<Assignment 3>

Student:Crisan Oana Andra

**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 goal of this project is to design and implement a client-server application that can be used by a restaurant to record orders, handle payments, and manage the employees' activity. The system should allow waiters to record orders, handle payments, and apply discounts based on the loyalty of the client. Kitchen employees should be able to send notifications to waiters when the ordered food is ready. A manager should be able to perform CRUD operations on employees' information and check their activity.

The order should have the following attributes: table number, client ID, waiter ID, the list of ordered products (drinks, food, dessert, etc.), and their quantity. The loyalty of a client should be evaluated based on the number of times the client ordered from the restaurant and the cost of each order.

# Functional Requirements

User Authentication:

The system should provide authentication mechanisms for waiters, kitchen employees, and the manager.

Waiters, kitchen employees, and the manager should have separate login credentials with appropriate access levels.

Order Recording:

Waiters should be able to record orders in the system.

The system should allow waiters to specify the table number, client ID, waiter ID, and the list of ordered products with their quantities for each order.

The system should validate and store the recorded orders securely.

Payment Handling:

The system should calculate and apply discounts based on the loyalty of the client.

Loyalty should be evaluated based on the number of times the client has ordered from the restaurant and the cost of each order.

Notifications:

Kitchen employees should be able to send notifications to waiters when the ordered food is ready.

The system should provide a mechanism for the kitchen employees to notify the relevant waiter about the readiness of specific orders.

Employee Management:

The manager should be able to perform CRUD operations (Create, Read, Update, Delete) on employees' information.

The manager should be able to add, view, update, and remove employee details such as name, contact information, etc.

The system should provide appropriate access controls to ensure only the manager can perform these operations.

Security and Data Privacy:

The system should ensure the security and privacy of user data, including authentication credentials, order details, employee information, and activity logs.

Data transmission between the client and server should be encrypted to protect sensitive information.

# Non-functional Requirements

*-* Data will be stored in a relational database.

- Use the Layers architectural pattern to organize your application.

- Passwords must be encrypted when stored to the database with a one-way encryption algorithm (base 64).

- Postman collection of operations

- ORM Hibernate framework to access the database

- API design should be RESTful, not SOAP

2. Use-Case Model

Use case: Place an order

Level: User-goal level

Primary actor: Waiter

Main success scenario:

1.The waiter logs into the system.

2.The waiter selects "Create a new order"

3.The system prompts the waiter to enter the Lab Class details (table number, client id, title, list of ordered products id with quantity).

4.The teacher enters the order details with order status “in making”.

5.When the food is ready a kitchen employee changes the status to “ready”.

USE CASE DIAGRAM FOR CLIENT:

A diagram of a person

Description automatically generated with medium confidence

USE CASE DIAGRAM FOR WAITER:

A diagram of a person with text

Description automatically generated with low confidence

USE CASE DIAGRAM FOR ADMIN:

A diagram of a person with text

Description automatically generated with low confidence

USE CASE DIAGRAM FOR KITCHEN EMPLOYEE:  
A picture containing diagram, line, plot, circle

Description automatically generated

3. System Architectural Design

**3.1 Architectural Pattern Description**

A layered architecture is a design pattern in which software components are organized into horizontal layers, with each layer having a specific responsibility and communicating only with its adjacent layers. Typically, the layers are ordered from top to bottom, with the top layer being the user interface layer, the middle layer being the business logic layer, and the bottom layer being the data access layer. This architecture promotes modularity, separation of concerns, and maintainability, as changes made to one layer do not affect the other layers.

**3.2 Diagrams**

Package Diagram:

Diagram

Description automatically generated

Component Diagram:

**Diagram

Description automatically generated**

4. UML Sequence Diagrams

Scenario 1: Client placing an order

A picture containing text, diagram, line, parallel

Description automatically generated

Scenario 2: Update a product (admin)

A picture containing text, diagram, line, screenshot

Description automatically generated

5. Class Design

**5.1 Design Patterns Description**

The Repository Design Pattern is a widely used approach in layered software architectures for managing data access and storage. It follows the principle of encapsulation, where each data entity or object has a corresponding repository that acts as an intermediary between the data source and the rest of the application. The repository abstracts the details of data storage and retrieval from the rest of the application by providing a set of methods for creating, reading, updating, and deleting data.

In a Java application with Spring, the Repository pattern can be implemented using Spring Data. This framework offers a set of interfaces and annotations that simplify data access and storage. Spring Data repositories provide a consistent API for performing CRUD (Create, Read, Update, Delete) operations, and can be easily customized and extended to meet the specific needs of your application.

Using the Repository pattern and Spring Data can provide several benefits for your application, including improved code organization, reduced complexity, and increased maintainability. Additionally, this approach can help you easily switch between different data sources, such as databases or external APIs, without impacting the rest of your application.

The Observer pattern is a design pattern used in software development. It allows objects to be notified automatically when the state of another object changes. In this pattern, there is a one-to-many relationship between the objects: one object, known as the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes.

In Java, the Observer pattern is implemented using the Observer interface and the Observable class. The Observer interface has a single method, update(), which is called by the Observable class when there is a change in its state. The Observable class has two methods, addObserver() and deleteObserver(), to add or remove observers from its list of dependents.

The Command pattern is another design pattern used in software development. It is used to encapsulate a request as an object, thereby allowing the request to be handled by different objects without knowing the request's details. In this pattern, there is a separation between the object that invokes the command (the invoker) and the object that knows how to perform the command (the receiver).

In Java, the Command pattern is implemented using the Command interface, which defines the execute() method, and the Invoker and Receiver classes. The Invoker class invokes the command by calling its execute() method, and the Receiver class performs the command.

**5.2 UML Class Diagram**

*Graphical user interface

Description automatically generated*

6. Data Model

Graphical user interface

Description automatically generated

7. System Testing

All the operations can be tested using Postman and Datagrip to check in the Database. Postman is a popular tool for API development and testing. It allows users to make HTTP requests to RESTful APIs and provides features such as request parameterization, response validation, and automated testing. With Postman, users can create collections of requests, organize and share them with other team members, and run tests on each request to ensure the API is working as expected. It also provides a friendly user interface that makes it easy to inspect and debug requests and responses. Postman supports a wide range of HTTP request methods, including GET, POST, PUT, DELETE, PATCH, and more, and it can be integrated with many popular development tools and services.

In combination, DataGrip and PostgreSQL provide a powerful toolset for developers to work with relational databases. DataGrip provides an intuitive interface for managing PostgreSQL databases, while PostgreSQL provides a powerful and scalable database management system. Together, they enable developers to build robust and scalable web applications that can handle large volumes of data.

I also made a Junit testing for adding a product into the database:

A screen shot of a computer

Description automatically generated with medium confidence

8. Bibliography