Online Grocery-Shopping System

Student: Antonescu Maria-Cristina

**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

In this assignment, the task is developing an Online Grocery Shopping System. The application will be designed to enable customers to order groceries online, and the system will have four types of users - admin, customer, provider, and courier. Admin will have full access to the system and perform CRUD operations on all other user types. Customers can add items to the cart and place an order. Providers can add new products and perform CRUD operations on existing products. Couriers will make deliveries to customers.

# Functional Requirements

Authentication and Authorization

• The application should allow users to log in with their email address and password.

• The application should have four types of users - admin, customer, provider, and courier.

• Only authenticated users should be allowed to access the application features.

CRUD on Customers

• Admin should be able to create, read, update, and delete customer records.

Add/Edit/Delete Products

• Providers should be able to add/edit/delete products.

• For each product, the application should track the name, description, price, and quantity in stock.

CRUD on Cart

• Customers should be able to add items to the cart, view items in the cart, and remove items from the cart.

• The application should track cart items, including product name, price, and quantity.

Place an Order

• Customers should be able to place an order, which includes the list of items in the cart and their delivery address.

• After an order is placed, the application should reduce the quantity of products in stock.

CRUD on Orders

• Admin should be able to create, read, update, and delete orders.

Assign Delivery

• Admin should be able to assign a courier to a delivery.

• Couriers should be able to view their assigned deliveries.

Update Delivery Status

• Couriers should be able to update the delivery status (e.g., delivered, in transit) for each assigned delivery.

Track Order Status

• Customers should be able to track the status of their orders.

Payment

• Customers should be able to pay for their orders online.

View Order History

• Customers should be able to view their order history, including previous orders, order details, and delivery status.

# Non-functional Requirements

1. Security - The application should use secure authentication and authorization mechanisms to ensure only authenticated users have access to the application features.
2. Performance - The application should be able to handle a large number of users and data without any performance issues.
3. Usability - The application should have an easy-to-use interface that is intuitive and easy to navigate.
4. Availability - The application should be available 24/7, with minimal downtime for maintenance.
5. Scalability - The application should be scalable to handle an increasing number of users and data.
6. Reliability - The application should be reliable and robust, with a low risk of failure or downtime.
7. Maintainability - The application should be easy to maintain and update, with clear documentation and code structure.

2. Use-Case Model

*Use case: Place Order*

*Level: User-goal level*

*Primary actor: Customer*

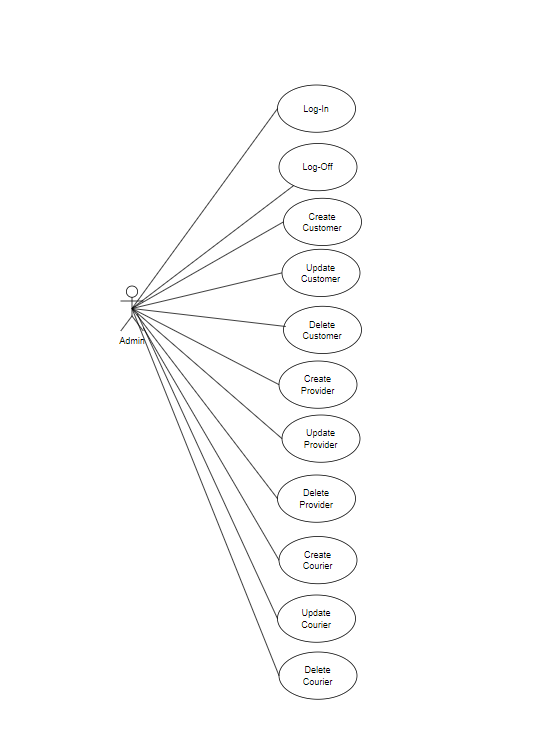
*Main success scenario:*

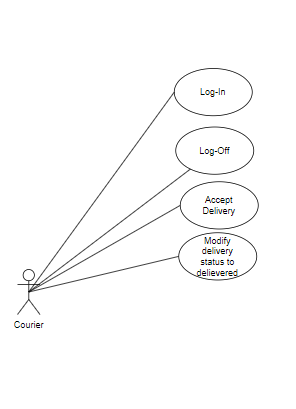
1. *The Customer logs in to the application.*
2. *The Customer selects the desired products and adds them to the cart.*
3. *The Customer reviews the items in the cart and proceeds to checkout.*
4. *The Customer selects a delivery address.*
5. *The Customer selects a payment method.*
6. *The Customer confirms the order.*
7. *The Courier picks up the order and delivers it to the Customer at the selected time slot.*

