Supermarket Manager

Analysis and Design Document

Student: Marin Andreea

**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 objective of this assignment was to allow us to become familiar with Common Design Patterns by designing and implementing an application for tracking the activity of a supermarket. The application should have two types of users (admin and employee) which must provide a username and a password to use the application.

# Functional Requirements

The application presents the following functional requirements:

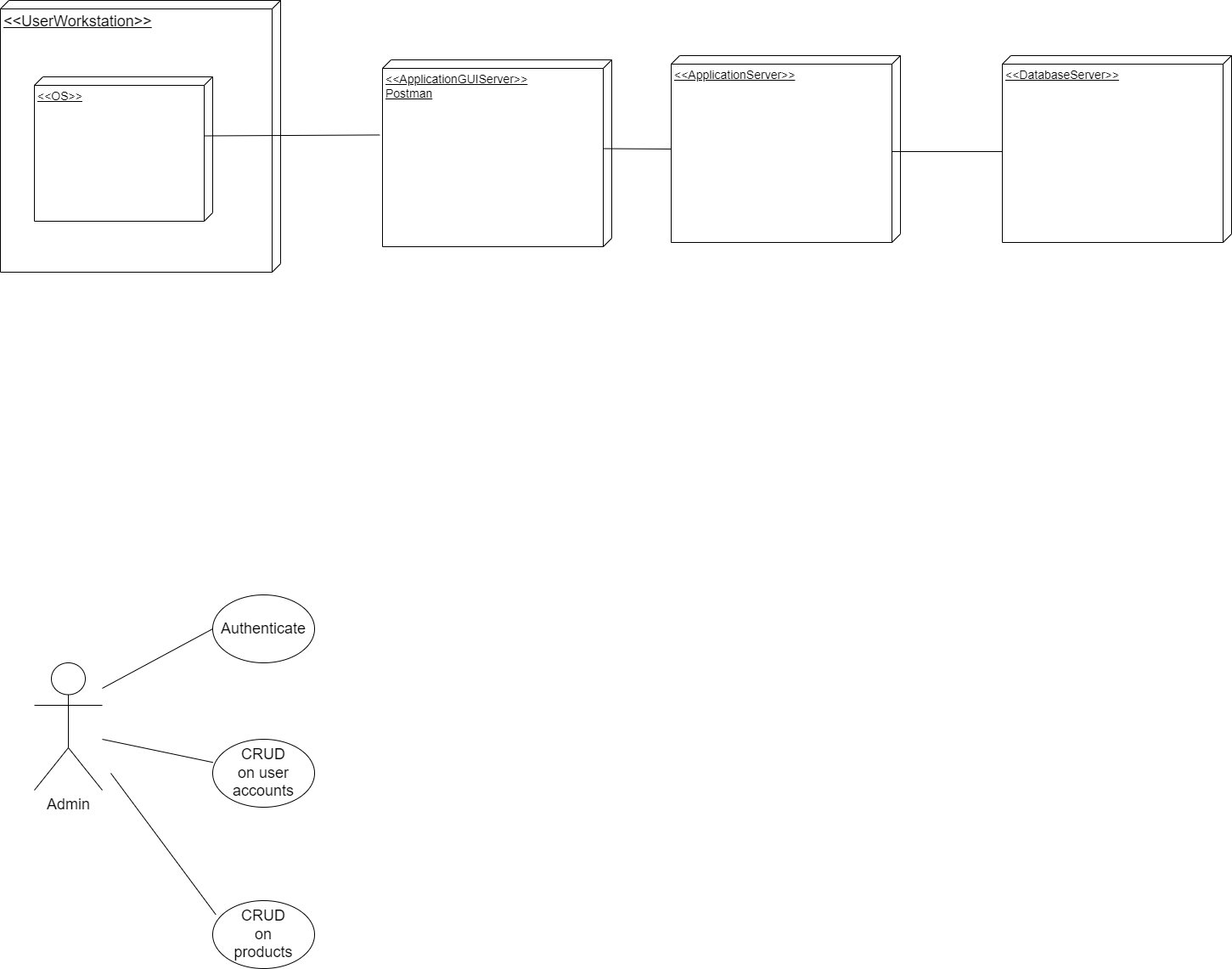
* Admin users can perform CRUD operations on user accounts
* Admin users can perform CRUD operations on products
* Employee users can perform CRUD operations on carts
* Employee users can perform CRUD operations on orders
* Employee users can issue an invoice for an order

