Addis Ababa University
አዲስ፡አበባ፡ዩኒቨርሲቲ
SEEK WISDOM, ELEVATE YOUR INTELLECT AND SERVE HUMANITY !

College of Natural and Computational science

School of information science

Web-based commodity exchange system

No	Name	ID_No
1	Abdulmenan Mohammed	ugr/2627/12
2	Anteneh Amare	ugr/1571/12
3	Biruk Amare	ugr/6400/12
4	Haileamlak waleligne	ugr/1406/12
5	Kirubel Asnakew	ugr/3194/12
6	Samuel Tesfaye	ugr/4893/12

Advisor: Mr. Melaku Girma

Submission Date: June 13, 2023

Submitted to: School of Information Science

Table of Contents

List of Figures iii
List of Tables iv
Acknowledgment vi
List of Acronyms vii
Chapter Five: Object oriented design 1
5.1 Introduction 1
5.2 System architecture 1
5.2.1 Architecture style 1
5.2.2 Components of the system and their interaction 4
5.2.3 Principles used to design the system 5
5.3 Design class modelling 7
5.3.1 Class diagram 7
5.3.2 Description of classes 8
5.4 Relational persistent model 19
5.5 Component diagram 21
5.6 Deployment diagram 22
5.7 User interface 22
5.7.1 User interface flow diagram 22
5.7.2 User interface design (selected samples) 24
Chapter Six: Object oriented implementation 28
6.1 Introduction 28
6.2 Testing and testing procedures 28
6.2.1 Unit Testing 28
6.2.2 Integration Testing 29
6.2.3 System Testing 29
6.3 deployment / installation process 33
Chapter Seven: Conclusion and recommendations 34
7.1 Conclusion 34
7.2 Recommendation 34
Appendix 36
References 42

List of Figures

Figure 5.1 MVC model --3

Figure 5.2 Class diagram ---7

Figure 5.3 Relational persistent model--20

Figure 5.4 Component diagram ---21

Figure 5.5 Deployment diagram ---22

Figure 5.6 User interface flow diagram for buyer ---23

Figure 5.7 User interface flow diagram for admin --23

Figure 5.8 Login page ---24

Figure 5.9 Home page ---24

Figure 5.10 Sign up page ---25

Figure 5.11 Cart page ---25

Figure 5.12 Manage product page --26

Figure 5.13 Add category page ---26

Figure 5.14 Update product page ---27

Figure 5.15 Manage order page ---27

List of Tables

Table 6.1 login test --29

Table 6.2 signup test ---30

Table 6.3 Add product Test --30

Table 6.4 Update category Test --31

Table 6.5 Order management ---32

Table 6.6 Logout test ---32

Examination Board

Advisor Name	Signature	Date
Examiner Name	Signature	Date
Examiner Name	Signature	Date

Acknowledgment

First and foremost, we would like to thank god for giving us the necessary guidance, energy, and support to complete the whole process of the project.

We owe a great deal of gratitude to our advisor instructor Melaku Girma for his unfailing patience, continuous support, and valuable advises during the course of writing the whole paper. He gave us support for all the difficulties encountered starting from writing the proposal to the finalization of the project.

The team would also like to give out the deepest appreciation and gratitude to those who gave us every information that required for the completion of this project.

The development team thanks are also forwarded to individuals who directly or indirectly supported the team during the project work.

Last but not least, we would like to thank our families and friends for providing us priceless support and encouragement during our study.

List of Acronyms

- **RDBMS**: Relational database management system
- **UML**: Unified modeling language
- **HTML**: Hypertext markup language
- **CSS**: Cascading stylesheet
- **JS**: JavaScript
- **API**: Application programming interface
- **NPM**: Node package manager
- **MVC**: model view controller

Chapter five

Object oriented design

5.1 Introduction

The purpose of object-oriented design is to design a system and gather the necessary information to bring the actual implementation of the system to life. It differs from analysis, which primarily focuses on understanding what will be built, by placing emphasis on the process of how the system will be built. [1]

In this chapter we will discuss major object oriented design artifacts to help us in implementing the system. We will be discussing class modelling, component modelling, deployment modelling, user interface design and more design artifacts.

5.2 System architecture

A system architecture is a set of principles that define the way system is designed and developed. An architecture defines the structure of the software and how it is organized. It also describes the relationships between components, levels of abstraction, and other aspects of the software system. An architecture can be used to define the goals of a project, or it can be used to guide the design and development of a new system. A software architecture is a set of principles that define the way software is designed and developed. [2]

5.2.1 Architecture style

The Model-View-Controller (MVC) framework is an architectural pattern that separates an application into three main logical components Model, View, and Controller. Each architecture component is built to handle specific development aspects of an application. [3]

Model

The model is responsible for managing the data of the application.

List of models:

- User
- Product
- Category
- Order
- Cart
- Notification
- Contact

View

It means a presentation of data in a particular format, triggered by a controller's decision to present the data.

List of views:

- Home page
- Login page
- Sign up page
- Cart page
- Checkout page
- Manage product page
- Manage category page

Controller

The controller is responsible for responding to the user input and performing interactions on the data model objects.

List of controllers:

- User controller
- Product controller
- Category controller
- Cart controller
- Order Controller

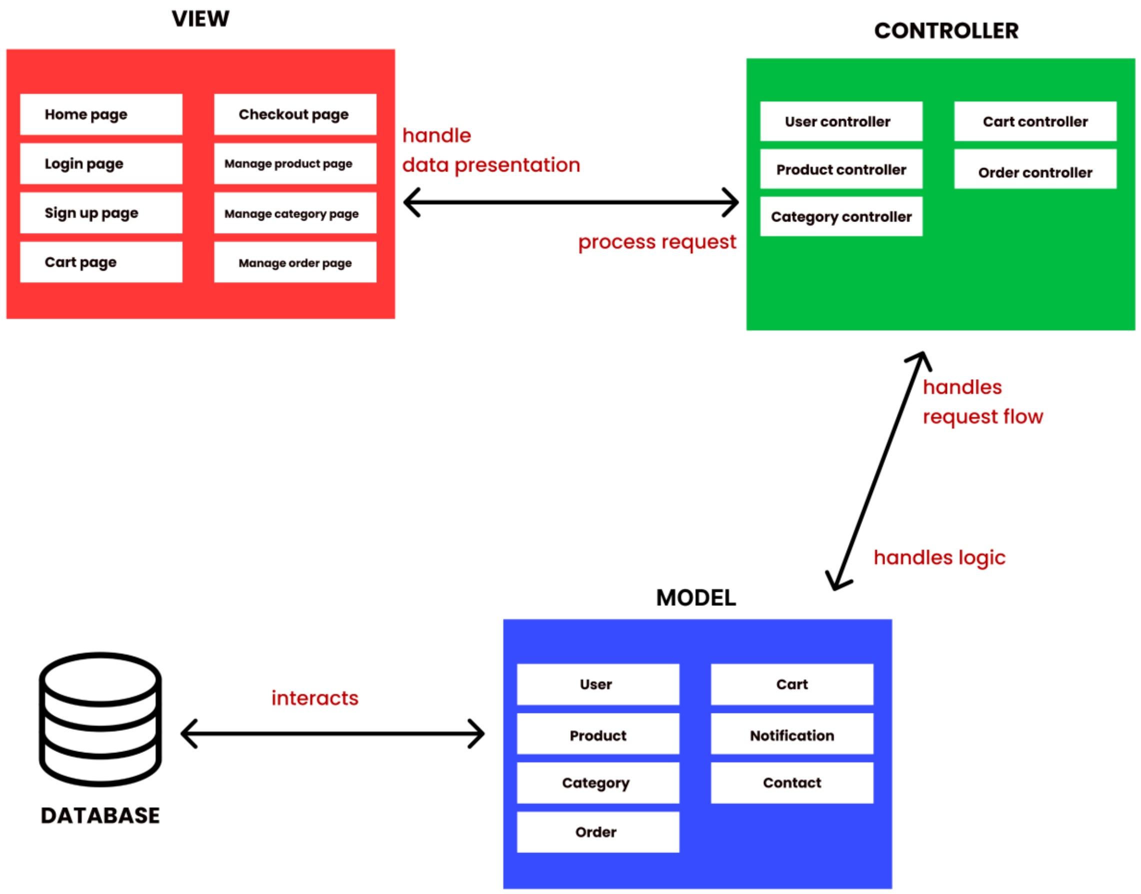


Figure 5.1 MVC model

5.2.2 Components of the system and their interaction

Frontend and User Interface:

The user interface, developed using frontend technologies, provides a visual platform for user interaction. Users access the web-based platform and perform tasks like registration, login, search, and messaging through the frontend. The frontend communicates with the backend through APIs to exchange and request data.

