**Software Requirements Specification (SRS)**

**For Dressify - An Online Fashion Store**

**1. Introduction**

**1.1 Purpose**

This document serves as the Software Requirements Specification (SRS) for **Dressify**, an online fashion store. The purpose of this document is to outline the functional and non-functional requirements, technical details, design considerations, and other aspects of the system to ensure that the project meets user needs and expectations.

**1.2 Scope**

**Dressify** is a web-based e-commerce platform designed to sell fashion items like clothing and accessories. Customers can browse, search, and buy products. They can also manage their profiles, track orders, and proceed with payments. Admins can manage products, customers, and orders through a backend interface. The frontend is developed using **HTML**, **CSS**, and **JavaScript**, while the backend uses the **Django framework** with a **PostgreSQL database**.

**1.3 Definitions, Acronyms, and Abbreviations**

• **Frontend:** The visual part of the website that customers interact with, built with **HTML**, **CSS**, and **JavaScript**.

• **Backend:** The server-side of the application, where business logic is processed and database interactions happen, developed using **Django** and **PostgreSQL**.

• **User:** Refers to both customers and admin users in the system.

• **Admin Panel:** A web interface for administrators to manage the system’s content, products, orders, and user accounts.

**2. Overall Description**

**2.1 Product Perspective**

**Dressify** is a standalone application, and all operations like adding products, placing orders, and user authentication happen on the same platform. It will be hosted on a secure web server, allowing customers to easily access the site and make transactions. It integrates with **PostgreSQL** to store all product, user, and order information, and uses **Django** as the backend web framework.

**2.2 Product Features**

The key features of the **Dressify** platform include:

• **User Management**: Registration, login, profile management, and password recovery.

• **Product Management**: Adding, editing, and deleting fashion products in various categories.

• **Shopping Cart**: Users can add products to their shopping cart, modify quantities, and proceed to checkout.

• **Order Management**: Users can place orders, view order history, and track deliveries.

• **Admin Panel**: Admins can manage products, users, orders, and view reports.

• **Responsive Design**: The site adapts seamlessly to mobile, tablet, and desktop views.

**2.3 User Classes and Characteristics**

• **Customers**: Individuals who visit the website to browse, shop, and make purchases.

• **Admins**: Users with higher privileges who can manage the product catalog, user data, and orders.

**2.4 Operating Environment**

• **Frontend:** The frontend is built using **HTML**, **CSS**, and **JavaScript** to provide an interactive user experience.

• **Backend:** The backend runs on the **Django framework**, with a **PostgreSQL** database for persistent storage.

• **Hosting:** The application is hosted on a secure web server with **HTTPS** enabled for secure transactions.

**3. Functional Requirements**

**3.1 User Registration and Authentication**

• **Description**: The system must allow users to sign up, log in, and manage their account information securely.

• **Use Case**: A new user can create an account with an email and password. After registration, the user can log in and manage their profile.

• **Constraints**: Passwords must be securely hashed and stored using Django’s built-in authentication system. Third-party authentication (like Google login) should be implemented.

**3.2 Product Management**

• **Description**: Admin users should be able to add, update, and delete products from the product catalog.

• **Use Case**: Admins can add new products with details such as name, description, price, images, and available sizes.

• **Constraints**: Products must be categorized and made searchable by name, category, price, and size.

**3.3 Shopping Cart**

• **Description**: Customers can add products to their shopping cart, view the cart, modify quantities, and proceed to checkout.

• **Use Case**: A customer can add multiple items to the cart, adjust the quantities, and view the total cost.

• **Constraints**: The cart must persist across sessions, even if the user logs out and logs back in.

**3.4 Order Management**

• **Description**: Customers can place orders, providing shipping and billing information. The system will calculate the total cost based on the products in the cart.

• **Use Case**: A customer can place an order by completing the checkout process. They will receive an order confirmation with tracking details.

• **Constraints**: The order must be linked to the user’s account, and the total price must be calculated correctly.

**4. Non-Functional Requirements**

**4.1 Performance Requirements**

• The system should handle at least **1,000 concurrent users** without significant performance degradation.

• All pages should load in **under 2 seconds** under normal usage conditions.

**4.2 Security Requirements**

• **Data Protection**: Use **HTTPS** for all communications. Secure user data, especially passwords and payment information, with industry-standard encryption.

• **Authentication**: Implement strong password policies and multi-factor authentication for admin users.

**4.3 Usability**

• The site must be **responsive**, ensuring a seamless experience on mobile devices, tablets, and desktops.

• The user interface should be intuitive and easy to navigate, with clear instructions and error messages.

**4.4 Reliability**

• The system must be **highly available**, with a goal of 99.9% uptime.

• Regular database backups should be performed to ensure data integrity.

**4.5 Maintainability**

• The codebase should follow best practices and be modular, ensuring that it can be easily maintained and updated in the future.

• Use version control (Git) for managing the code and collaborating with the development team.

**5. Tools and Technologies**

**5.1 Frontend Tools**

• **HTML5**: For the structure and content of the web pages.

• **CSS3**: For styling and responsive design.

• **JavaScript**: For interactivity and dynamic content loading.

**5.2 Backend Tools**

• **Django**: A Python-based web framework used to implement the backend.

• **PostgreSQL**: A powerful relational database used to store user and product data.

**5.3 Development Tools**

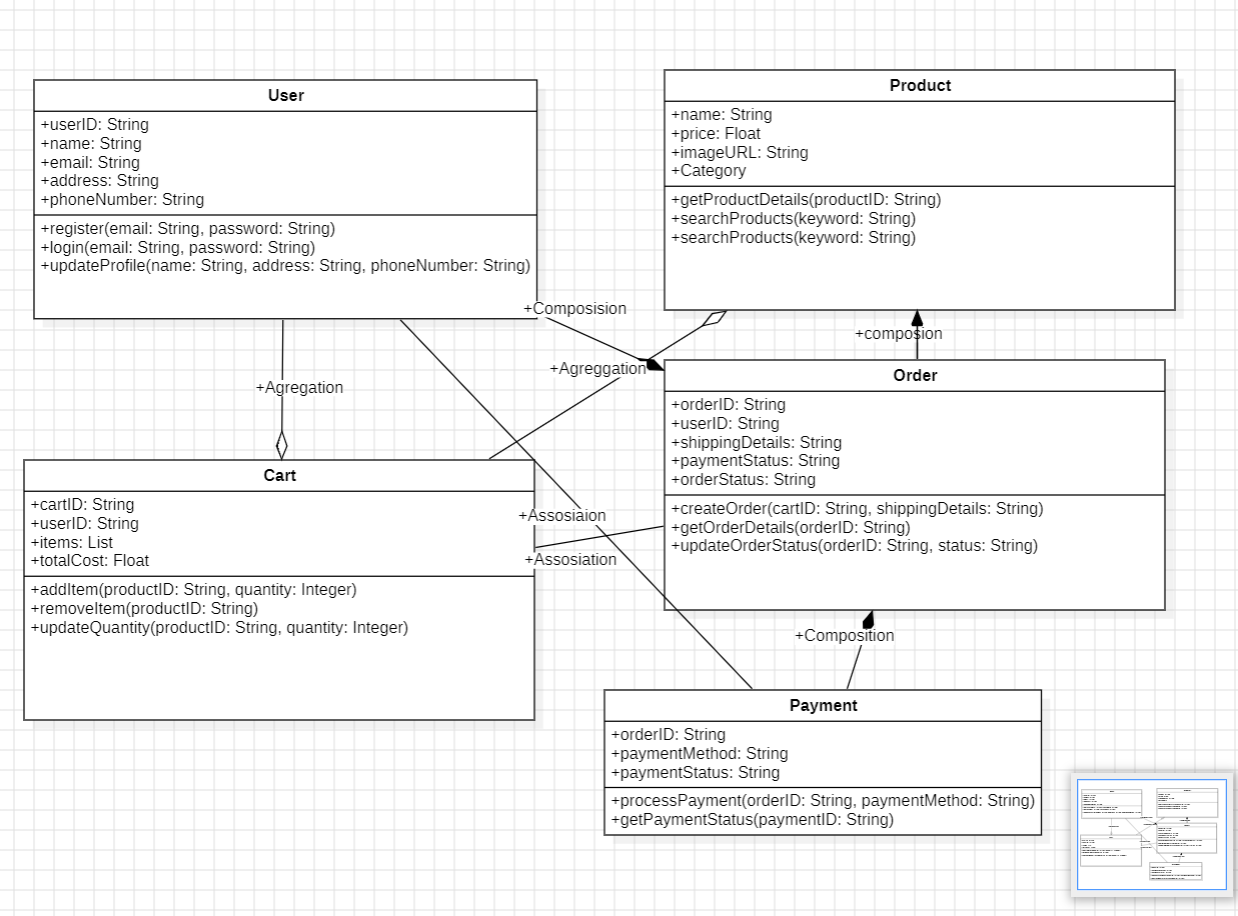
• **Git**: For version control.

