UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

PC retailer WebApp documentation

-Software Design lab project-

Table of Contents:

1. Introduction.....2
2. Bibliographic study.....2
3. Architecture.....3
4. Functional requirements.....4
5. Non-functional requirements.....4
6. Diagrams.....5
7. Use case definition.....12
8. Bibliography.....13

1. Introduction

The PC retailer WebApp is a modern, user-friendly platform aimed at facilitating the online purchase of PC components and accessories. Built using Java Spring Boot framework, the application provides ease of access for customers while offering robust management tools for administrators.

2. Bibliographic study

In the development of the PC retailer WebApp project, various technologies and frameworks were employed to ensure robustness, scalability, and maintainability. One of the core technologies utilized is Spring Boot, which serves as the foundation of the application. In addition to Spring Boot, several other technologies complement its functionalities, contributing to the overall architecture and performance of the project.

Spring Boot:

Spring Boot is a powerful Java-based framework designed to simplify the development of production-grade, stand-alone, and web applications.

Key features:

- Auto-configuration: Spring Boot automatically configures the application based on the dependencies present in the classpath, reducing the need for manual configuration.
- Embedded server: Spring Boot comes with embedded servers such as Tomcat, Jetty, and Undertow, enabling developers to package the application as a single, executable JAR file.
- Spring ecosystem: Spring Boot seamlessly integrates with other Spring projects such as Spring Data, Spring Security, and Spring MVC, providing a cohesive development experience.

Other Used Technologies:

- Spring Data JPA
    - Spring Data JPA is a part of the larger Spring Data project, which aims to simplify data access in Java applications. It provides a high-level abstraction over JPA (Java Persistence API), allowing developers to interact with relational databases using repository interfaces.
- MySQL Database
    - MySQL is a widely used open-source relational database management system (RDBMS) known for its reliability, scalability, and performance.
- RESTful API
    - RESTful (Representational State Transfer) architecture is a style of designing networked applications based on the principles of HTTP. RESTful APIs provide a standardized way for clients to interact with server-side resources using HTTP methods such as GET, POST, PUT, and DELETE. In the PC retailer WebApp project,

RESTful APIs are implemented using Spring MVC framework, enabling seamless communication between the client-side and server-side components.

3. Architecture

The PC retailer WebApp project follows a layered architecture pattern, incorporating principles of modularity, separation of concerns, and scalability. The architecture is designed to facilitate the development, testing, deployment, and maintenance of the application while ensuring robustness and flexibility.

- Presentation Layer:
  - Handled by RESTful APIs implemented using Spring MVC framework.
  - controllers handle incoming HTTP requests, process business logic, and delegate data to the appropriate services.
  - Each controller corresponds to a specific set of functionalities, such as user management, product catalog management, and order processing.
- Service Layer:
  - The service layer encapsulates the business logic of the application and orchestrates interactions between different components.
  - Service Classes:
    - Service classes implement business logic and transactional operations, such as user authentication, product management, and order processing.
    - They interact with repository interfaces to perform CRUD (Create, Read, Update, Delete) operations on data entities.
- Repository Layer:
  - The repository layer provides a data access interface for interacting with the underlying data storage mechanism, typically a relational database.
  - It abstracts away the details of database interactions and enables seamless integration with Spring Data JPA.
  - Repository Interfaces:
    - Repository interfaces define methods for accessing and manipulating data entities.
    - They extend Spring Data JPA interfaces such as JpaRepository, which provides out-of-the-box support for common database operations.
- Domain Model Layer:
  - The domain model layer represents the core business entities and their relationships within the application.
  - It defines the structure and behavior of data entities and serves as the foundation for data persistence and manipulation.
  - Entity Classes:
    - Entity classes represent data entities mapped to database tables.

- They contain attributes and methods that define the properties and behaviors of each entity, such as user, product, order, and order-product relationship.

4. Functional requirements

1. User Registration and Authentication
   - Allow users to create accounts and securely authenticate to access personalized features and make purchases.
2. Product Catalog Management
   - Enable administrators to add, update, and remove products from the catalog, ensuring an up-to-date inventory for users to browse and purchase.
3. Order Placement and Tracking
   - Allow users to add products to their cart, place orders securely, and track the status of their orders to ensure a smooth purchasing experience.
4. User Profile Management
   - Provide users with the ability to update their personal information, such as shipping address and contact details, for accurate order processing and delivery.
5. Administrative Dashboard
   - Offer administrators a centralized dashboard to monitor sales, manage orders, and perform administrative tasks efficiently, facilitating effective management of the online store.

