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

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

2. 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

3. 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.

4. Validation

```
@Service
public class ValidationUtils {

    public  ResponseEntity<?> handleValidationErrors(BindingResult result) {
        if (result.hasErrors()) {
            Map<String, String> errorMessages = new HashMap<>();
            for (FieldError error : result.getFieldErrors()) {
                String fieldName = error.getField();
                String errorMessage = error.getDefaultMessage();
                errorMessages.put(fieldName, errorMessage);
            }
            return ResponseEntity.badRequest().body(errorMessages);
        }
        return null; // No validation errors
    }
}
```

- **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

5. OCL Constraints

1) **User**
   - Email ends with (@gmail.com / @outlook.com / @yahoo.com)
   - Phone starts with 01
   - Password size 8 characters
2) **Address**
   - Country = Egypt
   - City (helwan / ain-shams) discount
3) **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
4) **Product**
   - Rate >= 2 product remove
   - Offer price = *0.3

6. ASOP (Aspect Oriented programming)

```java
@Aspect
@Component
public class ErrorHandlingAspect {

    private static final Logger logger = LoggerFactory.getLogger(ErrorHandlingAspect.class);

    @Pointcut("execution(* com.dreamcraze.dreamcraze.service..*(..))")
    public void serviceMethods() {
    }

    @AfterThrowing(pointcut = "serviceMethods()", throwing = "ex")
    public void handleServiceError(JoinPoint joinPoint, Exception ex) {
        String methodName = joinPoint.getSignature().getName();
        String className = joinPoint.getTarget().getClass().getName();
        logger.error("An error occurred in service method {} of class {}: {}", methodName, className, ex.getMessage(),ex);
    }

    @AfterThrowing(pointcut = "serviceMethods()", throwing = "ex")
    public void handleGenericError(Exception ex) {
        throw new ApiRequestException("An unexpected error occurred: " + ex.getMessage());
    }
}
```

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.

```java
@Aspect
@Component
public class ValidationAspect {

    @Autowired
    private ValidationUtils validationUtils;

    @Around("@annotation(org.springframework.web.bind.annotation.PostMapping) && args(.., result)")
    public ResponseEntity<?> handleValidation(ProceedingJoinPoint joinPoint, BindingResult result) throws Throwable {
        ResponseEntity<?> response = validationUtils.handleValidationErrors(result);
        return (response != null) ? response : (ResponseEntity<?>) joinPoint.proceed();
    }
}
```

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.

7. Microservices

Discount Microservice Container:

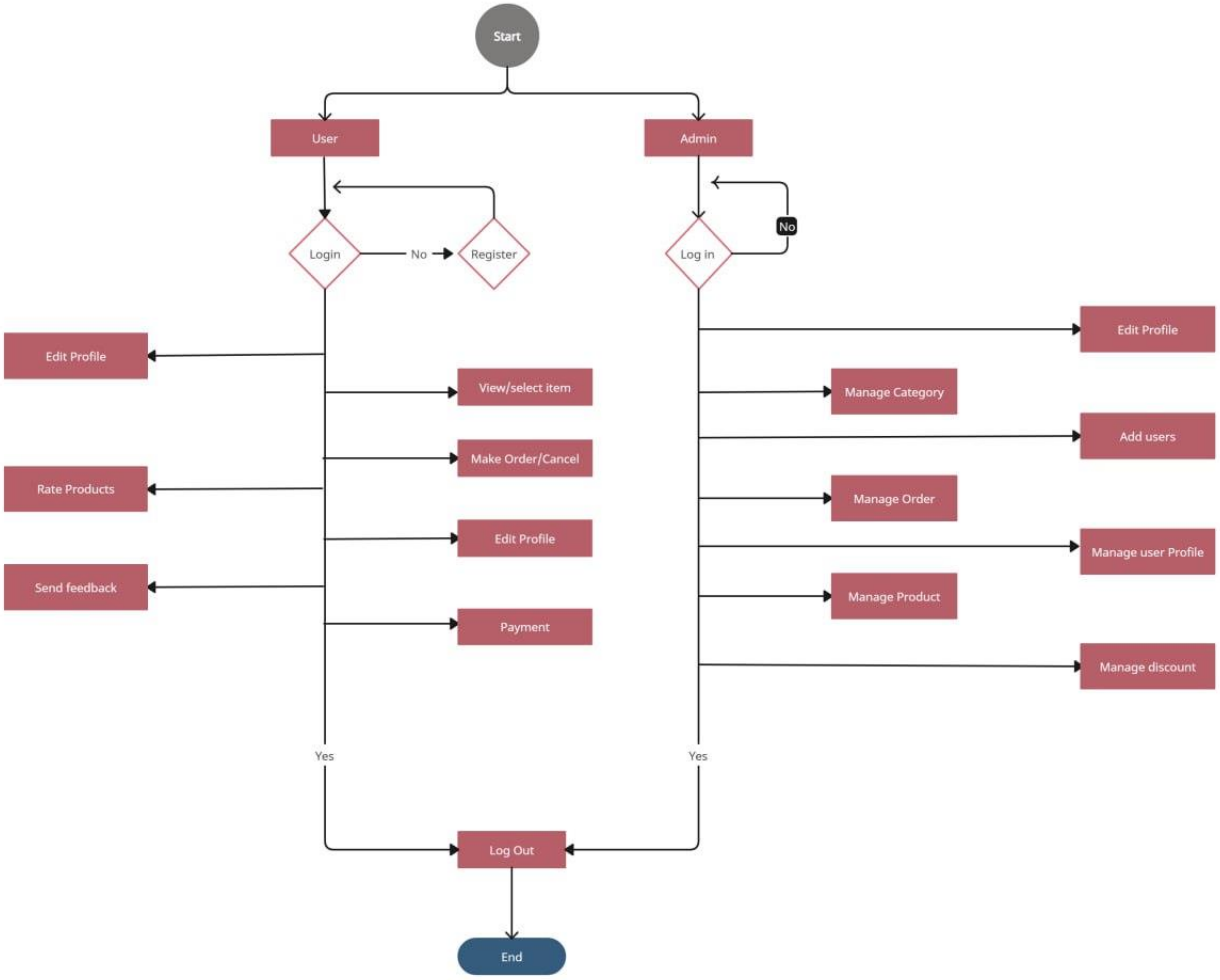
The discount microservice container is a self-contained, scalable component within the architecture of the application. It is responsible for managing and applying discounts to products or services offered by the e-commerce platform.
Key features and functionalities of the discount microservice container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

