# Electronics Store

**Our Team**

|  |  |
| --- | --- |
| Name | id |
| يارا عماد درويش يوسف | 20211040 |
| هادي محمد عبد الهادي | 20211028 |
| ولاء اسامه عبدالقادر مصطفي | 20211039 |
| احمد عبد النبي عبد المعطي عبد العزيز | 20210076 |
| شهد احمد محمد مصطفي | 20210464 |
| محمد صلاح الدين عبد السميع عبد الجواد | 20210794 |
| مريم ربيع دياب السيد | 202000877 |

**Grading Criteria**

|  |  |  |
| --- | --- | --- |
|  | Name | ID |
| 1 | SRS (Use Case Diagram, Activity Diagram, Sequence Diagram, Class Diagram, ERD) | **10%** |
| 2 | SDD (Project detailed documentation) | **10%** |
| 3 | Validation | **15%** |
| 4 | OCL | **20%** |
| 5 | ASOP (Aspect Oriented programming) | **20%** |
| 6 | Microservices | **5%** |

Content

1. **INTRODUCTION**
2. **OVERALL DESCRIPTION**

* SOFTWARE REQUIREMENT
* FUNCTIONAL REQUIREMENT
* NON-FUNCTIONAL REQUIREMENTS
* Validation
* OCL
* ASOP (Aspect Oriented programming)
* Microservices

1. **UML DIAGRAMS**

* USE-CASE DIAGRAM
* ACTIVITY DIAGRAM
* SEQUENCE DIAGRAM
* CLASS DIAGRAM
* ER DIAGRAM
* ERD DIAGRAM

Introduction

**Electronics Store Our platform provides a seamless shopping experience, allowing users to browse, select, search, and order a wide range of products including phones, computer equipment, and more. With just a few simple steps, users can easily place orders, add the product to their cart, track shipments, and even cancel the order, ensuring maximum flexibility and convenience. In addition to that, there is an admin dashboard that enables the admin to add, update, or delete any product or user, and also add offers, track everything, and other things.**

Overall Description

1. **Software Requirements**

* Frontend: Angular Framework.
* Backend: Spring Boot Framework and MySQL Database

1. **Functional Requirements**

**User Module**

* Users can log in and register
* User can handle his profile information (name, picture, password, phone, address)
* Users can search for products
* Users can filter /sort products after searching according to price, rating
* Users can view orders
* Users can add products to their cart or favorite list
* User can order product(s) and also cancel it
* Users can add promo codes (discounts)
* User can choose payment method cash/visa
* User can give feedback, rate a product
* User can change languages (Arabic / English)
* User can log out

**Admin Module**

* Admin edit profile system
* Admin login and sign up
* Admin release system update
* Admin view users, products, and orders of the system
* Admin add users, products
* Admin update user account, products, orders
* Admin delete user account, products, orders
* Admin modify the system according to feedback from users
* Admin add discount on product

1. **Non Functional Requirements**

**Performance**

* The system shall respond to the request from the user in less than 3 seconds.
* The system shall be reliable and flexible.
* Application shall handle more than 1user simultaneously.
* Application shall on average operate without failure for as long a time as possible

**Maintainability**

* The system shall be easy to change/ update.

**Serviceability**

* The system shall be easy to maintain, including monitoring the system, repairing problems that arise, and adding and removing users from the system.

**Data integrity**

* The system shall be described by accuracy, completeness, and data consistency, meaning safety of data and security.

**Manageability**

* System shall be efficient and easy to monitor and maintain to keep the system performing, secure, and running smoothly.

**Interface**

* The system must have a good interface so any user can understand it.

**Safety and Security**

* The system shall ensure that only authorized users can gain access.
* The system shall distinguish between authorized and non-authorized users.

**Usability**

* The system shall have a user-friendly interface.
* The system shall be easy to use by different users.