• **Visual Studio Code**: IDE used for development.

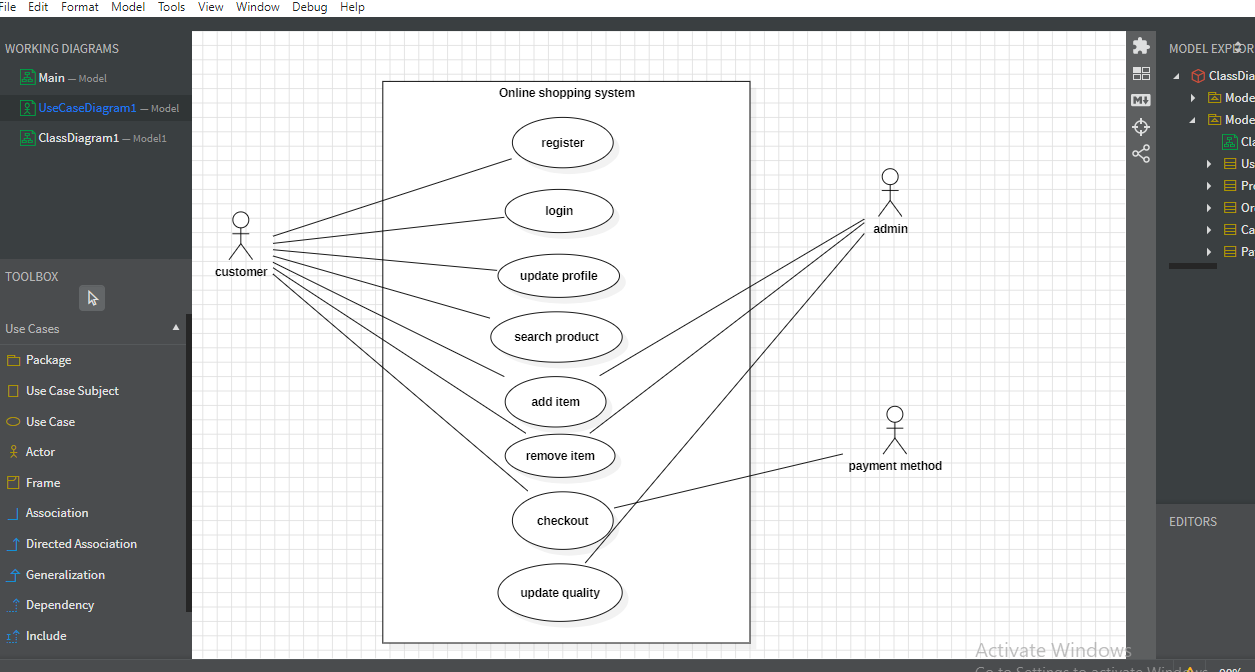
• **Postman**: For testing the backend API.

