Faculty of Computer Science
and Artificial Intelligence,
Computer Science Department

Software Engineering 2

Team Members

	Name	ID
1	أحمد عبد الرشيد حسن	20210071
2	رمزي أشرف رمزي	20210333
3	آيات احمد محمد	20210196
4	الاء عاطف مصطفى	20210168
5	الاء مجدي بيومي	20210170
6	انجي موسى محمد	20210187

E-commerce website:

- Project idea and overview

E-commerce website specializing in electronic devices such as laptops, smartwatches, and mobile phones. This document aims to provide an in-depth understanding of the system's functionalities, user roles, features, and operational processes.

-Architectural Design

Introduction to Architecture: The architectural design of the project leverages modern software development principles and technologies to build a scalable, modular, and resilient system. By employing Spring Boot, Java, Microservices architecture, and Docker, and Maven.

At its core, Spring Boot utilizes a layered architecture, including controllers, services, repositories, and models. Controllers handle incoming requests, services contain business logic, repositories manage data access, and models represent the data structure. we aim to achieve flexibility, maintainability, and ease of deployment.

Architectural Styles and Patterns: The project follows the Microservices architectural pattern, which decomposes the application into small, loosely coupled services, each responsible for a specific business domain or functionality. This approach allows for

independent development, deployment, and scaling of individual services.

System Layers: Each microservice typically follows a layered architecture, including:

- **Presentation Layer:** Exposes RESTful APIs for communication with clients.
- **Service Layer:** Implements business logic and orchestrates interactions between components.
- **Data Access Layer:** Communicates with databases or external systems for data retrieval and persistence.

Scalability and Performance Considerations: Microservices architecture inherently supports scalability by allowing individual services to be scaled independently based on demand. Performance considerations include:

Technologies and Frameworks:

- **Spring Boot:** Used for rapid development of microservices with built-in support for dependency injection, RESTful services, and containerization.
- **Java:** The primary programming language for building microservices due to its maturity, scalability, and ecosystem.
- **Docker:** Employed for containerization to package microservices and their dependencies into lightweight, portable containers.
- **Maven:** Utilized for project build automation, dependency management, and project structure organization.

Project Requirements:

Functional Requirements:

1. **User Registration and Authentication:**
- Users should be able to register for an account with a valid email address and password.
- Registered users should be able to log in securely using their credentials.
2. **Product Management:**
- Admin should be able to add, edit, and delete product listings.
- Each product listing should include details such as name, description, price, image, and specifications.
3. **Shopping Cart Management:**
- Users should be able to add items to their shopping cart and view the contents.
- They should be able to modify quantities, remove items, or proceed to checkout.

4. **Order Management:**
- Users should be able to place orders securely.
- They should receive order confirmation emails upon successful completion of transactions.
5. **User Reviews and Ratings:**
- Users should be able to submit reviews and ratings for products they have purchased.
- Reviews should include text feedback and a rating scale to indicate satisfaction levels.
- Users should have the option to edit or delete their reviews.
6. **Review Moderation:**
- Admins should have the ability to moderate user reviews to ensure they meet community guidelines.
- Moderation actions may include approving, rejecting, or editing reviews as necessary.
7. **Category Management:**
- Admins should be able to create, edit, and delete product categories.
- Each category should have a unique name and optional description.
8. **Category Navigation:**
- Users should be able to navigate through categories and subcategories to explore products.

- Category navigation should be intuitive and provide clear paths for users to find relevant items.

Non-Functional Requirements:

1.Performance:

- The website should load quickly and respond promptly to user interactions.
- Response times for search queries, page loads, and checkout processes should be minimal.

2.Security:

- User data should be encrypted and stored securely to prevent unauthorized access.
- Sensitive user data, including personally identifiable information and payment details, should be encrypted both in transit and at rest.

3.Usability:

- The website should have an intuitive and user-friendly interface that is easy to navigate.
- Usability testing should be conducted with representative users to identify and address any usability issues.

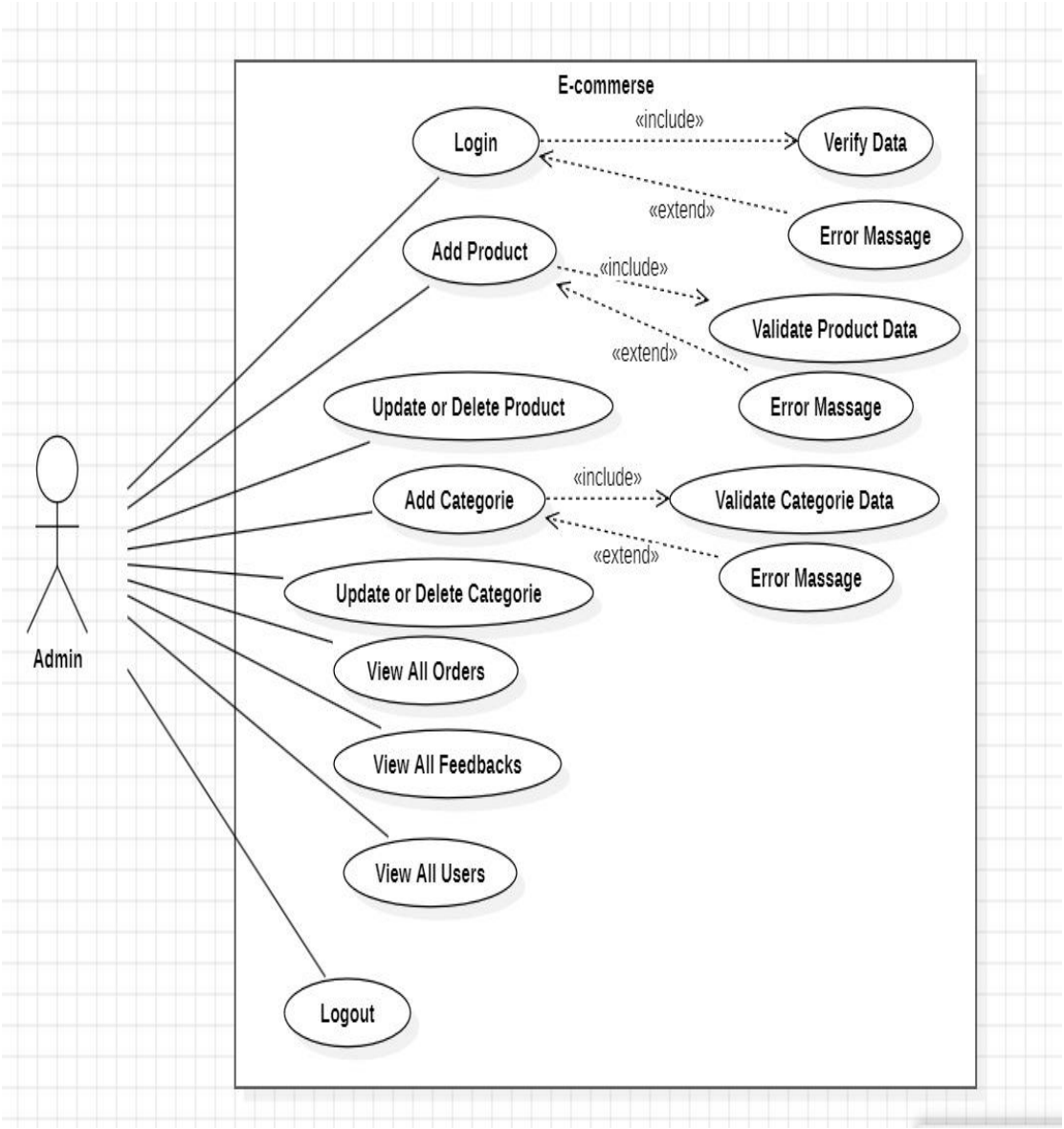
4.Availability:

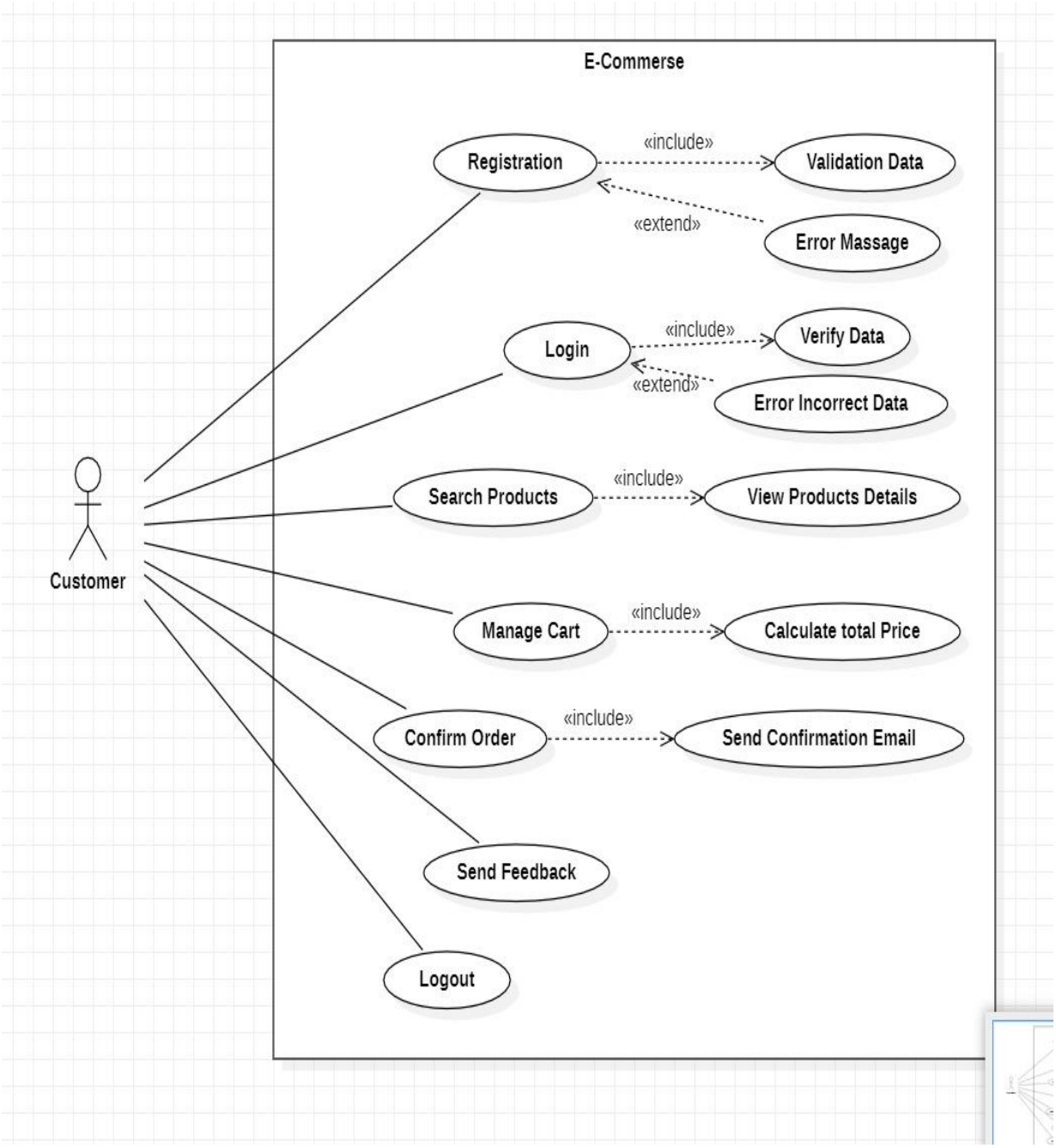
- The website should be available 24/7, with scheduled maintenance windows communicated to users in advance.

5.Regulatory Compliance:

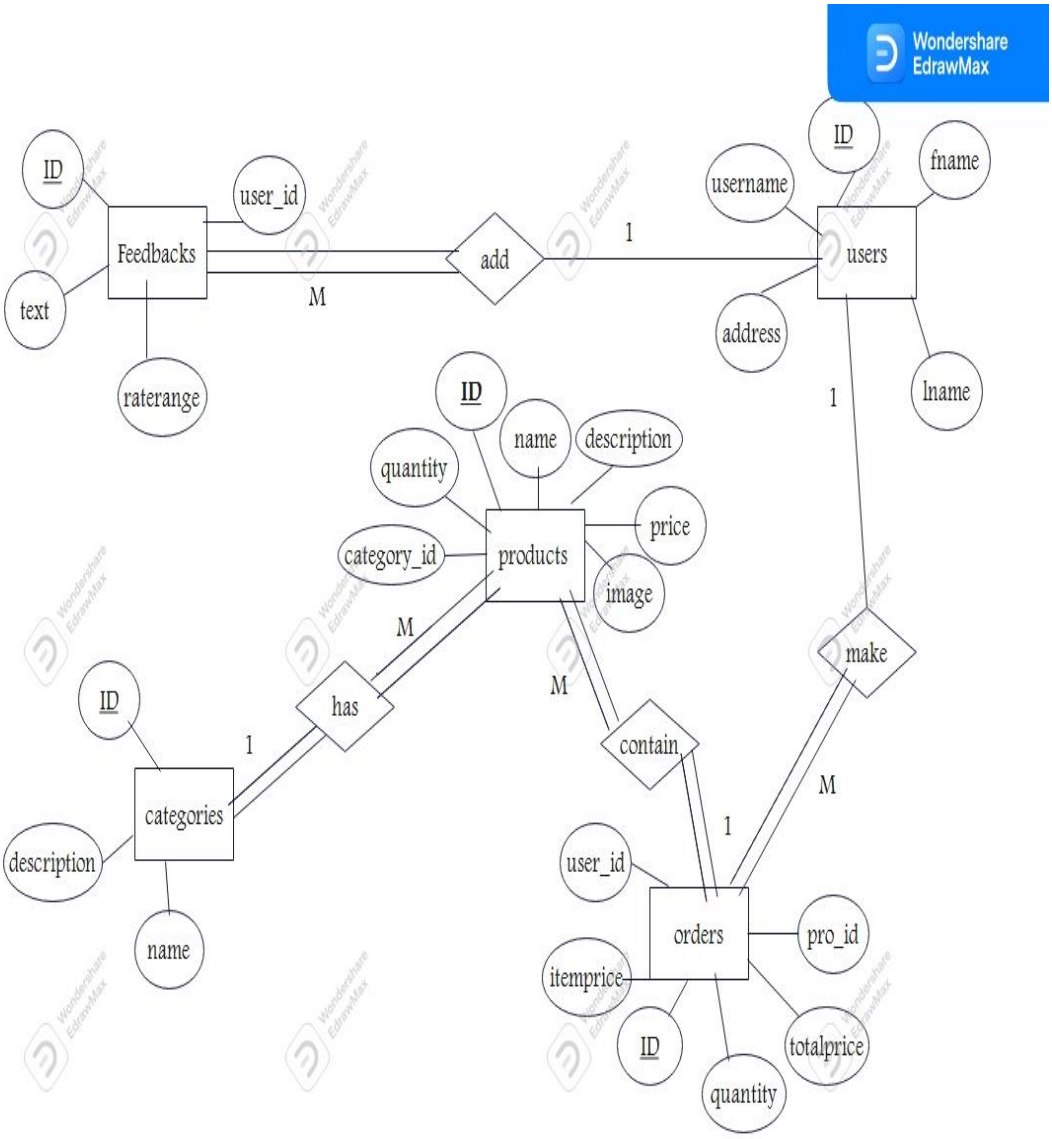
- Compliance measures should be implemented for user privacy and data security.