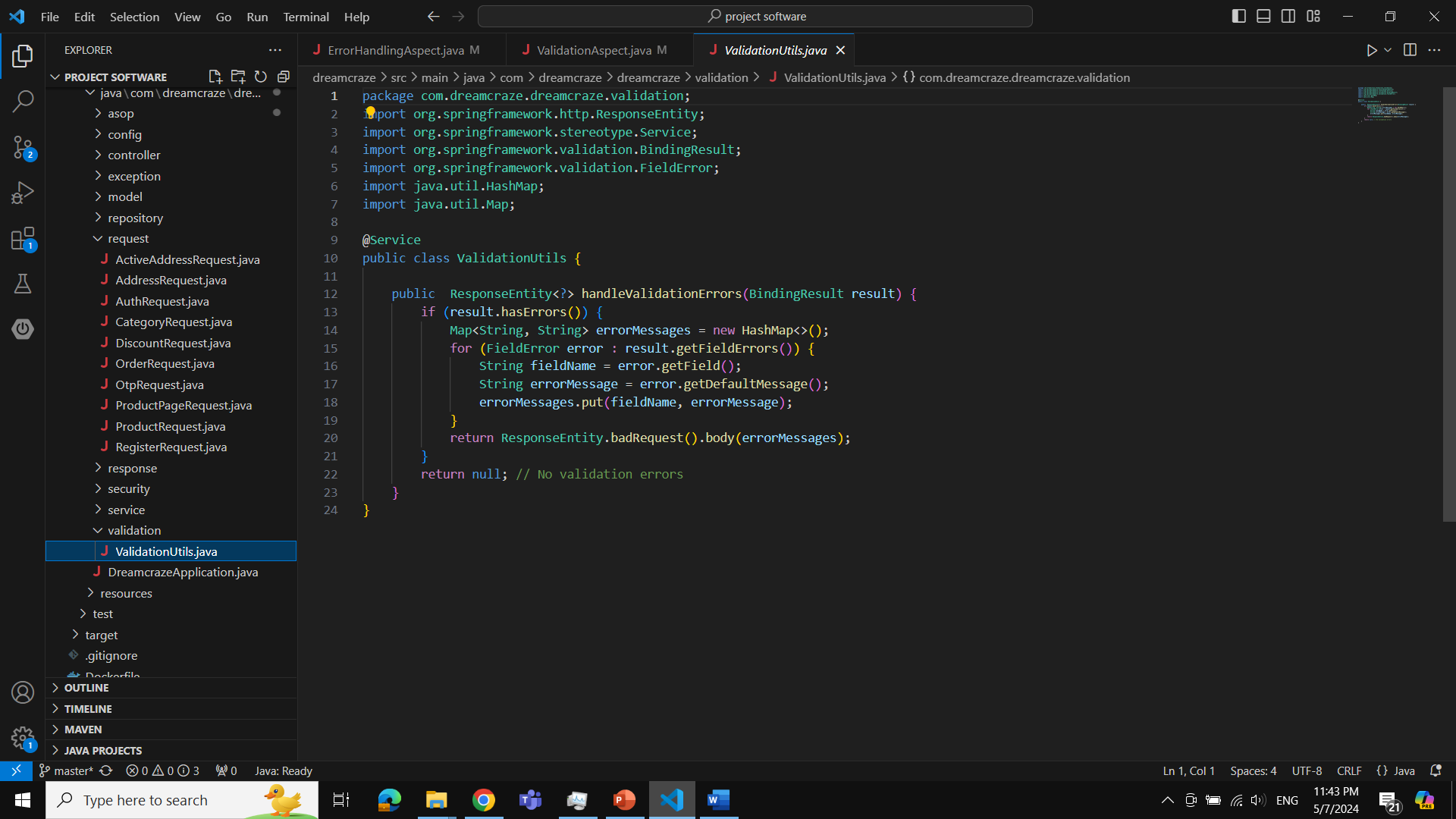
**Availability**

* The system shall be available 24 hours a day.

**Reliability**

* The system shall be reliable and do not have data validation failures.