**6. Diagrams**

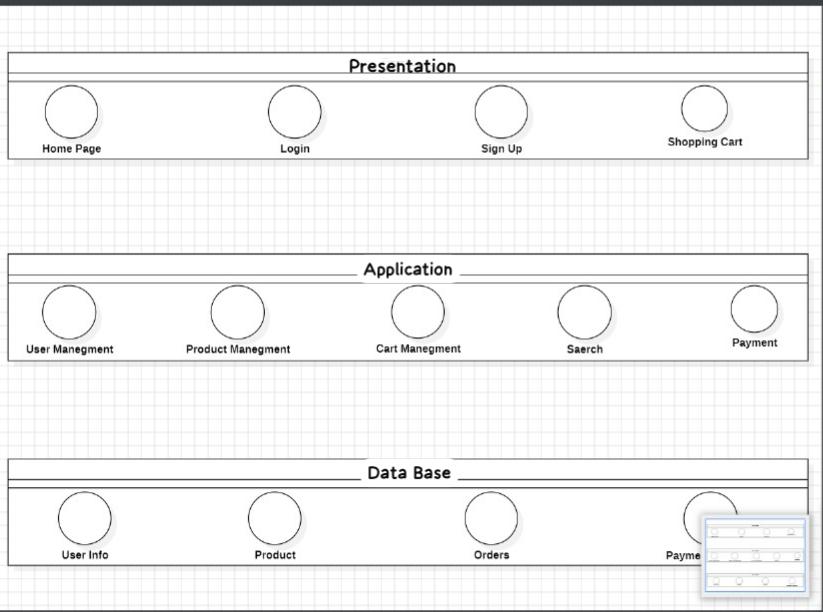
**Class Diagram**



**Use Case Diagram:**



**Architecture Diagram:**

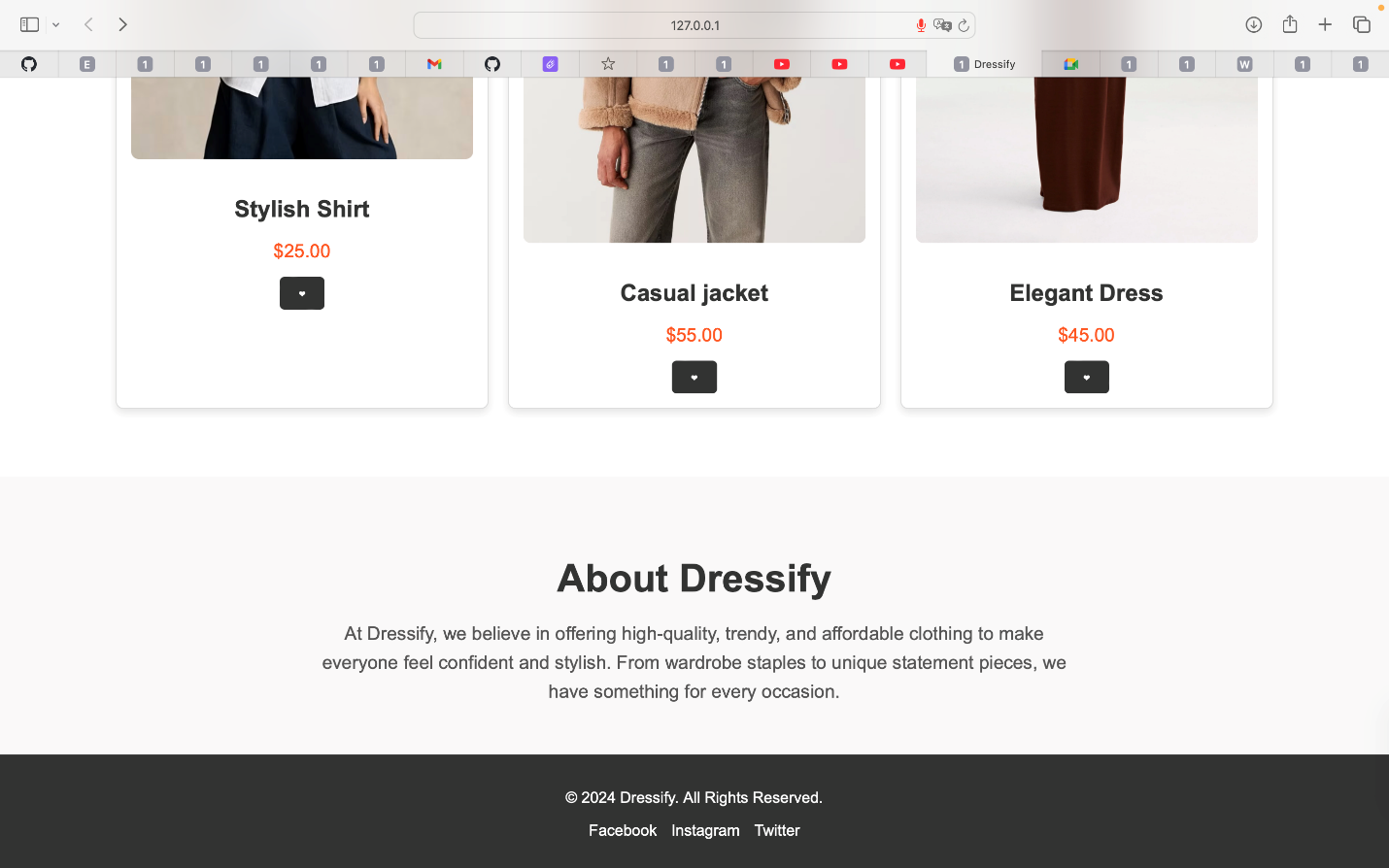
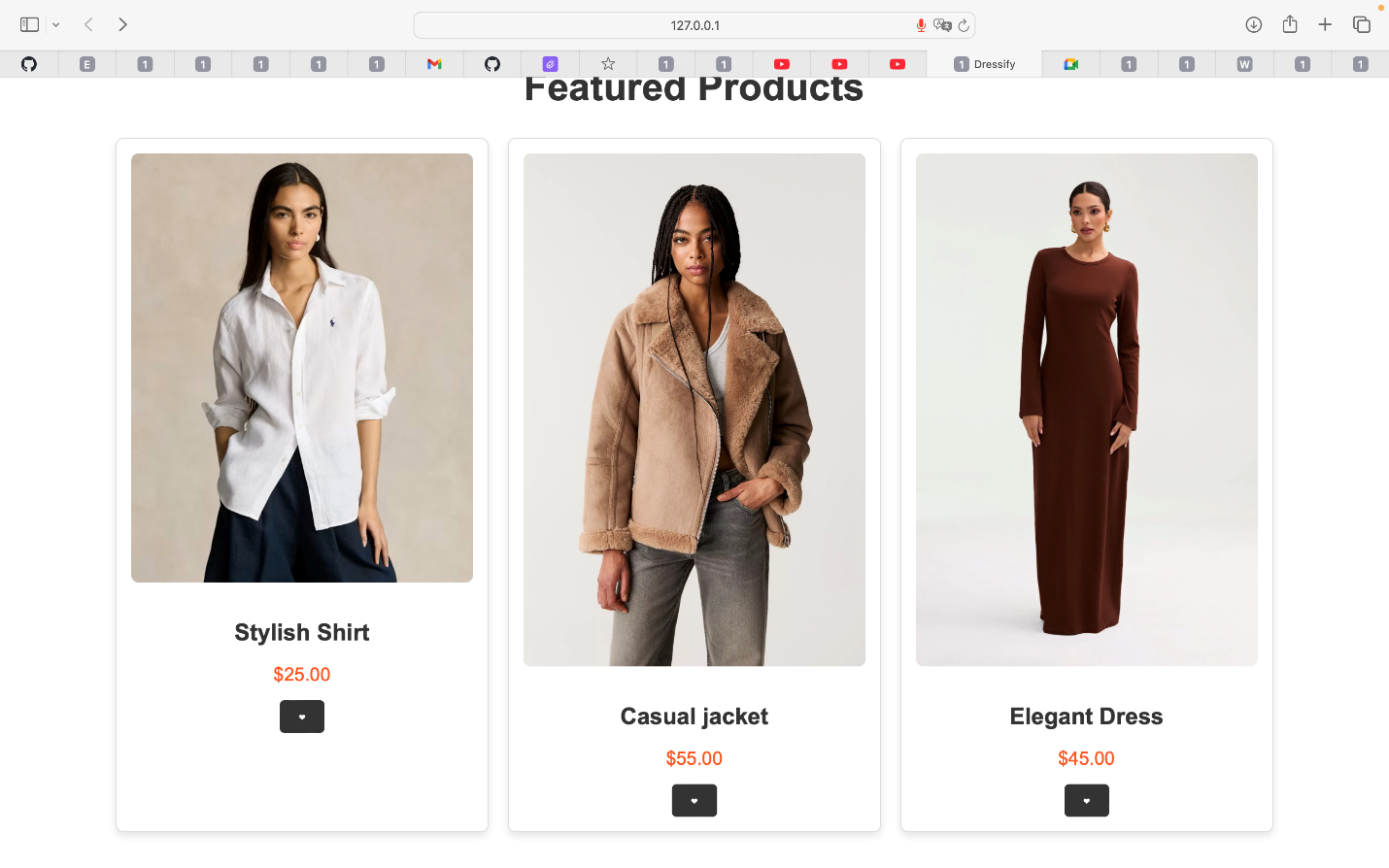
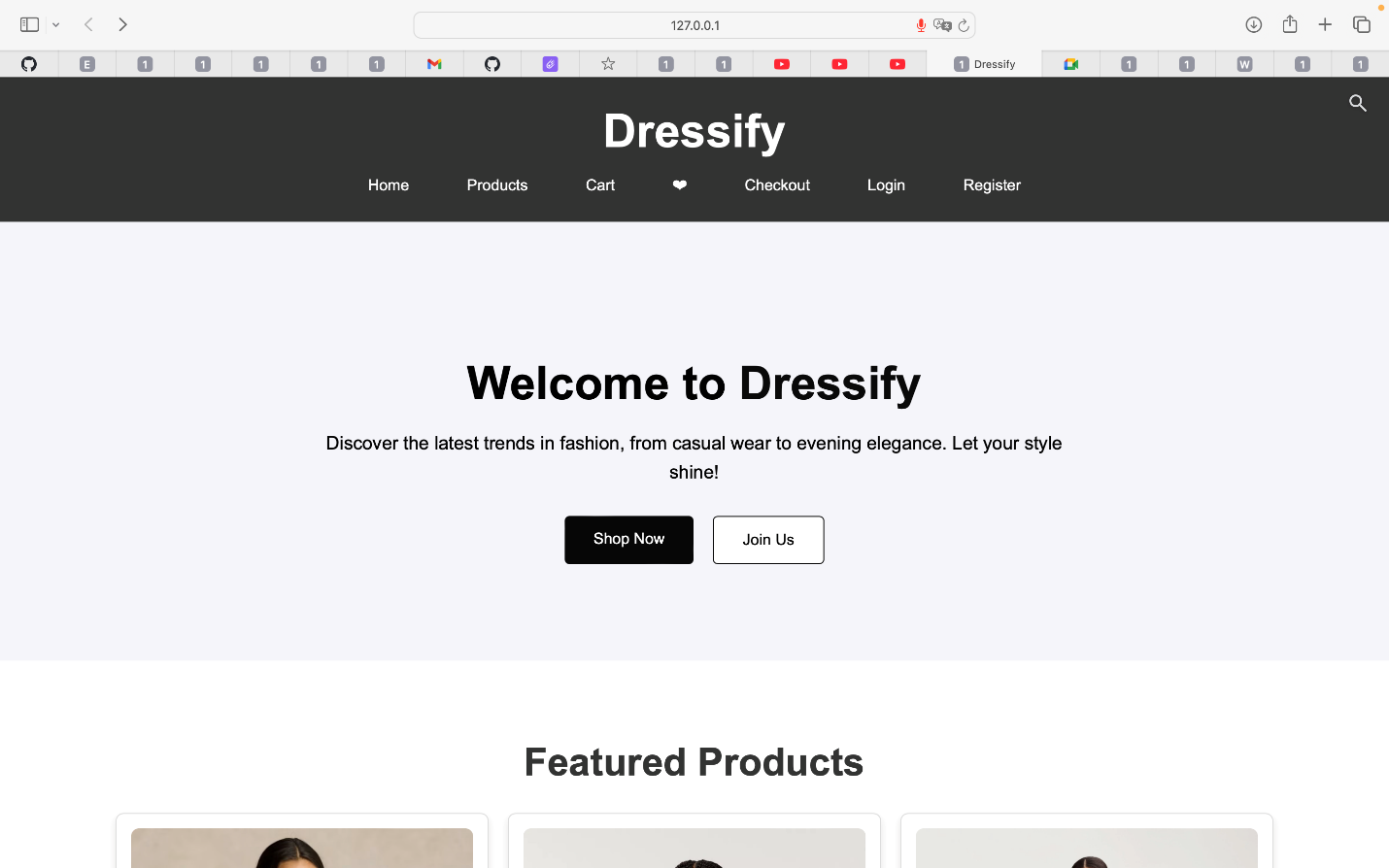