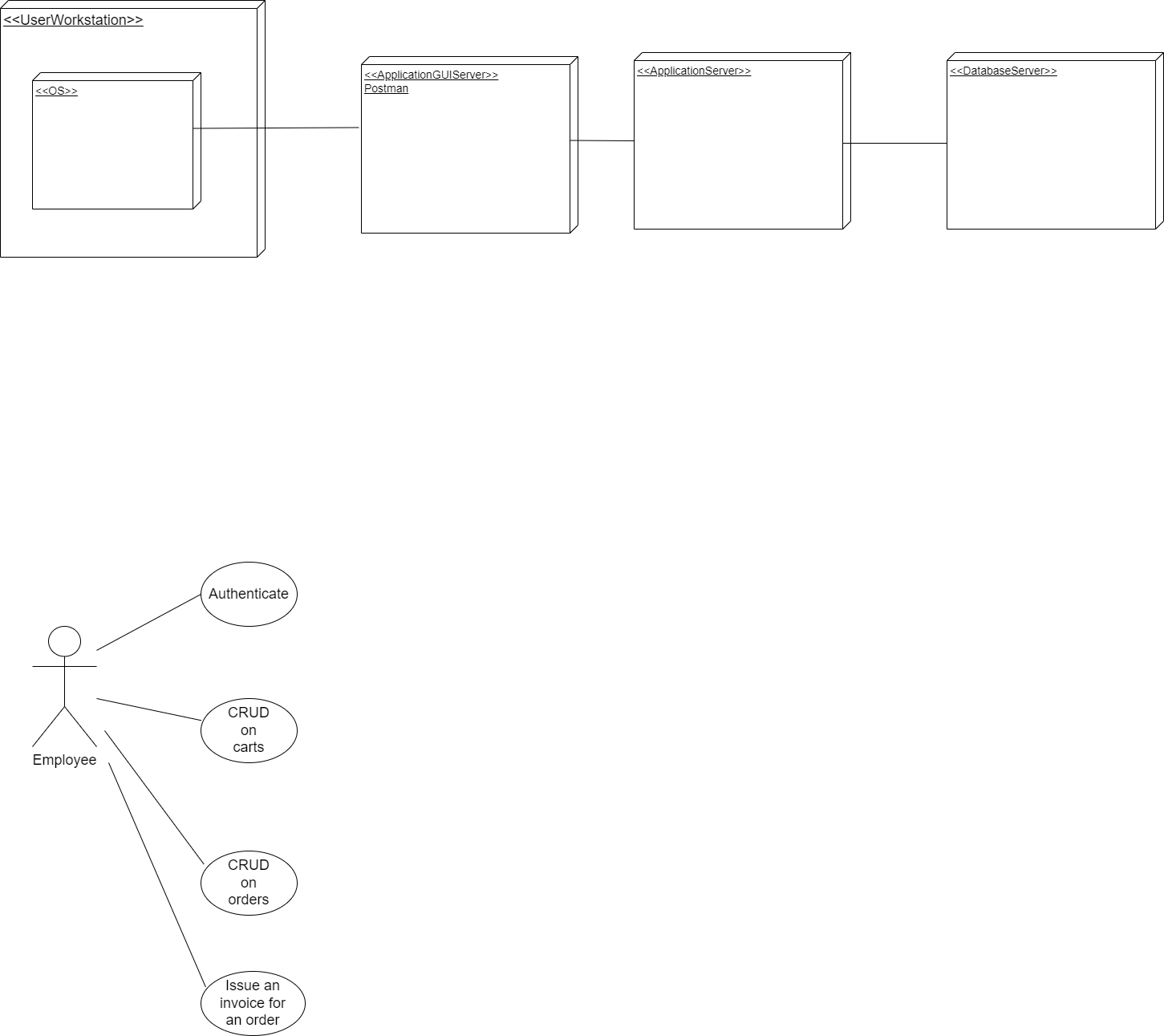
# Non-functional Requirements

The application presents the following non-functional requirements:

* The system requires authentication
* The system is easy to work with

2. Use-Case Model





**Use-case:** Create a new order

**Level:**

**Primary actor:** Employee

**Main success scenario:** The employee introduces his credentials. If the login is successful, the employee is able to create a new order. If there is no cart created, then the employee has to create a new cart and add products to the cart. The employee creates a new order by specifying the id of the cashier, the id of the client that requested the order, the id of the cart that contains the desired products, the total price and the status of the order.

**Extensions:** The system will notify that the credentials or that no cart exists and the login or creation of a new order were unsuccessful.

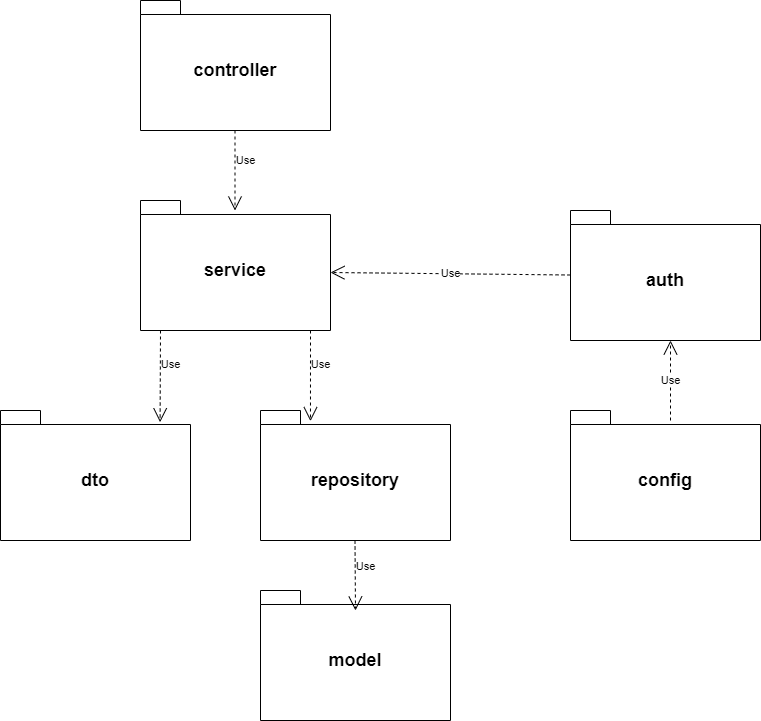
3. System Architectural Design

**3.1 Architectural Pattern Description**

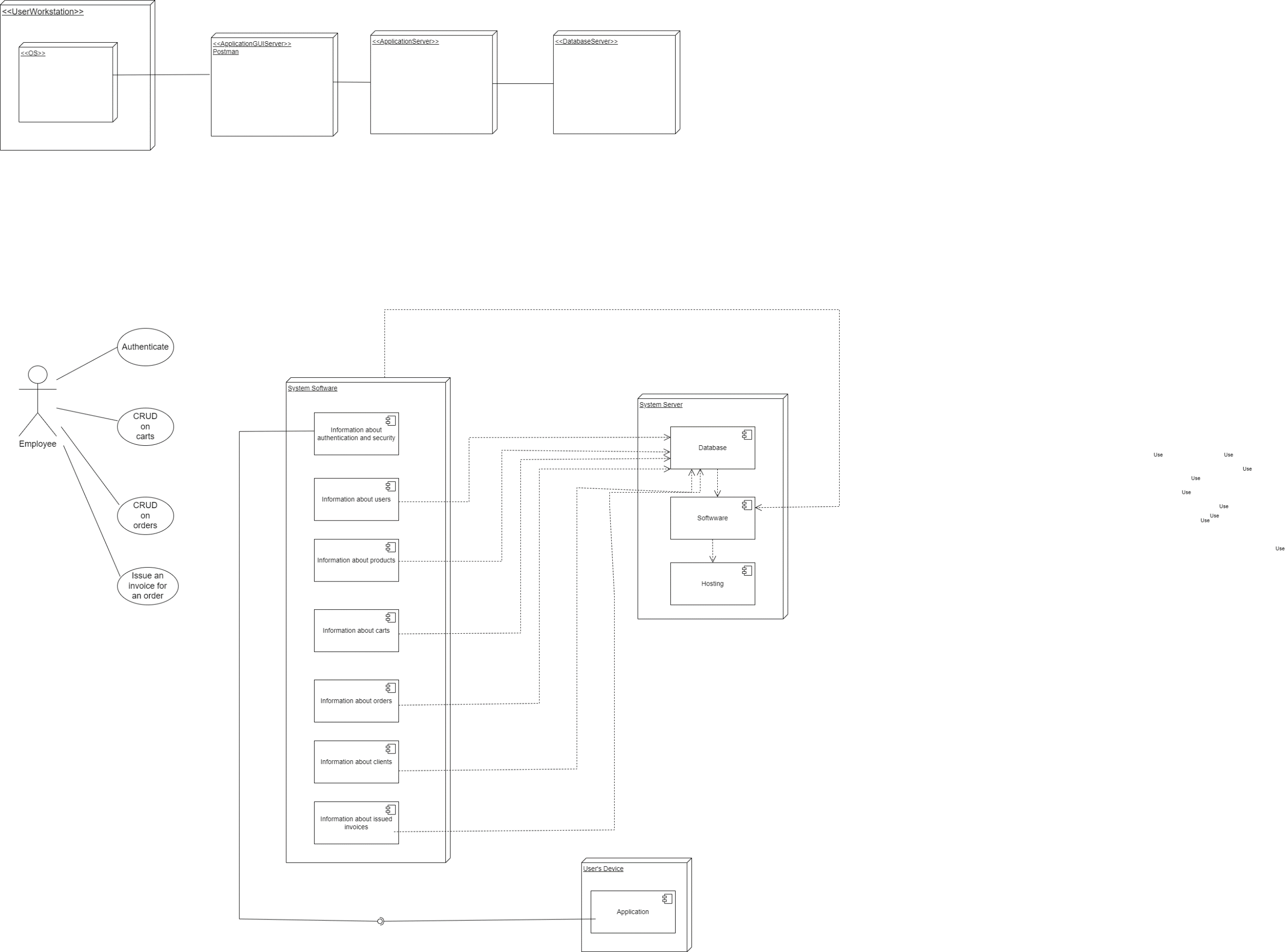
The architectural pattern used is MVC, but without the view part, which means the architectural pattern used is layered architecture. It is also known as an n-tier architecture and describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software. A major characteristic of this framework is that layers are only connected to the layers directly below them. It comes with the advantage that we can make changes in one layer without affecting the other layers.