1. **Validation**



* **Service Annotation:**
  + The **@Service** annotation marks the class as a Spring service component, indicating its role in the application's business logic layer.
* **handleValidationErrors Method:**
  + This method receives a **BindingResult** object, which contains validation errors detected during data binding and validation processes.
  + If validation errors are **present (result.hasErrors()),** the method constructs a map of field names and error messages from the **FieldError** objects within the **BindingResult**.
  + The map **errorMessages** contains field names as keys and corresponding error messages as values.
  + It then returns a **ResponseEntity** with a status of 400 (bad request) and the error messages map as the response body.
  + If no validation errors are found, the method returns null to indicate that no error response is needed.

This **ValidationUtils** service encapsulates the logic for handling validation errors in a reusable and centralized manner. It simplifies the process of converting validation errors into meaningful HTTP responses, enhancing the overall robustness and maintainability of the application.

Top of Form

1. **OCL**

**Constraints**

1. **User**

* Email ends with (@gmail.com / @outlook.com / @yahoo.com)
* Phone starts with 01
* Password size 8 characters

1. **Address**

* Country = Egypt
* City (helwan / ain-shams) discount

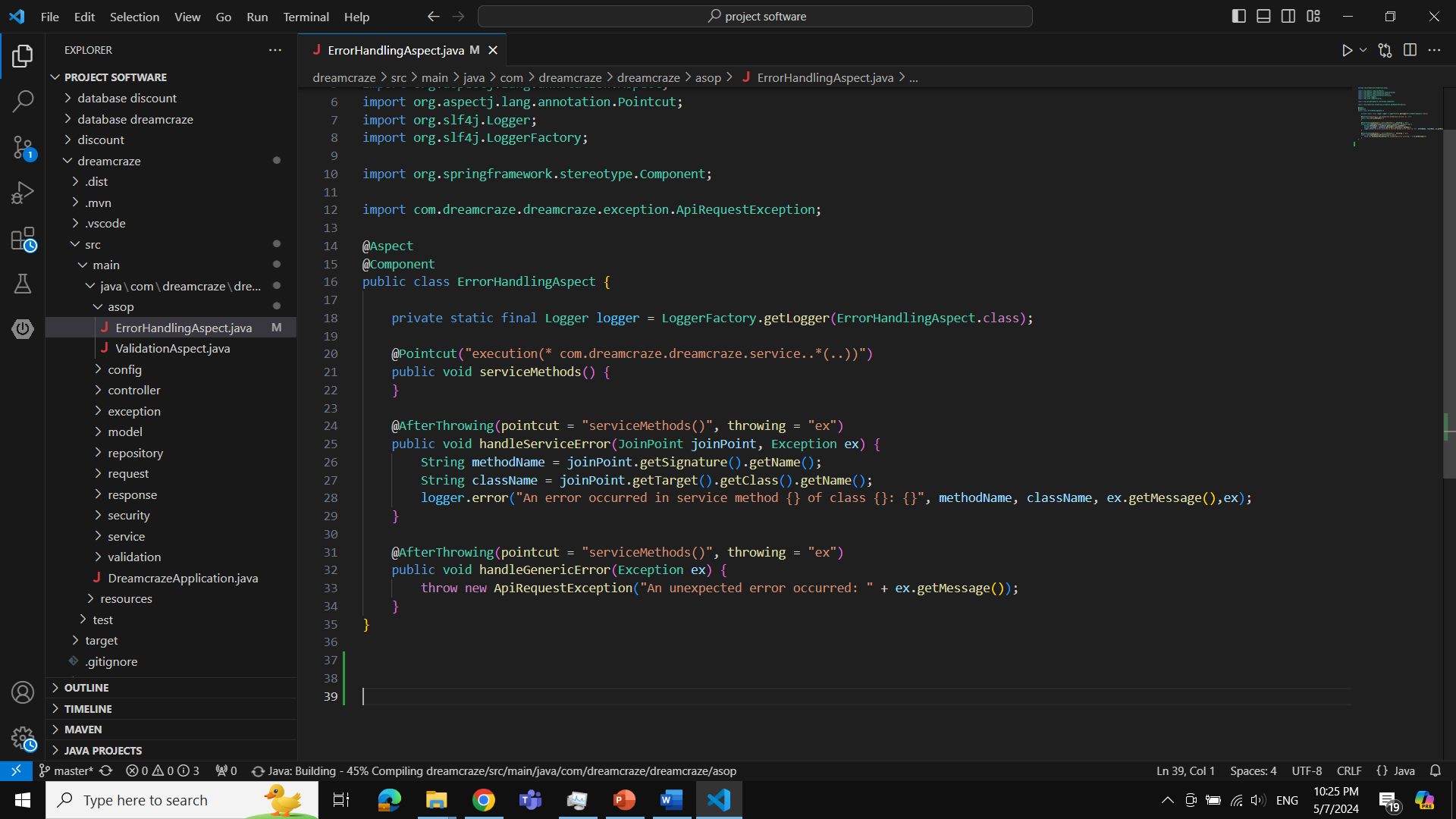
1. **Order**

* order price if higher than 50000 discount
* order quantity more than 5 items discount
* order quantity more than 3 items from the same product discount

1. **Product**

* Rate >= 2 product remove
* Offer price = \*0.3

1. **ASOP (Aspect Oriented programming)**



**error handling**

* **Aspect Annotation:**
* The @Aspect annotation marks the class as an aspect, allowing it to define pointcuts and advice.
* Pointcut Declaration:
* The **serviceMethods()** method defines a pointcut that matches all methods within the **com.dreamcraze.dreamcraze.service** package and its sub-packages.
* **After Throwing Advice:**
* The @AfterThrowing annotation marks methods that act as advice, executed after an exception is thrown by methods matched by the pointcut.
