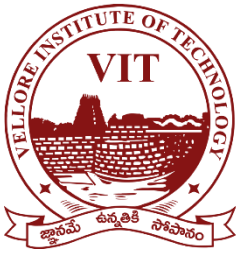


VIT-AP UNIVERSITY

CSE 1005 – Software Engineering

Report Title

Bellezza - Online Shopping Platform

Guided by

Dr. Hussain Syed

Department

School of Computer Science and Engineering

Submitted By

PABBISETTY PRANAVI	**21BCE9459**
KOTHA LAVANYA	**21BCE9452**
B TANU SREE	**21BCE9938**
AMASALA LIKITHA	**21BCE9931**

May 2024

1. Introduction

Brief overview of Bellezza:

Bellezza is an online shopping platform designed to provide a seamless experience for both sellers and customers. Built using Python Flask along with HTML, CSS, and JavaScript, Bellezza offers a user-friendly interface for users to browse, buy, and sell products. The platform aims to bridge the gap between sellers and buyers by providing a secure and efficient marketplace for transactions.

Purpose of the report:

The purpose of this report is to provide a comprehensive overview of the Bellezza project. It outlines the objectives, functionalities, and implementation details of the platform. Additionally, the report discusses the system architecture, user interface design, testing procedures, and potential future enhancements. This report serves as a documentation and reflection of the development process, highlighting key aspects of the project for stakeholders and developers alike.

2. Project Overview

Description of the project:

The project aims to develop Bellezza, an online shopping platform where sellers can list their products for sale, and customers can browse and purchase those products. Bellezza provides a user-friendly interface for both sellers and customers to interact, facilitating seamless transactions. Sellers can register, add products, manage their profiles, and track sales, while customers can register, browse products, add items to their cart, and make purchases. The platform prioritizes security, usability, and efficiency to ensure a positive experience for all users.

Objectives and goals:

- Create a robust online shopping platform that caters to the needs of both sellers and customers.
- Enable sellers to easily list their products, manage their inventory, and track sales.

- Provide customers with a wide range of products to choose from, along with a smooth shopping experience.
- Implement secure authentication and authorization mechanisms to protect user data and transactions.
- Ensure scalability and maintainability of the platform for future growth and updates.

Technologies used:

The project utilizes a combination of technologies to achieve its goals:

- Python Flask: Used as the backend framework to handle server-side logic and routing.
- HTML: Utilized for creating the structure and layout of web pages.
- CSS: Employed for styling and formatting the appearance of the web pages.
- JavaScript: Implemented for client-side interactions and dynamic content.
- Flask-Session: Integrated for managing user sessions securely.
- SQLite: Employed for storing user data, product information, and transaction records.

These technologies work together to create a fully functional online shopping platform that meets the needs of both sellers and customers, providing a seamless and secure experience for all users.

3. System Architecture

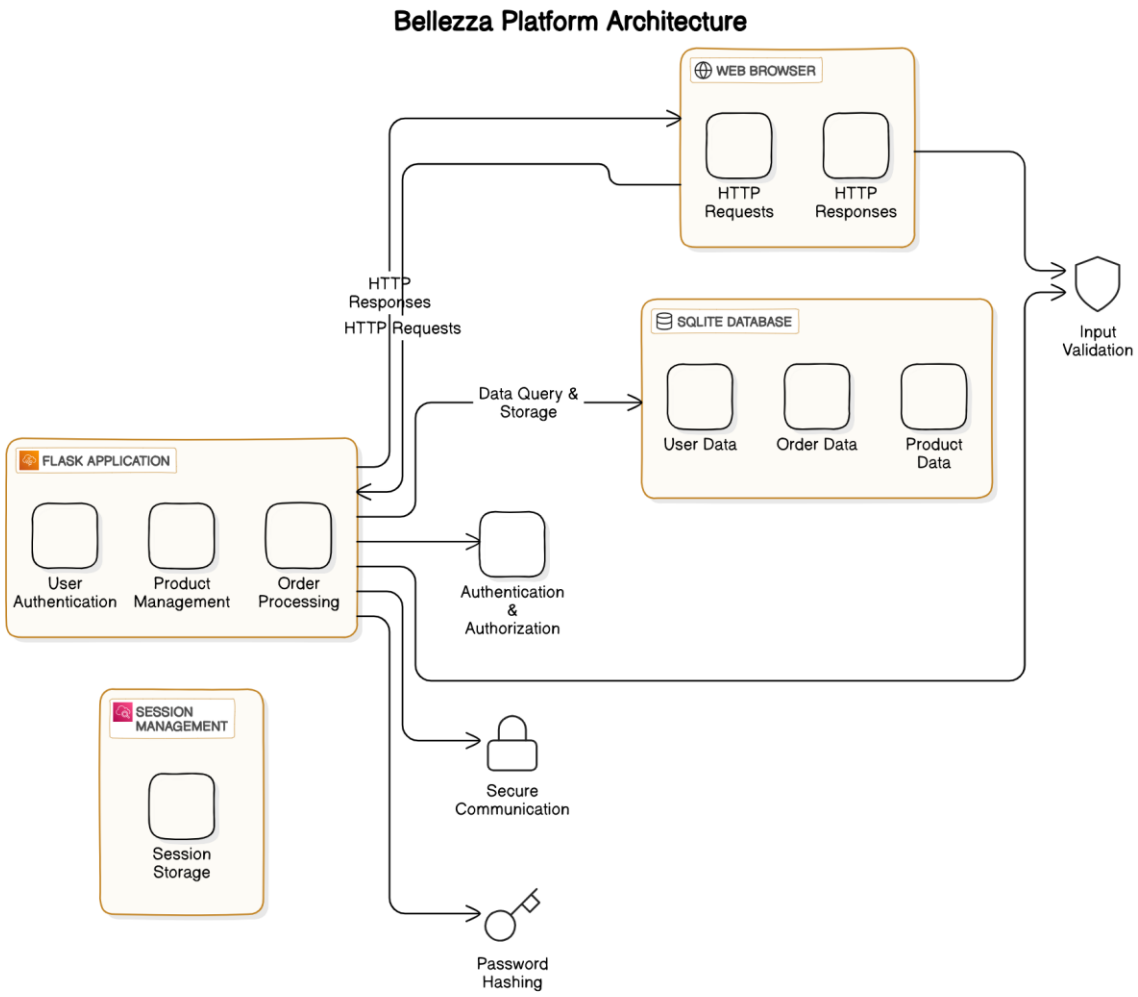
Overview of the system architecture:

The Bellezza online shopping platform follows a typical client-server architecture, where clients (web browsers) interact with a server to access and manipulate data. The architecture is divided into frontend and backend components, each serving specific purposes:

Frontend: The frontend of Bellezza consists of HTML, CSS, and JavaScript code that runs in the client's web browser. It renders the user interface and handles client-side interactions such as form submissions, user input validation, and dynamic content updates.

Backend: The backend of Bellezza is built using Python Flask, serving as the server-side application. It handles incoming requests from clients, processes data, interacts with the

database, and generates dynamic web pages. The backend implements business logic, user authentication, authorization, and session management.



Client-server interaction:

The interaction between clients and the server follows the typical request-response model:

1. Client Sends Request: A client (web browser) sends an HTTP request to the server when a user performs an action, such as logging in, browsing products, adding items to the cart, or updating profile information.

2. Server Processes Request: The Flask server receives the HTTP request, routes it to the appropriate endpoint, and executes the corresponding Python function. This involves

processing data, querying the database, and performing necessary operations based on the request.

3. Server Sends Response: After processing the request, the server generates an HTTP response, which includes the requested data or a rendered HTML page. The response is sent back to the client's web browser.

4. Client Receives Response: The client's web browser receives the HTTP response and renders the content to the user. This may involve displaying product listings, profile information, shopping cart contents, or confirmation messages.

This client-server interaction enables users to interact with Bellezza's features and functionalities seamlessly, with the server handling the underlying logic and data management.

Database schema:

The database schema for Bellezza includes tables to store various types of data, such as user information, product details, orders, and session data (if using Flask-Session). While the specific schema may vary based on the project's requirements and design, a typical schema may include the following tables:

- customer
- metadata
- product
- seller
- orders
- cart

These are handled by python’s sqlite3

Table: customer

custID - varchar(10)

name - varchar(30)

email - varchar(30)

phone - varchar(13)

area - varchar(20)

locality - varchar(20)

city - varchar(20)

state - varchar(20)

country - varchar(20)

zipcode - varchar(6)

password - varchar(20)

Table: metadata

custnum - INTEGER

sellnum - INTEGER

prodnum - INTEGER

profit_rate - decimal(6,3)

ordernum - INTEGER

Table: product

prodID - varchar(10)

name - varchar(30)

quantity - INTEGER

category - varchar(30)

cost_price - decimal(9,2)

sell_price - decimal(9,2)

description - varchar(100)

sellID - varchar(10)

Table: seller

sellID - varchar(10)

name - varchar(30)

email - varchar(30)

phone - varchar(13)

area - varchar(20)

locality - varchar(20)

city - varchar(20)

state - varchar(20)

country - varchar(20)

zipcode - varchar(6)

password - varchar(20)

Table: orders

orderID - varchar(10)

custID

prodID

quantity - INTEGER

date - datetime

cost_price - decimal(9,2)

sell_price - decimal(9,2)

status - varchar(15)

Table: cart

custID - varchar(10)

prodID - varchar(10)

quantity - INTEGER

app.py abc.py

abc.py > ...

```python
import sqlite3
conn = sqlite3.connect('onlineshop.db')
cursor = conn.cursor()
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
tables = cursor.fetchall()
for table in tables:
    print(table[0])
conn.close()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE CODE REFERENCE LOG

```
Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine for com
ort-Module PSReadLine'.