For example, when a user initiates a search, the user interacts with the frontend interface by entering a search query and clicking the search button. This triggers a request from the frontend to the backend API, passing the search query as a parameter. The backend, configured with a specific route for search requests, has a corresponding controller that handles the incoming request. The controller processes the search query, performing tasks such as querying a database or making API calls.

Once the query is processed, the backend controller generates a response in JSON format containing the search results. On the frontend side, the response is received and parsed, extracting the relevant data to dynamically update the user interface. The updated UI presents the search results, displaying the necessary information to the user, such as titles and descriptions. Through this integration, our system seamlessly provides users with a search feature that efficiently retrieves and displays the requested information.

Backend:

The backend component handles the core logic and functionality of the web based commodity exchange system. It receives requests from the frontend and processes them accordingly. User authentication, input validation, and secure storage of user data in the database are handled by the backend. In response to search requests, the backend collects the necessary information based on the search criteria and sends it to the frontend. The backend interacts with the database to store and retrieve information related to users, products, orders, and other entities.

APIs:

APIs act as the bridge between front end and backend components, allowing them to communicate and exchange data. The frontend component makes requests to the backend API to fetch data, submit forms, or perform other actions. The backend API processes these requests, retrieves or manipulates the necessary data, and sends back the response to the frontend.

Database:

The database component manages the persistent data of the web based commodity exchange system. The backend communicates with the database to store and retrieve data. When a user registers, for example, the backend creates a new user entry in the database. information submitted by users is stored in the appropriate database tables by the backend. The database provides the necessary tools for data storage, retrieval, and manipulation.

5.2.3 Principles used to design the system

The following design principles were applied when we were creating the web based commodity exchange system to provide a seamless user experience and carry out the required functionalities.

Simplicity:

This principle highlights the significance of simplicity in the design of our web-based commodity exchange system. By prioritizing simplicity, we ensure that the system remains straightforward, focused, and effectively addresses the specific problems and requirements of the commodity exchange domain. This approach makes the system easier to understand, maintain, and troubleshoot, enhancing its overall efficiency.

Legibility:

Legibility is crucial in the design of our web-based commodity exchange system. A well-designed system should be easily comprehensible to developers, enabling them to modify, enhance, and fix issues efficiently. By ensuring the legibility of the system's architecture, code, and interfaces, we facilitate smooth development processes, facilitate future enhancements, and improve the system's adaptability to changing market dynamics.

Core-focused:

Our system design should prioritize solving core problems directly related to the commodity exchange process. By avoiding the inclusion of non-core functionalities or unnecessary features, we ensure that the system remains streamlined, user-centric, and aligned with the primary objectives of facilitating efficient commodity trading. This approach eliminates potential complexities and distractions, enabling users to focus on essential trading activities.

Scalability:

The design of our web-based commodity exchange system should cater to scalability, allowing it to handle both simple and complex trading scenarios. The initial system design should accommodate basic use cases, while also providing flexibility for future growth and the evolution of trading requirements. This scalability enables the system to seamlessly adapt to changing market dynamics and increasing user demands. [4]

5.3 Design class modelling

5.3.1 Class diagram

A class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. [5]

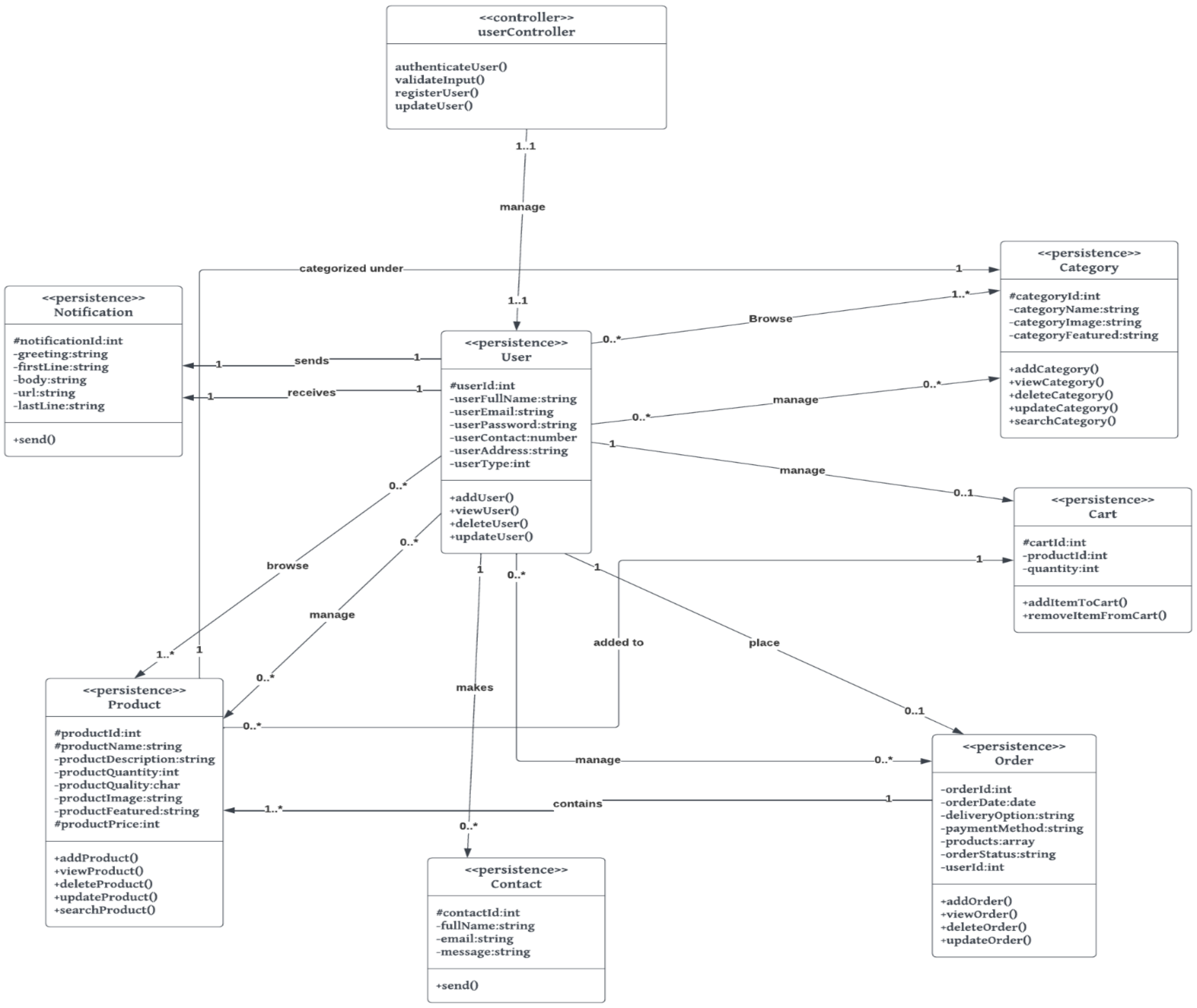


Figure 5.2 class diagram

5.3.2 Description of classes

Class Name: User (buyer or admin**)**

Attributes:

- **userId**
  - Data type: int
  - Visibility: Protected
  - Description: Represent unique identifier of a user

- **userFullname**
  - Data type: String
  - Visibility: Private
  - Description: Represent the complete name of an individual user

- **userEmail**
  - Data type: String
  - Visibility: Private
  - Description: Represent the email of an individual user

- **userPassword**
  - Data type: String
  - Visibility: Private
  - Description: Represent the password of an individual user

- **userContact**
  - Data type: Int
  - Visibility: Private
  - Description: Represent the phone number of an individual user

- **userAddress**
- Data type: String
- Visibility: Private
- Description: Represent the address of an individual user

- **userType**
- Data type: int
- Visibility: Private
- Description: Represent the type of an individual user(admin or user)

Methods:

- **AddUser**
- Visibility: Public
- Parameters: UserID ,Fullname ,Email, Password, Contact Number, Address
- Description : Add new user to the system

- **viewUser**
- Visibility: Public
- Parameters: None
- Description : View user data

- **deleteUser**
- Visibility: Public
- Parameters: None
- Description : Delete user data from the system

- **updateUser**
- Visibility: Public
- Parameters: UserID ,Fullname ,Email, Password, Contact Number, Address
- Description : Update user data

Class Name: Product

Attributes:

- **productId**
- Data type: int
- Visibility: Protected
- Description: Represent unique identifier of a product

- **productName**
- Data type: String
- Visibility: Protected
- Description: Represent the name of a product

- **productDescription**
- Data type: String
- Visibility: Private
- Description: Represent the description of a product

- **productQuantity**
- Data type: Int
- Visibility: Private
- Description: Represent the quantity of a product

- **productQuality**
- Data type: char
- Visibility: Private
- Description: Represent the quality of a product

- **productImage**
  - Data type: String
  - Visibility: Private
  - Description: Represent the image of a product

- **productFeatured**
  - Data type: String
  - Visibility: Private
  - Description: Tells whether the product is on home page or not