USE CASE

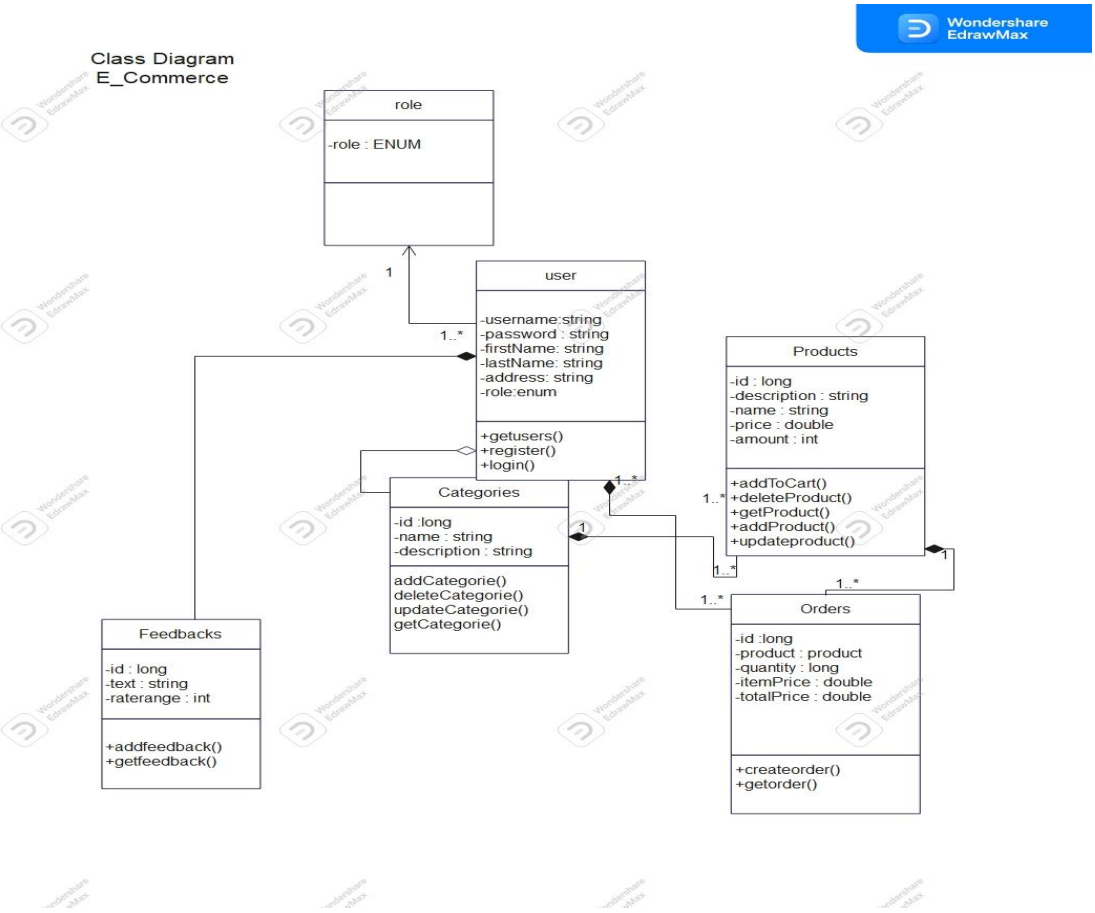

E-commerse
Login
«include»
Verify Data
«extend»
Error Massage
Add Product
«include»
Validate Product Data
«extend»
Error Massage
Update or Delete Product
Add Categorie
«include»
Validate Categorie Data
«extend»
Error Massage
Update or Delete Categorie
View All Orders
View All Feedbacks
View All Users
Logout
Admin


E-Commerse
Registration
«include»
Validation Data
«extend»
Error Massage
Login
«include»
Verify Data
«extend»
Error Incorrect Data
Search Products
«include»
View Products Details
Customer
Manage Cart
«include»
Calculate total Price
Confirm Order
«include»
Send Confirmation Email
Send Feedback
Logout