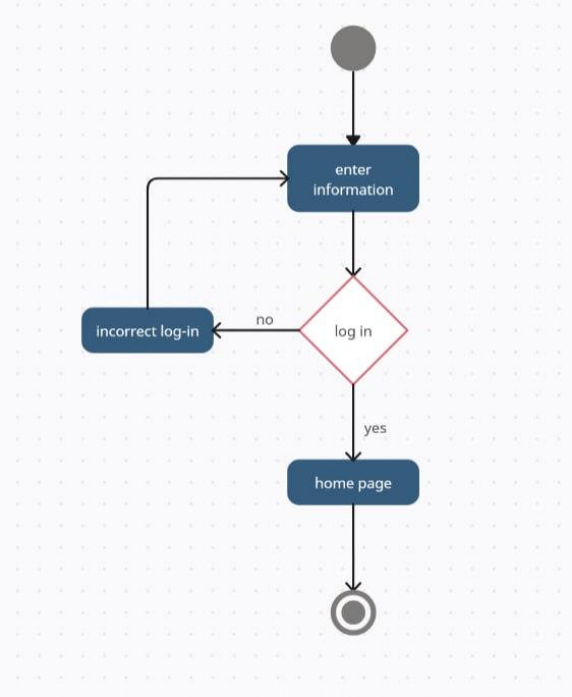
Diagrams

1. Use-Case Diagram

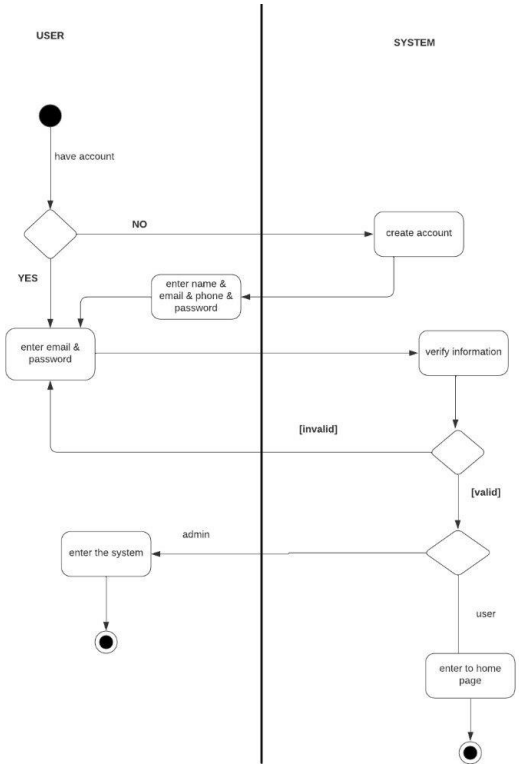
2. Activity Diagram


Start
User
Admin
Login
No
Register
Log in
No
Edit Profile
View/select item
Make Order/Cancel
Rate Products
Edit Profile
Send feedback
Payment
Yes
Edit Profile
Manage Category
Add users
Manage Order
Manage user Profile
Manage Product
Manage discount
Yes
Log Out
End