- **productPrice**
  - Data type: int
  - Visibility: Protected
  - Description: Represent the price of a product

Methods :

- **addProduct**
  - Visibility: protected
  - Parameters: product_id,product_name,product_description,product_quantity,product_quality,product_image,product_featured,product_price
  - Description : Add new product to the system

- **viewProduct**
  - Visibility: Public
  - Parameters: None
  - Description : View product data

- **deleteProduct**
- Visibility: Protected
- Parameters: None
- Description : Delete product data from the system

- **updateProduct**
- Visibility: Protected
- Parameters: product_id,product_name,product_description,product_quantity,product_quality,product_image,product_featured,product_price
- Description: Update product data

- **searchProduct**
- Visibility: Protected
- Parameters: product_name
- Description: search specific product

Class Name: Category

Attributes:

- **categoryId**
- Data type: int
- Visibility: Protected
- Description: Represent unique identifier of a category

- **categoryName**
- Data type: String
- Visibility: Public
- Description: Represent the name of a category

- **categoryImage**
  - Data type: String
  - Visibility: public
  - Description: Represent the image of a categroy

- **categoryFeatured**
  - Data type: String
  - Visibility: Public
  - Description: Tells whether the category is on home page or not

Methods :

- **addCategory**
  - Visibility: Public
  - Parameters: category_id,category_name,category_image,category_featured
  - Description : Add new category to the system

- **viewCategory**
  - Visibility: Public
  - Parameters: None
  - Description : View category data

- **deleteCategory**
  - Visibility: Public
  - Parameters: None
  - Description : Delete category data from the system

- **updateCategory**
  - Visibility: public
  - Parameters: category_id, category_name,category_image,category_featured
  - Description: Update category data

- **searchCategory**
- Visibility: Protected
- Parameters: category_name
- Description: search specific category

Class Name: Order

Attributes:

- **orderId**
- Data type: int
- Visibility: Protected
- Description: Represent unique identifier of an order

- **orderDate**
- Data type: Date
- Visibility: Private
- Description: Represent the date of an order

- **deliveryOption**
- Data type: String
- Visibility: Private
- Description: Represent delivery option of a user's order

- **paymentMethod**
- Data type: String
- Visibility: Private
- Description: Represent payment method of a user's order

- **products**

- Data type: Array
- Visibility: Private
- Description: Represent all products ordered by the buyer

- **orderStatus**

- Data type: String
- Visibility: Private
- Description: Represent status of an order

Methods :

- **addOrder**

- Visibility: Public
- Parameters: order_id,order_date,delivery_option,payment_method,order_status
- Description : Add new order to the system

- **viewOrder**

- Visibility: Public
- Parameters: None
- Description: View order data

- **deleteOrder**

- Visibility: Public
- Parameters: None
- Description: Delete order data from the system

- **updateOrder**

- Visibility: public
- Parameters: order_id,order_date,delivery_option,payment_method,order_status
- Description: Update order status

Class Name: Notification

Attributes:

- **notificationId**
- Data type: int
- Visibility: Protected
- Description: Represent unique identifier of a notification

- **greeting**
- Data type: String
- Visibility: Private
- Description: Represent the greeting section of the notification

- **firstLine**
- Data type: String
- Visibility: Private
- Description: Represent the first line section of the notification

- **body**
- Data type: String
- Visibility: Private
- Description: represent the body of the notification

- **lastLine**
- Data type: String
- Visibility: Private
- Description: Represent the last line section of the notification

- **url**
- Data type: String
- Visibility: Private
- Description: Represent the email address

Methods :

- **send**
- Visibility: Public
- Parameters: greeting,firstline,body,lastline,url
- Description : Add new category to the system

Class Name: Cart

Attributes:

- **cartId**
- Data type: int
- Visibility: Protected
- Description: Represent unique identifier of a cart

- **quantity**
- Data type: int
- Visibility: private
- Description: Represent the quantity of specific product

Methods :

- **addItemToCart**
- Visibility: Public
- Parameters: productId,quantity
- Description : Add new item to the cart

- **removeItemFromCart**
- Visibility: Public
- Parameters: cartId
- Description : remove item from cart

Class Name: Contact

Attributes:

- **contactId**
- Data type: int
- Visibility: Protected
- Description: Represent unique identifier of a contact message

- **fullName**
- Data type: string
- Visibility: protected
- Description: Represent the full name of sender

- **email**
- Data type: string
- Visibility: protected
- Description: Represent the email of sender

- **message**
- Data type: string
- Visibility: protected
- Description: Represent the message of sender

Methods :

- **send**
- Visibility: Public
- Parameters: fullname,email,message
- Description : send message

Class Name: userController

Methods :

- **authenticateUser**
- Visibility: Protected
- Parameters: email,password
- Description : authenticate a user by verifying their email and password

- **validateInput**
- Visibility: Protected
- Parameters: none
- Description : ensuring the user input meets the required criteria

5.4 Relational persistent model

Persistence models are used to design the schema of the database. Persistence model is created whenever relational database is used to store objects and relational database is used as a mechanism to object persistence. In persistence model, class is conceptually the same as the table of relational database and attributes are the same as table columns. [6]

Our team used the following methods to map the objects and classes to the RDBMS:

Mapping class to tables: the tables in the RDBMS correspond back to the classes in our system

Mapping attributes to columns: Attributes in a class will be mapped to corresponding columns inside a table in the RDBMS.

Mapping relationships in to foreign key: under persistence modelling, the mapping of association results in creating a foreign key in one or more tables in the set of relational tables.

Below are list of tables with their columns:

- User(userId,userFullName, userEmail, userPassword, userContact,userAddress,userType)
- Product(productId,productName,productDescription,productQuantity,productQuality,productImage,productFeatured,productPrice)
- Category(categoryId,categoryName,categoryImage,categoryFeatured)
- Notification(notificationId,greeting,firstLine,body,url,lastLine)
- Order(orderId,orderDate,deliveryOption,paymentMethod,productId,orderStatus,userId)
- Cart(cartId,productId,quantity)
- userController()

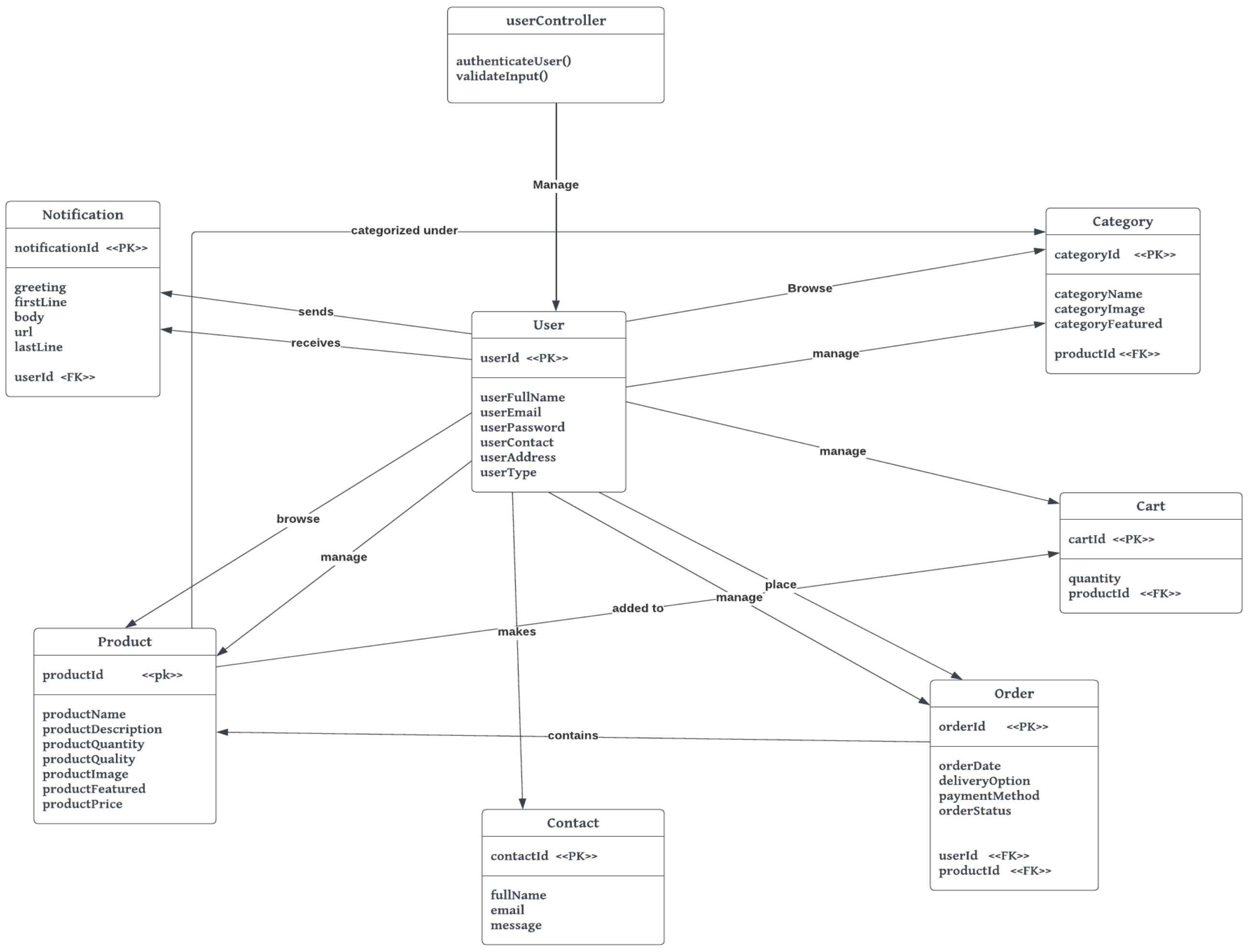


Figure 5.3 Relational persistent model

5.5 Component diagram

A component diagram, also known as a UML component diagram, describes the organization and wiring of the components in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development. [7]