**3.2 Diagrams**

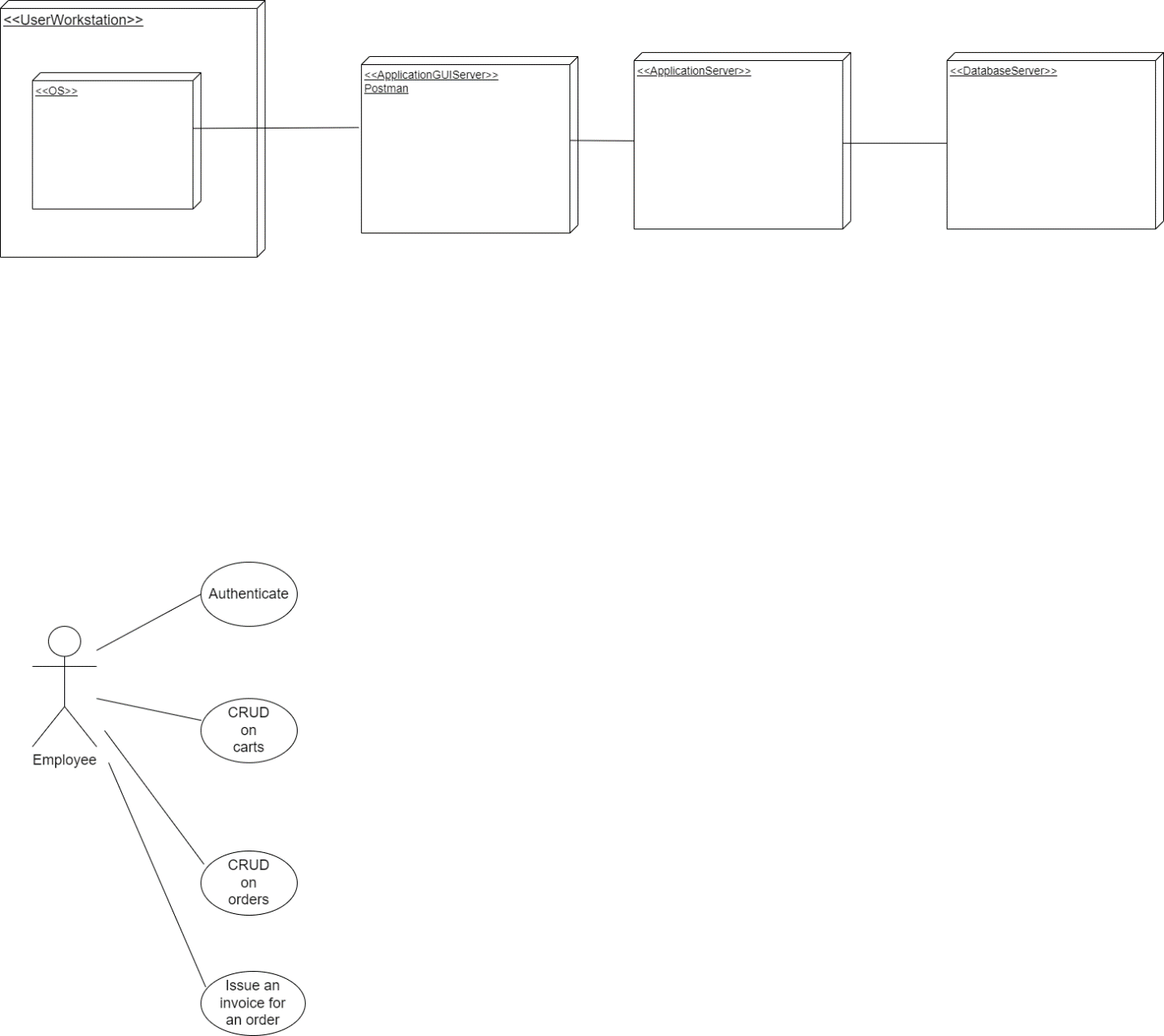
* Package Diagram



* Component Diagram

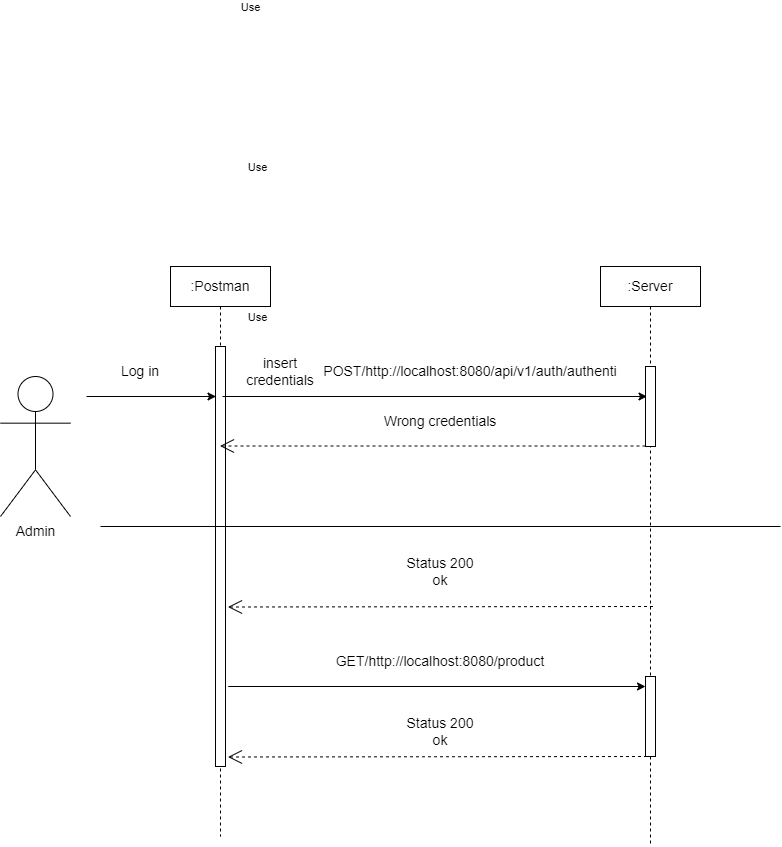


* Deployment Diagram



4. UML Sequence Diagrams