* **handleServiceError()** logs specific service method errors, capturing the method name, class name, and exception message.
* **handleGenericError()** throws a custom **ApiRequestException** for unexpected errors, including the original exception message.
* **Logger Usage:**
* The **LoggerFactory.getLogger()** method initializes a logger instance for logging error messages.
* **JoinPoint:**
* The **JoinPoint** parameter in advice methods provides access to information about the intercepted method, such as its signature and target class.
* **Exception Handling:**
* Both advice methods handle exceptions thrown by service methods, with **handleServiceError()** providing specific error logging and **handleGenericError()** throwing a generic exception with a custom message.

A screenshot of a computer program

Description automatically generated

**validation**

* **Aspect Annotation:**
* The **@Aspect annotation** marks the class as an aspect, allowing it to define pointcuts and advice.
* **Component Annotation:**
* The **@Component annotation** marks the class as a Spring component, enabling auto-detection and instantiation by the Spring container.
* **Autowired** Dependency Injection:
* The **ValidationUtils** bean is injected into the aspect using **@Autowired.**
* Around Advice:
* The **@Around annotation** marks the method as advice, executed around the target method invocation.
* The pointcut expression **@annotation(org.springframework.web.bind.annotation.PostMapping)** && args(.., result) matches any method annotated with **@PostMapping** that accepts a **BindingResult** parameter.
* **handleValidation()** intercepts method calls and executes additional logic before and after the target method invocation.
* **ProceedingJoinPoint:**
* The **ProceedingJoinPoint** parameter provides access to the intercepted method's execution.
* **joinPoint.proceed()** proceeds with the execution of the intercepted method.
* Input Validation Handling:
* Inside the advice, the **ValidationUtils** bean's **handleValidationErrors()** method is called to handle validation errors based on the **BindingResult.**
* If validation errors are found **(response != null),** the advice returns the error response. Otherwise, it proceeds with the execution of the target method.