PS C:\Users\PABBISETTY PRANAVI\OneDrive\Desktop\SEM 6\SWE\SWE Lab\OnlineShop-master\OnlineShop> py abc.p
customer
metadata
product
seller
orders
cart
PS C:\Users\PABBISETTY PRANAVI\OneDrive\Desktop\SEM 6\SWE\SWE Lab\OnlineShop-master\OnlineShop>
```

4. Functionalities

1. User Registration and Authentication:

Description: Users can register for a new account by providing their details such as username, email, and password. Upon registration, the system securely stores the user's credentials in the database. Registered users can then authenticate themselves by logging in with their username and password.

Implementation: The Flask application provides endpoints for user registration and login. Passwords are securely hashed before storing them in the database. Authentication is performed by verifying the hashed password during login.

2. User Login/Logout:

Description: Registered users can log in to their accounts using their username and password. Upon successful login, users gain access to their personalized dashboard and functionalities. Users can log out to terminate their session securely.

Implementation: Flask provides routes for user login and logout. Sessions are managed securely to maintain user authentication across requests. When a user logs out, their session is invalidated, and they are redirected to the homepage.

3. Profile Management (Viewing, Editing):

Description: Users can view and edit their profile information, including personal details, contact information, and address. They have the ability to update their profile picture, change passwords, and modify other account settings.

Implementation: Flask routes are implemented to render profile pages for viewing and editing. Forms are provided for users to update their information securely. Input validation ensures data integrity and prevents unauthorized changes.

4. Product Management (Adding, Viewing, Editing):

Description: Sellers can add new products to the platform by providing details such as product name, description, price, quantity, and images. Users can browse and view available products, including product details and images. Sellers have the ability to edit existing product listings to update information or make changes.