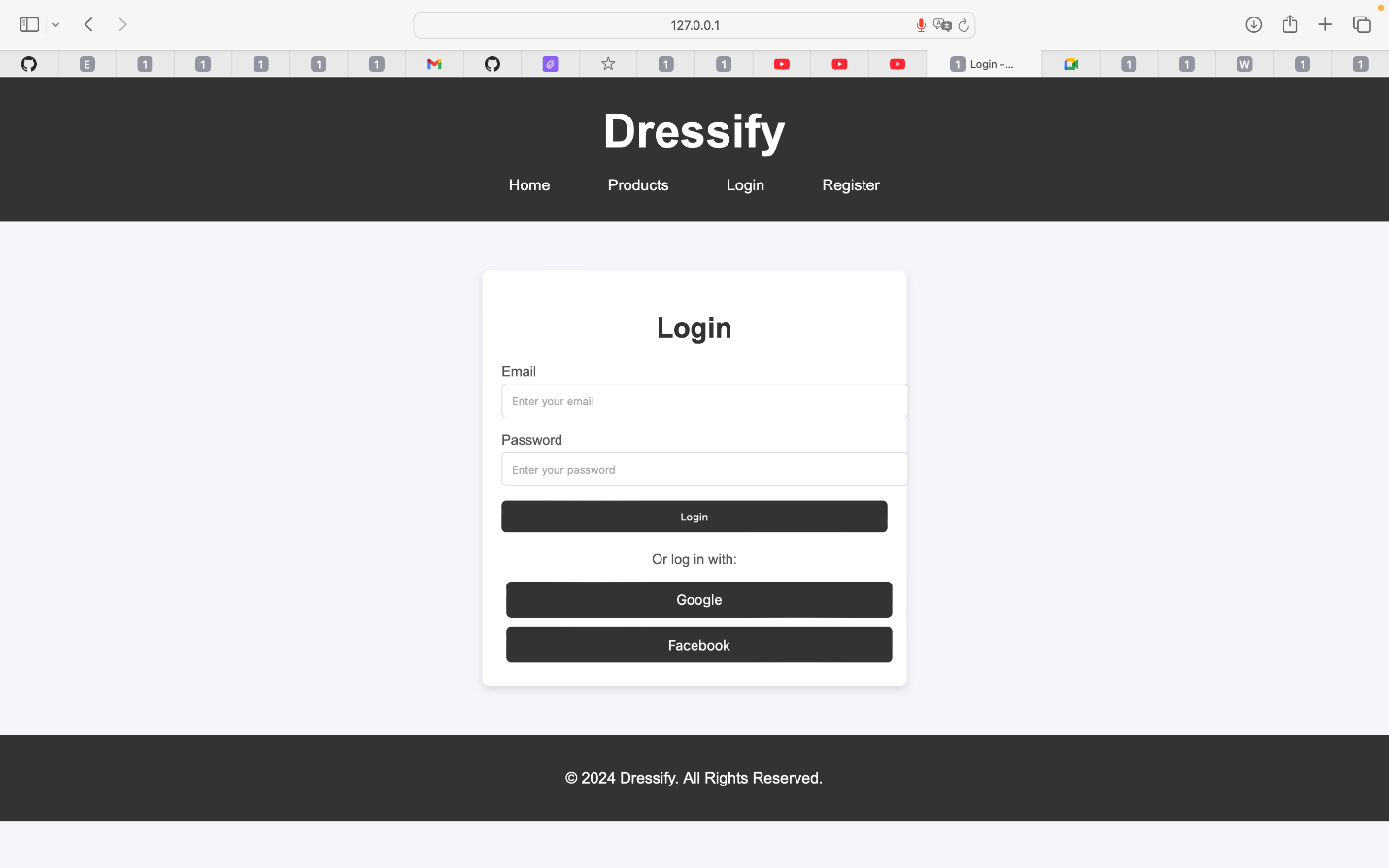
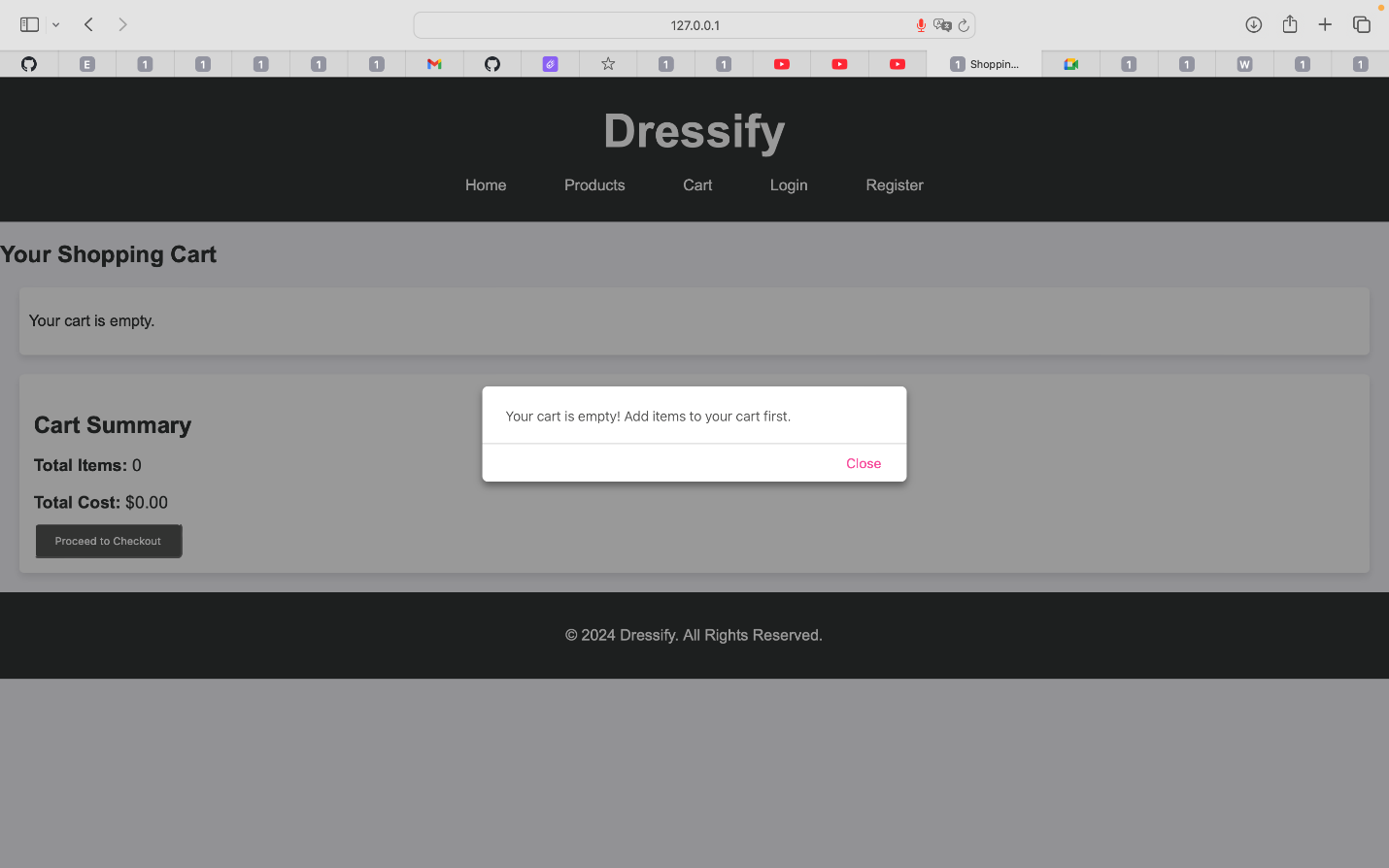