UML Sequence Diagram for viewing all the products



5. Class Design

**5.1 Design Patterns Description**

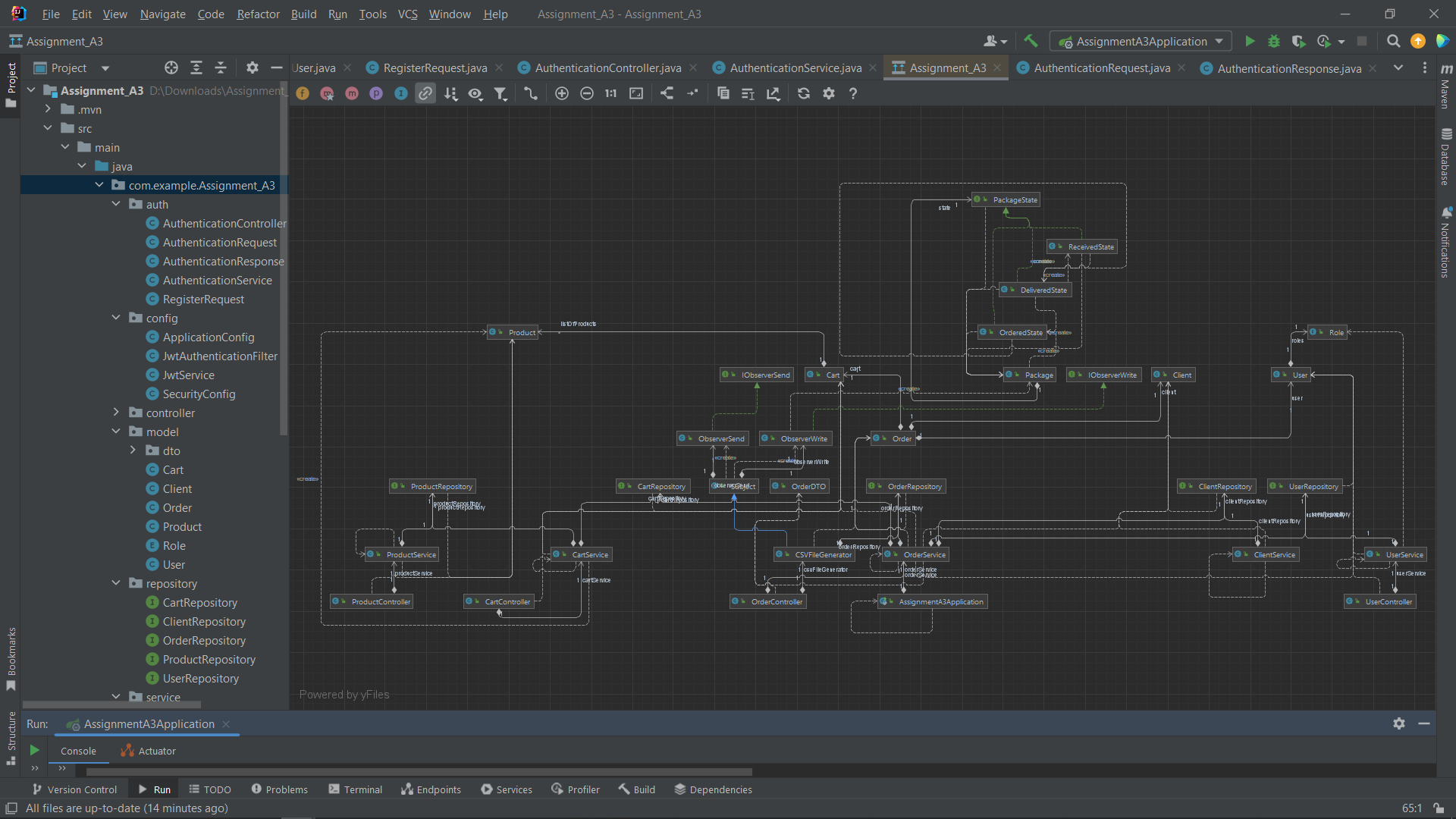
In software engineering, dependency injection is a design pattern in which an object or function receives other objects or functions that it depends on. A form of inversion of control, dependency injection aims to separate the concerns of constructing objects and using them, leading to loosely coupled programs.