Implementation: Flask routes handle product management functionalities, including adding, viewing, and editing products. Forms are provided for sellers to input product details. Images are uploaded securely and stored in a designated folder. Product information is retrieved from the database and displayed dynamically on product pages.

5. Shopping Cart Management:

Description: Users can add products to their shopping cart while browsing the platform. They can view the contents of their cart, update quantities, and remove items as needed. Users can proceed to checkout to complete their purchases.

Implementation: Flask routes manage shopping cart functionalities, including adding, viewing, and updating cart items. Session data is utilized to store cart information for authenticated users. Changes to the cart are reflected dynamically on the cart page.

6. Product Search and Purchase:

Description: Users can search for products using keywords or browse products by category. Search results are displayed based on user input. Users can view product details, including price, description, and seller information, and proceed to purchase products securely.

Implementation: Flask routes handle product search functionalities, querying the database based on user input. Search results are displayed dynamically on search result pages. Users can add products to their cart or proceed directly to purchase.

7. Order Management (Viewing, Cancelling, Dispatching):

Description: Users can view their order history, including details of past purchases and current order status. They have the ability to cancel pending orders before dispatch. Sellers can view and manage orders, mark orders as dispatched, and track order fulfilment.

Implementation: Flask routes manage order-related functionalities, including viewing order history, cancelling orders, and updating order status. Database queries retrieve order information based on user roles and permissions. Order status changes are reflected in real-time.

8. Sales Management (For Sellers):

Description: Sellers can view their sales history and track revenue generated from product sales. They have access to detailed sales reports, including product-wise sales, revenue trends, and customer insights.

Implementation: Flask routes provide sellers with access to sales-related functionalities, including viewing sales history, generating reports, and analyzing sales data. Database queries retrieve sales information and generate reports dynamically.