1. **Microservices**

**Discount Microservice Container:**

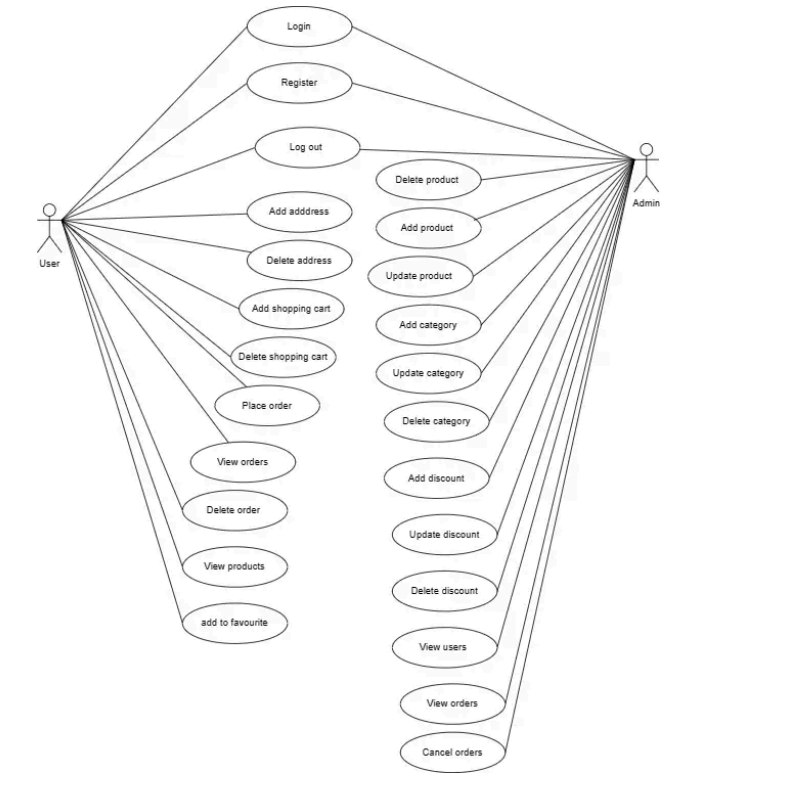
The discount microservice container is a self-contained, scalable component within the architecture of the application. It is responsible for managing and applying discounts to products or services offered by the e-commerce platform.

Key features and functionalities of the discount microservice container include:

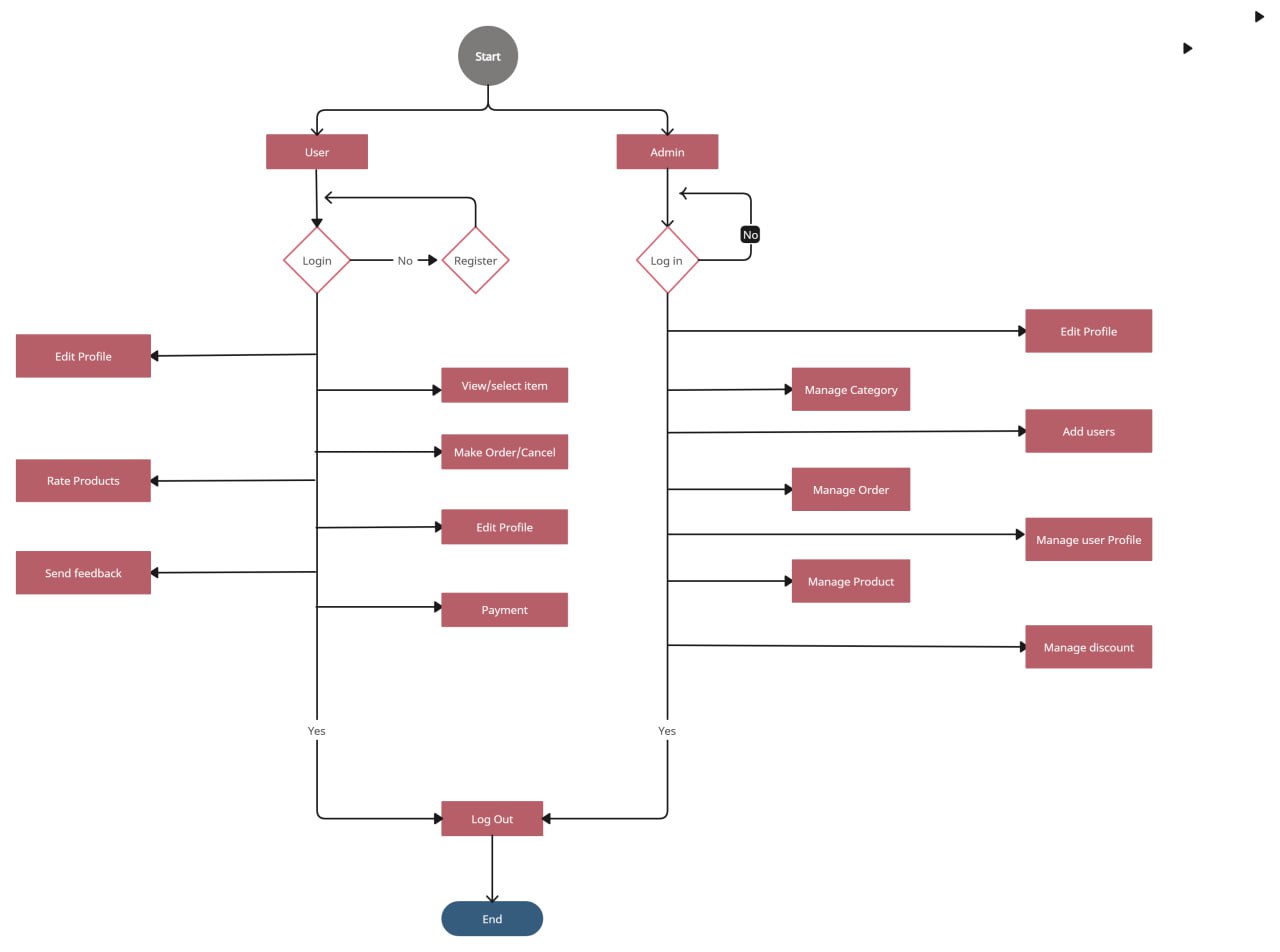
* **Discount Calculation:** The microservice calculates discounts based on various criteria such as promotional campaigns, user segments, product categories, and order quantities.
* **Flexible Discount Rules:** It supports flexible discount rule configurations, allowing administrators to define and modify discount rules dynamically without requiring code changes.
* **Integration with Order Management:** The microservice integrates seamlessly with the order management system to apply discounts to eligible orders during checkout.
* **Real-time Updates:** Discounts are applied in real-time, ensuring accurate pricing information is displayed to users throughout the shopping experience.
* **Performance and Scalability:** Designed for high performance and scalability, the microservice can handle a large volume of discount calculations and adjustments, even during peak traffic periods.
* **Logging and Monitoring:** Comprehensive logging and monitoring capabilities enable administrators to track discount activities, monitor performance metrics, and identify any issues or anomalies.
* **Security:** The microservice ensures data security and integrity by implementing appropriate authentication, authorization, and data encryption mechanisms to protect sensitive discount-related information.