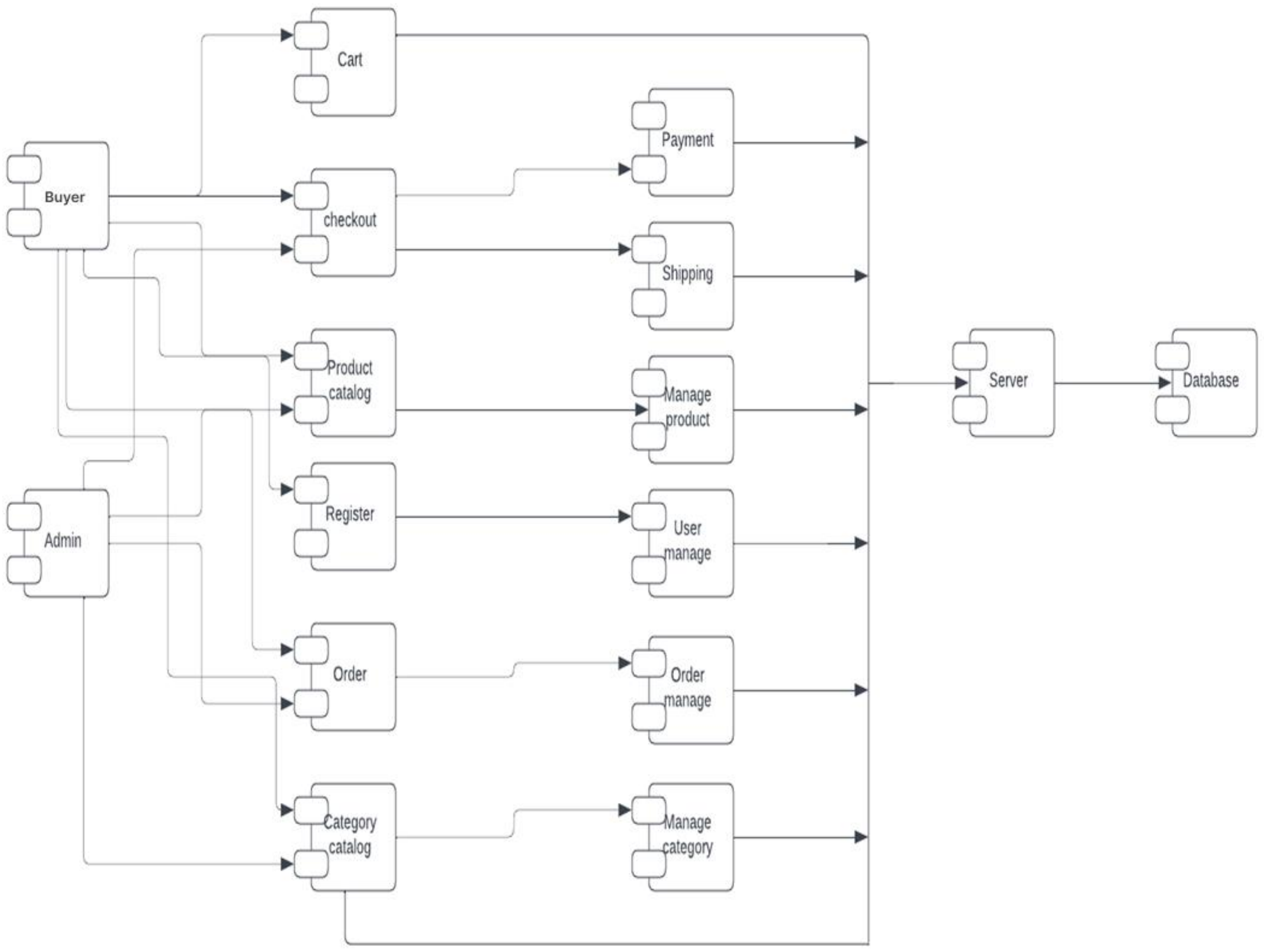


Figure 5.4 Component diagram

5.6 Deployment diagram

A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them. Deployment diagrams are typically used to visualize the physical hardware and software of a system. [8]

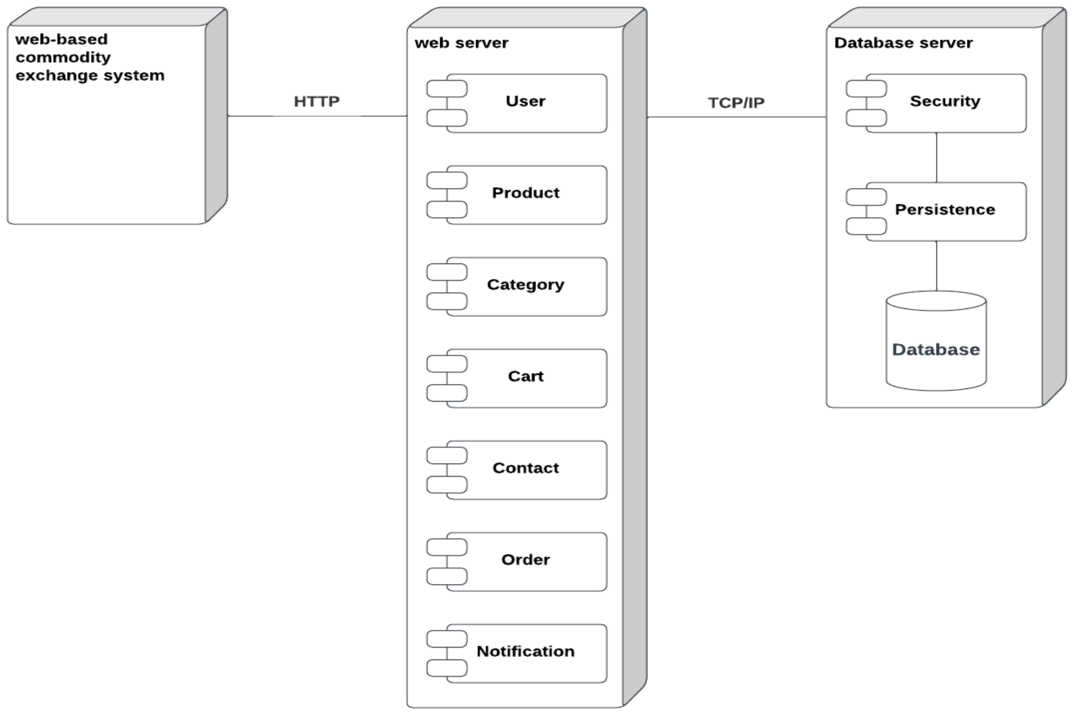


Figure 5.5 Deployment diagram

5.7 User interface

5.7.1 User interface flow diagram

User interface flow diagram enable us to model the high level relationships between major user interface elements. The diagram below will illustrate the major interface flow diagram in web based commodity exchange system.