9. Error Handling and Security Measures:

Description: The system includes robust error handling mechanisms to handle exceptions gracefully and provide informative error messages to users. Security measures such as input

validation, password hashing, HTTPS encryption, and session management are implemented to protect user data and prevent unauthorized access.

Implementation: Flask routes include error handling logic to catch and handle exceptions. Input validation is performed on user input to prevent injection attacks and ensure data integrity. Passwords are securely hashed before storage. HTTPS protocol is utilized to encrypt data transmission between the client and server. Flask-Session is used for secure session management, with session data stored securely on the server side.

5. Implementation Details

Overview of the Flask Main Code:

The Flask main code serves as the backbone of the Bellezza online shopping platform, handling routing, request processing, database interactions, and session management. It consists of various routes and functions to implement different functionalities such as user authentication, product management, order processing, and more.

Explanation of Key Components and Functions:

1. Routes: Defined using `@app.route()` decorator, routes handle incoming HTTP requests and call corresponding functions to process data and generate responses.

2. Functions: Implemented to perform specific tasks such as user authentication (`login()`, `logout()`), profile management (`edit_profile()`), product management (`add_products()`, `view_product()`), order processing (`place_order()`, `cancel_order()`), and more.

3. Session Management: Utilizes Flask-Session for secure session management. Session data is stored securely on the server side, allowing users to remain authenticated across multiple requests without compromising sensitive information.

4. Database Interactions: Database interactions are managed using functions such as `add_user()`, `auth_user()`, `fetch_details()`, `add_prod()`, `get_seller_products()`, etc. These functions interact with the database to perform CRUD (Create, Read, Update, Delete) operations on user data, product information, orders, and more.

File Upload Handling:

- File upload handling is implemented using Flask's `request.files` object.