I also used Observer Design Pattern to send notifications. Upon an important event in my application, such as issuing an invoice for an order, a notification is sent. The event has two observers: one that writes the invoice to a log event file (LogEventFile.txt) and the other one that sends the invoice via email. The two subscribers are implemented as separate observers and the notification is implemented as subject.

Another design pattern used in this project is the State Design Pattern that allows the object for changing its behavior without changing its class.Also, by implementing it, the code remains cleaner without many if/else statements. This design pattern is used for tracking in which state the order is. There are three states: 1.Package ordered, not delivered to the office yet; 2.Package delivered to the post office, not received yet; 3.The package was delivered by the client. The states are also printed on the generated invoice.

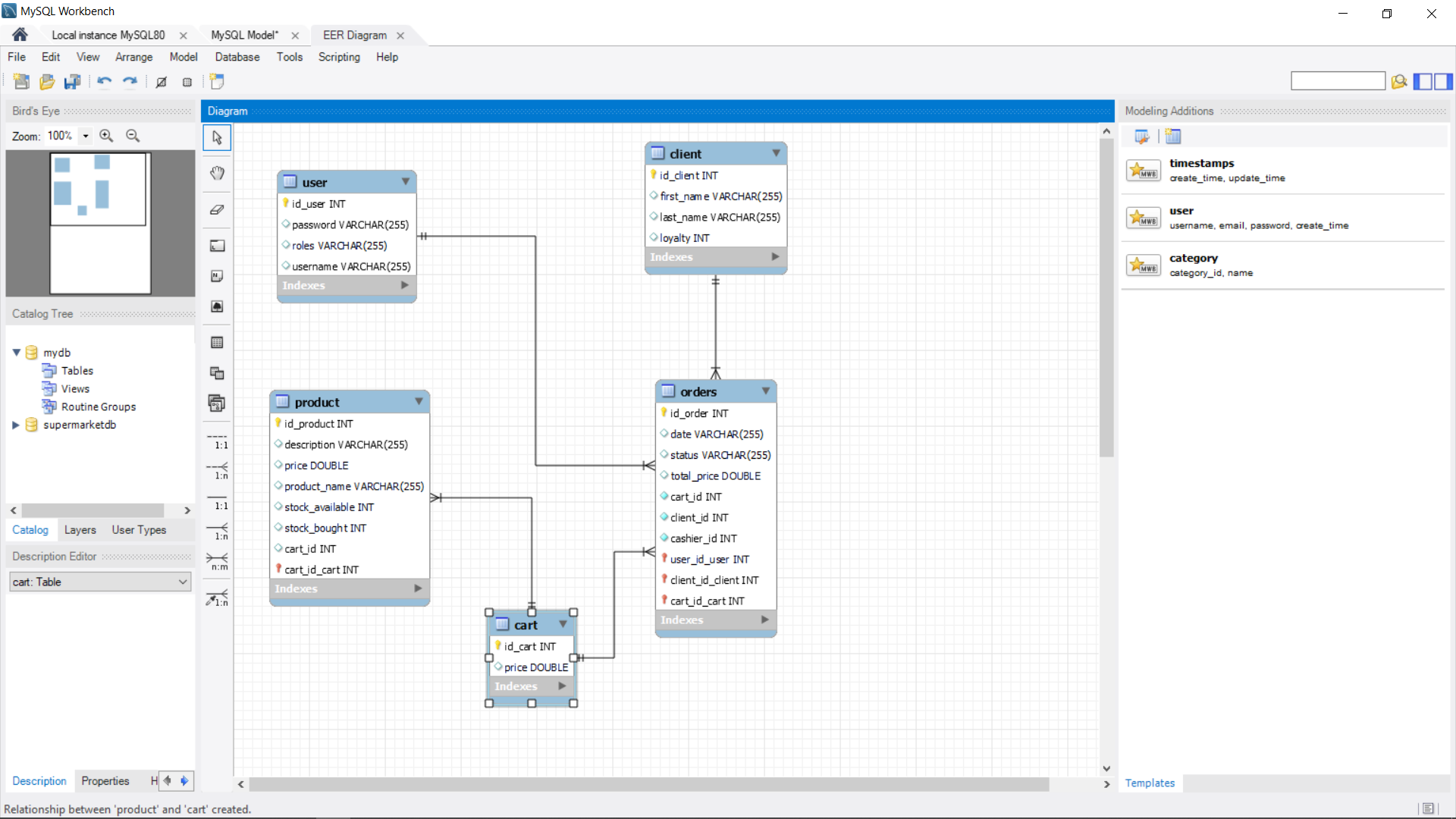
**5.2 UML Class Diagram**

The dependency injection pattern ensures that an object or function which wants to use a given [service](https://en.wikipedia.org/wiki/Service_(systems_architecture)) should not have to know how to construct those services.



6. Data Model

The system implements five data models: User, Client, Product, Cart and Order. The User data model contains an id, username and the password used for authentication and a role (admin or employee). The Client data model contains information about the clients that order products. Each client has a loyalty attribute that is incremented by 1 each time the client makes a new order. The Product data model contains information about the products that are available for sell. The Cart data model contains a list of products that will be bought and the total price of the cart. When a new product is added to the cart, its available and bought stock are updated. The Order data model contains the id of the cashier that initiated the sale, the id of the client that buys the products, the total price and the status of the order.



7. System Testing

For testing the system I used Mock. The application provides two tests:

* Test for creating a new cashier. A new user is created. It is saved in the database using the repository for the User and at the end it verifies if the string returned after creating the user is equal to “A new user with id 4 was created!”
* Test for creating a new cart and adding products to the cart. Two new products are created, a new cart is created containing the two products and at the end the test verifies if the string returned after creating the cart is equal to “A new cart with id 3 was created successfully!”

8. Bibliography

* <https://www.youtube.com/playlist?list=PLqq-6Pq4lTTbx8p2oCgcAQGQyqN8XeA1x>
* <https://www.youtube.com/watch?v=KxqlJblhzfI>
* <https://www.youtube.com/watch?v=9SGDpanrc8U>
* <https://www.baeldung.com/spring-boot-data-sql-and-schema-sql>
* <https://www.baeldung.com/java-state-design-pattern>