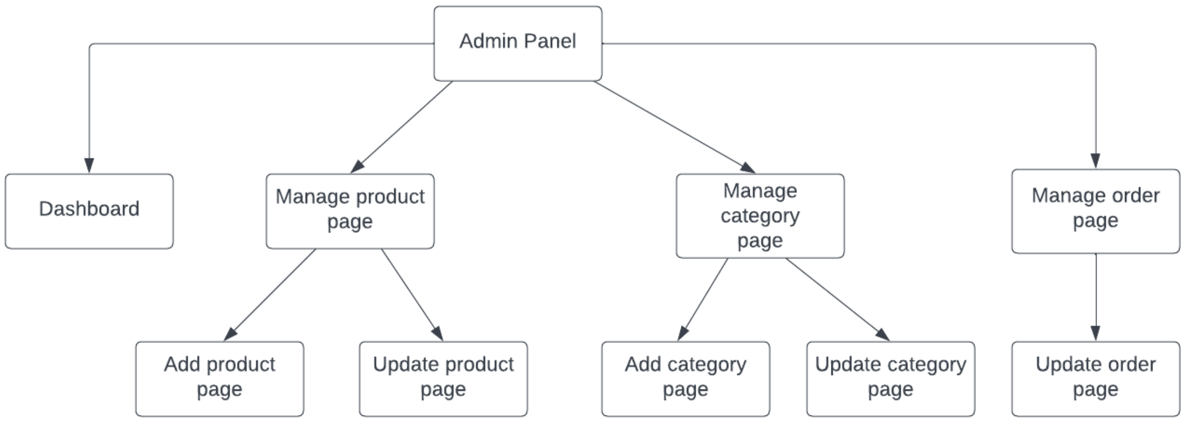


Figure 5.6 User interface flow diagram for buyer

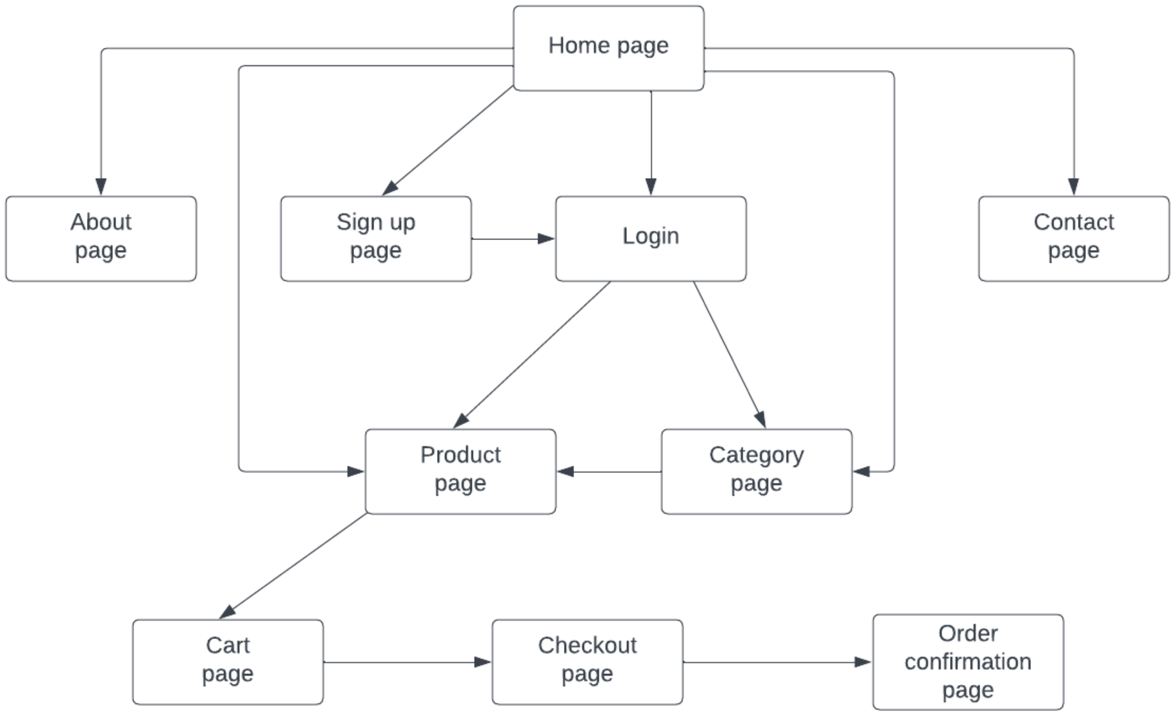


Figure 5.7 User interface flow diagram for admin

5.7.2 User interface design (selected samples)

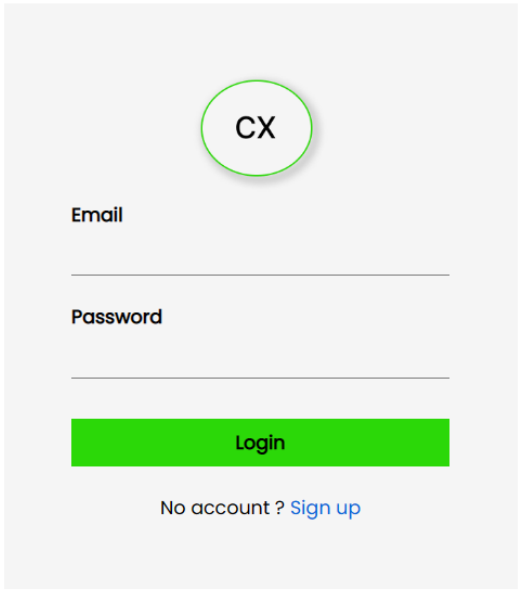


Figure 5.8 Login page

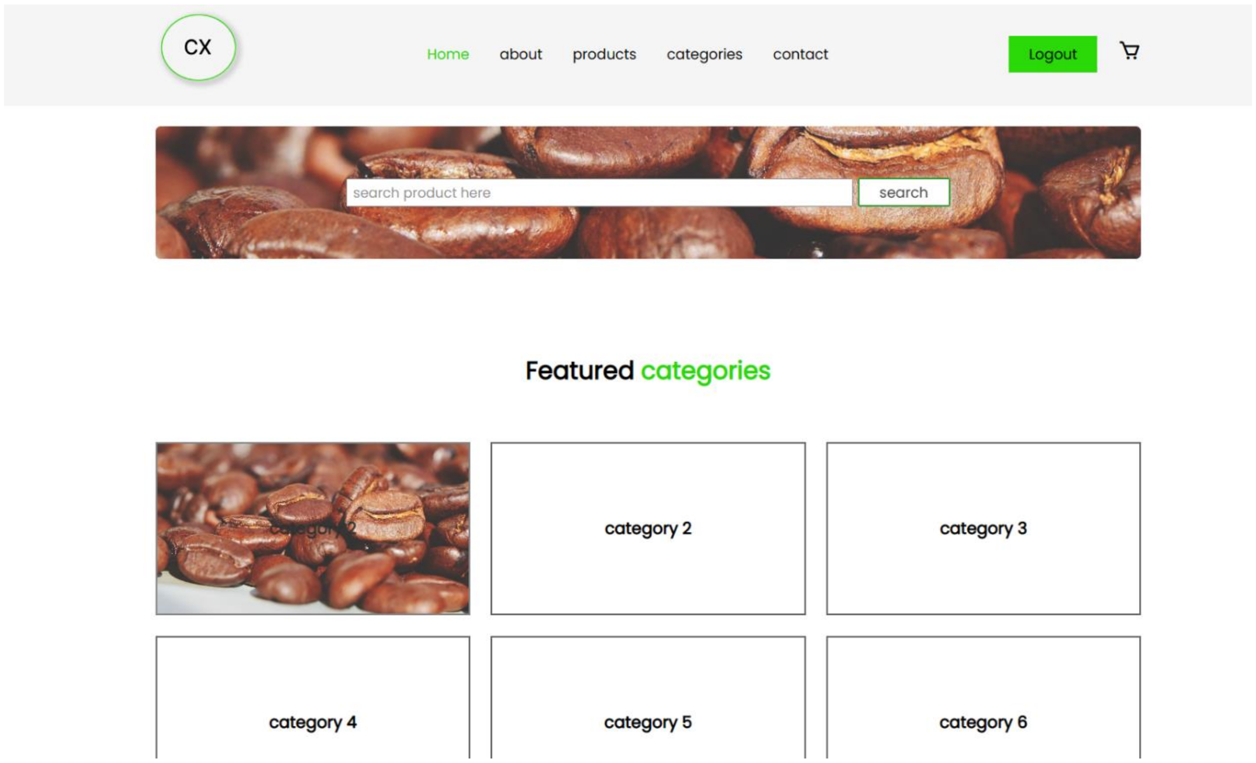


Figure 5.9 home page

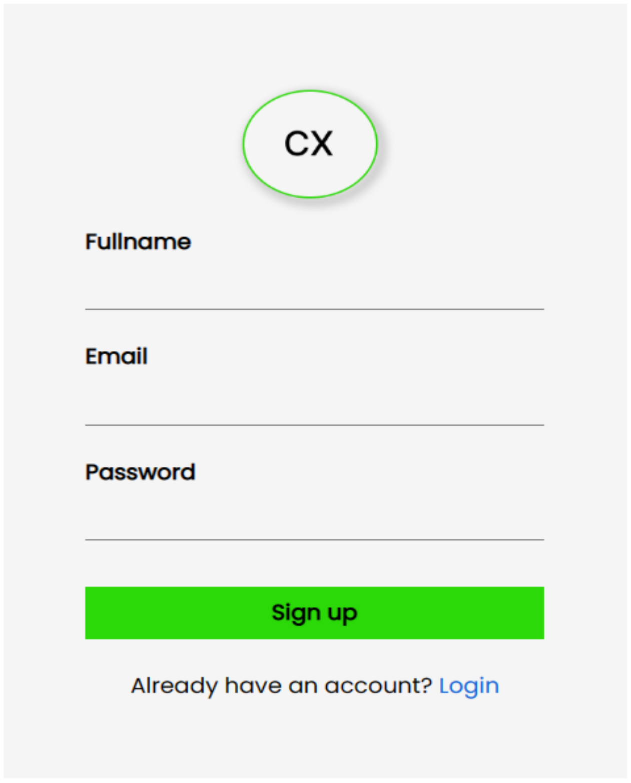
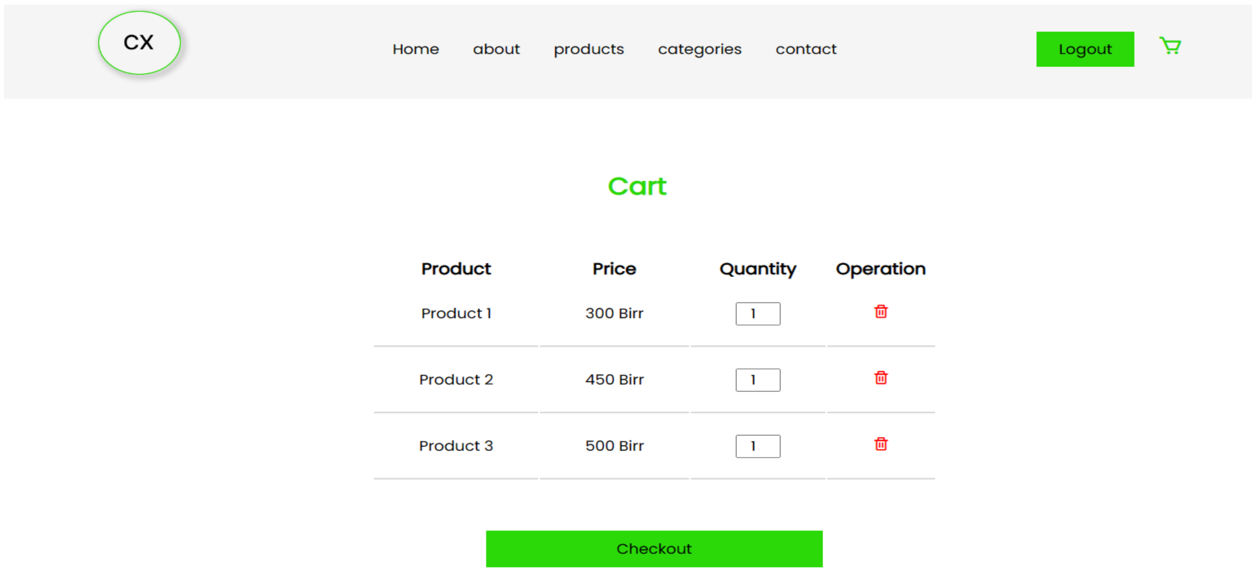
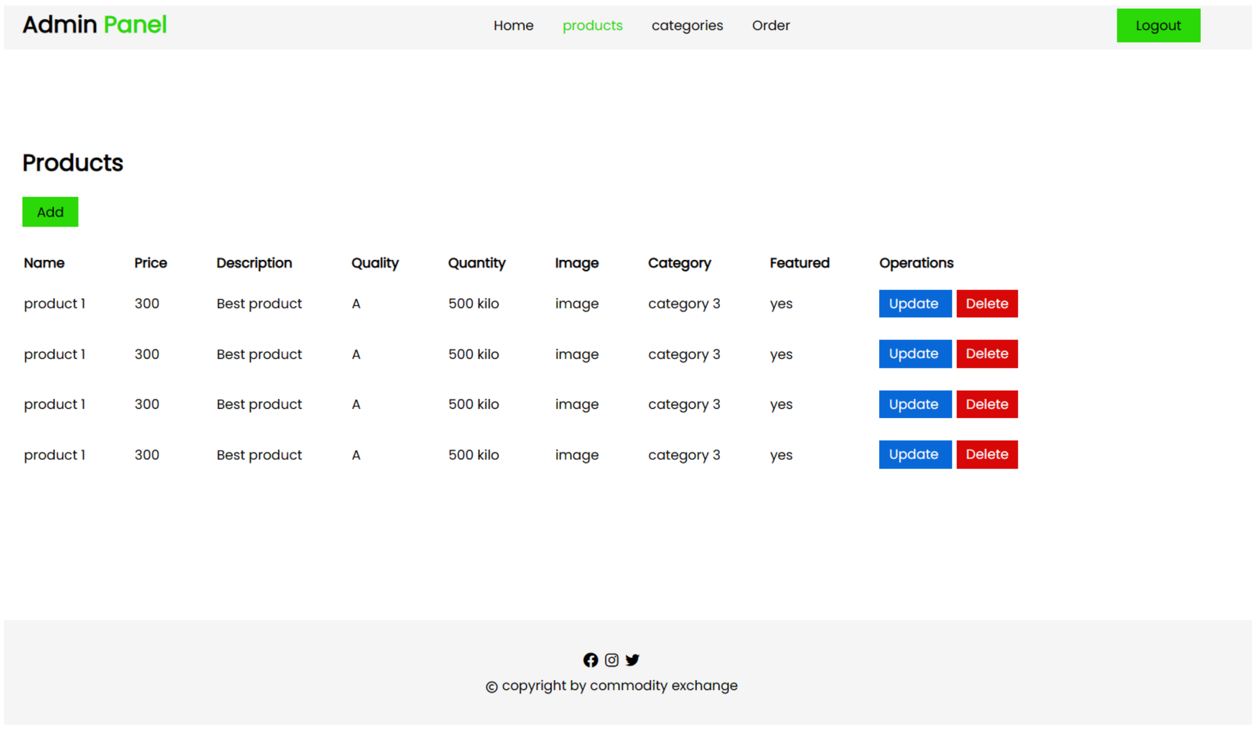


Figure 5.10 sign up page



Product	Price	Quantity	Operation