- The `upload_image()` route checks if a file is present in the request and verifies its filename and extension using the `allowed_file()` function.
- If the file is valid, it is saved to the designated upload folder (`UPLOAD_FOLDER`) using `secure_filename()` to prevent directory traversal attacks.

Session Management:

- Session management is implemented using Flask-Session extension.
- The `app.config['SECRET_KEY']` is set to a randomly generated value to securely sign session cookies.
- Sessions are initialized using `sess.init_app(app)`, allowing for session-based authentication and user state management across requests.

Database Interactions:

- Database interactions are managed using functions defined in `dbaccess.py`.
- Functions such as `add_user()`, `auth_user()`, `fetch_details()`, `add_prod()`, etc., interact with the SQLite database to perform CRUD operations on user data, product information, and orders.

HTML Files:

1. home.html:

Renders the landing page of the online shopping platform.

Displays featured products, login/signup options, and promotional content.

If a user is signed in, it may display personalized content such as user profile information.

2. login.html:

Provides a form for user login, including fields for username, password, and a submit button.

If login credentials are incorrect, it may display an error message.

3. signup.html:

Allows users to register for a new account by providing their details and submitting the form.

If registration is successful, it may redirect to a success page.

4. success_signup.html:

Displays a success message upon successful user registration.

5. profiles.html:

Renders a page for viewing user profiles.

Allows users to search for other users based on specific criteria (e.g., name, location).

6. view_profile.html:

Displays detailed information about a user's profile, including personal details, contact information, and address.

If the profile belongs to the currently signed-in user, it may include options for editing the profile.

7. edit_profile.html:

Provides a form for editing the user's profile information.

Pre-populates the form with the user's existing details for easy modification.

8. add_products.html:

Allows sellers to add new products to their inventory.

Provides a form for entering product details such as name, quantity, category, price, and description.