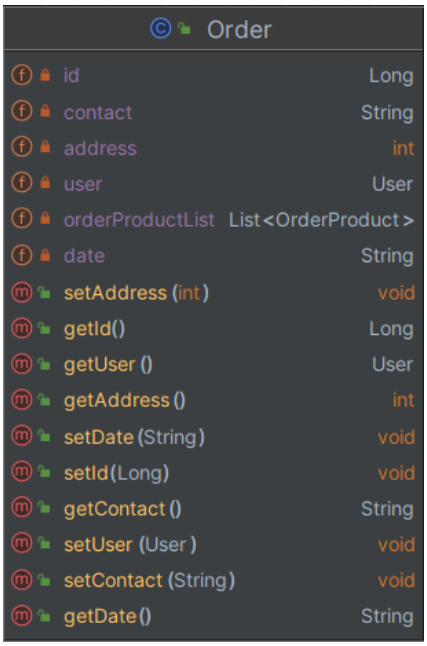
5. Non-functional requirements

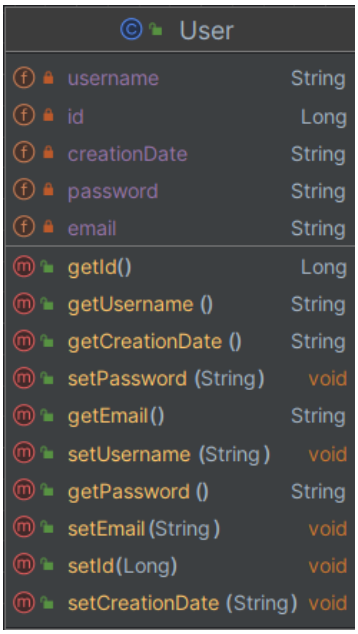
1. Security
   - Implement secure authentication mechanisms to protect user data and prevent unauthorized access.
2. Performance
   - Ensure fast response times and low latency to provide a seamless shopping experience for users, even during peak traffic periods.
3. Scalability
   - Design the system to scale horizontally and vertically to accommodate increasing user traffic and data volume over time.
4. Reliability
   - Ensure high availability and uptime of the application to minimize service disruptions and downtime for users.
5. Usability
   - Design an intuitive and user-friendly interface to enhance user satisfaction and minimize learning curves for new users.

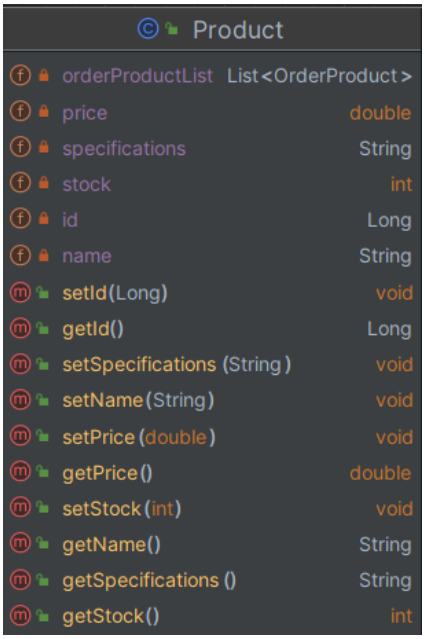
6. Diagrams

6.1. Class diagrams

- Entity classes:

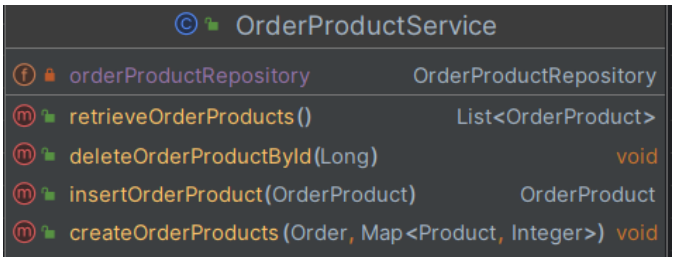


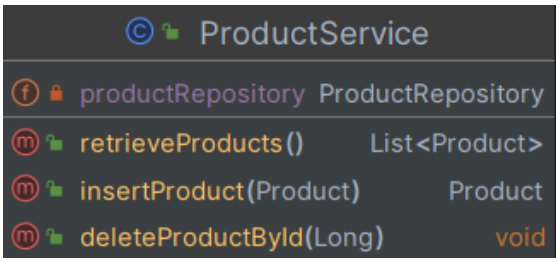


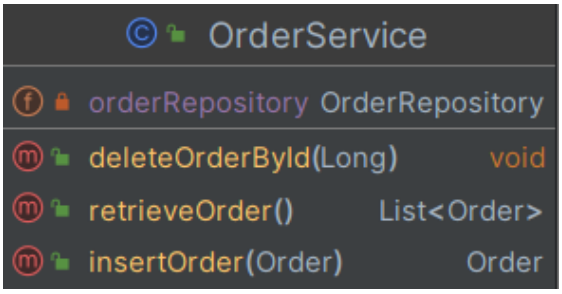


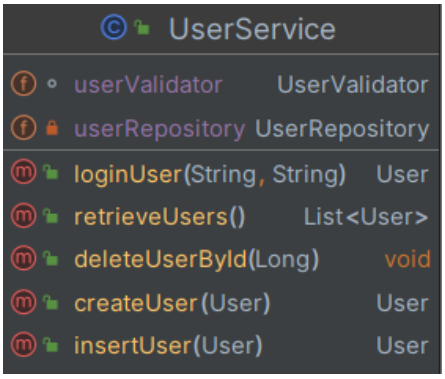
These classes represent the domain model layer (the data based on which the database is modeled)

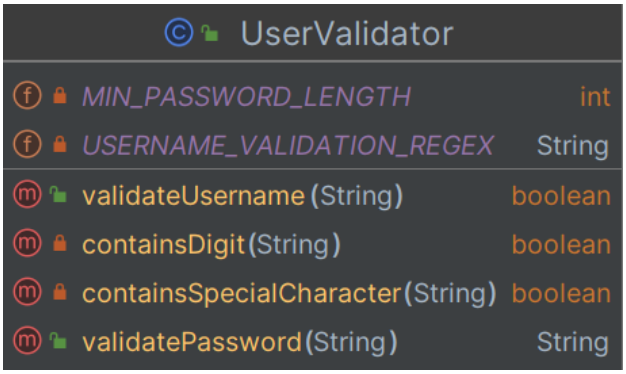
- Service classes:





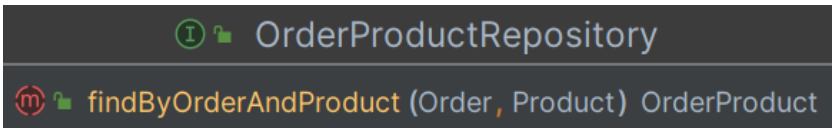


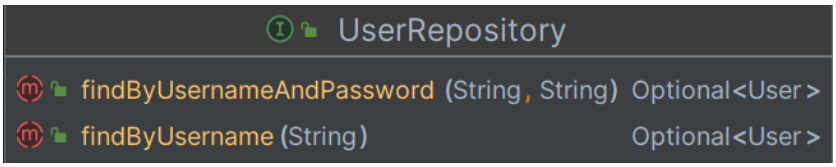




These classes represent the service layer. They handle the logic of the app which will be passed to the controller classes to be mapped.

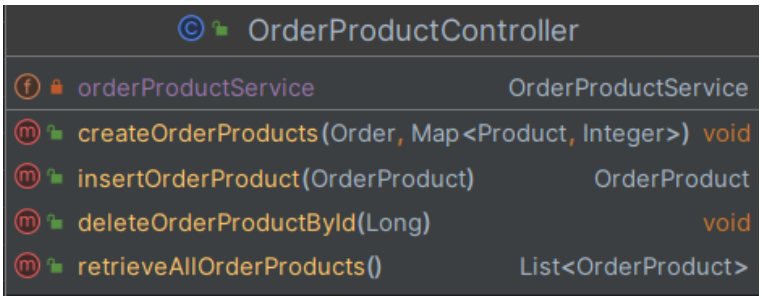
- Repository interfaces:

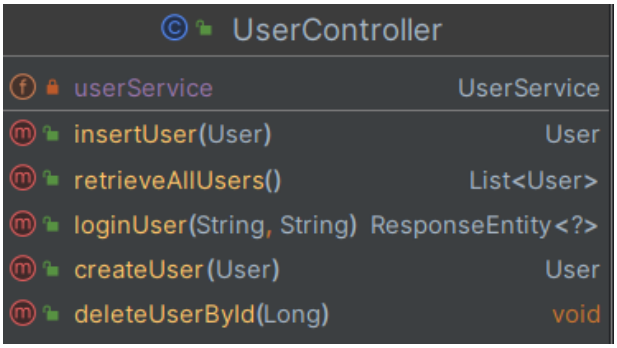




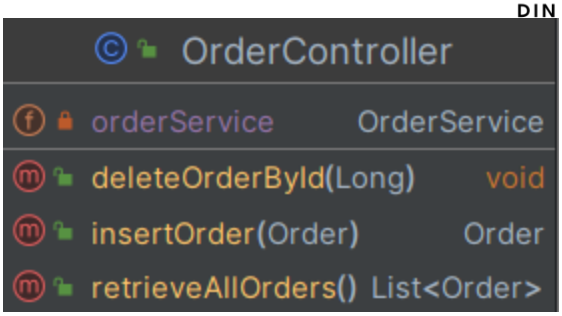
  - These classes are extensions of the JpaRepository interface, they facilitate interactions with the database and are used to pass the data to the service layer.
  - The ProductRepository and OrderRepository classes do not contain any explicit methods.

- Controller classes:

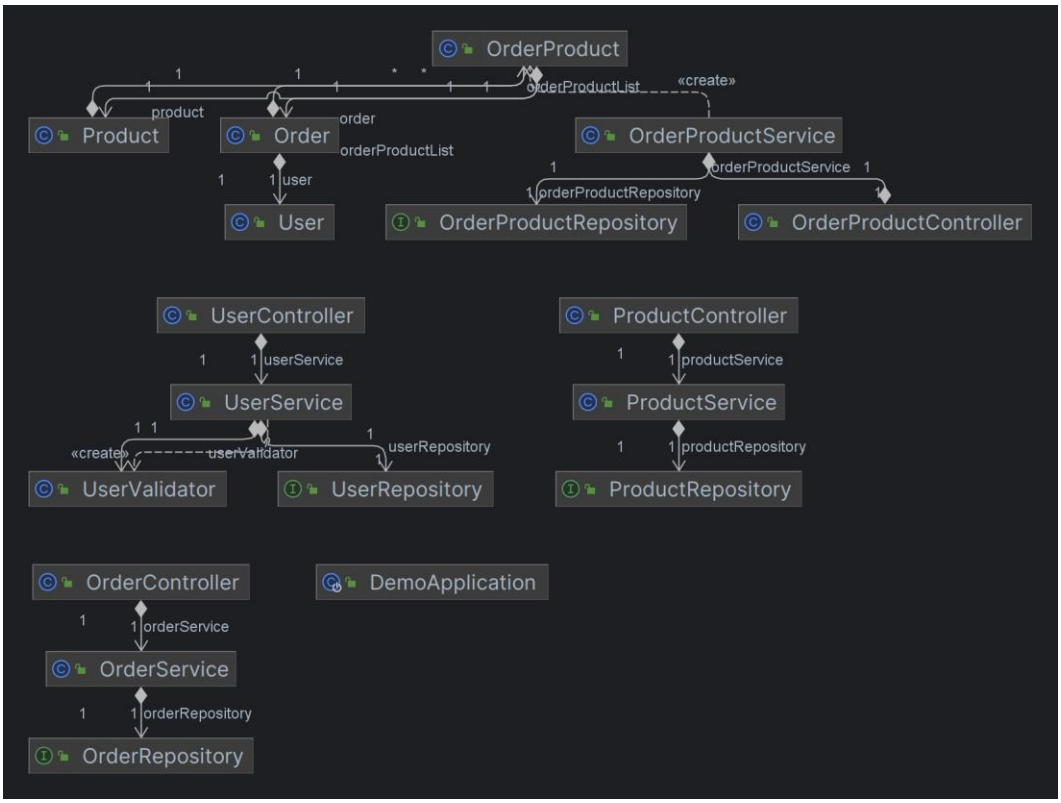




These classes represent the logic part of the presentation layer. They will be used to map the interaction of the users with the logic of the webapp.