Product 1	300 Birr	1	
Product 2	450 Birr	1	
Product 3	500 Birr	1	

Figure 5.11 cart page


Admin Panel

Home products categories Order Logout

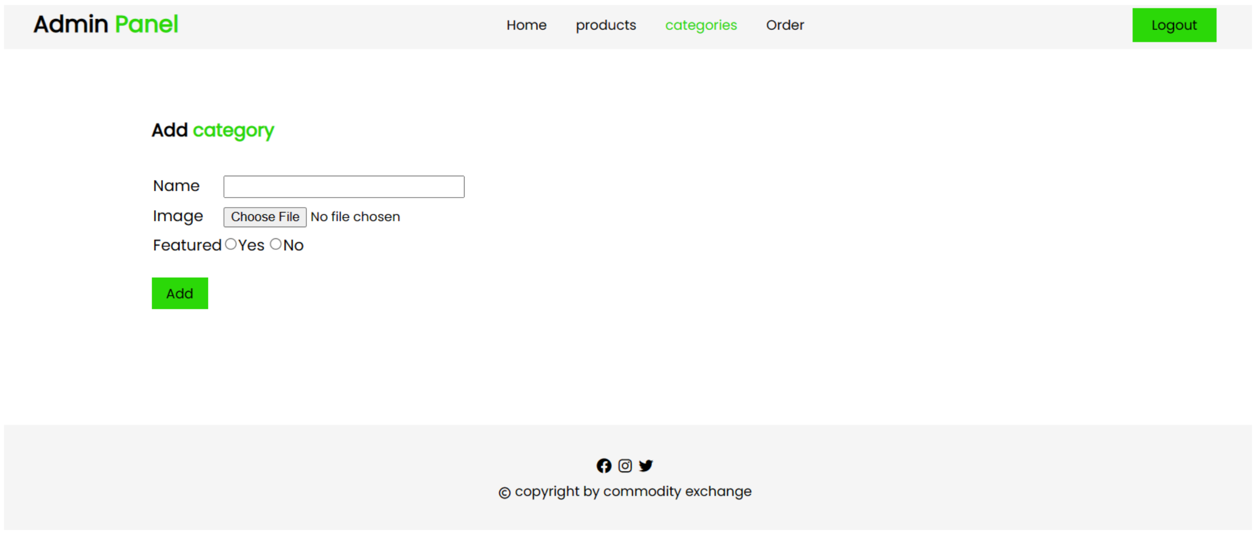
Products

Add

Name	Price	Description	Quality	Quantity	Image	Category	Featured	Operations
product 1	300	Best product	A	500 kilo	image	category 3	yes	Update Delete
product 1	300	Best product	A	500 kilo	image	category 3	yes	Update Delete
product 1	300	Best product	A	500 kilo	image	category 3	yes	Update Delete
product 1	300	Best product	A	500 kilo	image	category 3	yes	Update Delete