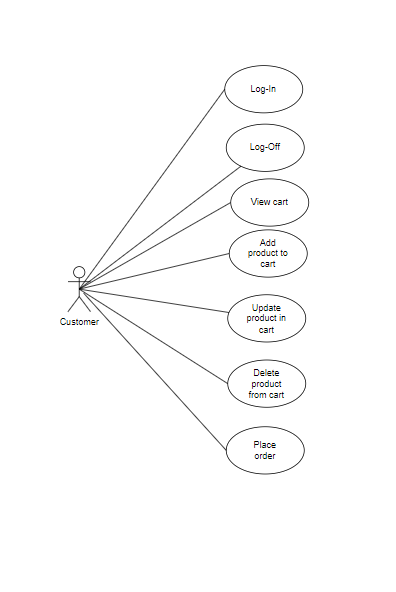
*Extensions:*

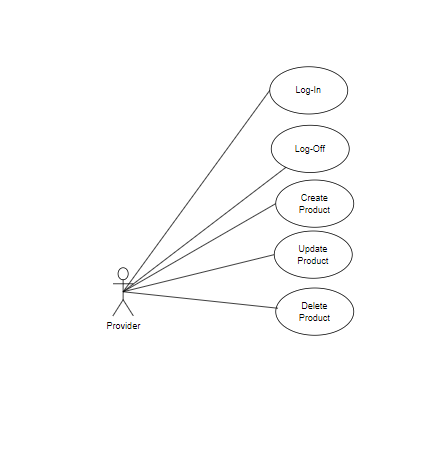
1. *If the Customer tries to place an order without logging in, the application prompts the Customer to log in.*
2. *If the selected products are out of stock or unavailable, the application tells the Customer that the order is invalid.*

*Use case diagrams:*

**

**

**

**

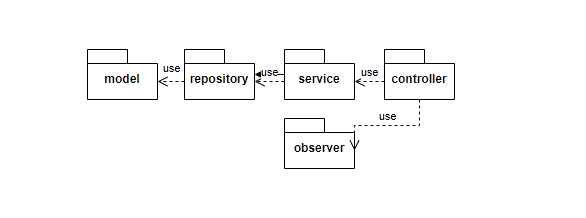
3. System Architectural Design

**3.1 Architectural Pattern Description**

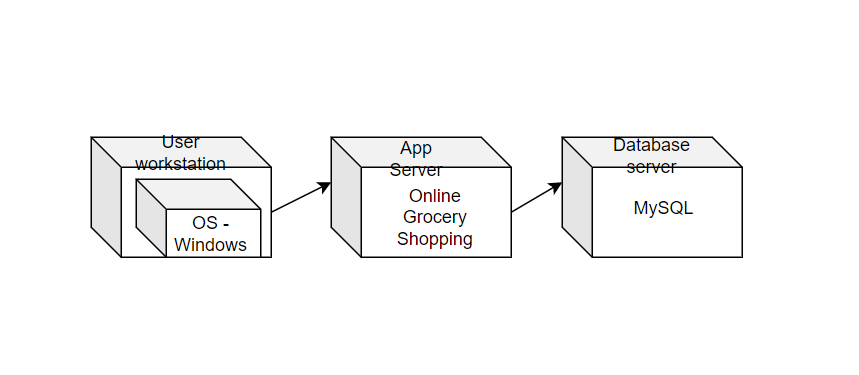
*The application will use the Layers architectural pattern, which separates the application into layers, each with its own set of responsibilities. The layers include the Presentation layer (user interface), Business layer (application logic), and Data Access layer (data storage and retrieval).*