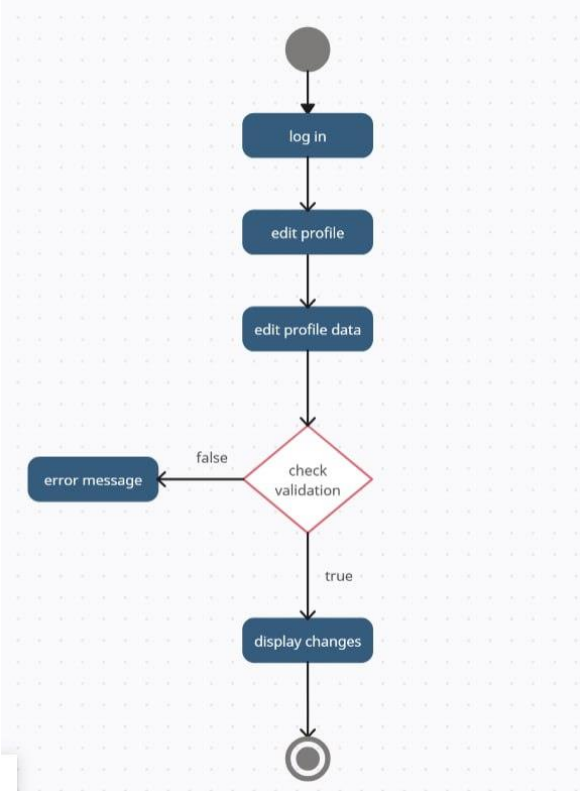
LOGIN



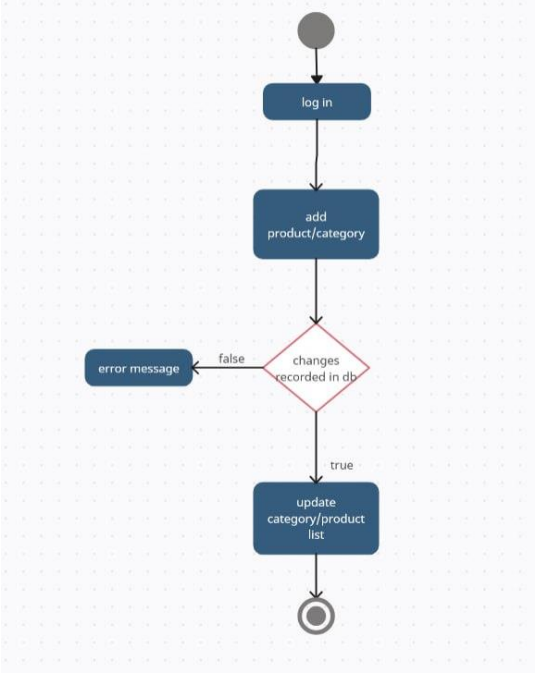
Sign up



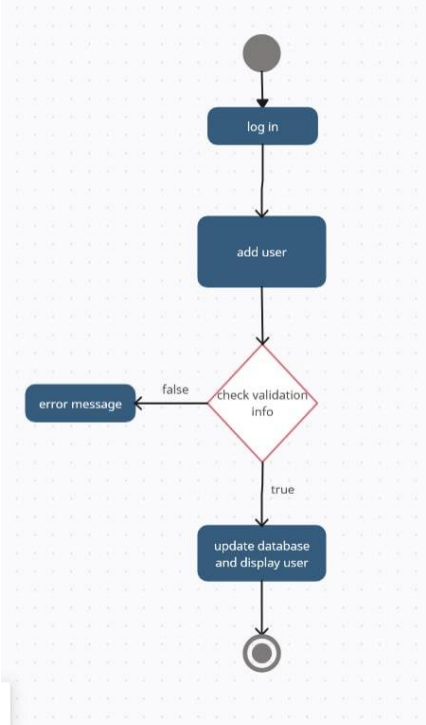
Check validation



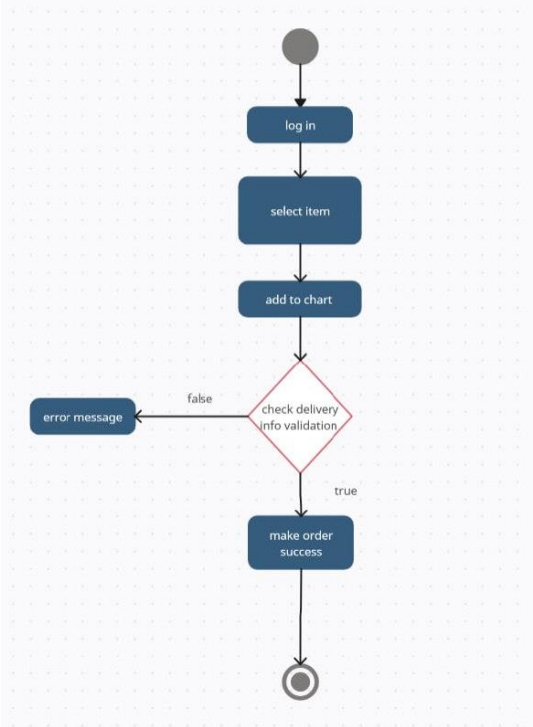
Add to cart



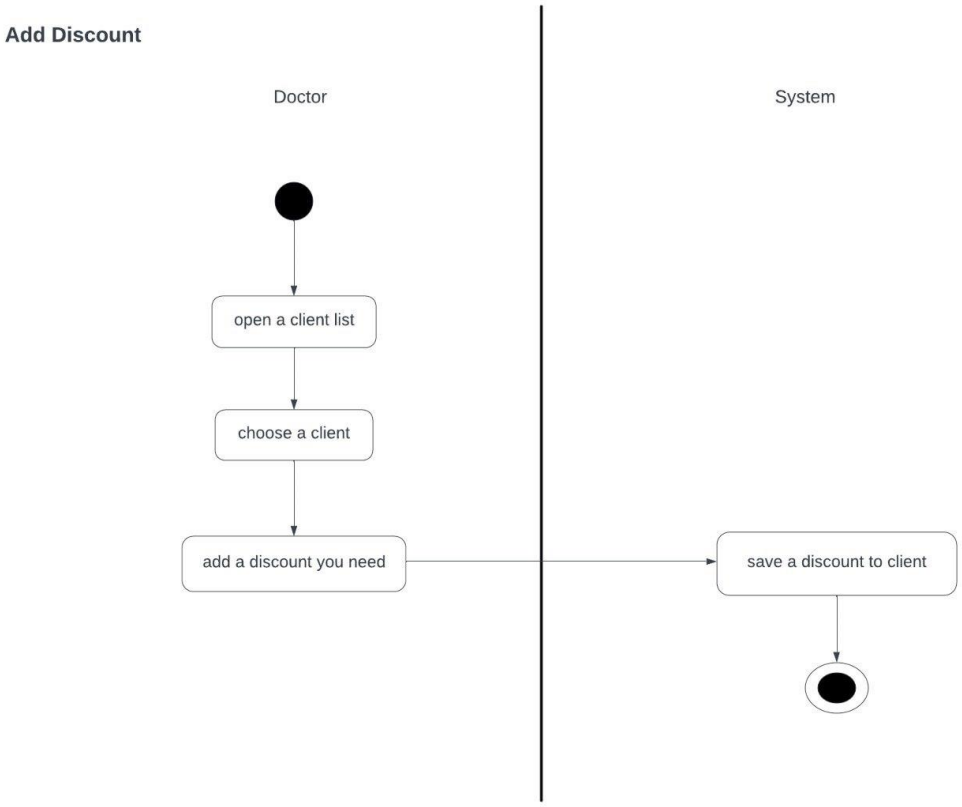
Add user



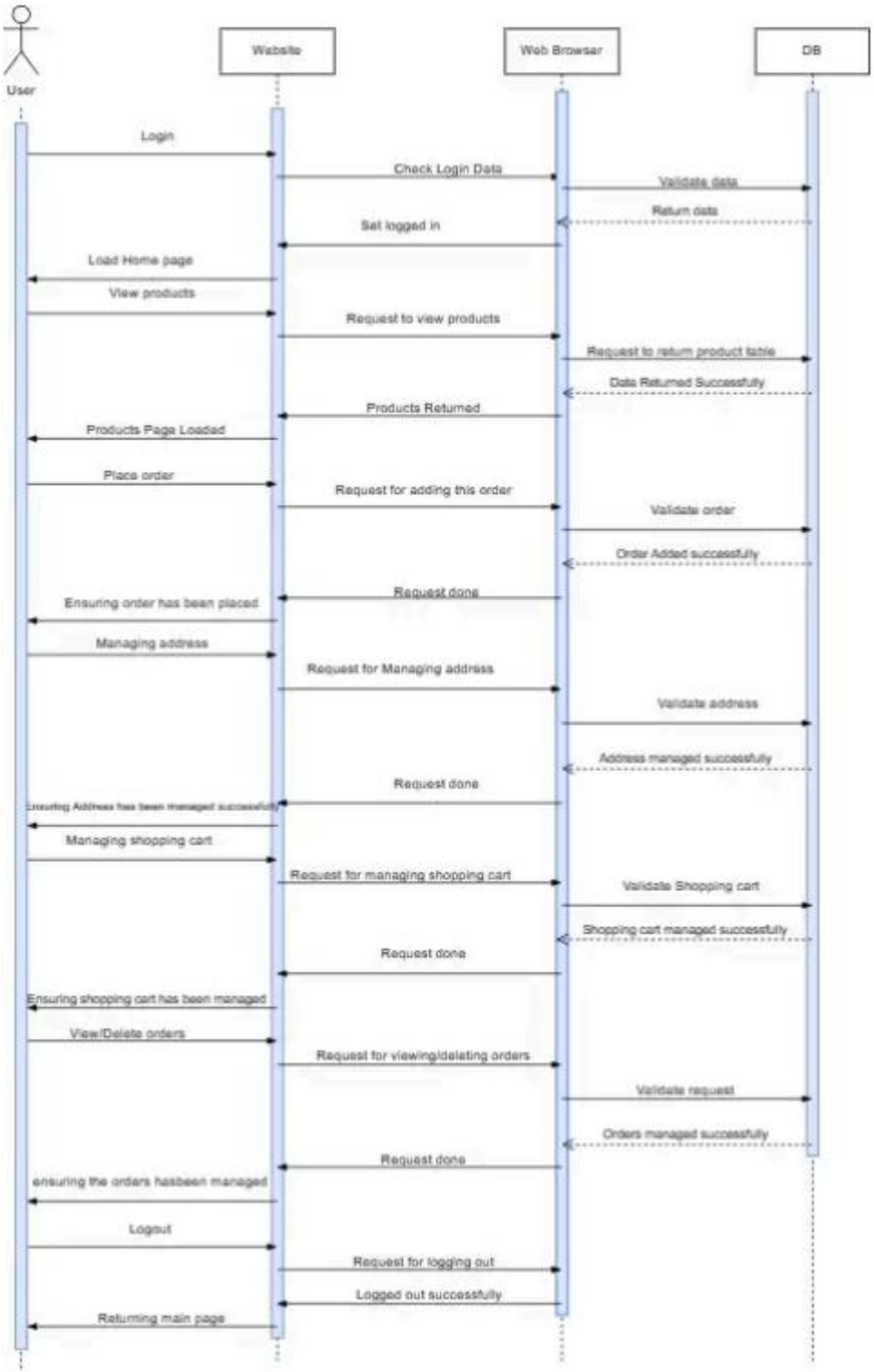
Order



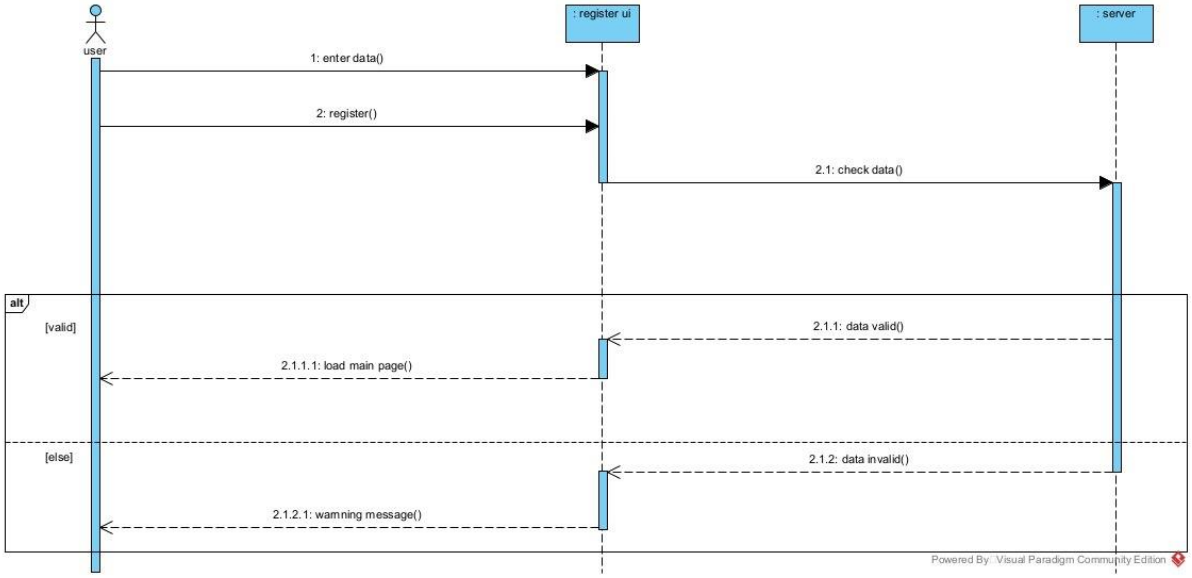
Add discount



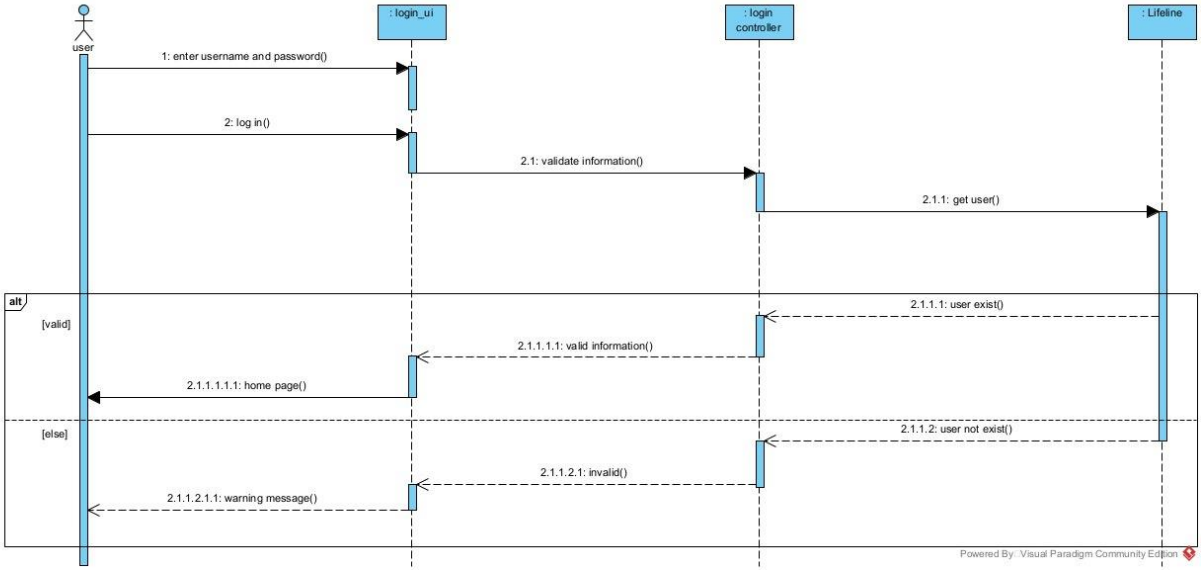
3. Sequence Diagram


User
Website
Web Browser
DB
Login
Check Login Data
Validate data
Return data
Set logged in
Load Home page
View products
Request to view products
Request to return product table
Data Returned Successfully
Products Returned
Products Page Loaded
Place order
Request for adding this order
Validate order
Order Added successfully
Request done
Ensuring order has been placed
Managing address
Request for Managing address
Validate address
Address managed successfully
Request done
Ensuring Address has been managed successfully
Managing shopping cart
Request for managing shopping cart
Validate Shopping cart
Shopping cart managed successfully
Request done
Ensuring shopping cart has been managed
View/Delete orders
Request for viewing/deleting orders
Validate request
Orders managed successfully
Request done
ensuring the orders hasbeen managed
Logout
Request for logging out
Logged out successfully
Returning main page