© copyright by commodity exchange

Figure 5.12 manage product page


Admin Panel

Home products categories Order Logout

Add category

Name

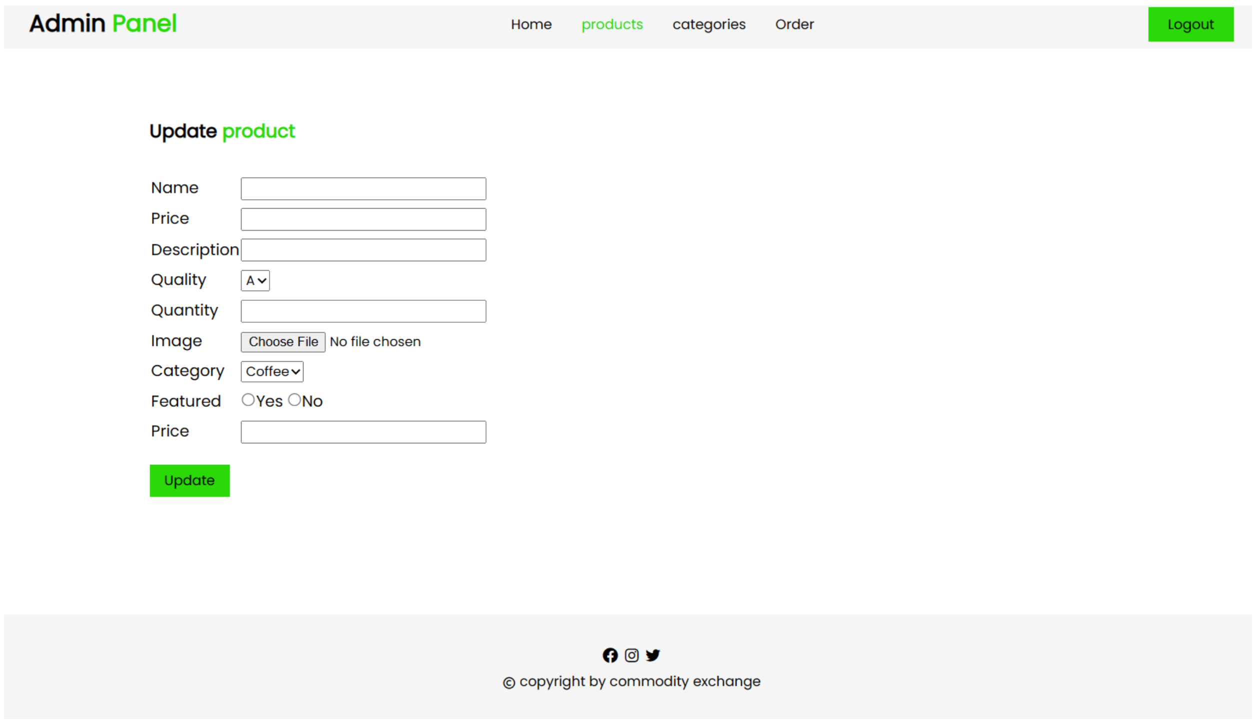
Image Choose File No file chosen

Featured ○Yes ○No

Add

© copyright by commodity exchange

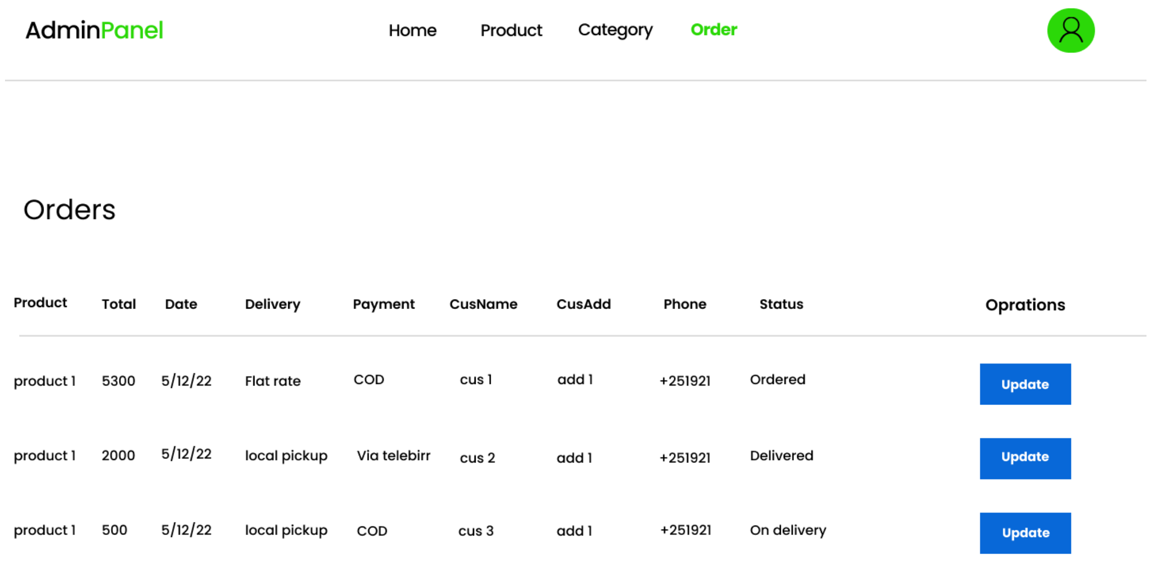
Figure 5.13 add category page



Update product

Name	
Price	
Description	
Quality	A
Quantity	
Image	Choose File No file chosen
Category	Coffee
Featured	○Yes ○No
Price	

Figure 5.14 update product page



Orders

Product	Total	Date	Delivery	Payment	CusName	CusAdd	Phone	Status	Oprations
product 1	5300	5/12/22	Flat rate	COD	cus 1	add 1	+251921	Ordered	Update
product 1	2000	5/12/22	local pickup	Via telebirr	cus 2	add 1	+251921	Delivered	Update
product 1	500	5/12/22	local pickup	COD	cus 3	add 1	+251921	On delivery	Update

Figure 5.15 manage order page

Chapter six

Object oriented implementation

6.1 Introduction

After systems are proposed they need to be implemented and tested afterwards before they are open to the end users. Implementation in the system includes implementing the attributes and methods of each object and integrating all the objects in the system to function as a single system. The implementation activity spans the gap between the detailed objects designed model and a complete of source code file that can be compiled together. [9]

After we the developers of the system described it intensively, we have moved to the next phase which is implementation and testing. In this chapter we discuss major implementation criteria of the system which include technologies used, testing, compatibility issue, testing procedures and deployment processes.

6.2 Testing and testing procedures

Software testing is defined as an activity to check whether the actual results match the expected results and to ensure that the software system is defect free. It involves execution of a software component or system component to evaluate one or more properties of interest. [10]

6.2.1 Unit Testing

Unit testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.

6.2.2 Integration Testing

Integration Testing is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated.

6.2.3 System Testing

System Testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems. System Testing is defined as a series of different tests whose sole purpose is to exercise the full computer-based system.

System testing samples

Table 6.1 Login Test

A : Login	
Test case identifier	Login
Test location	Login page
Feature to be tested	Authentication and authorization of the user
Data	Email and password
Test Cases	• If the system accepts a validated user input. • If the system access the information of the desired user account. • If the system validated the data entered with respect to the user account by comparing hashed password and phone number entered. • If the system takes the user to the dashboard that is authorized for that person only

Test data	• semira@gmail.com | password • ab@gmail.com | gsdg@$fnd
Result	Log user to the dashboard

Table 6.2 Signup Test

B : Sign up	
Test case identifier	Sign up
Test location	Sign up page
Feature to be tested	Creating a user account
Data	Information about user account
Test Cases	• If the system accepts a validated data from the user. • If the system creates a new account for the user based on their preference. • If the system stores the information in the desired format for example passwords should be hashed. • If the system takes the user to the dashboard that is authorized for that person only
Test data	• Kedir seid | kedir@gmail.com | password • Leul kebede| Leul@gmail.com | gsdg@$fnd
Result	Log user to the dashboard

Table 6.3 Add product Test

C : Add product	
Test case identifier	Add product
Test location	Add product page
Feature to be tested	Adding a new product

Data	productName,productDescription,productQuantity,product quality,productImage,productFeatured
Test Cases	• If the system accepts a validated input from the user. • If the system adds the new product to the system. • If the system validated the data entered with respect to the user account by comparing hashed password and phone number entered. • If the system takes the user to manage product page after a new product is added
Test data	• Yirga chefe Coffee,best coffee in the town,300 kilo,A,yes • Red bean, Nice bean, 200kilo , B , No
Result	New product is added to the system

Table 6.4 Update category Test

D : update category	
Test case identifier	Update category
Test location	Update category page
Feature to be tested	Updating existing category
Data	categoryName, categoryImage , category featured
Test Cases	• If the system accepts a validated input from the user. • If the system updates the desired category • If the system saves the changes that were made. • If the system takes the user to manage category page after the product is updated
Test data	• Sesame , Yes • Legumes , No
Result	Category is updated

Table 6.5 Order management

D : Order management	
Test case identifier	Order management
Test location	Manage order page and update order page
Feature to be tested	Viewing order and updating order
Data	Information about the order
Test Cases	• If the system accepts a validated data from the user. • If the system stores the order information based on the order identifier • If the system shows the order starting from recent order • If the system updates the desired field of the order
Test data	• Coffee, kaleb , 1000birr , 15 kilo , 12/06/2023,
Result	Order is managed

Table 6.6 Logout test

E : Log out	
Test case identifier	Logout
Test location	All user type dashboard
Feature to be tested	Logging out user
Data	User token from session
Test Cases	• If the system correctly deletes the token from session. • If the system doesn't log back in after reload
Test data	• kedir@gmail.com • Leul@gmail.com
Result	Logs out the user

6.3 deployment / installation process

The deployment process for a software application involves the steps and procedures for releasing the application into a production environment. The deployment process can vary depending on the complexity of the application, the deployment environment, and the requirements of the organization.