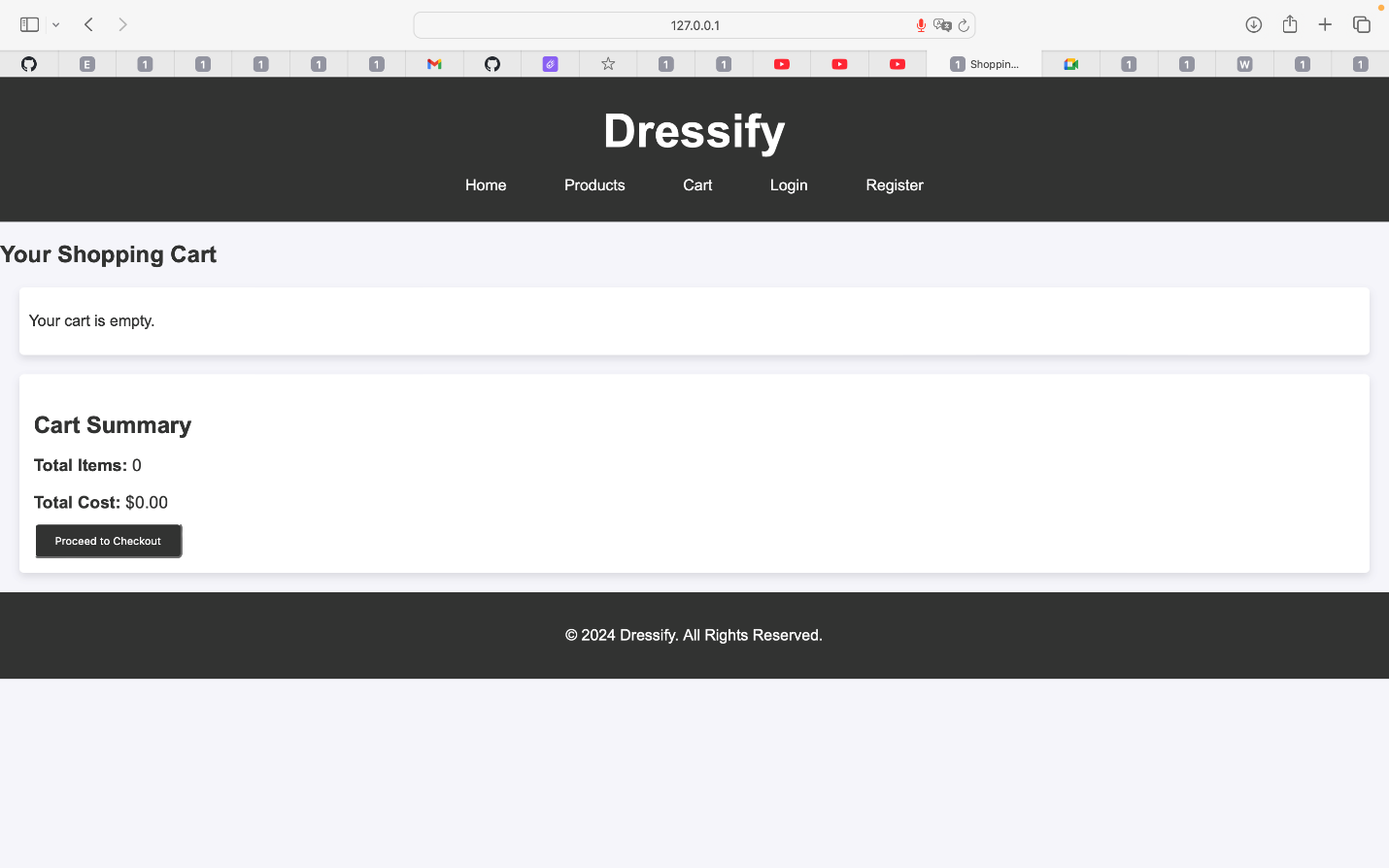
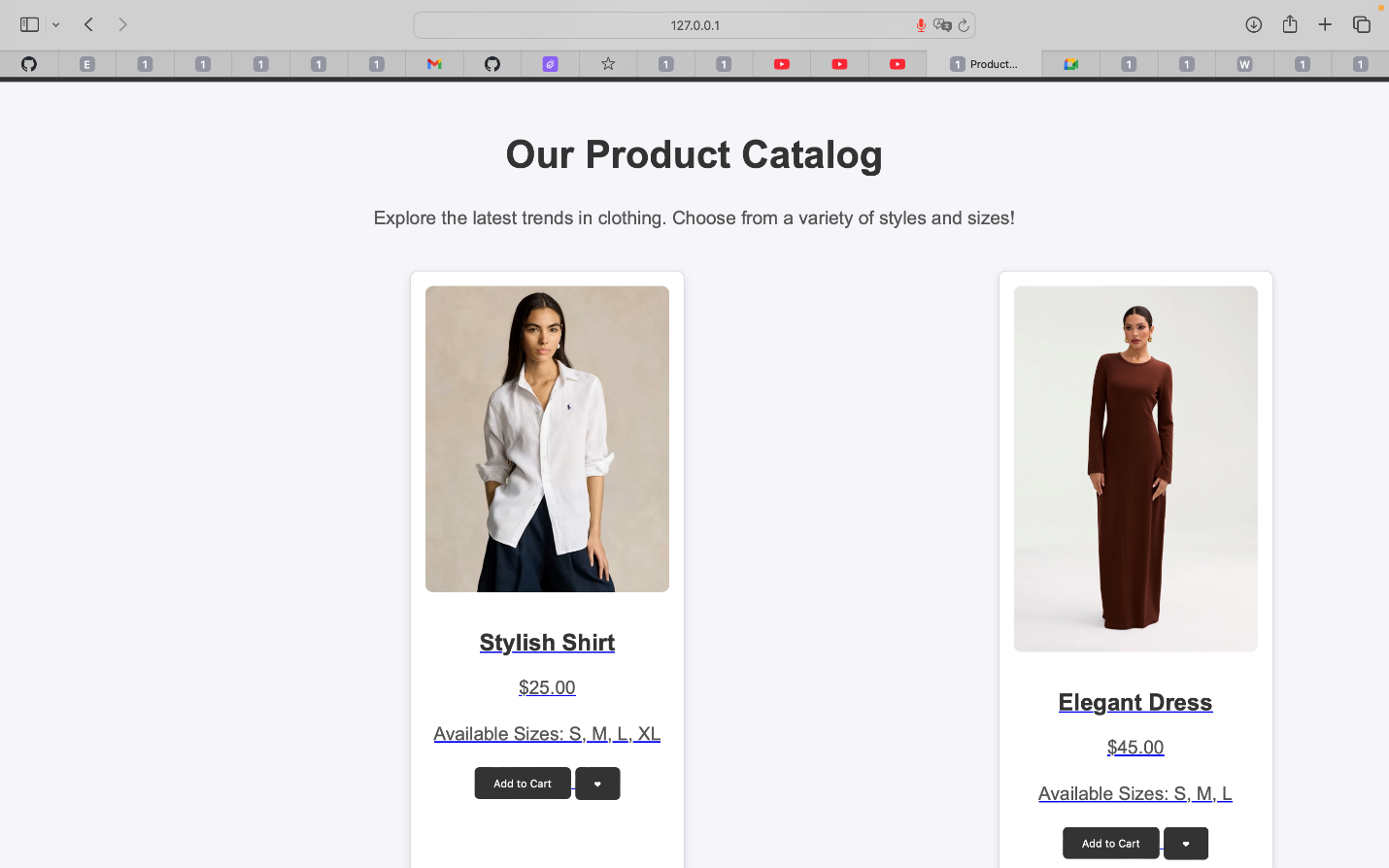
**Sequence Diagram:**