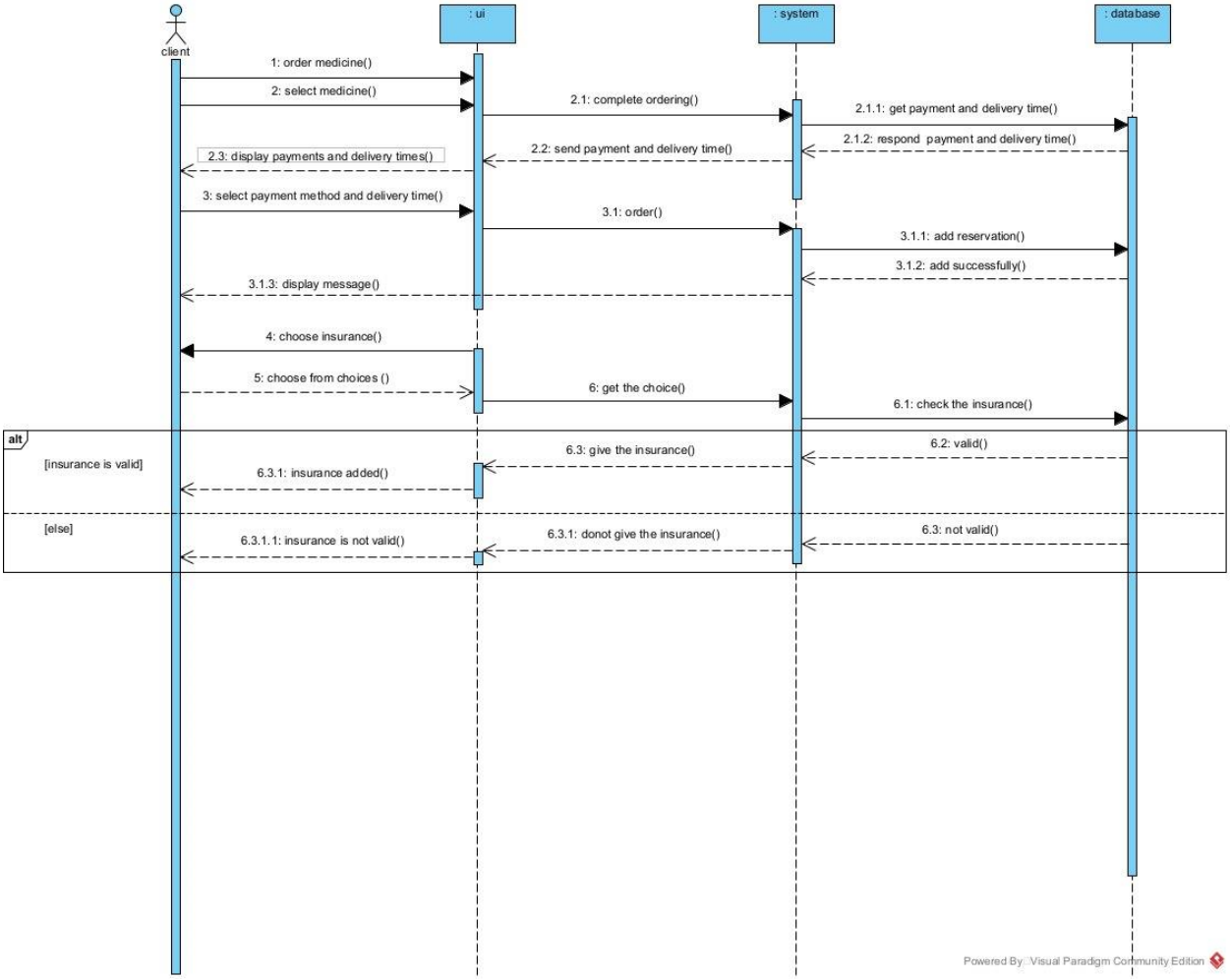
Sign up


user
: register ui
: server
1: enter data()
2: register()
2.1: check data()
alt
[valid]
2.1.1: data valid()
2.1.1.1: load main page()
[else]
2.1.2: data invalid()
2.1.2.1: warning message()
Powered By: Visual Paradigm Community Edition