Our hosting provider: **hostinger.com**

The deployment and installation process for our web based system looks like this: First, we ensure that our codebase is prepared with all the necessary HTML, CSS, JS, and Laravel files. Next, sign up for a hosting account on hostinger.com and configure our domain to point to the hosting account.

Once we have access to the hostinger.com control panel, upload our codebase using the file manager or file upload feature. After uploading the files, we create a database in the hostinger.com control panel. Then in the hostinger.com control panel's terminal or command line interface, we run the commands "composer install" and "npm install" to install the necessary Laravel dependencies.

Next, we update the Laravel environment file (usually named ".env") with the database credentials we obtained earlier. Then we configure required settings, such as cache and session drivers, based on our system requirements. With the database and environment configured, we run the Laravel migration commands to create the required database tables.

Finally, we test our deployment by accessing our website using our domain name. Ensure that all functionalities, including user authentication, data retrieval, and transactional processes, are working correctly.

Chapter seven

Conclusion and recommendations

7.1 Conclusion

In developing this system, we the developers have applied different software development methodologies and implemented the system in order to provide the advantages we believed to solve different problems the current trade system is facing.

While developing the system the development team has also found out that clear documentation and communication is key for the success of any kind of software project. We also gathered and analyzed different information from different people who participate in the current trade system which helped us a lot in developing the system by making us understand the problems that exist and shade light on the solutions that will best fit the individual problem.

While implementing this system the development team have used different technologies like HTML, CSS, JS and Laravel and also used different third party libraries like Jetstream and APIs like chapa. This experience has helped all of us acquire knowledge and practices in making a full stack applications.

In general, even if there are some limitations on the system and we face a lot of challenges we tried to develop the system which can satisfy the need of the sellers and buyers in the trade system.

7.2 Recommendation

The development team have implemented web based system that have different features like payment system capabilities that different parties in the trade industry can use and utilize the features to their needs. Most importantly our system is fully responsive on most browsers which helps users to use it on devices they are mostly comfortable in using including their phones, tablets, laptops and desktop computers.

We recommend that users who want to interact to our system should have some computer knowledge, basic experience with web based applications, basic skill on how to use browser application like google chrome. Organizations which want the system should also provide a good performance machines to make the system work smoothly.

And we also recommend that organizations interested in implementing the system consider integrating it with shipping functionality, as we were unable to locate an API that provides this service.

Appendix

Sample code

- Admin home page code

```
<! DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Home - admin</title>
   <link rel="stylesheet" href="home/css/style.css">
   <link href='https://unpkg.com/boxicons@2.1.4/css/boxicons.min.css' rel='stylesheet'>
</head>
<body>

<!-- header section-->
<header style="padding: 5px 10%;">
   <h2>Admin <span>Panel</span> </h2>
   <ul>
      <li><a style="color:#2BD808" href="{{ route('adminhome') }}">Home</a></li>
      <li><a href="{{ route('product') }}">products</a></li>
      <li><a href="{{ route('category') }}">categories</a></li>
      <li><a href="admin-order-manage.html">Order</a></li>
   </ul>
   <div class="logout-cart">
      <div class="">
         <x-app-layout>
         </x-app-layout>
      </div>
```

```
    </div>
</header>

<!-- Dashboard section -->
<section class="dashboard">
  <div class="dashboard-container">
    <div class="content-box">
        <h1><span>35</span> </h1>
        <p>products</p>
    </div>

    <div class="content-box">
        <h1><span>10</span> </h1>
        <p>Categories</p>
    </div>

    <div class="content-box">
        <h1><span>15</span> </h1>
        <p>Orders</p>
    </div>

    <div class="content-box">
        <h1><span>55K birr</span> </h1>
        <p>Revenue generated</p>
    </div>
  </div>
</section>
<!-- footer section -->
<footer>
    <div class="social">
        <i class='bx bxl-facebook-circle'></i>
        <i class='bx bxl-instagram' ></i>
        <i class='bx bxl-twitter' ></i>
    </div>
    <i class='bx bx-copyright'></i> copyright by commodity exchange
</footer>
</body>
</html>
```

- Product controller code

```
<? php
namespace App\Http\Controllers;
use App\Models\Product;
use Illuminate\Http\Request;
class ProductController extends Controller
{
    /* Display a listing of the resource.      */
    public function index()
    {
        return view('admin.product',[
            'products'=>Product::orderBy('updated_at','asc')->get()
        ]);
    }
   /* Show the form for creating a new resource.*/
    public function create()
    {
        return view('admin.createProduct');
    }
    /*  Store a newly created resource in storage. */
    public function store(Request $request)
    {
        $request->validate([
    {

            'name'=>'required|unique:categories',
            'description' =>'required',
            'quantity'=>'required',
            'quality'=>'required',
            'image'=>['required','mimes:jpg,png,jpeg'],
            'price'=>'required'
        ]);
        Product::create([
            'name'=>$request->name,
            'description'=>$request->description,
            'quantity'=>$request->quantity,
            'quality'=>$request->quality,
            'image'=>$this->storeImage($request),
            'featured'=>$request->featured==='yes',
            'price'=>$request->price
        ]);

        return redirect(route('product'))->with('message','Product has been created Successfully');
    }
    /
     * Display the specified resource.
     */
    public function show(string $id)
    {
        return view('admin.product',[
            'product'=>Product::findOrFail($id)
        ]);
    }
    /*  Show the form for editing the specified resource. */
    public function edit(string $id)
    {
        return view('admin.updateProduct',
        [
            'product'=>Product::where('id', $id)->first()
        ]
    );
    }
    /* Update the specified resource in storage. */
    public function update(Request $request, string $id)
    {
        $request->validate([
            'name'=>'required|unique:categories,name,'.$id,
            'image'=>['mimes:jpg,png,jpeg']
        ]);
        Product::where('id', $id)->update($request->except(['_method', '_token']));
        return redirect(route('product'))->with('message','Category has been updated');
    }
    /**
     * Remove the specified resource from storage.
     */
    public function destroy(string $id)
    {
       Product::destroy($id);
        return redirect(route('product'))->with('message','Category has been deleted');
    }

    private function storeImage($request){
        $newImageName = uniqid().'-'.$request->name.'.'.$request->image->extension();
        return $request->image->move(public_path('images'),$newImageName);
    }
}
```

- Product model code

```
<? php
namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
class Product extends Model
{
    use HasFactory;
    protected $fillable = [
        'name',
        'description',
        'quantity',
        'quality',
        'image',
        'featured',
        'price'
    ];
}
```

References

[1] Berner-mayer. (1997).*Object-oriented design and methodologies 2nd edition*.[Accessed on 5th may 2023]

[2] InterviewBit.(2022). *System Architecture – Detailed Explanation*.[online][Accessed on 5th may 2023] https://www.interviewbit.com/blog/system-architecture/

[3] GeeksforGeeks.(2022). *MVC framework introduction[*online][Accessed on 7th may 2023] MVC Framework Introduction - GeeksforGeeks

[4] intercom(2022).*Six principles of system design*.[online][Accessed on 8th may 2023] https://www.intercom.com/blog/six-principles-of-system-design/

[5] VisualParadigm.(No date).*What is Class Diagram*.[online][Accessed on 8th may 2023] https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

[6] Ambler S. W. (2001). *Relational persistent model.* [Accessed on 10th may2023]

[7] SmartDraw. (No date). *component diagram.* [online] [Accessed on 12th may 2023] https://www.smartdraw.com/component-diagram/

[8] Creately.(2022).*The Easy Guide to UML Deployment Diagrams*.[online][Accessed on 13th may 2023]https://creately.com/guides/deployment-diagram-tutorial/

[9] Ambler S. W. (2001). *Object-oriented implementation.* [Accessed on 15th may2023]

[10] Guru.(2021).*Testing Guide*.[online][Accessed on 15th may 2023] https://www.guru99.com/testing-guide.html