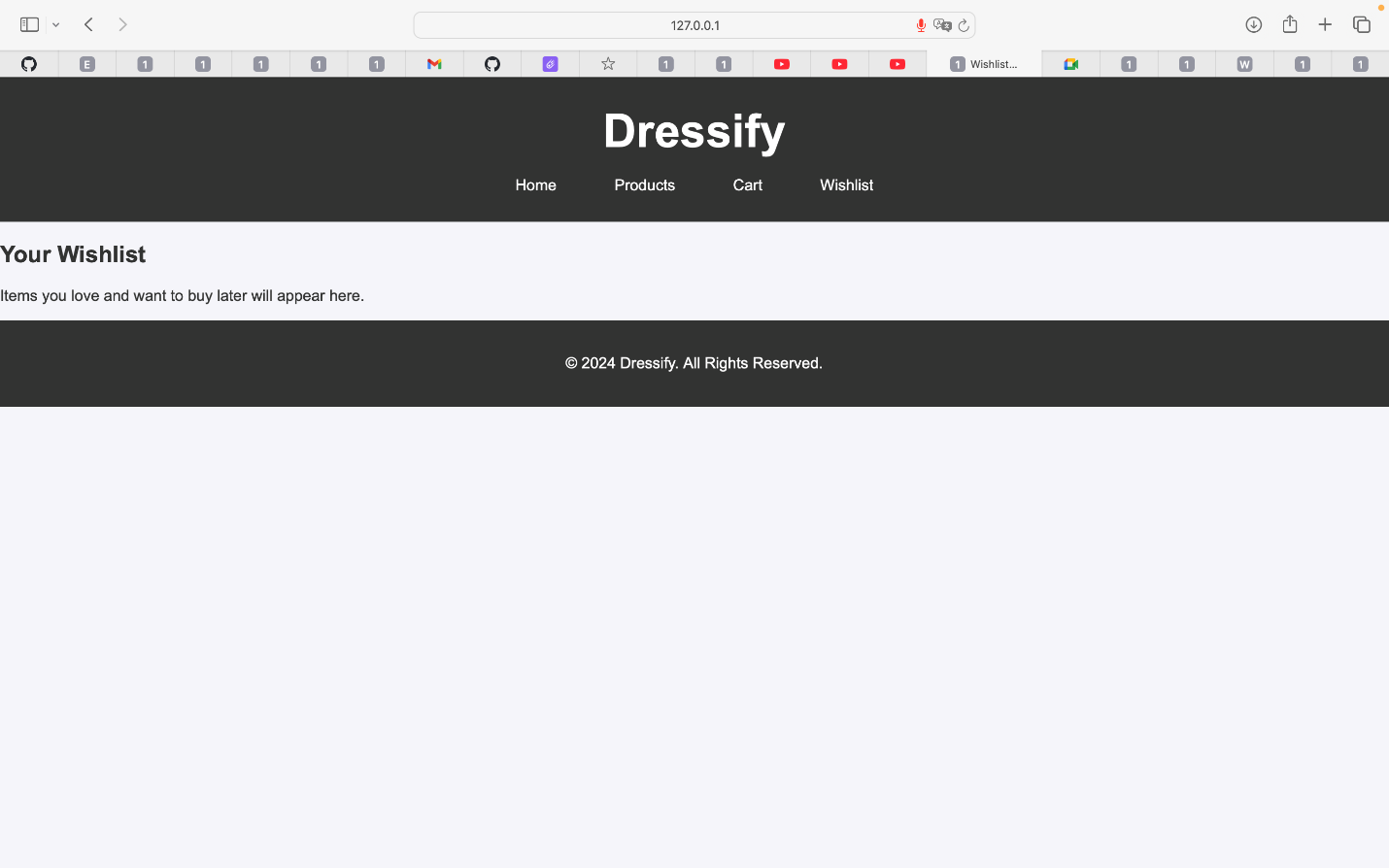
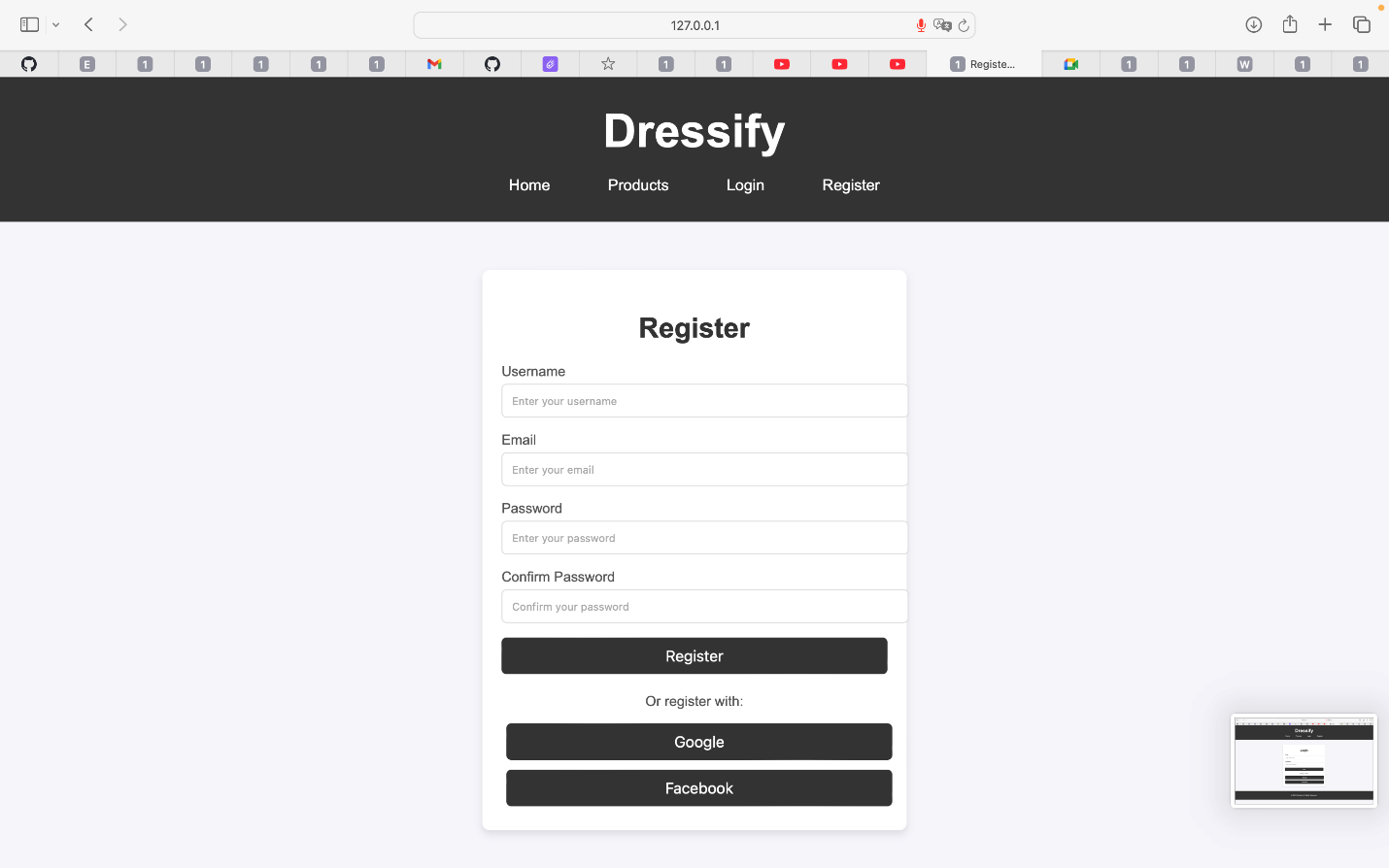
A diagram of a product