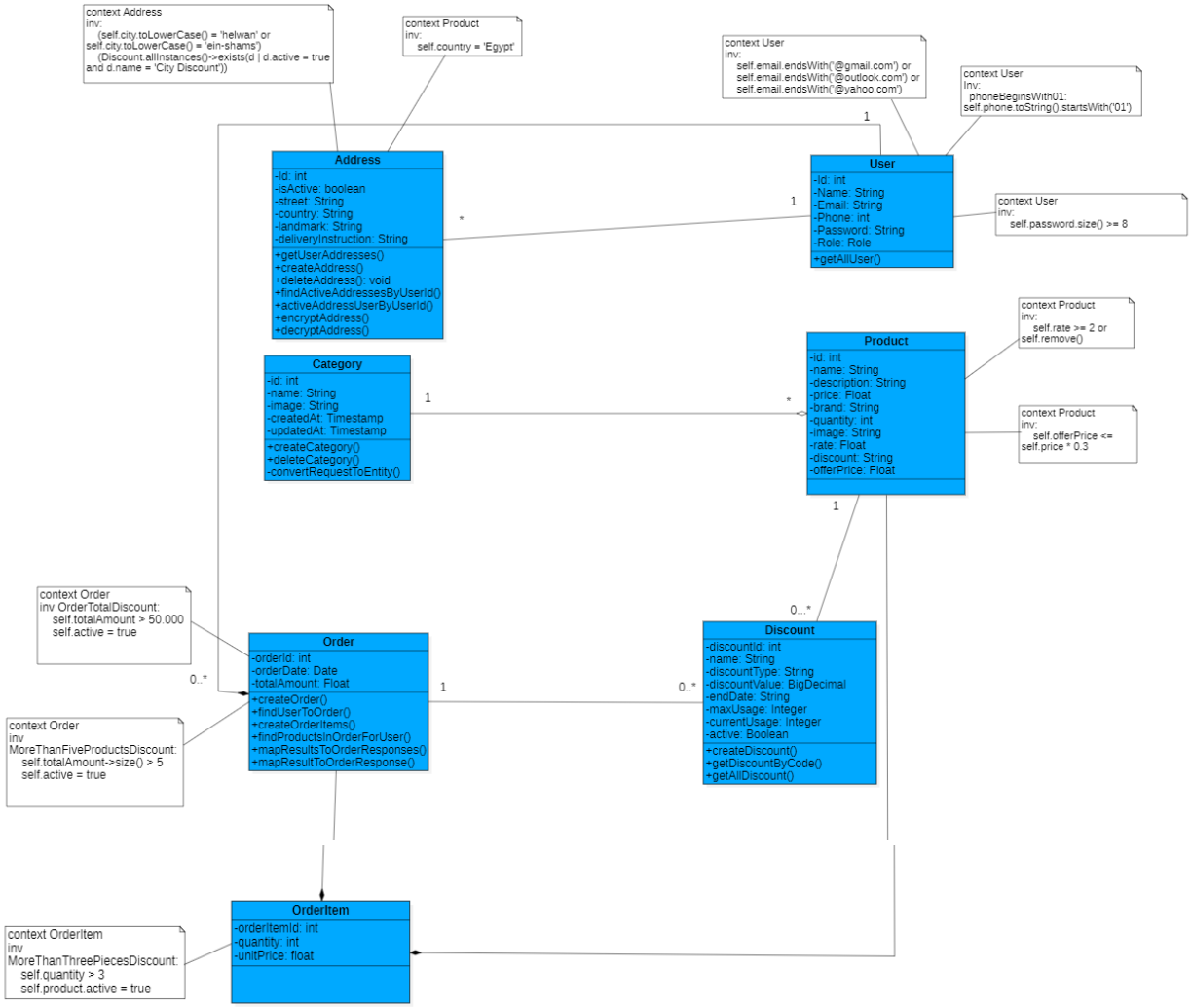
Login


user
: login_ui
: login controller
: Lifeline
1: enter username and password()
2: log in()
2.1: validate information()
2.1.1: get user()
alt
[valid]
2.1.1.1: user exist()
2.1.1.1.1: valid information()
2.1.1.1.1.1: home page()
[else]
2.1.1.2: user not exist()
2.1.1.2.1: invalid()
2.1.1.2.1.1: warning message()
Powered By: Visual Paradigm Community Edition