Supports uploading product images.

9. my_products.html:

Displays a list of products owned by the currently signed-in user (seller).

Allows sellers to search for their own products based on specific criteria.

10. view_product.html:

Shows detailed information about a specific product, including its name, quantity, category, price, description, and seller details.

If the user is a seller, it may include options for editing the product.

11. buy_product.html:

Displays information about a product available for purchase.

Allows customers to specify the quantity they wish to purchase.

12. buy_confirm.html:

Confirms the purchase of a product by displaying the selected items and total cost.

Provides options for placing the order or canceling the purchase.

13. search_products.html:

Allows customers to search for products based on specific criteria (e.g., category, keyword).

Displays search results and allows customers to view product details or make a purchase.

14. my_cart.html:

Displays the contents of the customer's shopping cart.

Allows customers to update the quantity of items or proceed to checkout.

15. my_orders.html:

Displays a list of orders placed by the currently signed-in customer.

Provides options for cancelling orders.

16. new_orders.html:

Displays a list of new orders received by the currently signed-in seller.

Provides options for dispatching orders.

17. my_purchases.html:

Displays a list of purchases made by the currently signed-in customer.

18. my_sales.html:

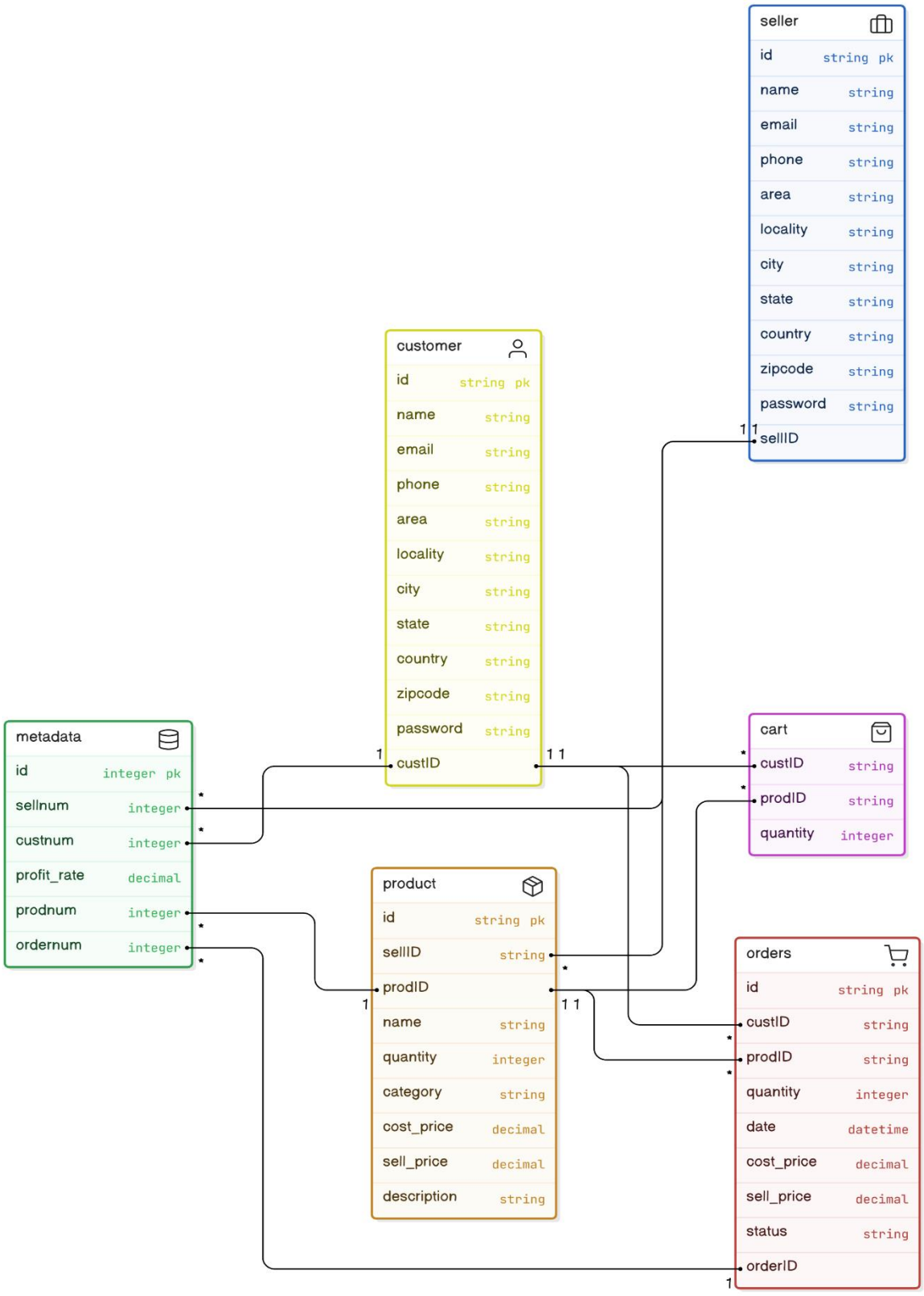
Displays a list of sales made by the currently signed-in seller.

HTML templates utilize Flask's template engine (Jinja2) to render dynamic content, including variables, loops, and conditionals. For example, product listings are dynamically generated based on data retrieved from the backend database.

6. UML Diagrams:

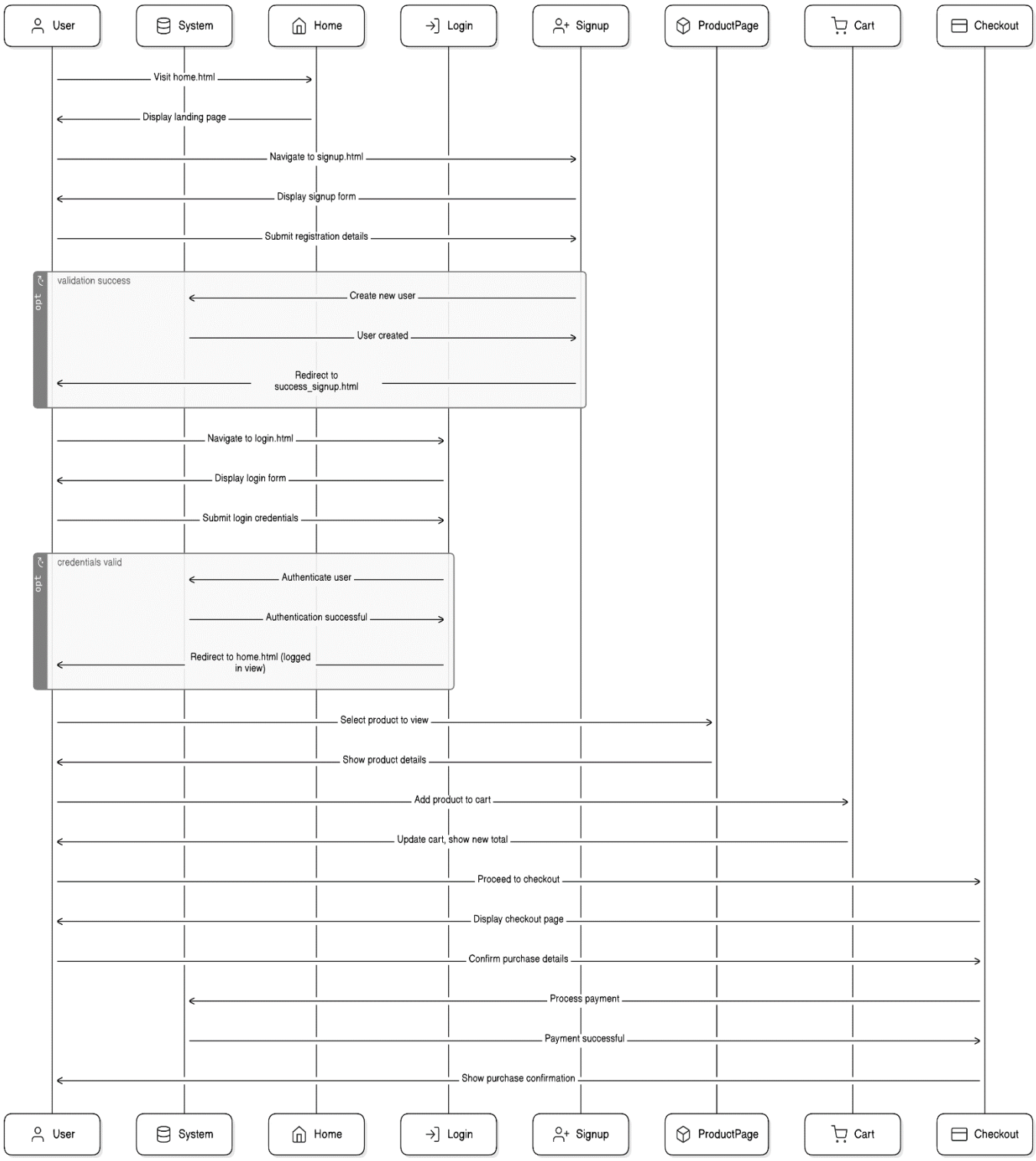
1. Use Case Diagram:

2. Class Diagram:

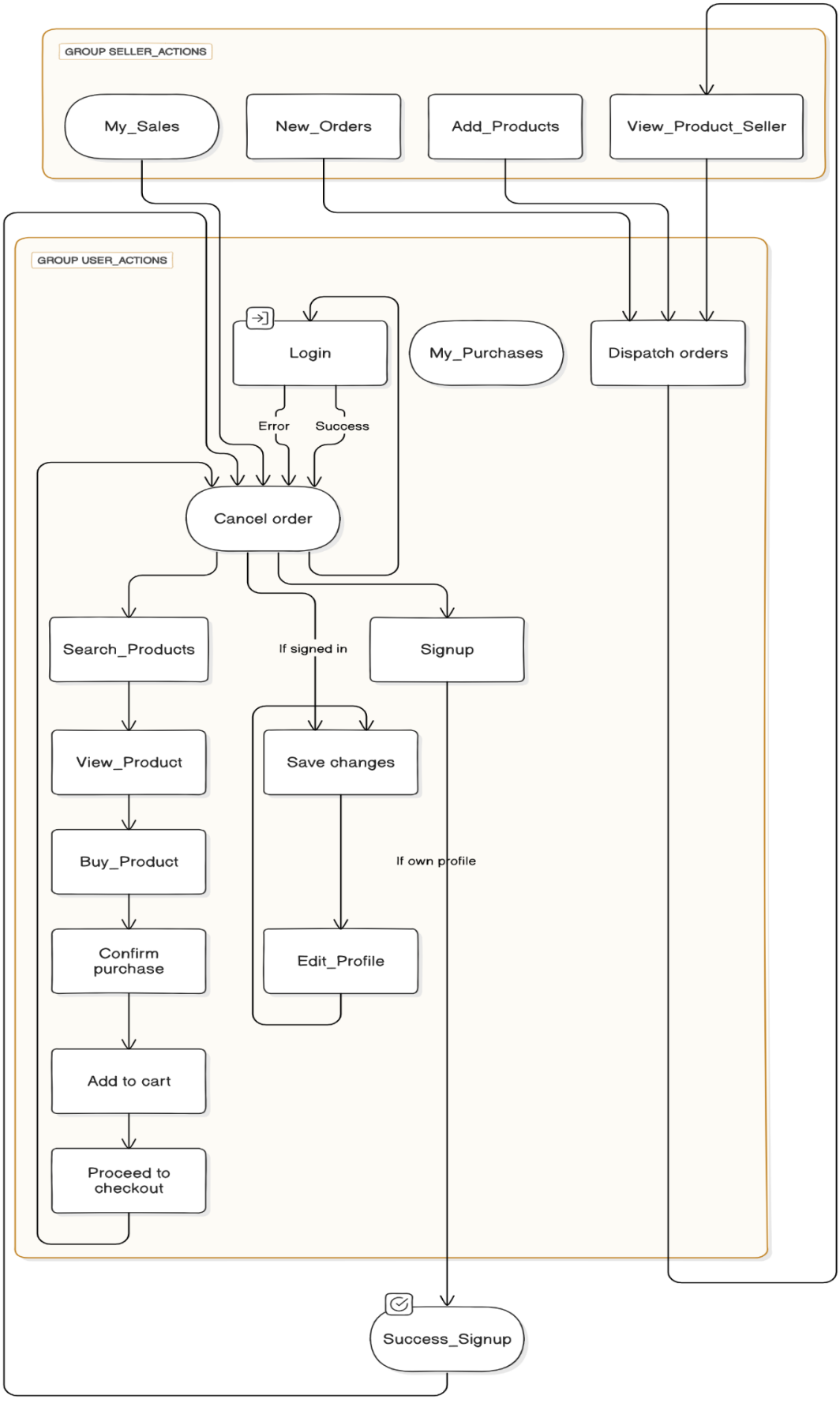


3. Sequential Diagram

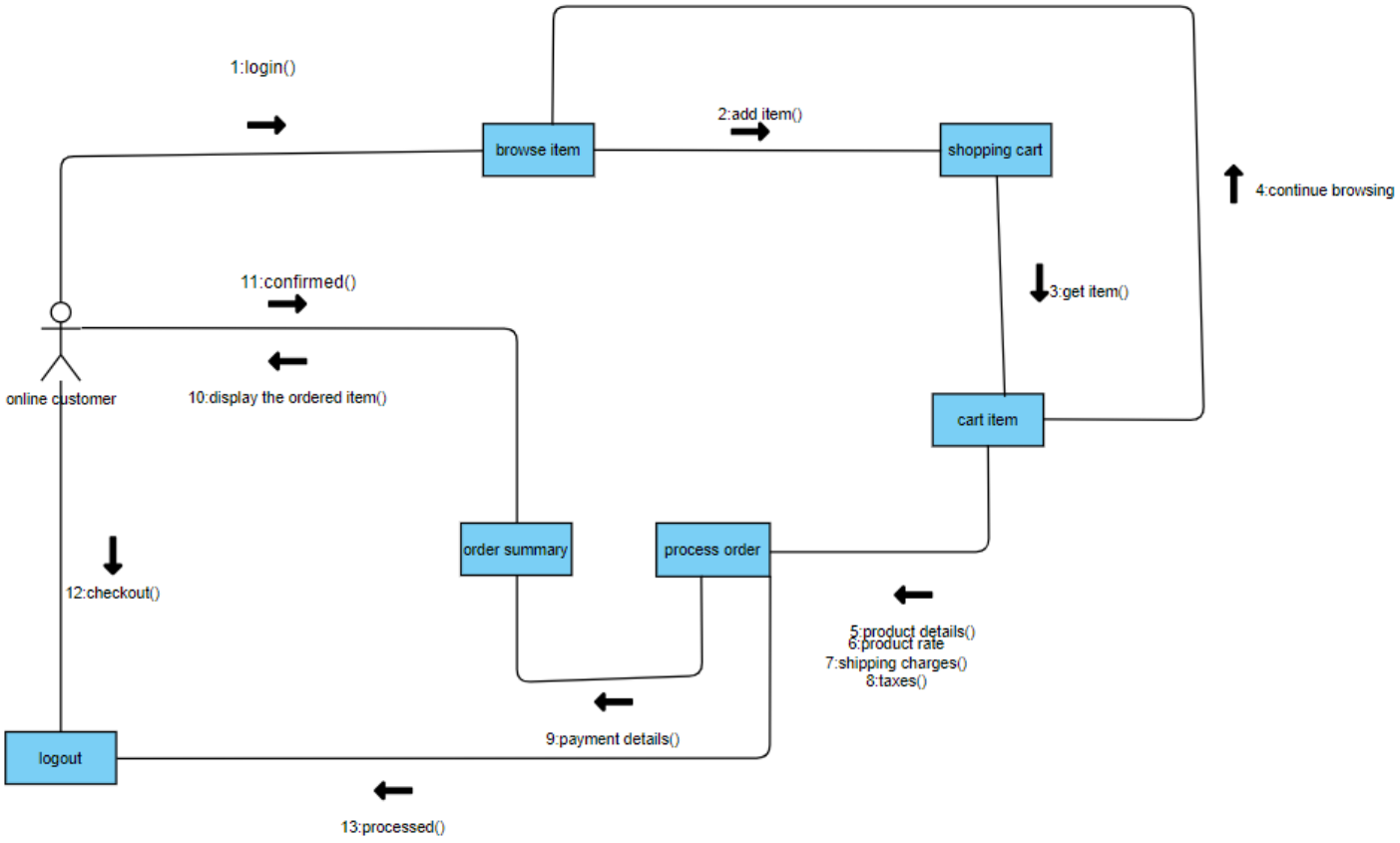
Online Shopping Platform Interaction



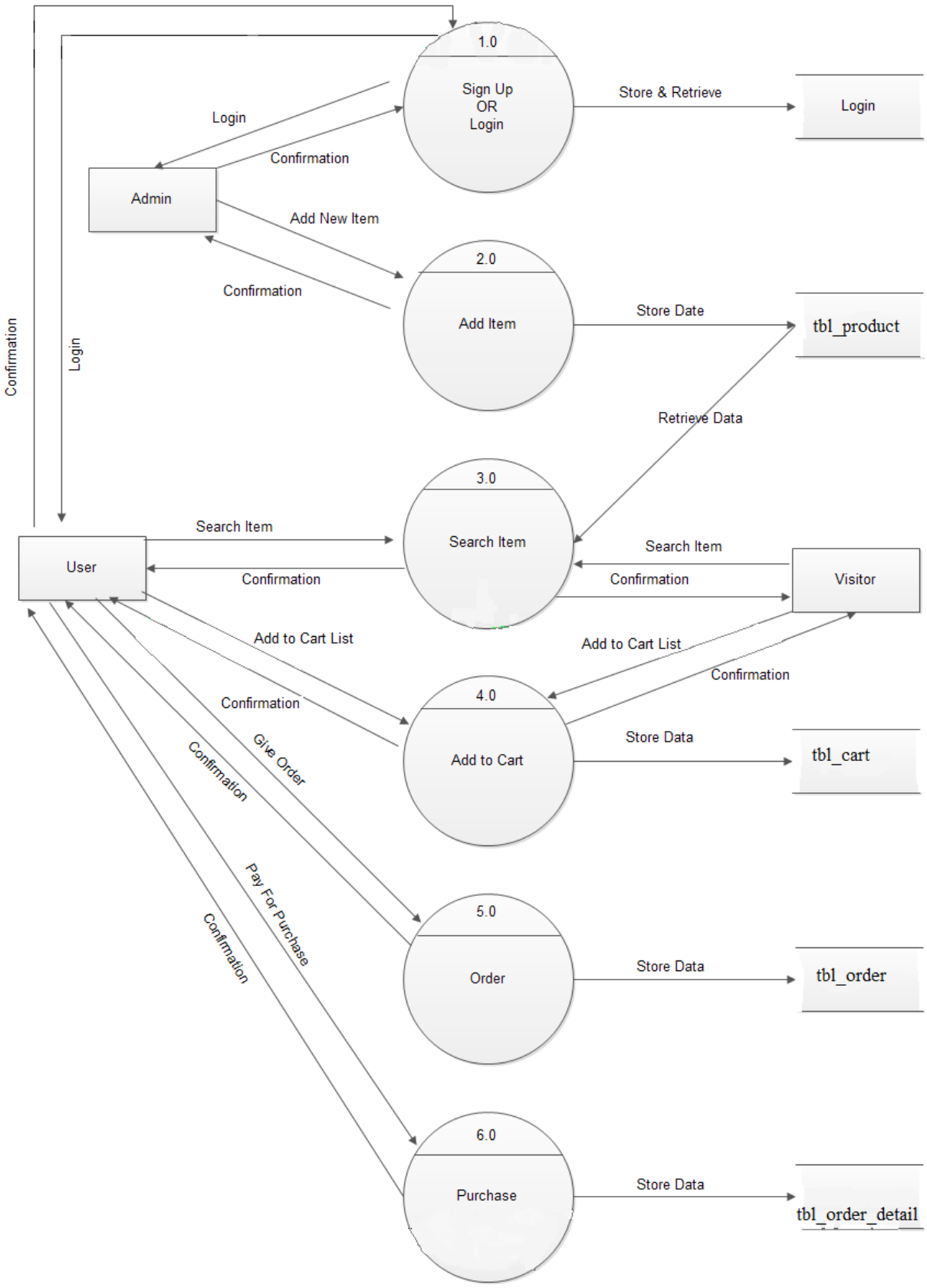
4. Activity Diagram:



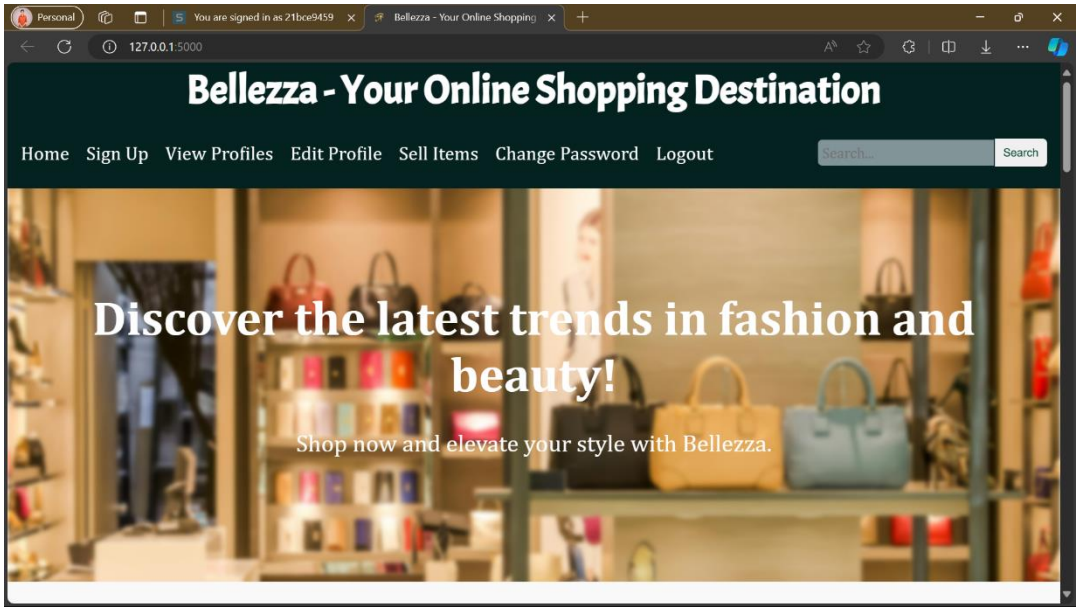
5. Collaboration Diagram:

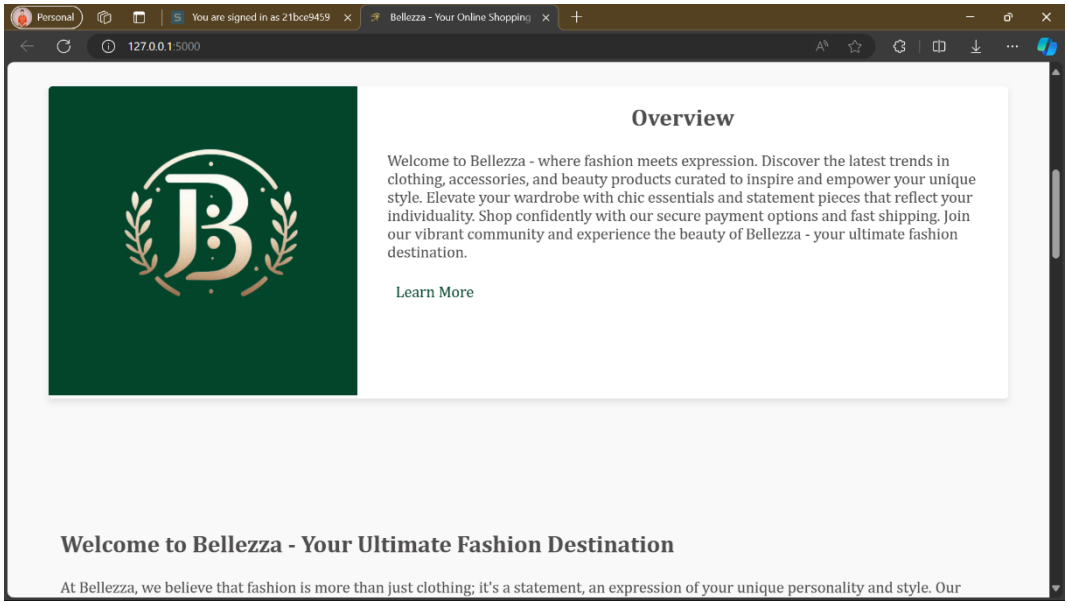


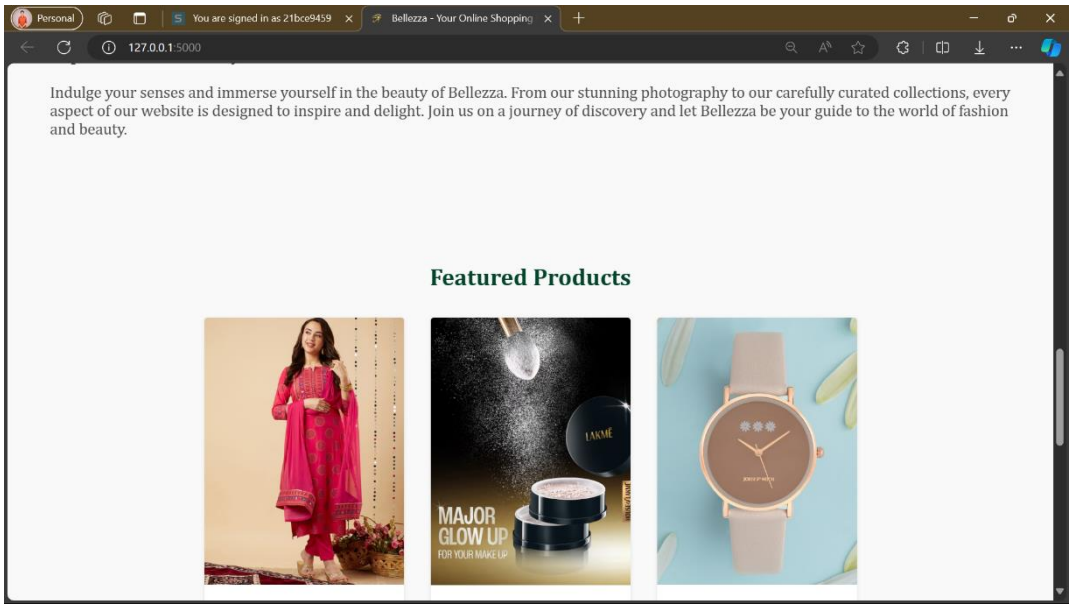