**3.2 Diagrams**

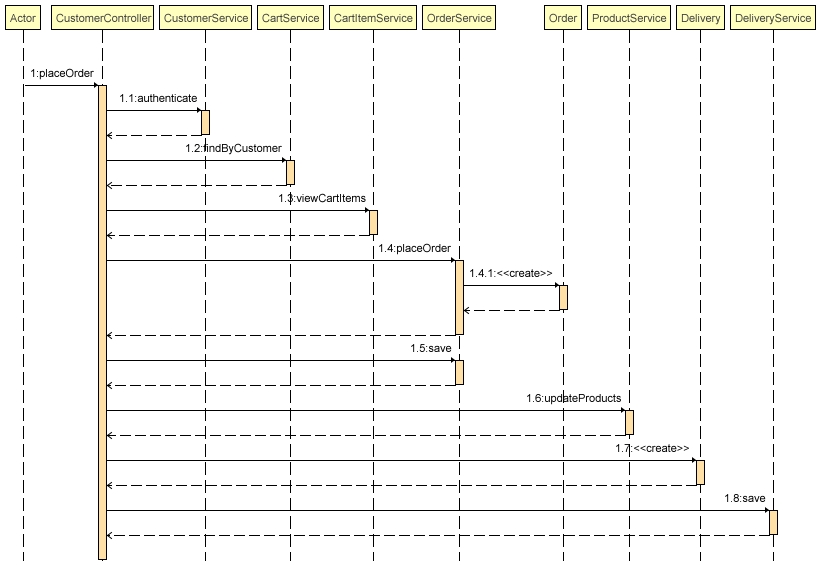
*Package diagram:*

**

*Deployment diagram:*

**

1. UML Sequence Diagrams

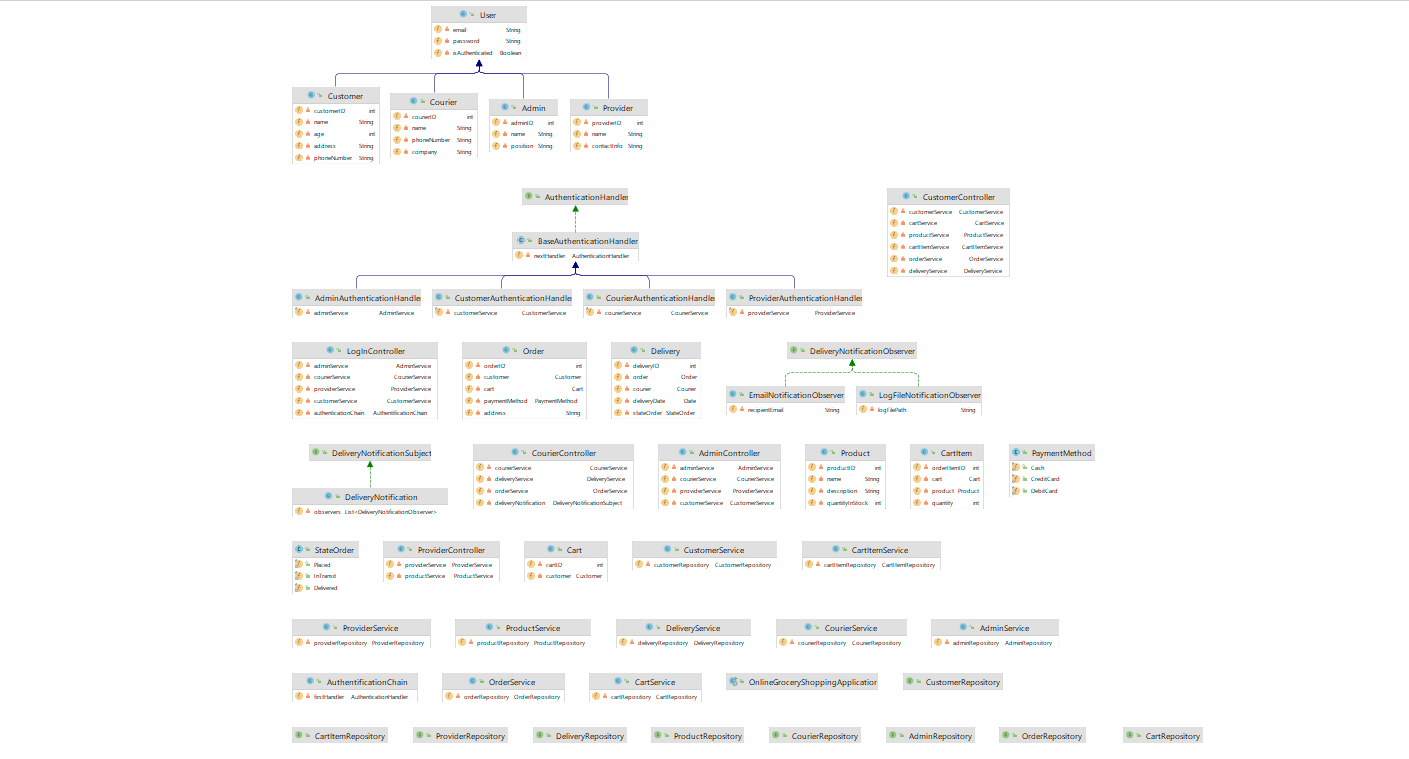
**

5. Class Design

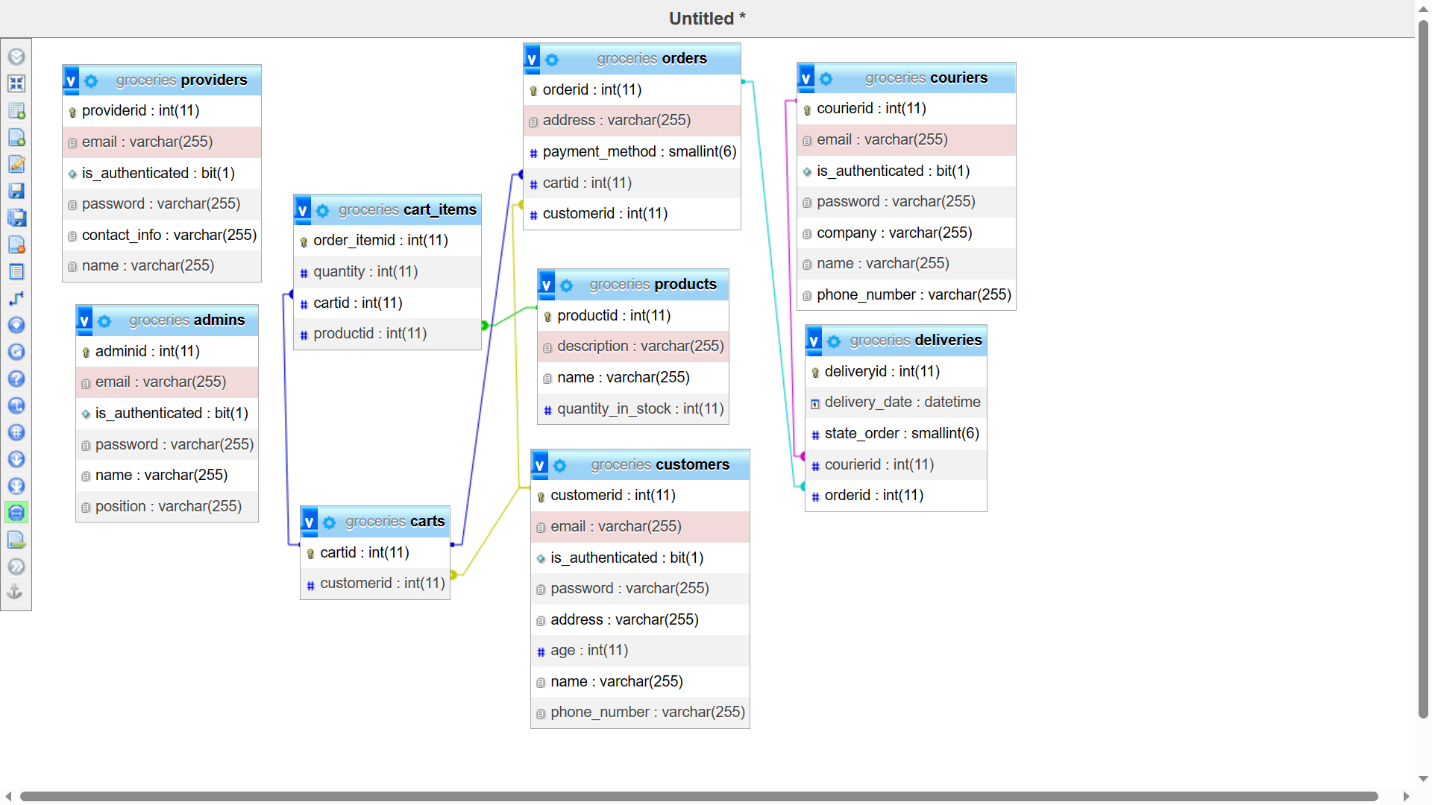
**5.1 Design Patterns Description**

1. ***Repository Pattern*** *- We will use this pattern to interact with the database. The repository pattern will allow us to abstract the data access layer from the rest of the application.*
2. ***Service Layer Pattern*** *- We will use this pattern to handle the business logic of the application. The service layer pattern will allow us to abstract the business logic from the rest of the application.*
3. ***Chain of responsibility pattern*** *– We will use this pattern to have a chain of handlers, where each handler is responsible for handling the login request for a specific type of user. For example, we can have handlers for admin, customer, provider, and courier.*
4. ***Observer Pattern*** *- We can use the Observer pattern to send an email notification to the user after a successful login. The Observer pattern allows an object (the subject) to notify a list of other objects (the observers) of any changes to its state. In the case of the login system, we can have an email sender object as the subject and a list of email notification objects as the observers.*