Diagrams

1. **Use-Case Diagram**

****

1. **Activity Diagram**

****

**LOGIN**

**A diagram of a log in

Description automatically generated**

**Sign up**

**Diagram

Description automatically generated**

**Check validation**

**A diagram of a flowchart

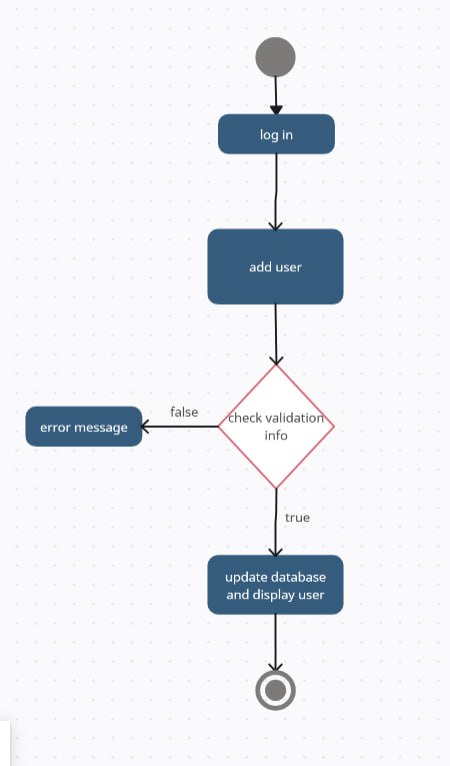
Description automatically generated**

**Add to cart**

**A diagram of a program

Description automatically generated**

**Add user**

****

**Order**

**A diagram of a process flow

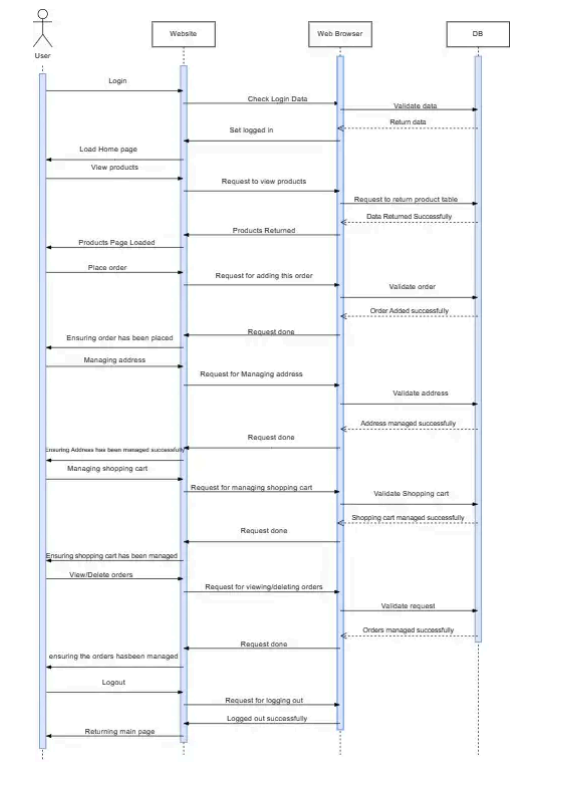
Description automatically generated**

**Add discount**

**Diagram

Description automatically generated**

1. **Sequence Diagram**

****

**Sign up**

**Diagram

Description automatically generated**

**Login**

**Chart, box and whisker chart

Description automatically generated**

**Order**

**Table

Description automatically generated with low confidence**

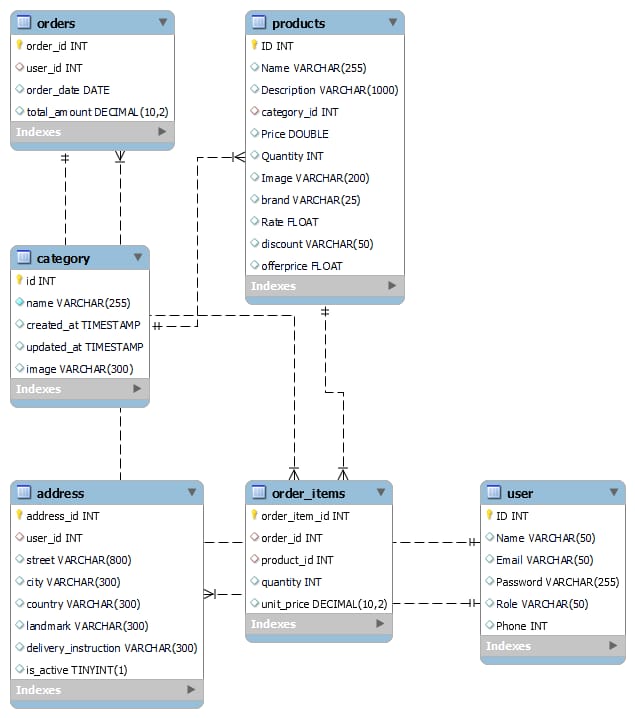
1. **Class Diagram**

**A computer screen shot of a computer

Description automatically generatedA computer screen shot of a diagram

Description automatically generated**

1. **ERD**

****