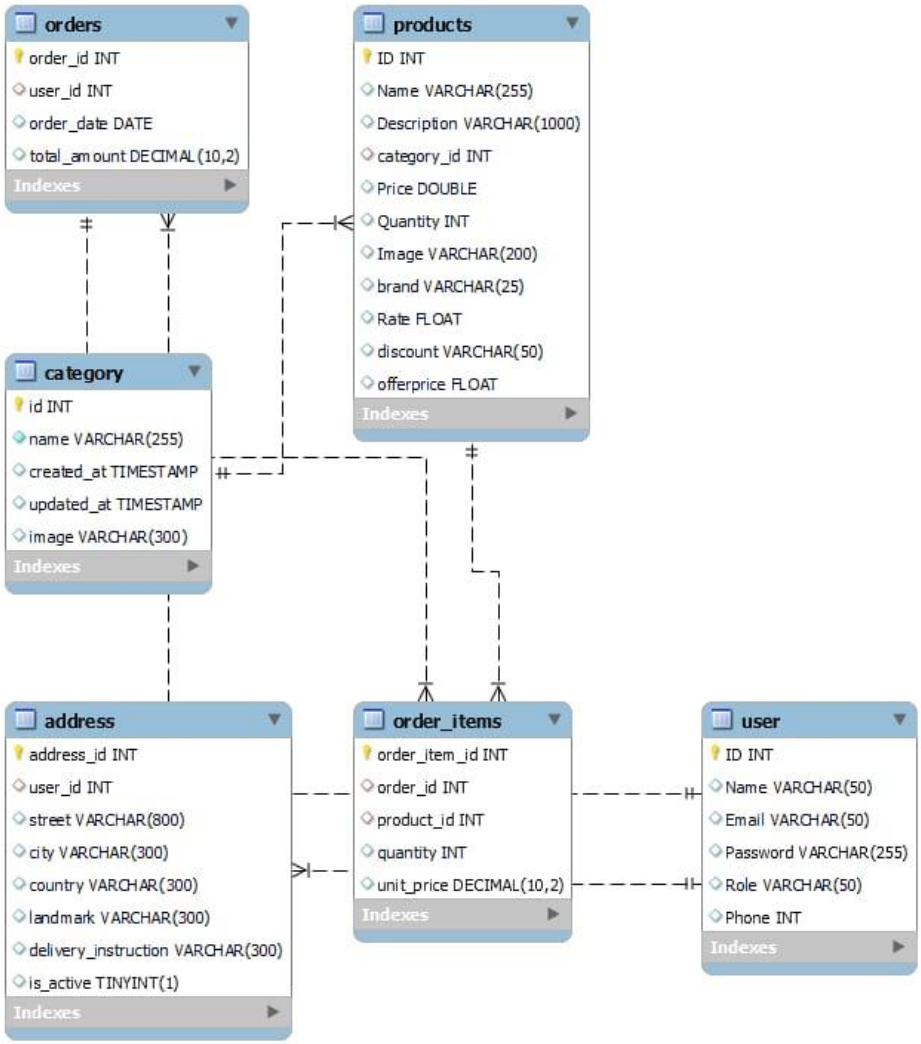
Order



4. Class Diagram


context Address
inv:
(self.city.toLowerCase() = 'helwan' or
self.city.toLowerCase() = 'ein-shams')
(Discount.allInstances()->exists(d | d.active = true
and d.name = 'City Discount'))
context Product
inv:
self.country = 'Egypt'
context User
inv:
self.email.endsWith('@gmail.com') or
self.email.endsWith('@outlook.com') or
self.email.endsWith('@yahoo.com')
context User
inv:
phoneBeginsWith01:
self.phone.toString().startsWith('01')
Address
-Id: int
-isActive: boolean
-street: String
-country: String
-landmark: String
-deliveryInstruction: String
+getUserAddresses()
+createAddress()
+deleteAddress(): void
+findActiveAddressesByUserId()
+activeAddressUserByUserId()
+encryptAddress()
+decryptAddress()
User
-Id: int
-Name: String
-Email: String
-Phone: int
-Password: String
-Role: Role
+getAllUser()
context User
inv:
self.password.size() >= 8
Category
-id: int
-name: String
-image: String
-createdAt: Timestamp
-updatedAt: Timestamp
+createCategory()
+deleteCategory()
-convertRequestToEntity()
Product
-id: int
-name: String
-description: String
-price: Float
-brand: String
-quantity: int
-image: String
-rate: Float
-discount: String
-offerPrice: Float
context Product
inv:
self.rate >= 2 or
self.remove()
context Product
inv:
self.offerPrice <=
self.price * 0.3
context Order
inv OrderTotalDiscount:
self.totalAmount > 50.000
self.active = true
Order
-orderId: int
-orderDate: Date
-totalAmount: Float
+createOrder()
+findUserToOrder()
+createOrderItems()
+findProductsInOrderForUser()
+mapResultsToOrderResponses()
+mapResultToOrderResponse()
Discount
-discountId: int
-name: String
-discountType: String
-discountValue: BigDecimal
-endDate: String
-maxUsage: Integer
-currentUsage: Integer
-active: Boolean
+createDiscount()
+getDiscountByCode()
+getAllDiscount()
context Order
inv
MoreThanFiveProductsDiscount:
self.totalAmount->size() > 5
self.active = true
OrderItem
-orderItemId: int
-quantity: int
-unitPrice: float
context OrderItem
inv
MoreThanThreePiecesDiscount:
self.quantity > 3
self.product.active = true
1
*
0..*
0...*

5. ERD


orders
order_id INT
user_id INT
order_date DATE
total_amount DECIMAL(10,2)
Indexes
products
ID INT
Name VARCHAR(255)
Description VARCHAR(1000)
category_id INT
Price DOUBLE
Quantity INT
Image VARCHAR(200)
brand VARCHAR(25)
Rate FLOAT
discount VARCHAR(50)
offerprice FLOAT
Indexes
category
id INT
name VARCHAR(255)
created_at TIMESTAMP
updated_at TIMESTAMP
image VARCHAR(300)
Indexes
address
address_id INT
user_id INT
street VARCHAR(800)
city VARCHAR(300)
country VARCHAR(300)
landmark VARCHAR(300)
delivery_instruction VARCHAR(300)
is_active TINYINT(1)
Indexes
order_items
order_item_id INT
order_id INT
product_id INT
quantity INT
unit_price DECIMAL(10,2)
Indexes
user
ID INT
Name VARCHAR(50)
Email VARCHAR(50)
Password VARCHAR(255)
Role VARCHAR(50)
Phone INT
Indexes