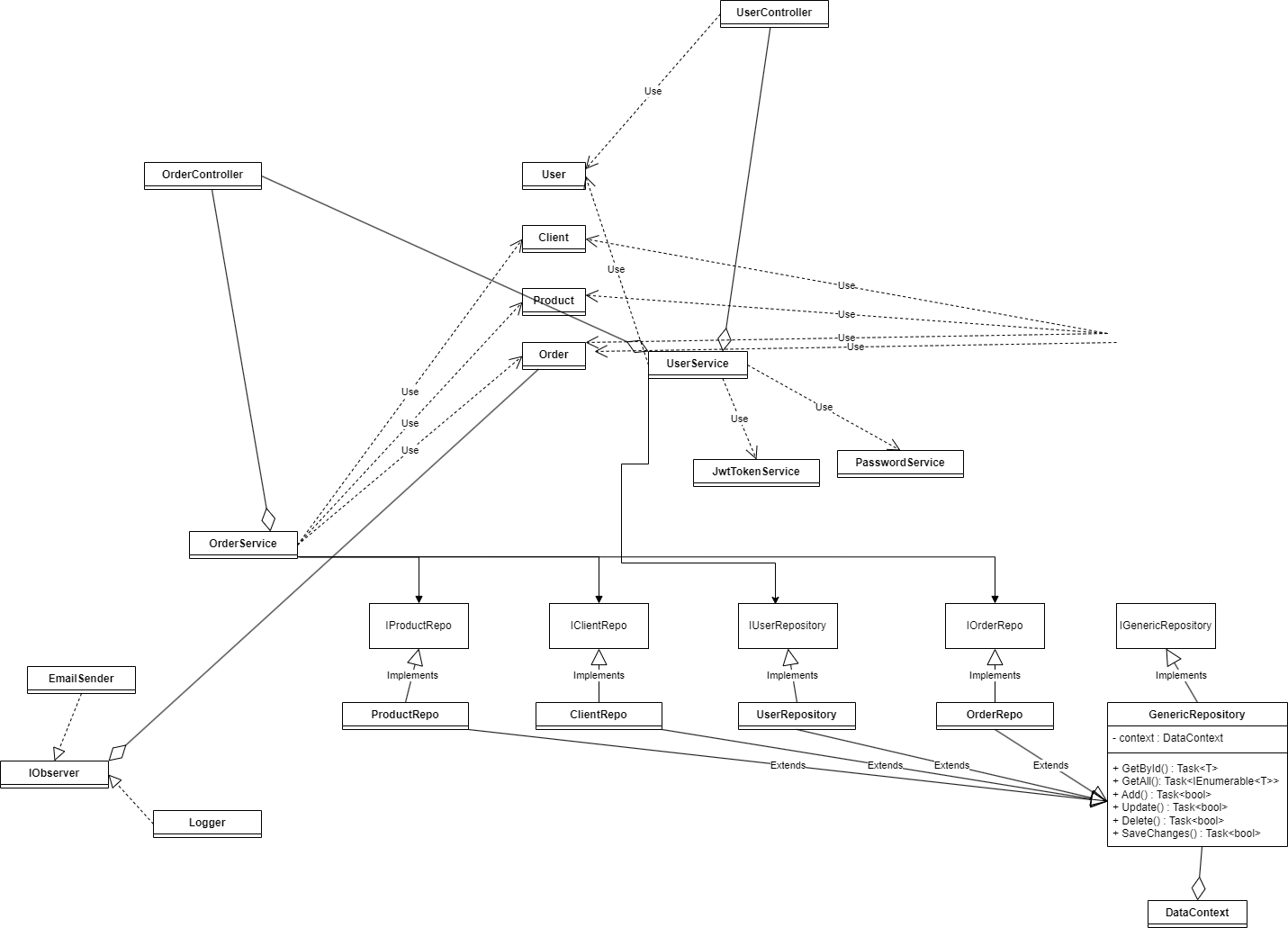
**5.1 Design Patterns Description**

One design pattern I used was the Observer Design Pattern, in which the subject (observable) was the order, which would notify its observers when the status has changed. The observers were an EmailSender and a Logger, that would, based on the status of the order act:

* When “Pending”, it would be logged that a waiter has taken an order.
* When “Finished”, it would be logged that a kitchen worker has prepared the order.
* When “Paid”, an email would be sent to the client acknowledging them that the order was paid.

I also used the Command Design Pattern, in which the commands were one for both the waiters and the kitchen staff, one would be handling the payment of the order and the other one preparing the order. These commands were invoked by the order service object.

**5.2 UML Class Diagram**

**

6. Data Model

* User has the needed information about the user of an application, including the role.
* Client contains the contact information of clients of restaurant.
* Category represents a static entity in the database, only having the name.
* Product contains some detail information, the category and the price.
* Order contains the id of the waiter, the id of the client, the total, the table number, the date and time of the order and the list of products.
* I also used some DTOs for the user, for logging in the application and in case of needing to return an user (to avoid displaying sensitive information).

7. System Testing

For testing the application I created a Tester class that contains some unit tests, to validate the functionality of some methods of the application. For a wider testing, I used Insomnia to call each request and verify the output with what I was expecting it to be.

8. Bibliography

* C# fundamentals recap (https://app.pluralsight.com/library/courses/object-oriented-programming-fundamentals-csharp/table-of-contents)
* C# collection fundamentals (https://app.pluralsight.com/player?course=csharp-collections&author=simon-robinson&name=csharp-collections-fundamentals-m1-welcome&clip=0)
* Head first C# book (https://www.amazon.com/NET-Head-First-Microsoft-Programming/s?ie=UTF8&page=1&rh=n%3A764452%2Cp\_lbr\_books\_series\_browse-bin%3AHead%20First)
* Web API (https://app.pluralsight.com/library/courses/implementing-restful-aspdotnet-web-api/table-of-contents)
* C# basic authentication check with user & password and authorization using roles (for assignment 3) https://www.c-sharpcorner.com/article/basic-authentication-using-message-handlers-in-web-api/
* Message handlers in C# (https://www.infoworld.com/article/2991458/application-architecture/how-to-work-with-message-handlers-in-web-api.html)
* http://www.tutorialsteacher.com/webapi/consuming-web-api-in-dotnet-using-httpclient
* https://www.aspsnippets.com/Articles/Call-Consume-Web-API-in-Windows-Forms-Application-using-C.aspx