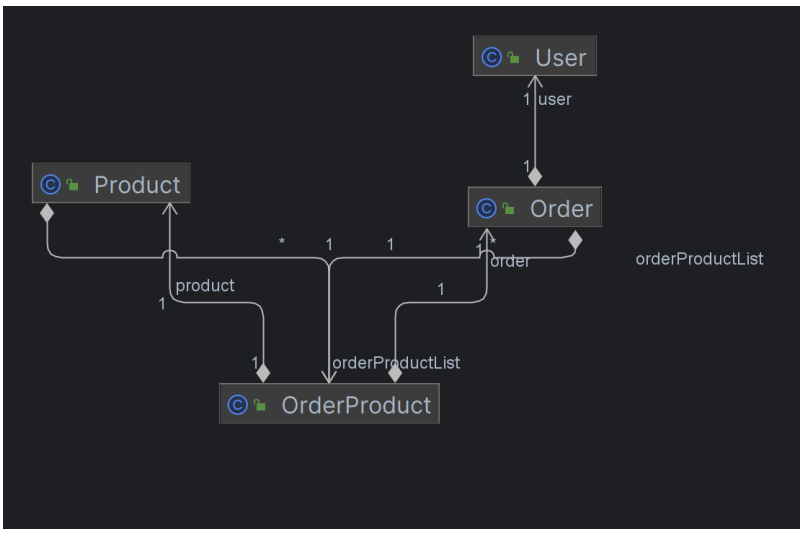
- Class diagram with all dependencies between classes displayed:



- This diagram represents the connectivity of the back end of the application

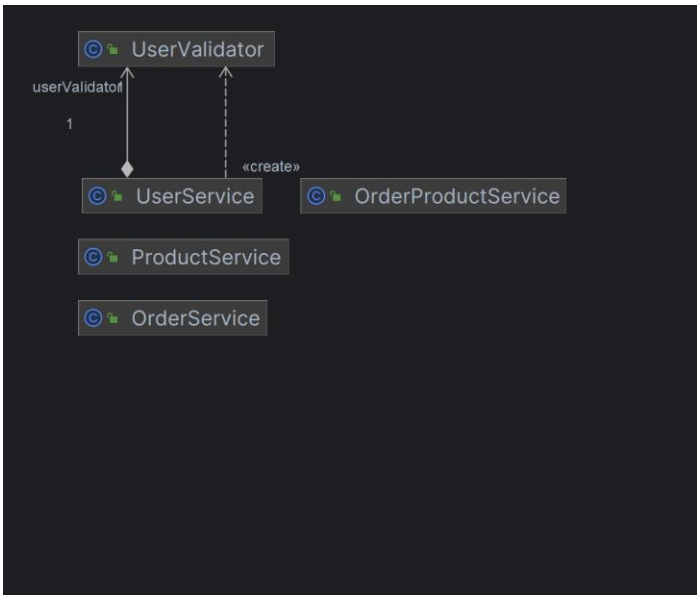
6.2. Package diagrams.

- Entity package:



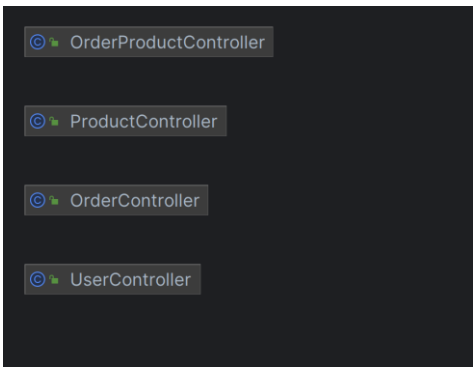
- This diagram displays the dependencies within the entity package.

- Service package:

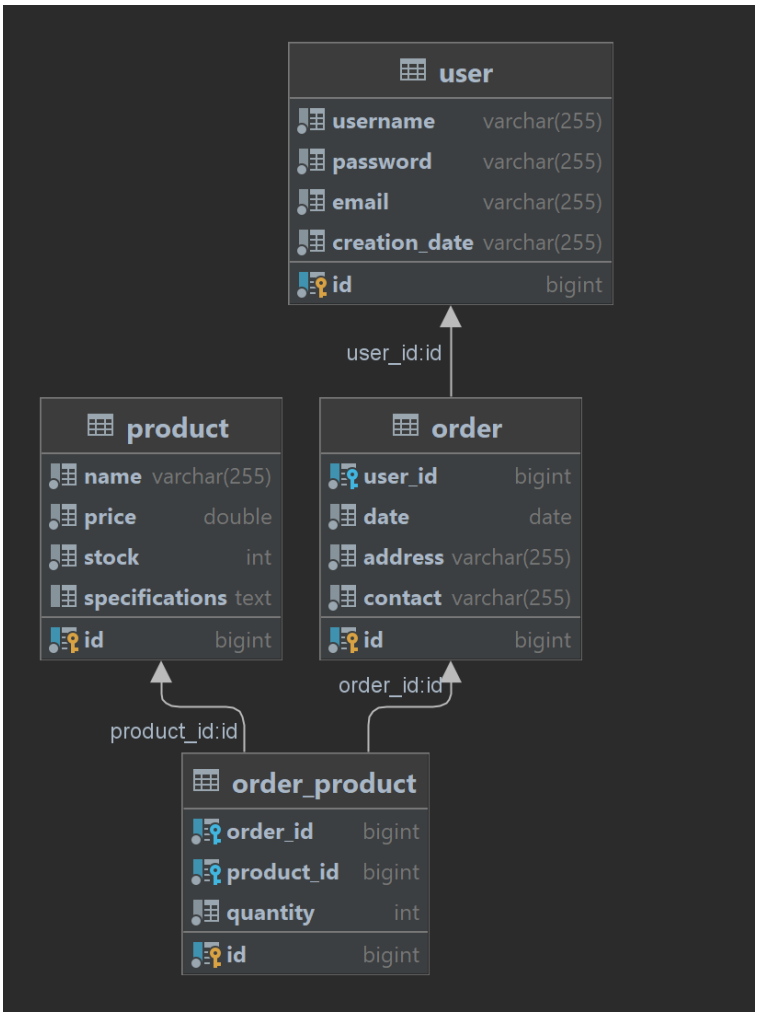


- This diagram shows the dependencies within the service package.

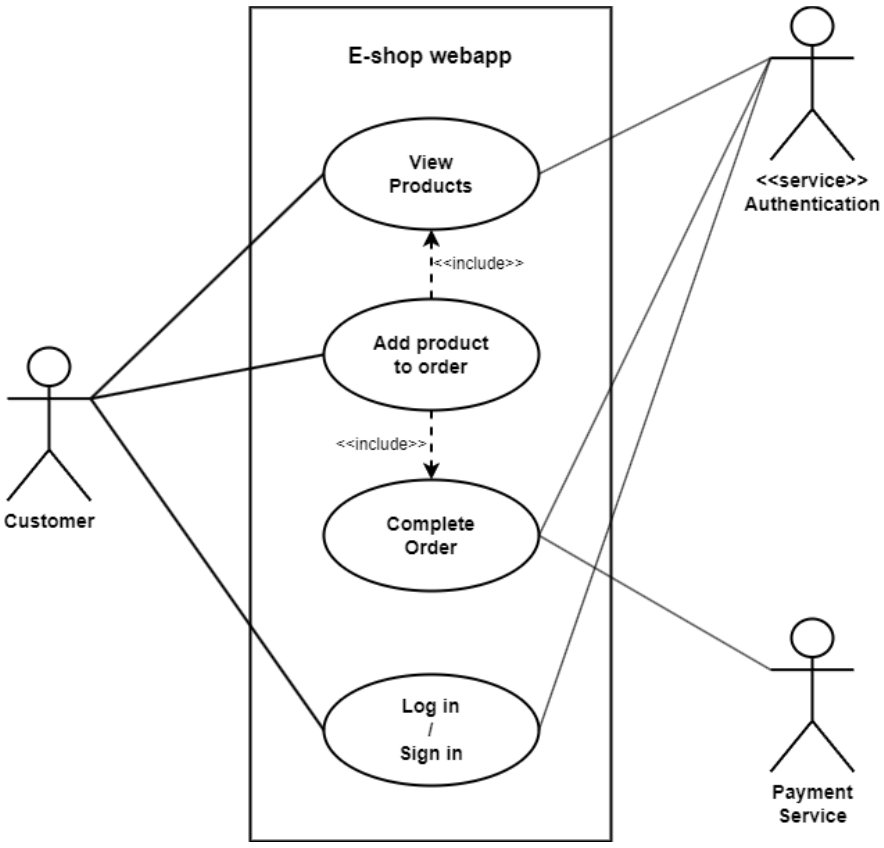
- Controller package:



6.3. Database diagram:

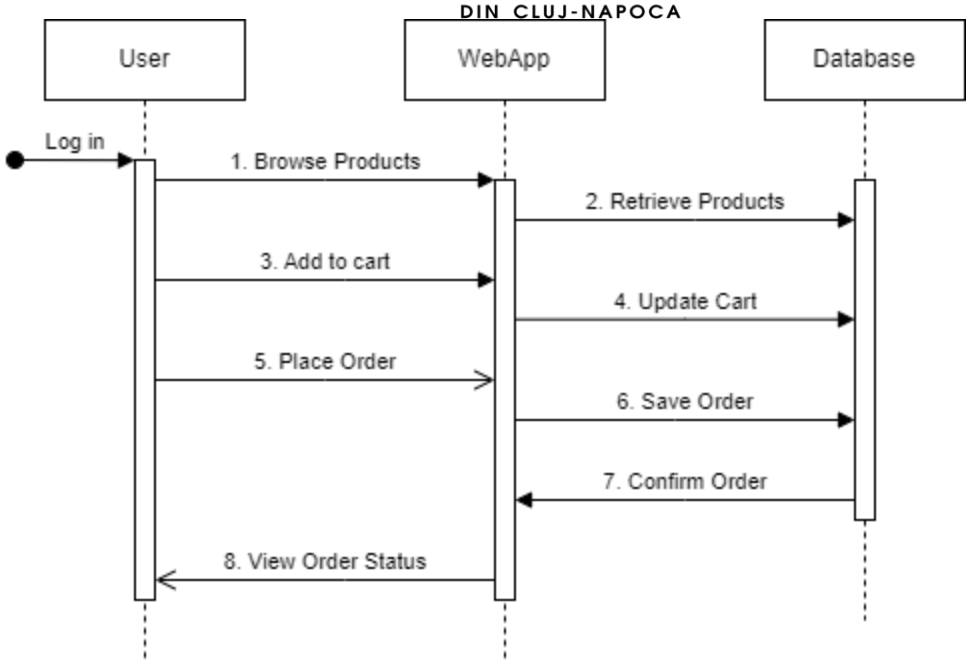


6.4. Usecase diagram:



- This use case diagram provides a comprehensive overview of the various interactions between users and the system, highlighting the core functionalities supported by the PC retailer WebApp project.

6.5. Sequence diagram:



Sequence Description:

- **User Browse Products**: The user browses available products on the WebApp.
- **WebApp Retrieve Products**: The WebApp retrieves product information from the database.
- **User Add to Cart**: The user adds selected products to their shopping cart.
- **WebApp Update Cart**: The WebApp updates the user's shopping cart in the database.
- **User Place Order**: The user proceeds to place an order.
- **WebApp Save Order**: The WebApp saves the order details in the database.
- **WebApp Confirm Order**: The WebApp confirms the order and initiates order processing.
- **User View Order Status**: The user views the status of their order.

7. Use case definition

- Register User
  - Actors: Guest
  - Description: Allows a guest user to register for a new account in the system.
- Authenticate User
  - Actors: User
  - Description: Allows a registered user to authenticate (login) to the system using their credentials.
- Manage Products

  - Actors: Administrator

  - Description: Allows an administrator to manage products in the catalog, including adding, updating, and removing products.
- View Product Catalog
  - Actors: User
  - Description: Allows a user to view the available products in the catalog without the need to authenticate.
- Add Product to Cart
  - Actors: User
  - Description: Allows a user to add products to their shopping cart for purchase.
- Place Order
  - Actors: User
  - Description: Allows a user to place an order for the products in their shopping cart.
- Track Order
  - Actors: User
  - Description: Allows a user to track the status of their orders, including shipping and delivery updates.
- Manage User Profile
  - Actors: User
  - Description: Allows a user to manage their profile information, including updating contact details and shipping address.
- View Sales Dashboard
  - Actors: Administrator
  - Description: Allows an administrator to view sales data and performance metrics via a dashboard interface.
- Manage Orders
  - Actors: Administrator
  - Description: Allows an administrator to manage orders, including processing, updating, and canceling orders.

8. Bibliography

- https://www.baeldung.com
- https://www.geeksforgeeks.org
- https://www.javatpoint.com
- https://stackoverflow.com
- https://www.nuclino.com/articles/functional-requirements
- https://www.lucidchart.com