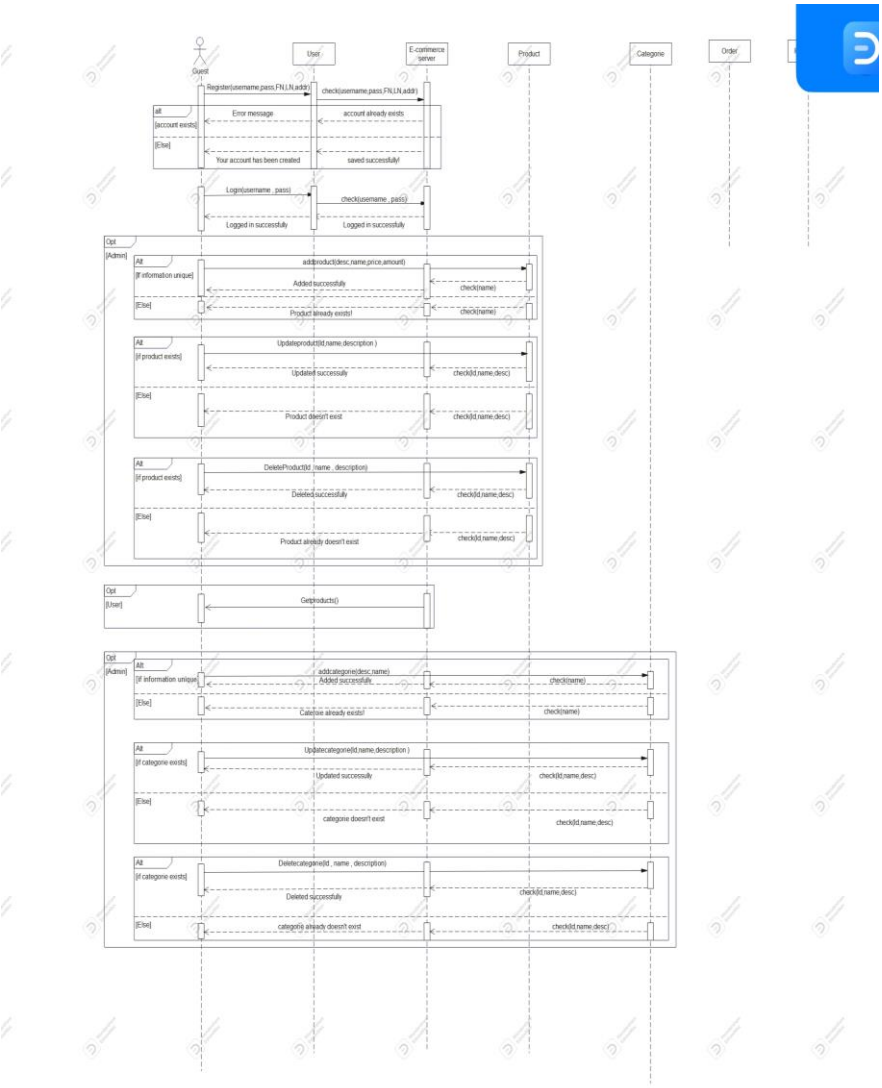
ERD

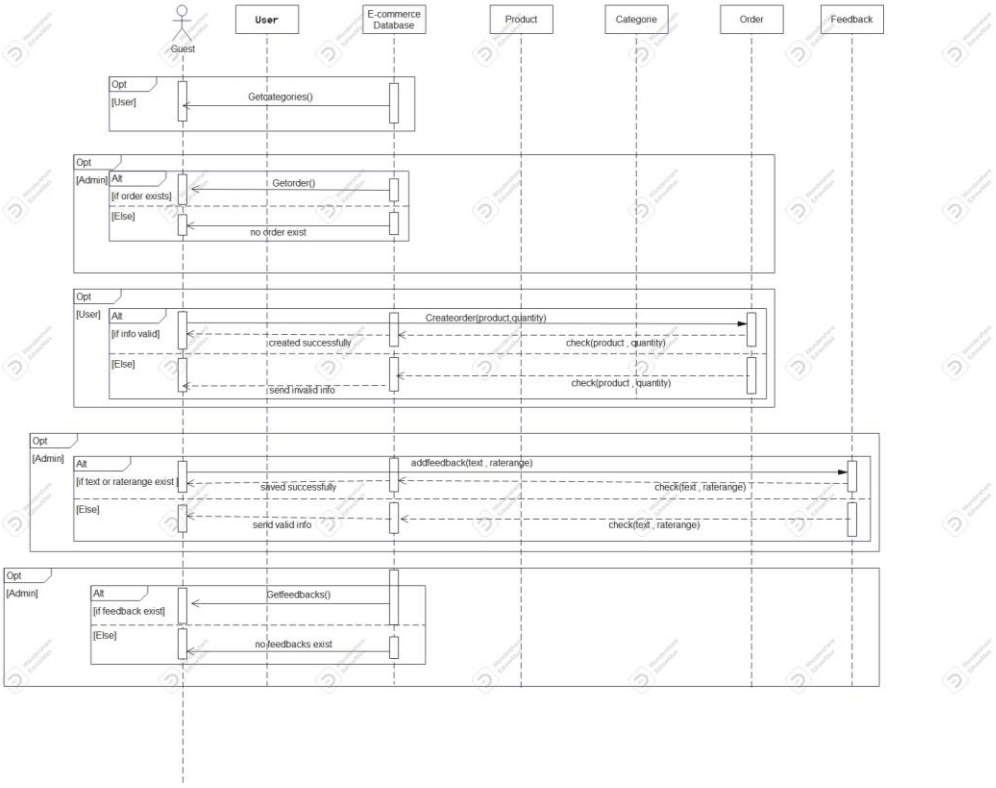

Wondershare EdrawMax
ID
user_id
Feedbacks
text
raterange
add
M
1
ID
username
fname
users
address
lname
1
ID
name
description
quantity
price
category_id
products
image
M
has
1
ID
categories
description
name
M
make
contain
M
1
user_id
orders
pro_id
itemprice
ID
totalprice
quantity