Description automatically generated

**7. Screnshots**







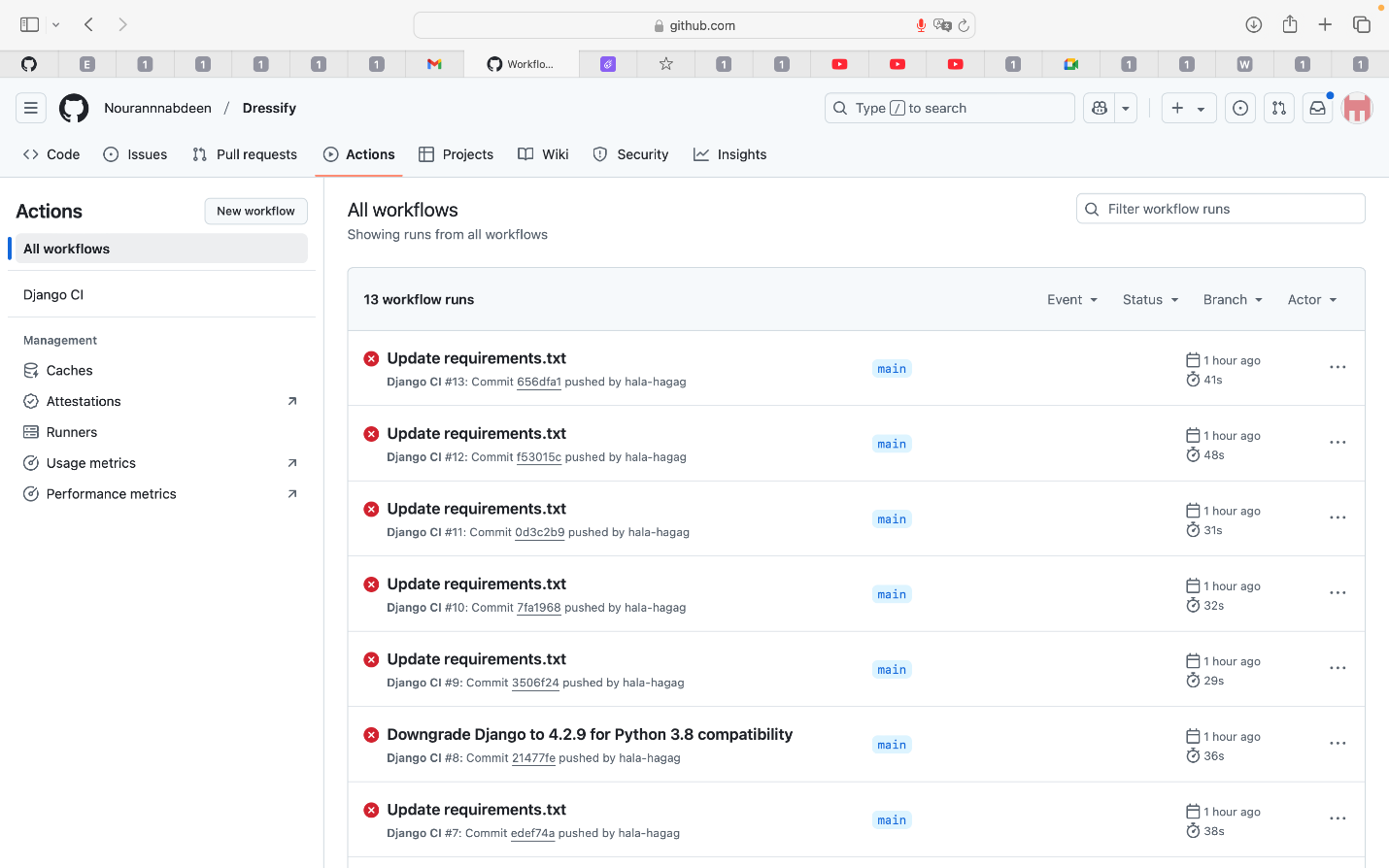
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