**5.2 UML Class Diagram**



6. Data Model

**

*There are 5 entities in the system:*

*• Admin – is a type of User, has email, password and has permission to perform CRUD operations on all other types of users.*

*• Customer – is a type of User, has email, password, full name, address, phone numberand age.*

*• Provider – is a type of User, has email, password, name and contactinfo.*

*• Courier – is a type of User, has email, password, name, company and phonenumber.*

*• Product – is an entity which describes a product, it has a name, description, price and total quantity in stock.*

*• Cart – is an entity which represents a customer's cart, it has a reference to the customer.*

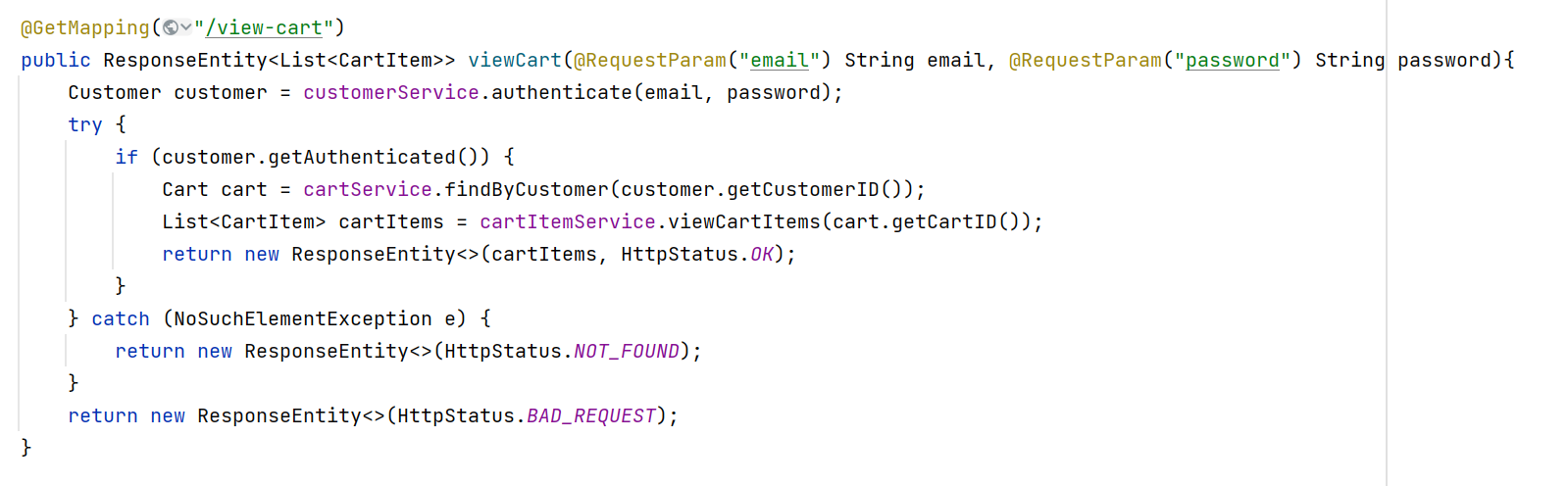
*• CartItem – is an entity which represents a product added to a cart, it has a reference to the cart, a reference to the product and a quantity.*

*• Order – is an entity which represents a customer's order, it has a reference to the customer, a reference to the cart and a payment method and an address.*

*• Delivery – is an entity which represents a delivery, it has a reference to the courier, a reference to the order and a status.*

7. System Testing

*For this online grocery shopping system, I tested the requests in Postman. I returned a ResponseEntity to make sure that my info is update correctly. I also return HTTP codes to make testing much easier. Here is how a class from the controller looks:*

**

*I have also written 2 unit tests, which validate the log-in and the placeOrder methods:*

**

8. Bibliography