CLASS DIAGRAM

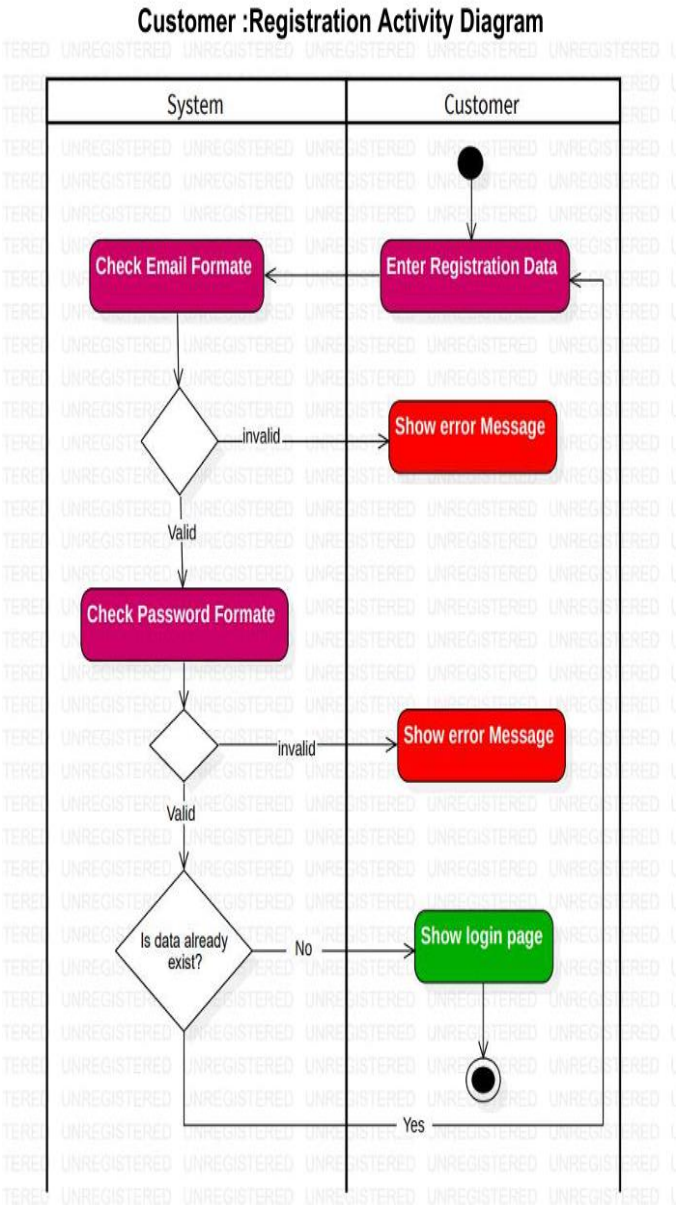


SEQUENCE DIAGRAM

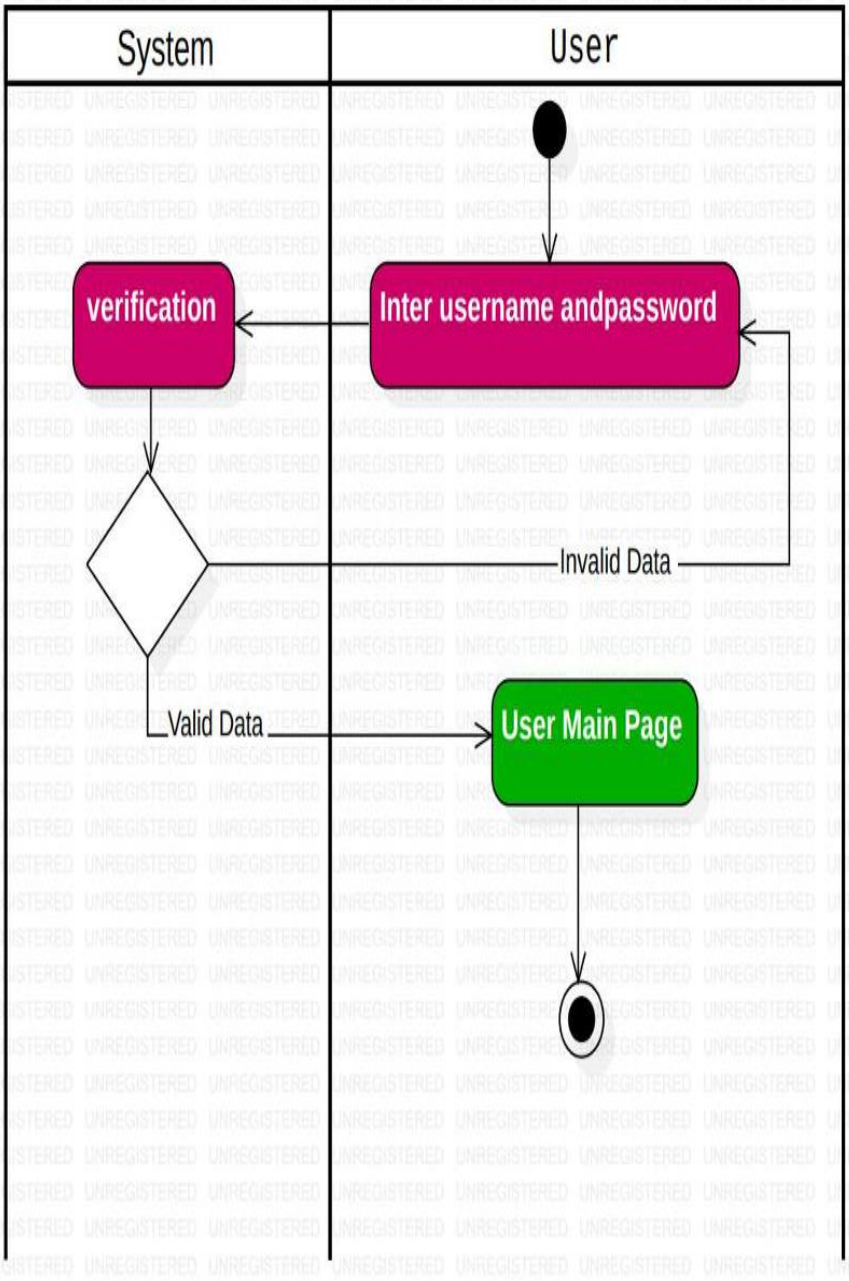



Guest
User
E-commerce Database
Product
Categorie
Order
Feedback
Opt
[User]
Getcategories()
Opt
[Admin]
Alt
[if order exists]
Getorder()
[Else]
no order exist
Opt
[User]
Alt
[if info valid]
Createorder(product,quantity)
created successfully
check(product , quantity)
[Else]
send invalid info
check(product , quantity)
Opt
[Admin]
Alt
[if text or raterange exist]
addfeedback(text , raterange)
saved successfully
check(text , raterange)
[Else]
send valid info
check(text , raterange)
Opt
[Admin]
Alt
[if feedback exist]
Getfeedbacks()
[Else]
no feedbacks exist

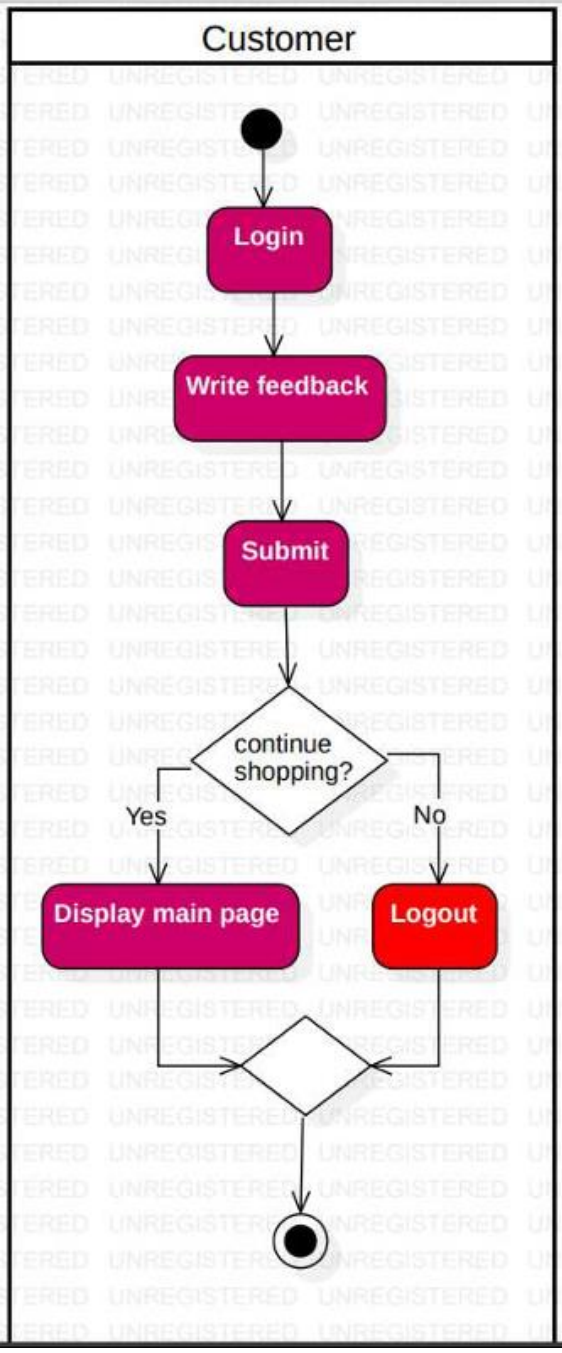
ACTIVITY DIAGRAM



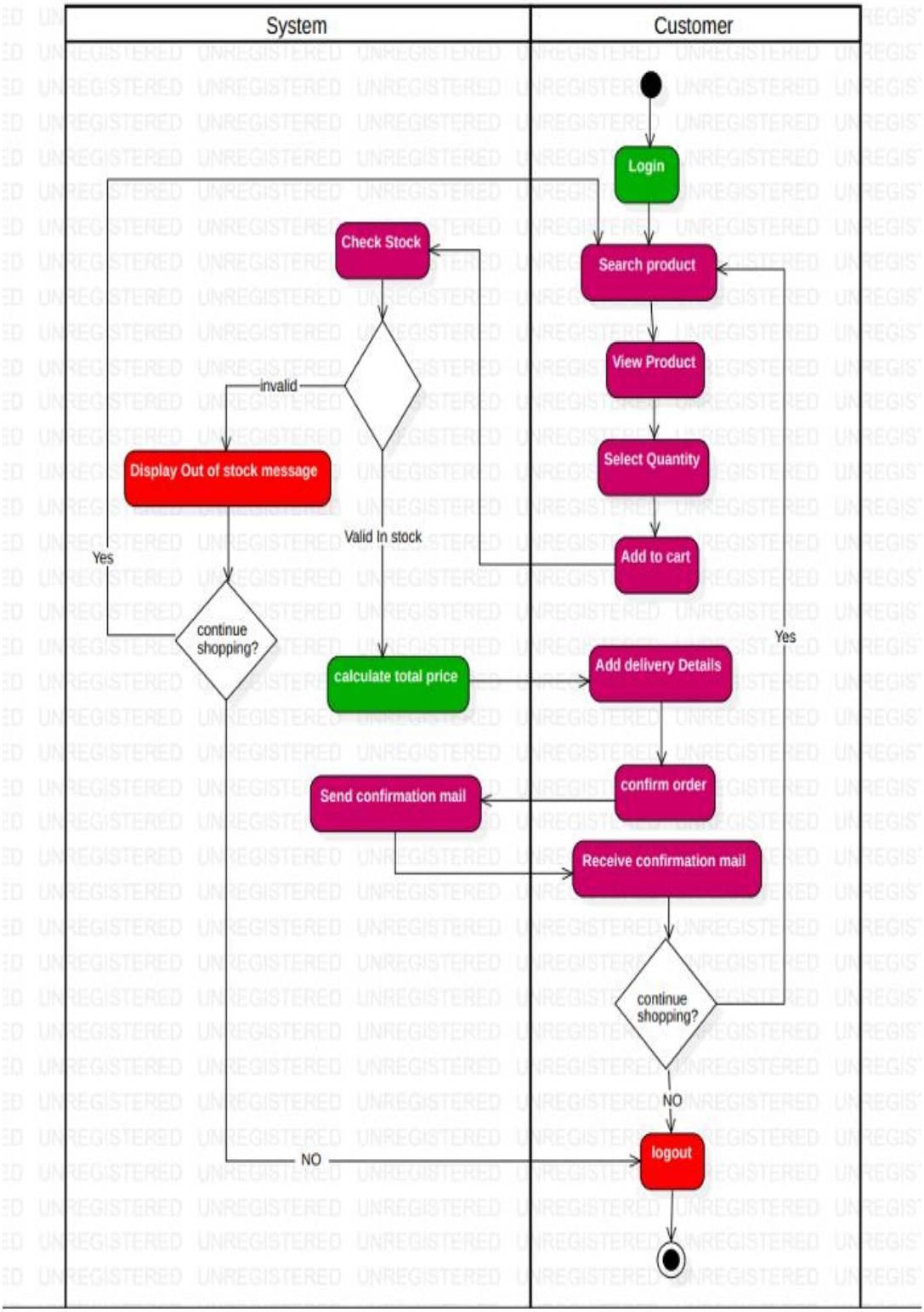
User(Customer/Admin) :Login Activity Diagram