Data Flow Diagram:



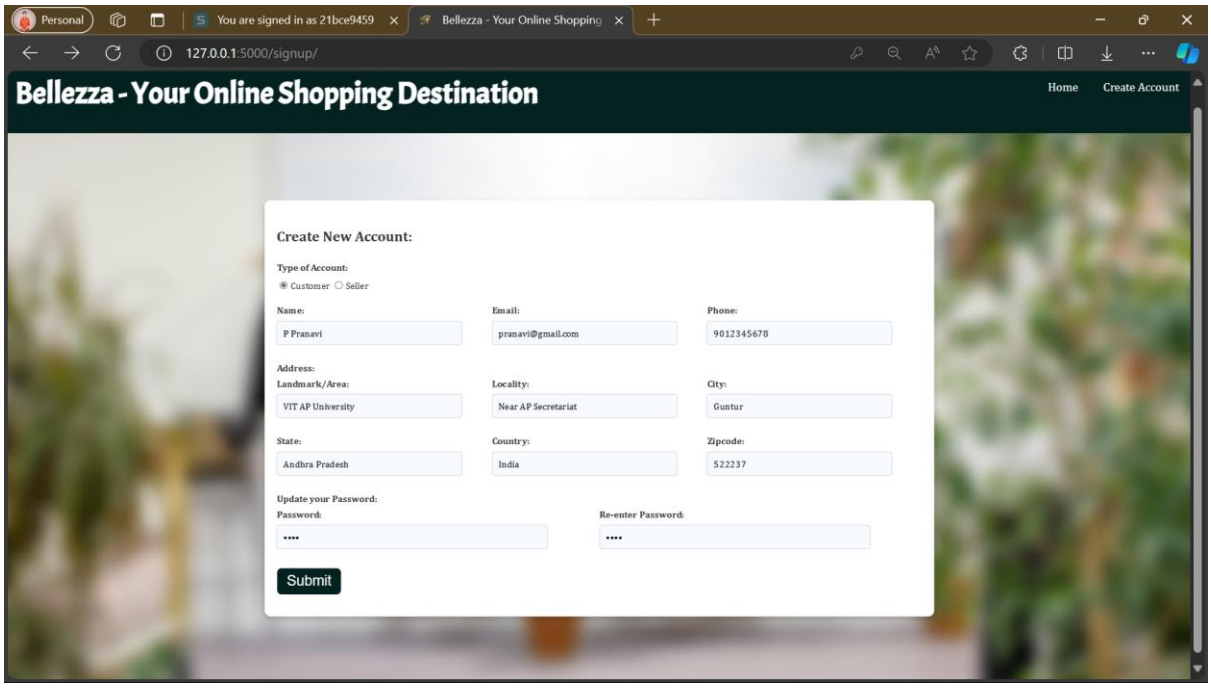
7. User Interface – Landing Page



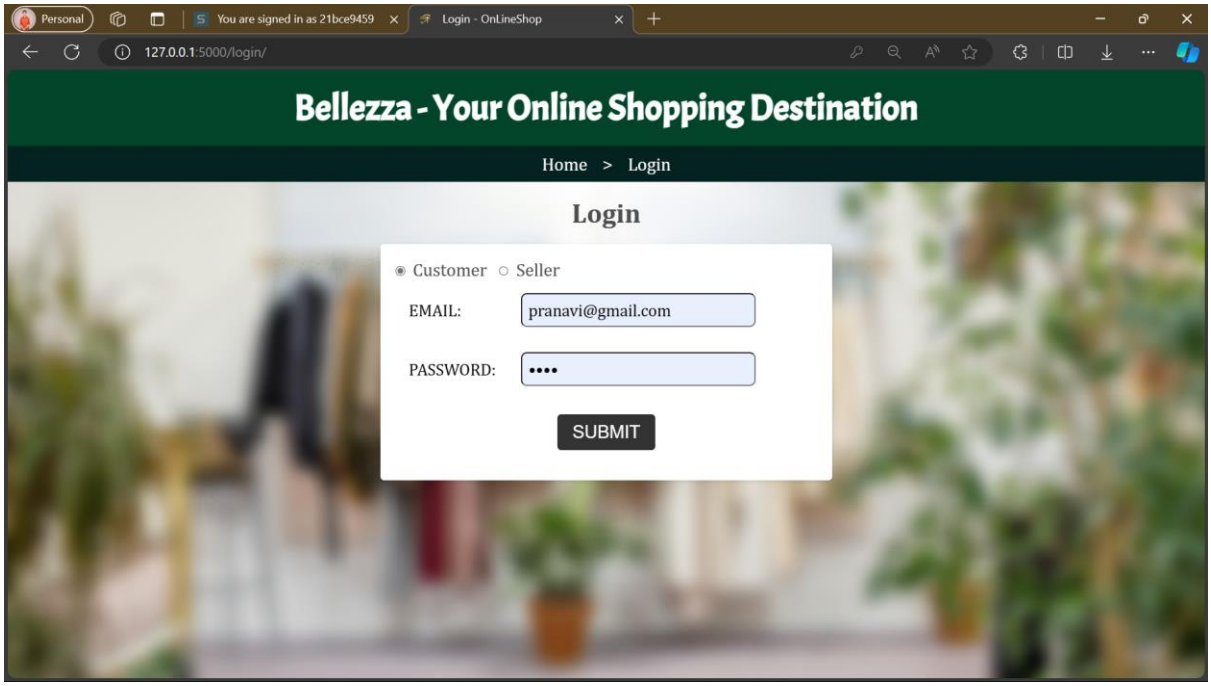




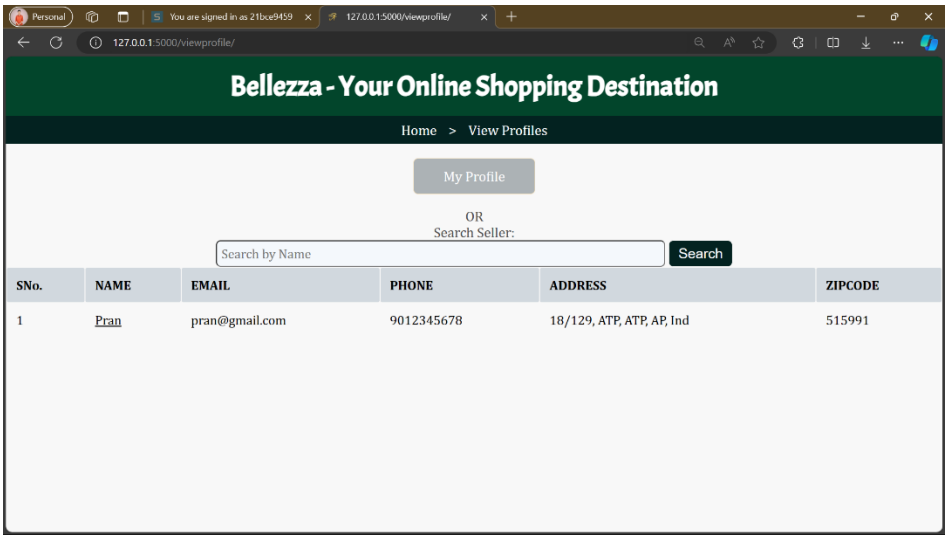
Sign Up Page – Customer:

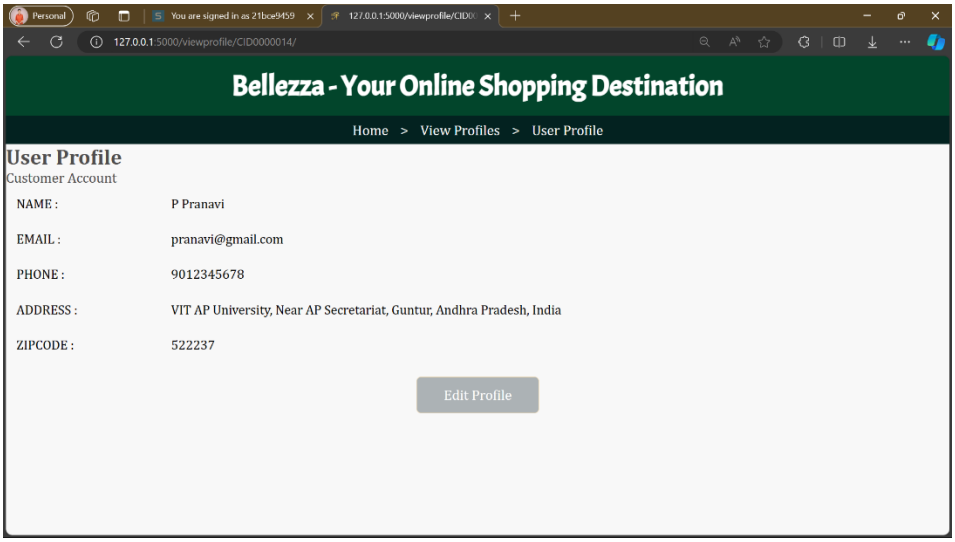


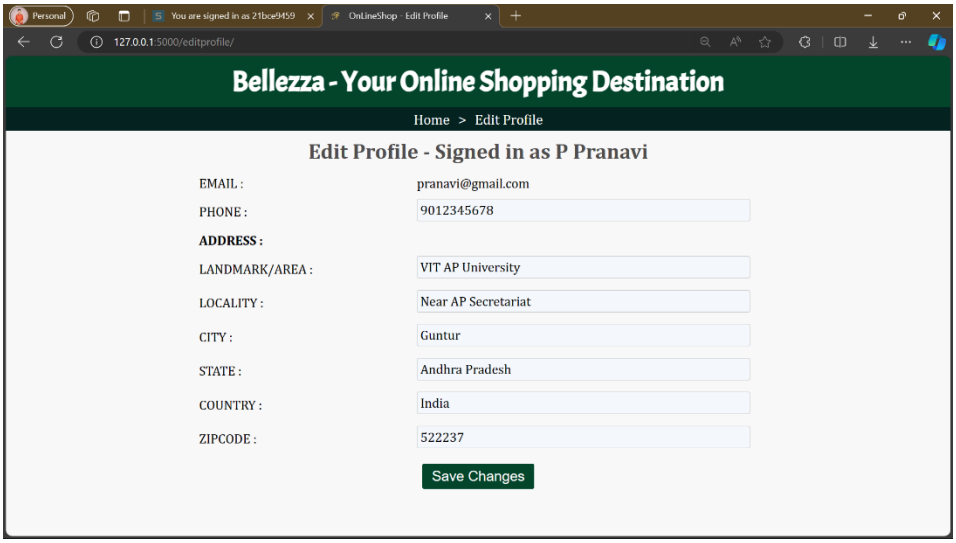
Login Page: Customer



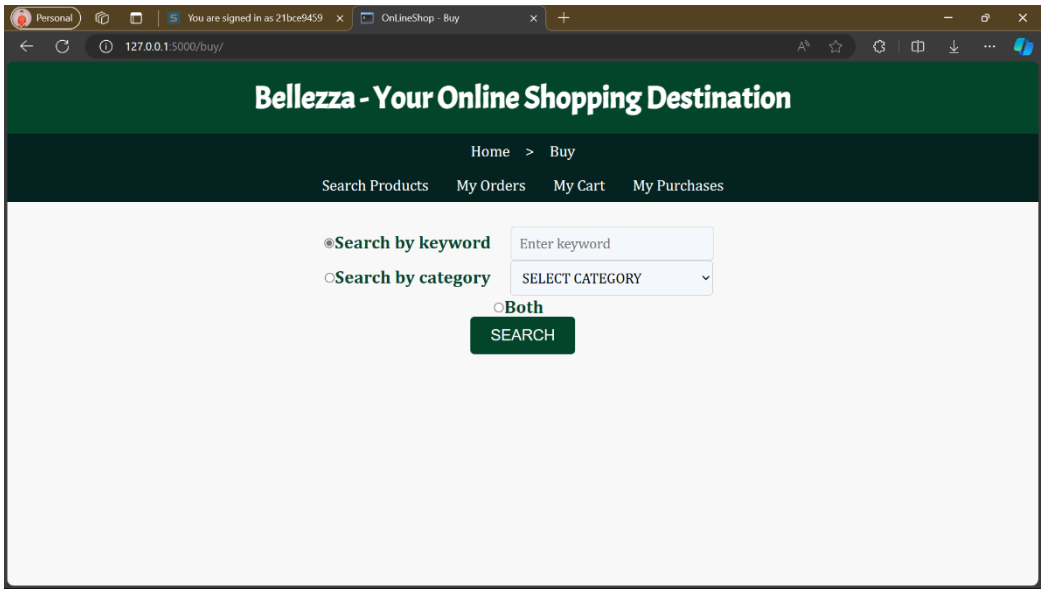
View Profiles:

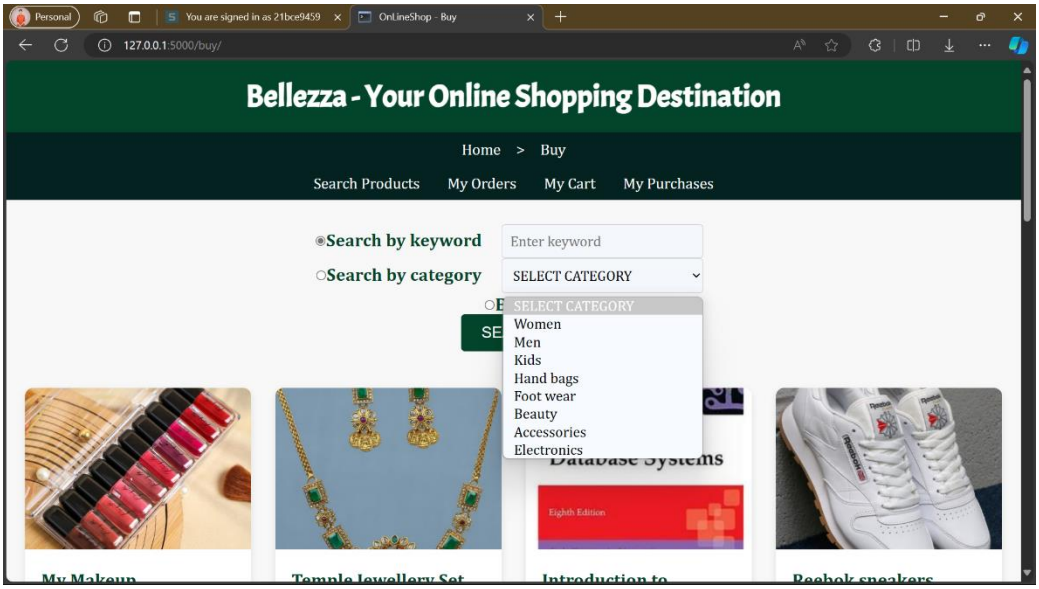


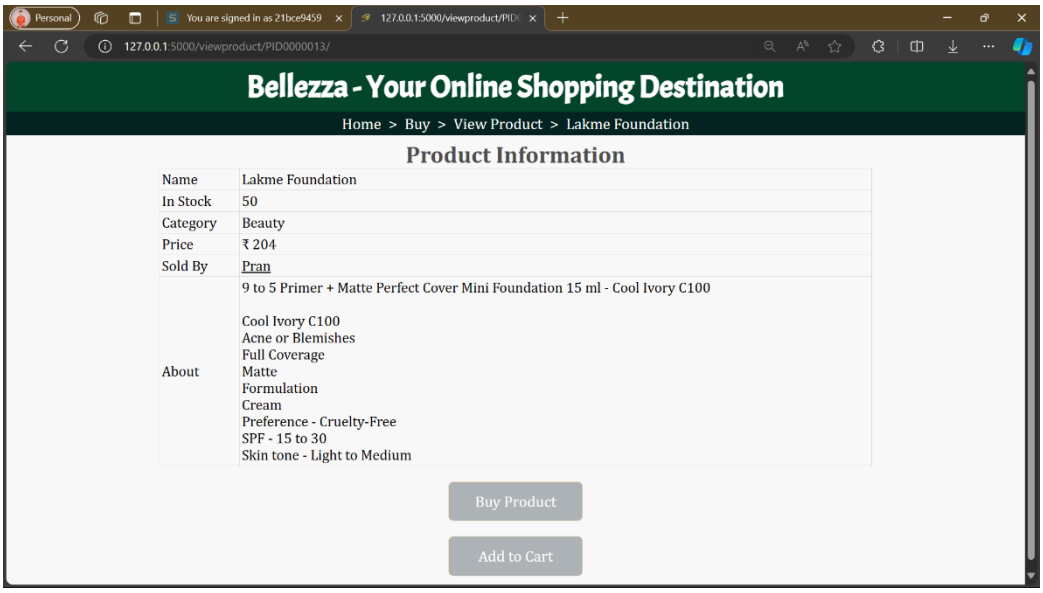




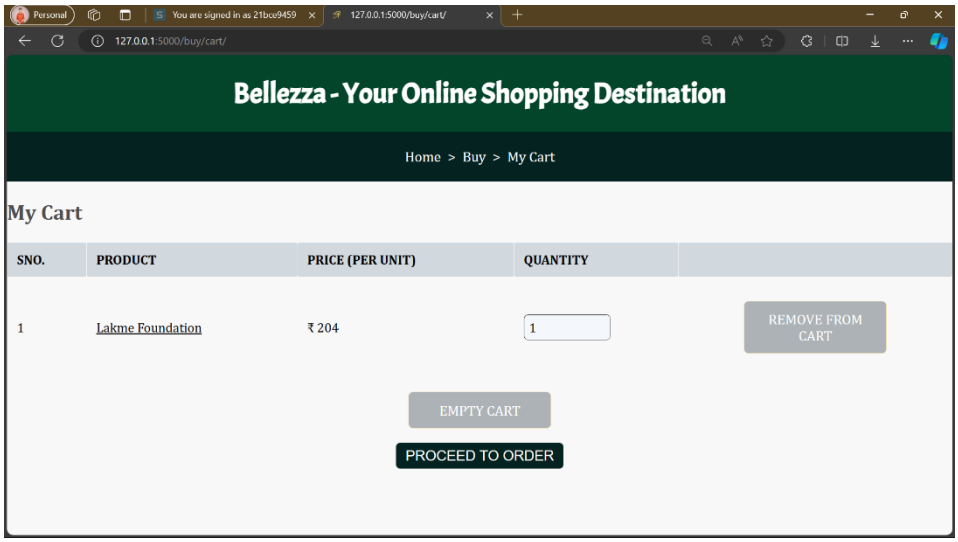
Buy Items:


Bellezza - Your Online Shopping Destination
Home > Buy
Search Products
My Orders
My Cart
My Purchases
Search by keyword
Enter keyword
Search by category
SELECT CATEGORY
Both
SEARCH

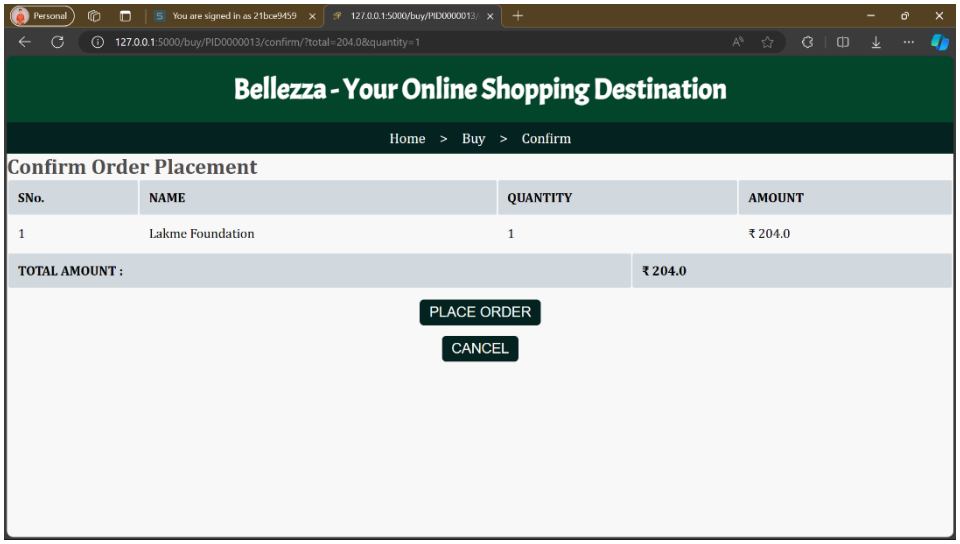

Bellezza - Your Online Shopping Destination
Home > Buy
Search Products
My Orders
My Cart
My Purchases
Search by keyword
Enter keyword
Search by category
SELECT CATEGORY
SELECT CATEGORY
Women
Men
Kids
Hand bags
Foot wear
Beauty
Accessories
Electronics


Bellezza - Your Online Shopping Destination
Home > Buy > View Product > Lakme Foundation
Product Information
Name
Lakme Foundation
In Stock
50
Category
Beauty
Price
₹ 204
Sold By
Pran
About
9 to 5 Primer + Matte Perfect Cover Mini Foundation 15 ml - Cool Ivory C100
Cool Ivory C100
Acne or Blemishes
Full Coverage
Matte
Formulation
Cream
Preference - Cruelty-Free
SPF - 15 to 30
Skin tone - Light to Medium
Buy Product
Add to Cart

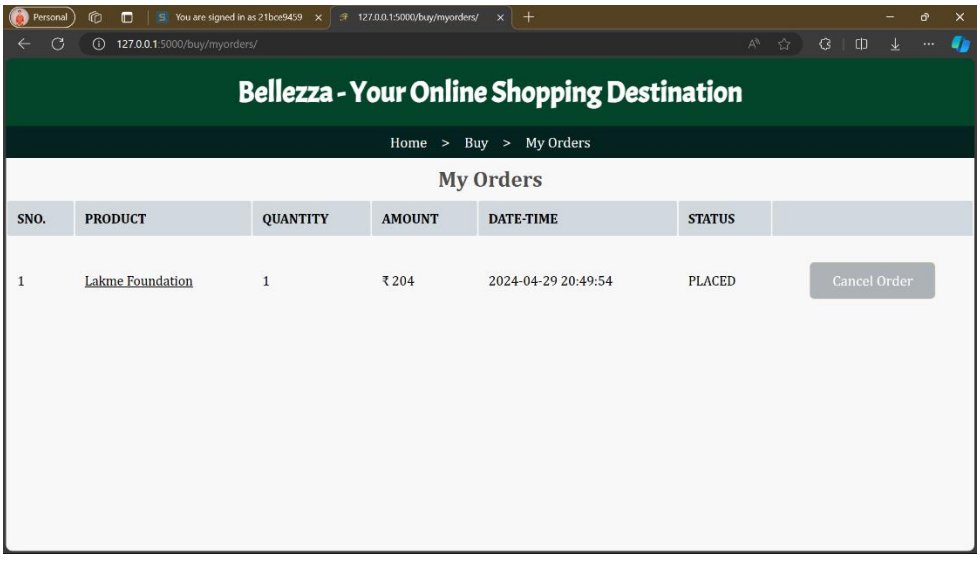
My Cart:



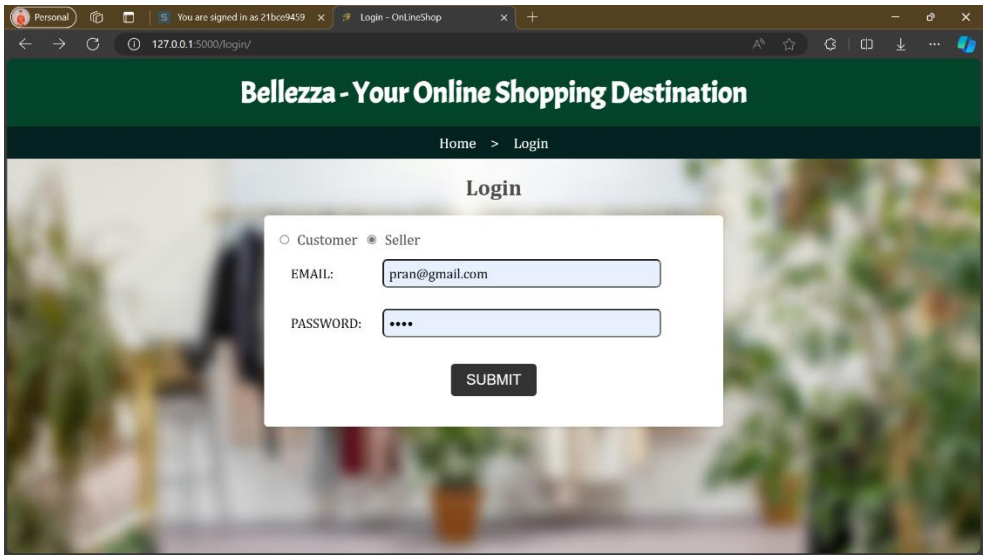
Confirm Order:



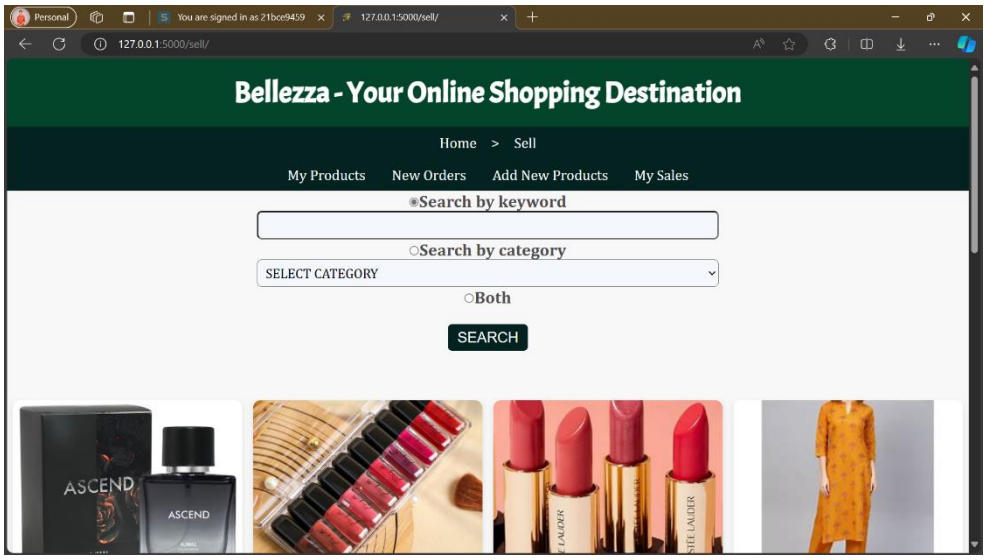
My Orders:



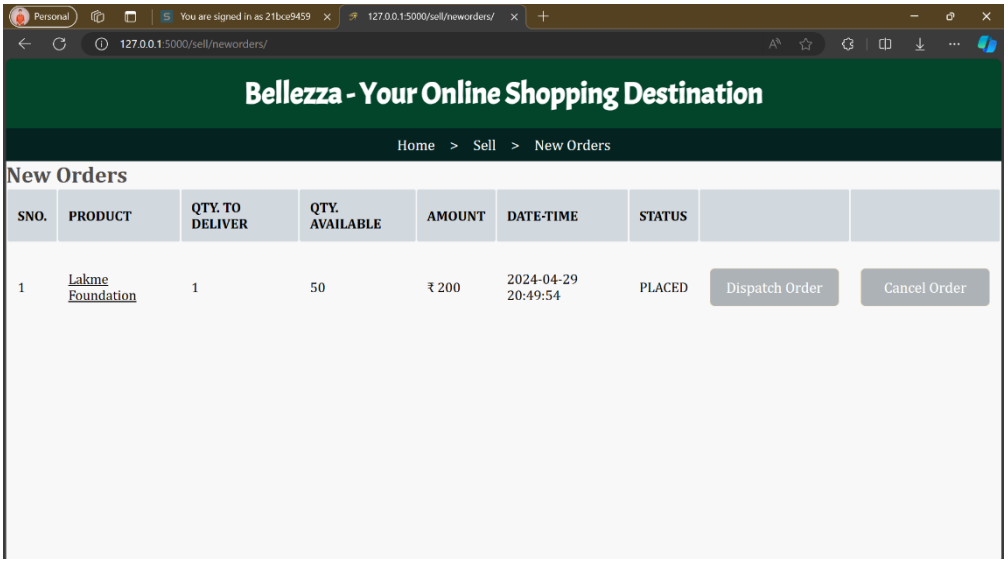
Seller Login:



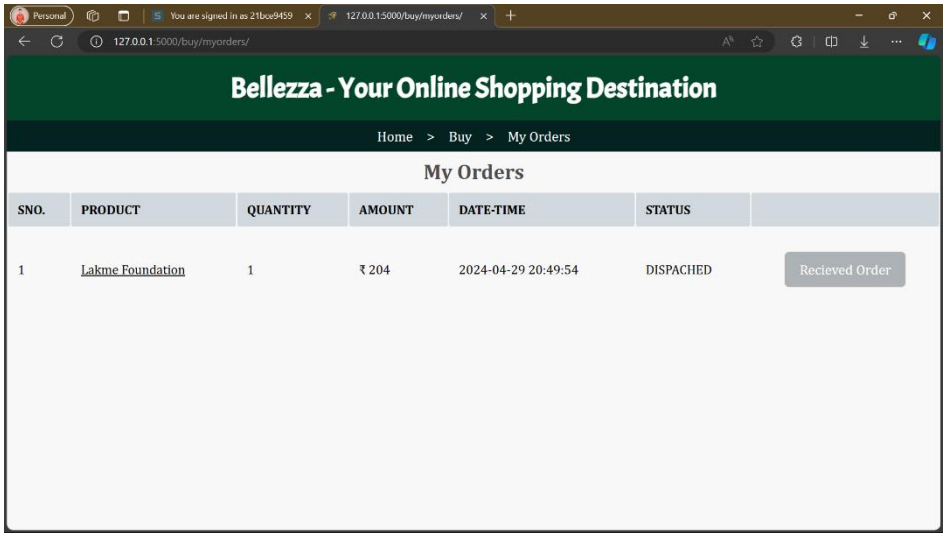
Seller Products:



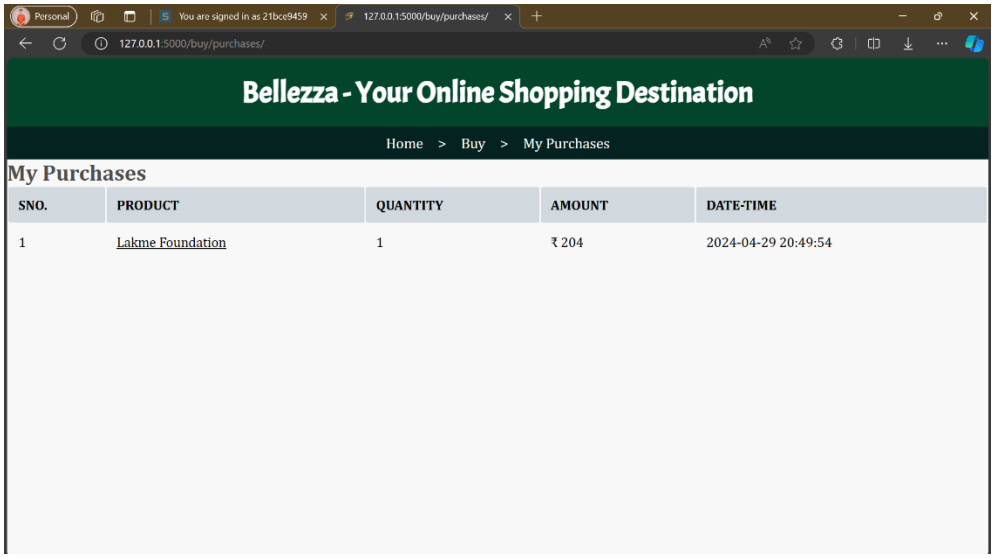
New Orders to dispatch:



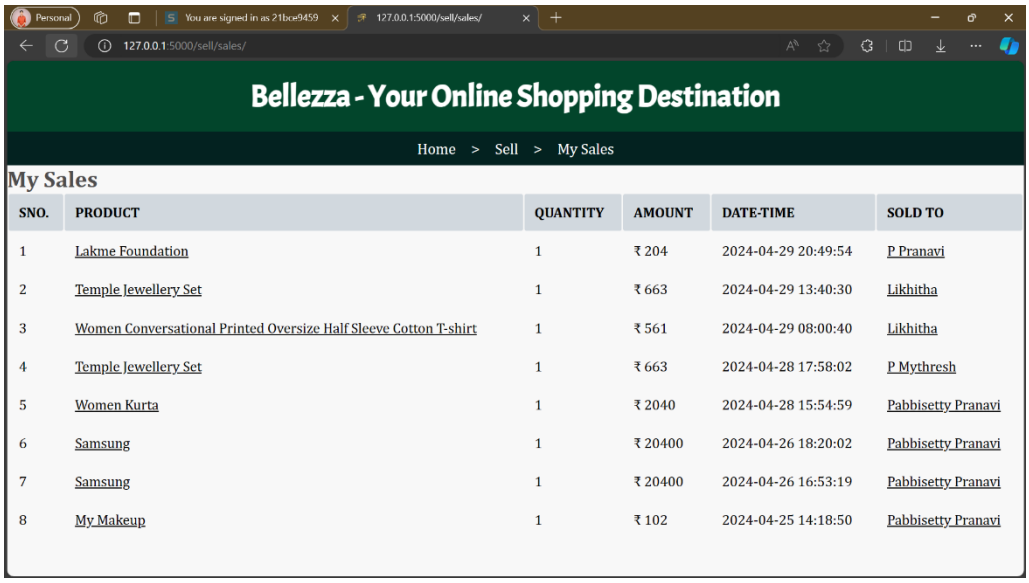
Customer's My Orders Page:



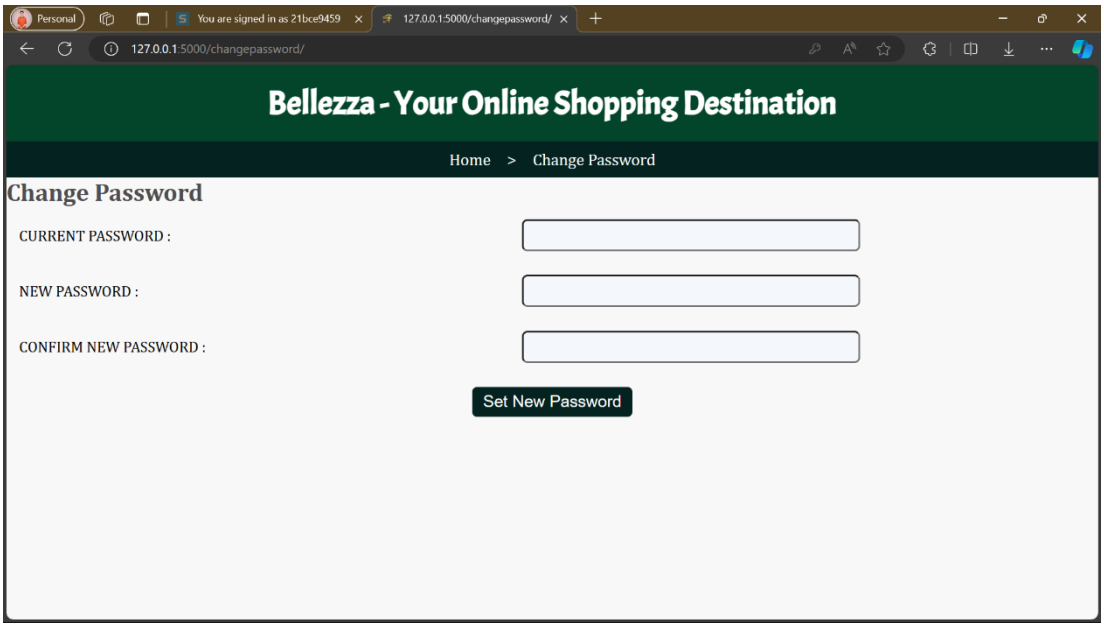
My Purchases of Customer Account:



Updated My Sales of Seller:



Change Password:



8. Test Case Report:

Test Cases:

Test Case Id	Description	Expected Result	Actual Result	Pass/Fail
TC- 01	User Registration - Valid Input	New user is registered successfully	New user is registered successfully	Pass
TC-02	User Registration - Invalid Input	Proper error message is displayed for invalid input	Proper error message is displayed for invalid input	Pass