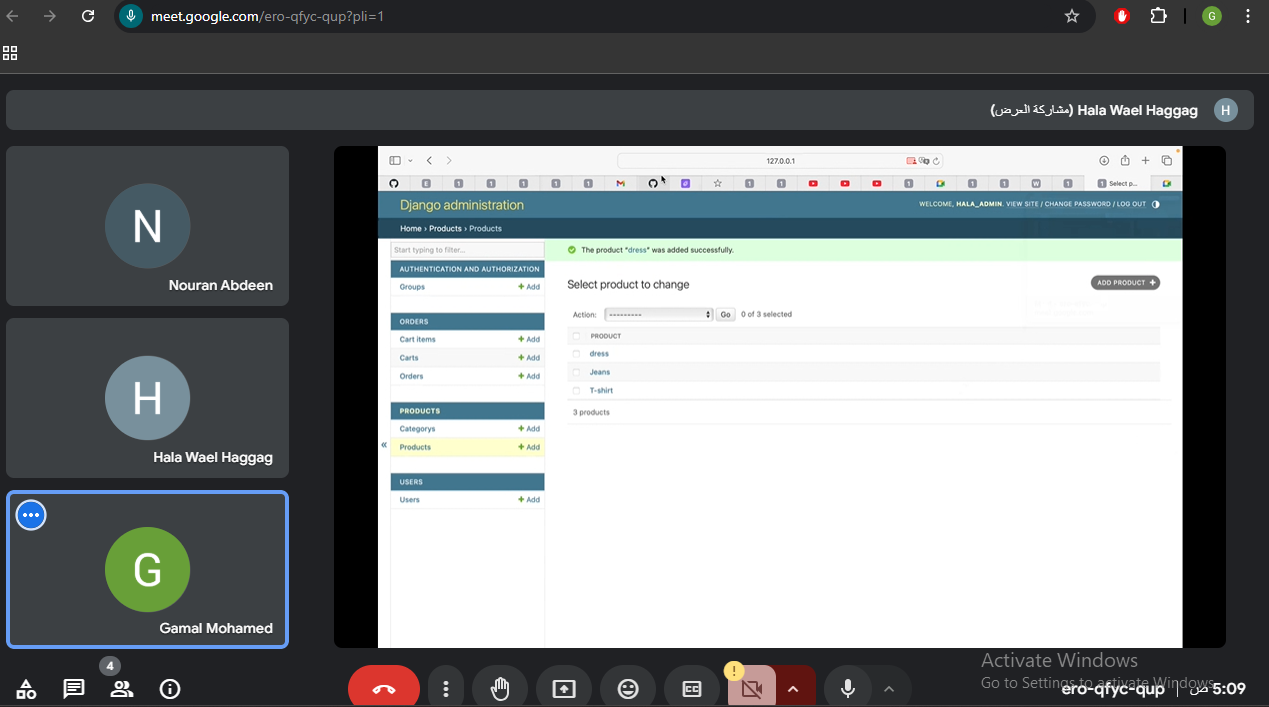
**8. Test Cases**

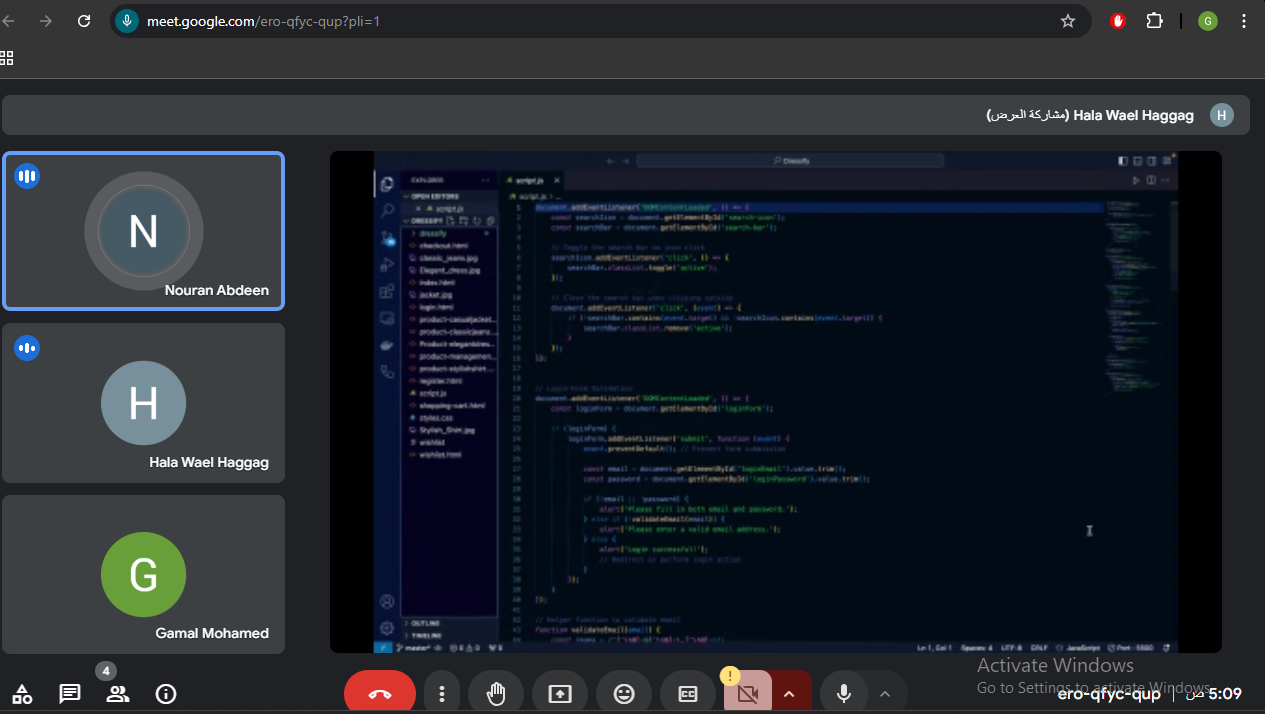


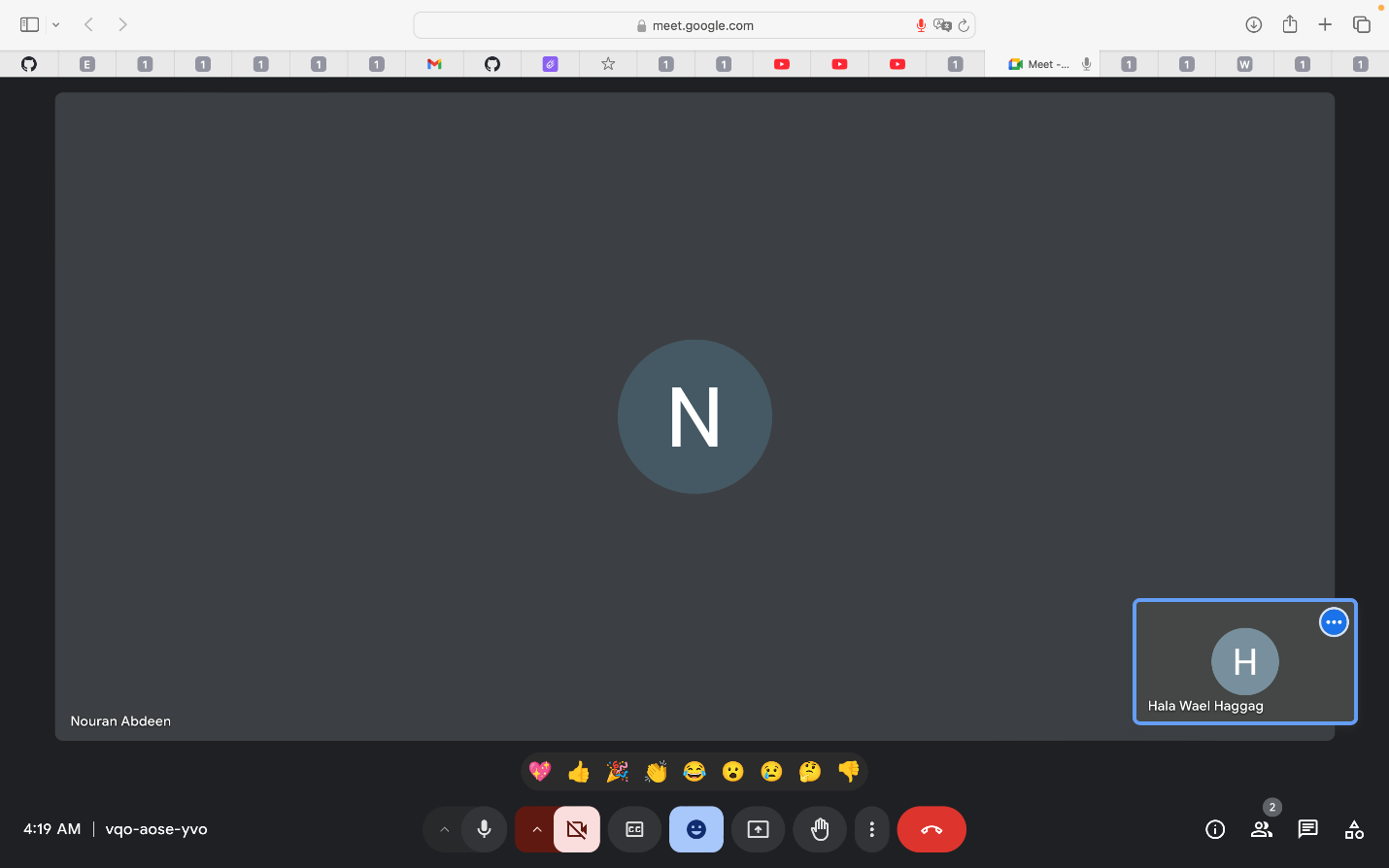
**9. Risk Analysis**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Risk | Description | Liklhood | Impact | Mitigation Strategy |
| Service Interruptions | The application may experience downtime due to issues with the server or database infrastructure. | Medium | Critical | Utilize cloud-based database solutions and ensure high availability of servers. |
| Overloading System Resources | System performance degradation when handling a large number of concurrent users. | Medium | High | Optimize codebase for better resource management and load testing to handle peak traffic. |
| Integration Failures | Issues with connecting services like AI features or third-party APIs could cause disruptions. | Low | Medium | Ensure thorough testing of all integrations and implement fallback mechanisms for third-party failures. |
| User Interface Issues | Poor user interface design or unexpected user behaviors might reduce the app’s usability. | High | Moderate | Focus on conducting user testing, gathering feedback, and refining the design to ensure simplicity and intuitiveness. |
| User Experience Problems | Complex or unintuitive UI could reduce the app’s usability, leading to user dissatisfaction. | High | Moderate | Gather user feedback regularly and improve UI elements for better user experience. |
| Version Control Conflicts | Development team members might face conflicts when merging changes in the codebase, leading to delays or errors in the system. | Medium | Low | Establish clear Git workflow guidelines, conduct frequent code reviews, and encourage better collaboration among team members. |

**10. Scrum Meetings**

****

****



**11. Conclusion**

This document outlines the full scope of the **Dressify** project, covering both functional and non-functional requirements. By adhering to these guidelines, the system will meet the needs of customers, admins, and other stakeholders. The project aims to provide a reliable, secure, and user-friendly e-commerce platform for fashion enthusiasts.