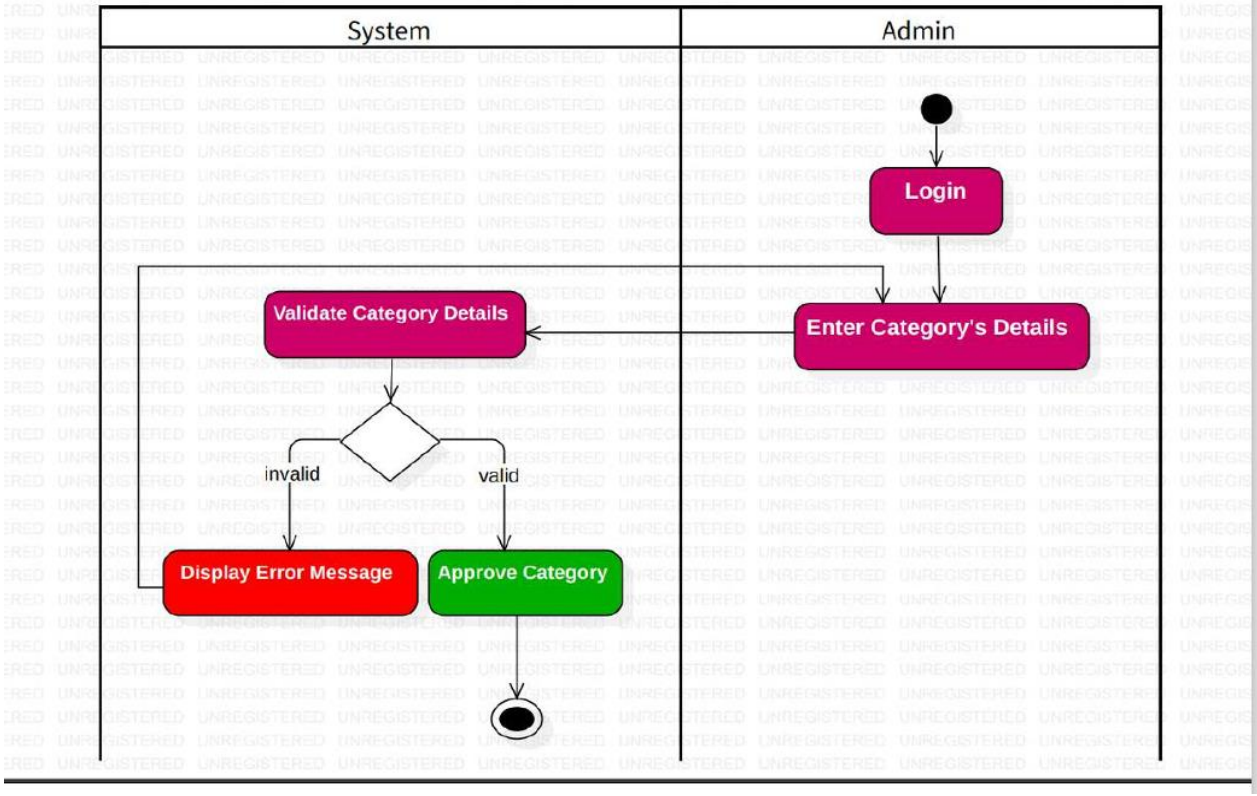
Customer : Feedback Activity Diagram



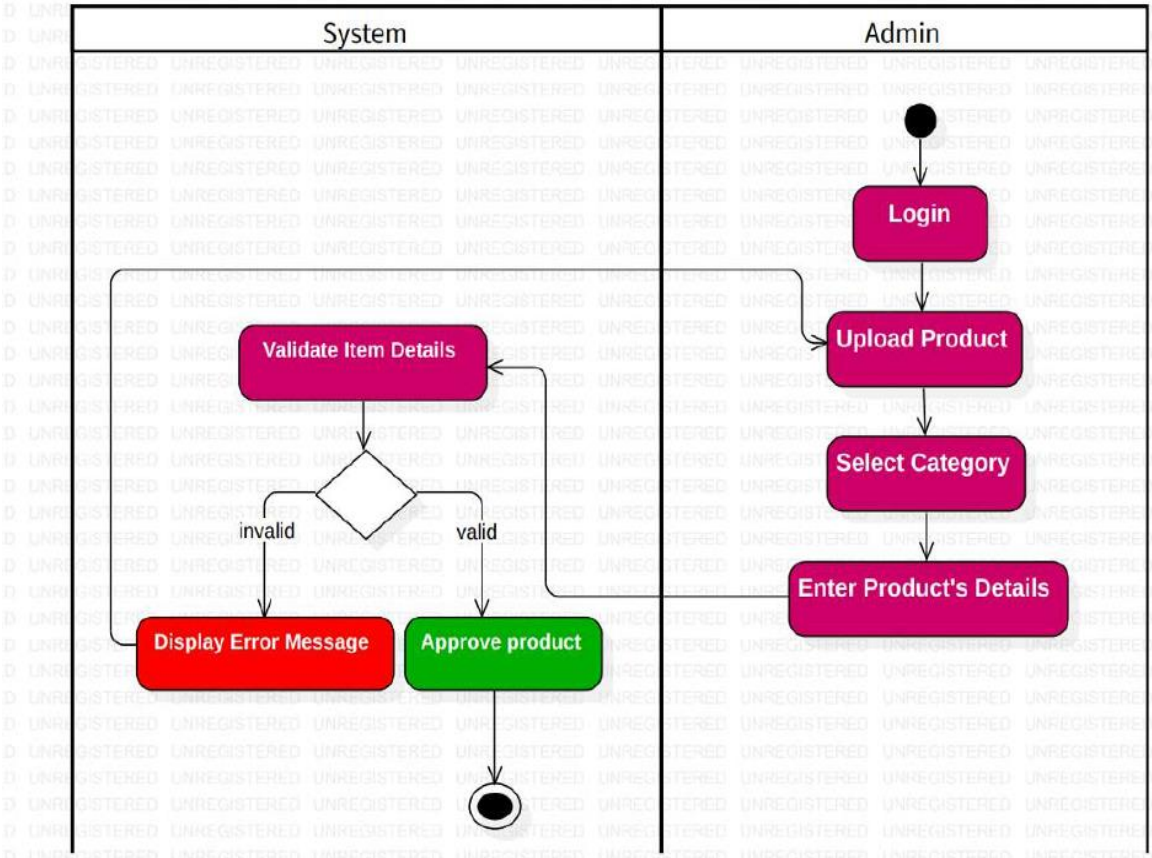
Customer:Order Activity Diagram



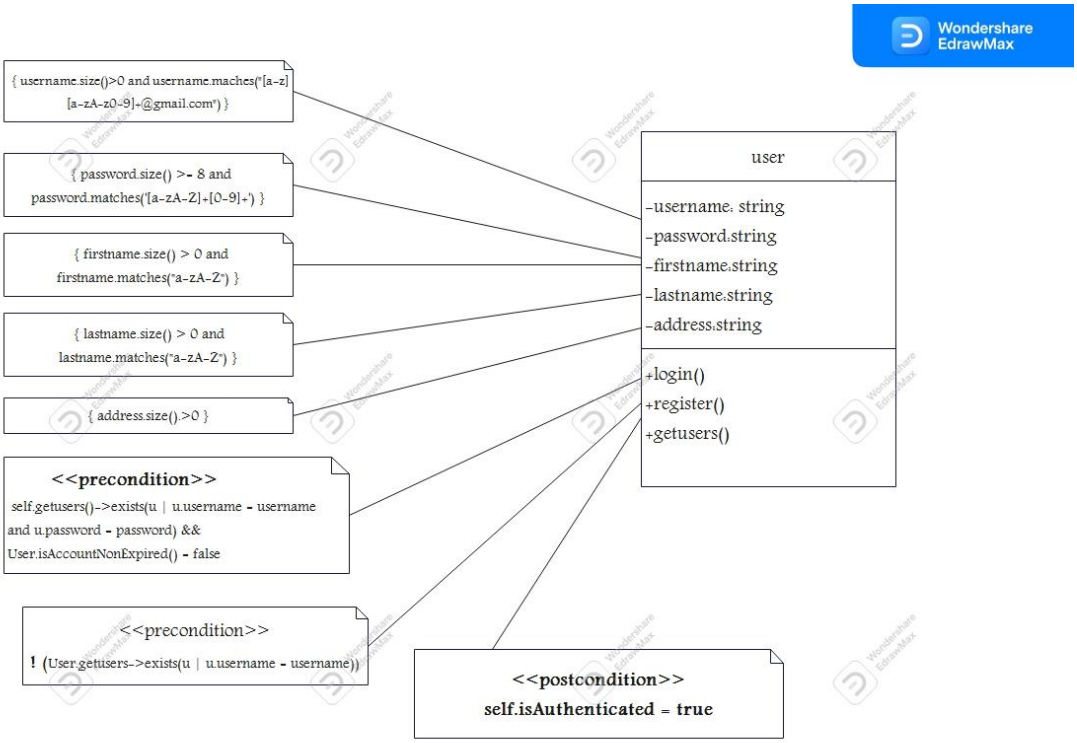
Admin:Add Category Activity Diagram

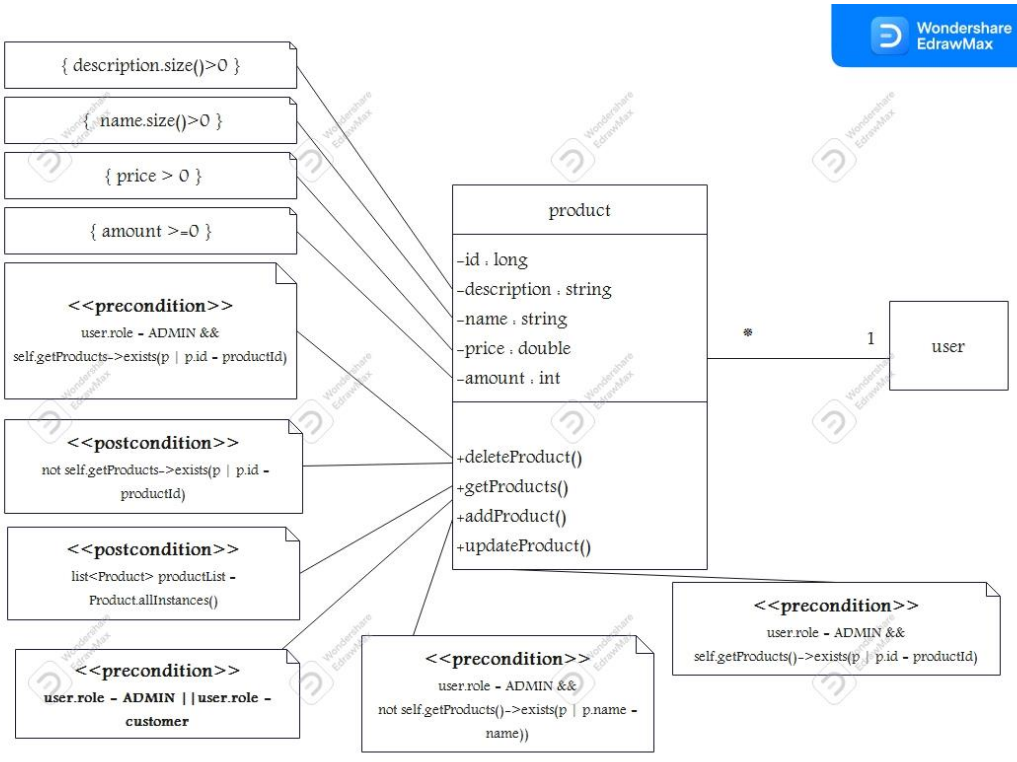


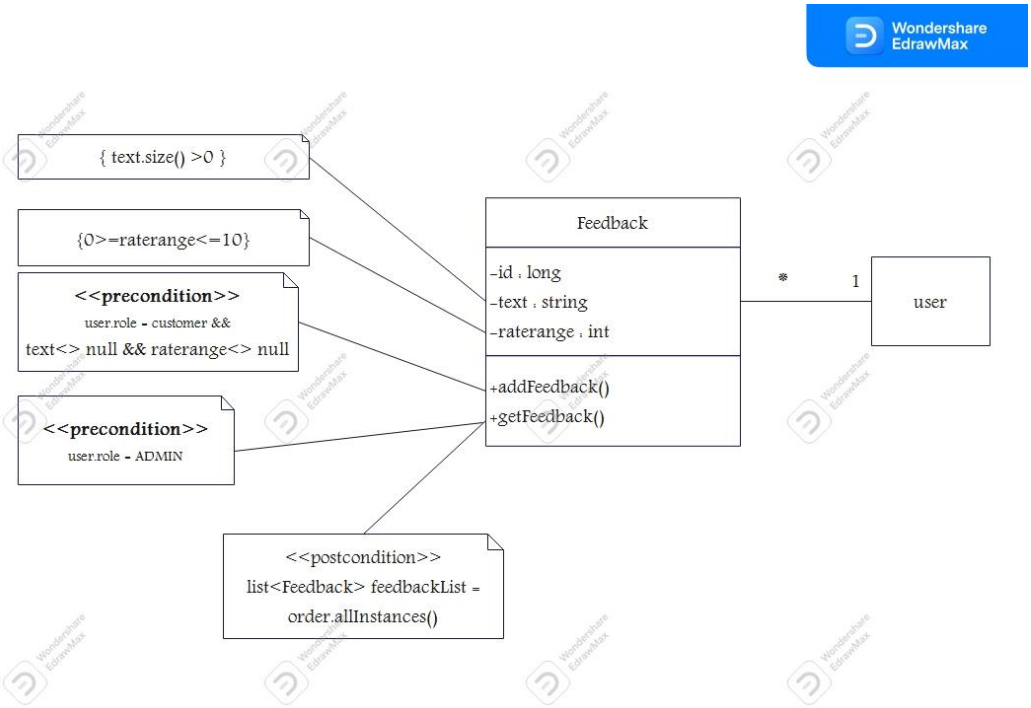
Admin:Add Product Activity Diagram

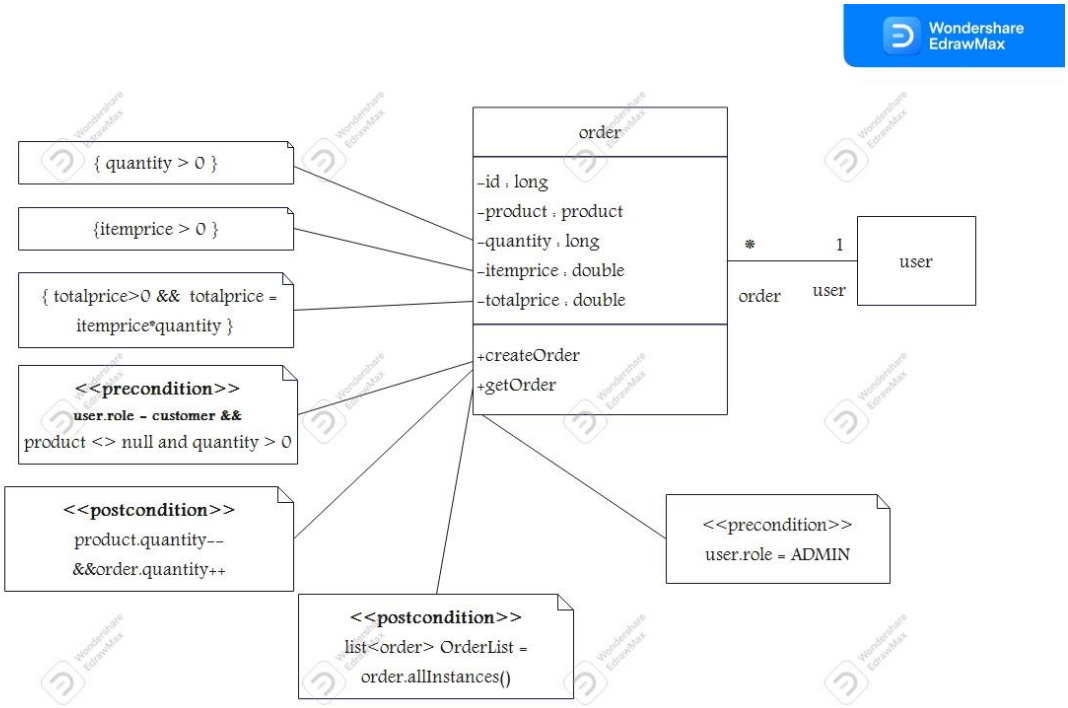


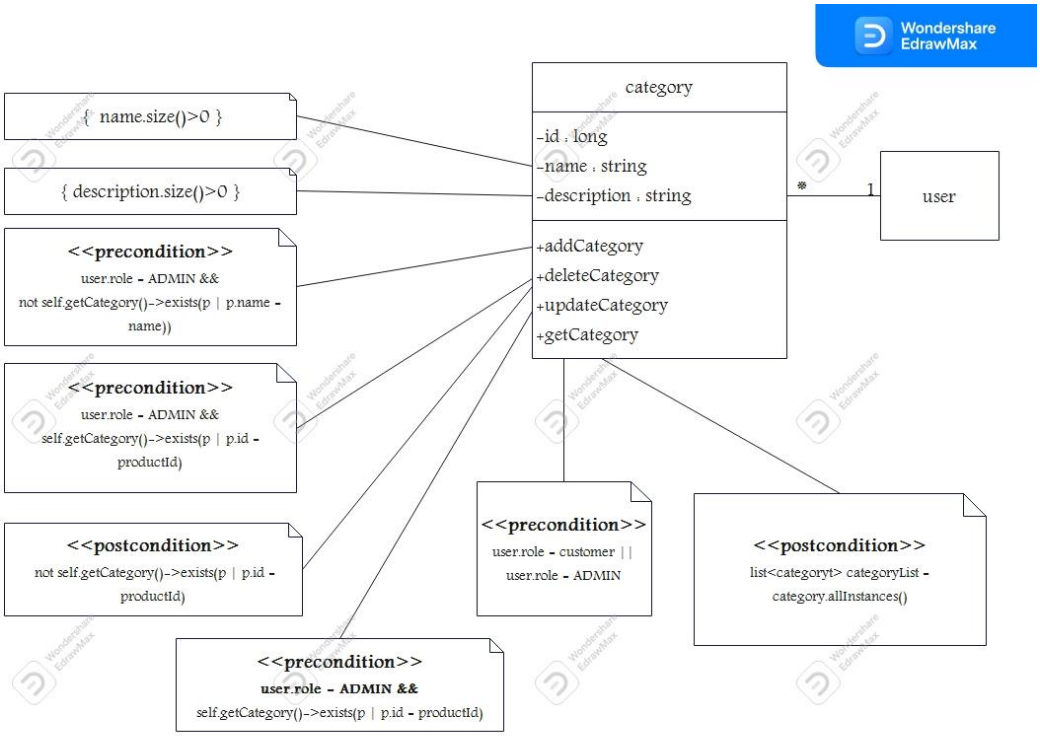
OCL


{ username.size()>0 and username.maches("[a-z][a-zA-z0-9]+@gmail.com") }
{ password.size() >= 8 and password.matches("[a-zA-Z]+[0-9]+") }
{ firstname.size() > 0 and firstname.matches("a-zA-Z") }
{ lastname.size() > 0 and lastname.matches("a-zA-Z") }
{ address.size().>0 }
<<precondition>>
self.getusers()->exists(u | u.username = username and u.password = password) && User.isAccountNonExpired() = false
<<precondition>>
! (User.getusers->exists(u | u.username = username))
user
-username: string
-password:string
-firstname:string
-lastname:string
-address:string
+login()
+register()
+getusers()
<<postcondition>>