TC-03	User Login - Valid Credentials	User is logged in successfully	User is logged in successfully	Pass
TC-04	User Login - Invalid Credentials	Proper error message is displayed for	Proper error message is displayed for	Pass

		invalid credentials	invalid credentials	
TC-05	Product Addition - Valid Input	New product is added to the inventory	New product is added to the inventory	Pass
TC-06	Product Addition - Invalid Input	Proper error message is displayed for invalid input	Proper error message is displayed for invalid input	Pass
TC-07	Search Functionality - Valid Input	Products matching search criteria are displayed	Products matching search criteria are displayed	Pass
TC-08	Search Functionality - Invalid Input	Proper message is displayed for no search results	Proper message is displayed for no search results	Pass
TC-09	Order Placement - Valid Input	Order is successfully placed	Order is successfully placed	Pass
TC-10	Order Placement - Invalid Input	Proper error message is displayed for invalid input	Proper error message is displayed for invalid input	Pass

Test Summary:

- Total Test Cases: 10
- Passed: 10
- Failed: 0
- Success Rate: 100%

Detailed Test Results:

1. Unit Tests:
   - Test Case 1: Passed

- Test Case 2: Passed
- Test Case 3: Passed
- Test Case 4: Passed
- Test Case 5: Passed
- Test Case 6: Passed
- Test Case 7: Passed
- Test Case 8: Passed
- Test Case 9: Passed
- Test Case 10: Passed

2. Integration Tests:
   - All integration tests passed successfully.
3. User Acceptance Tests:
   - Scenario 1: Passed
   - Scenario 2: Passed
   - Scenario 3: Passed
   - Scenario 4: Passed

Conclusion:

All tests passed successfully, indicating the robustness and reliability of the application.

9. Future Enhancements

Enhanced User Experience: Implement a more intuitive user interface to improve user engagement and satisfaction.

Advanced Search Functionality: Incorporate advanced search filters and sorting options to facilitate easier product discovery.

Social Media Integration: Allow users to share products or their shopping experiences on social media platforms to increase brand visibility.

Personalized Recommendations: Implement a recommendation system based on user preferences and browsing history to suggest relevant products.

Mobile Application Development: Develop a mobile application to expand the platform's reach and accessibility.