self.isAuthenticated = true


{ description.size()>0 }
{ name.size()>0 }
{ price > 0 }
{ amount >=0 }
<<precondition>>
user.role = ADMIN && self.getProducts->exists(p | p.id = productId)
<<postcondition>>
not self.getProducts->exists(p | p.id = productId)
<<postcondition>>
list<Product> productList = Product.allInstances()
<<precondition>>
user.role = ADMIN || user.role = customer
product
-id : long
-description : string
-name : string
-price : double
-amount : int
+deleteProduct()
+getProducts()
+addProduct()
+updateProduct()
*
1
user
<<precondition>>
user.role = ADMIN && not self.getProducts()->exists(p | p.name = name))
<<precondition>>
user.role = ADMIN && self.getProducts()->exists(p | p.id = productId)


{ text.size() >0 }
{0>=raterange<=10}
<<precondition>>
user.role = customer &&
text<> null && raterange<> null
<<precondition>>
user.role = ADMIN
<<postcondition>>
list<Feedback> feedbackList =
order.allInstances()
Feedback
-id : long
-text : string
-raterange : int
+addFeedback()
+getFeedback()
*
1
user


{ quantity > 0 }
{itemprice > 0 }
{ totalprice>0 && totalprice =
itemprice*quantity }
<<precondition>>
user.role = customer &&
product <> null and quantity > 0
<<postcondition>>
product.quantity--
&&order.quantity++
<<postcondition>>
list<order> OrderList =
order.allInstances()
<<precondition>>
user.role = ADMIN
order
-id : long
-product : product
-quantity : long
-itemprice : double
-totalprice : double
+createOrder
+getOrder
*
1
order
user
user


Wondershare EdrawMax
category
-id : long
-name : string
-description : string
+addCategory
+deleteCategory
+updateCategory
+getCategory
*
1
user
{ name.size()>0 }
{ description.size()>0 }
<<precondition>>
user.role = ADMIN &&
not self.getCategory()->exists(p | p.name = name))
<<precondition>>
user.role = ADMIN &&
self.getCategory()->exists(p | p.id = productId)
<<postcondition>>
not self.getCategory()->exists(p | p.id = productId)
<<precondition>>
user.role = customer | |
user.role = ADMIN
<<postcondition>>
list<categoryt> categoryList = category.allInstances()
<<precondition>>
user.role = ADMIN &&
self.getCategory()->exists(p | p.id = productId)