### **Project Title**: Product Catalog and Order Management System

### **Author**: Tahmid Ayman Alam

### **Document Version**: FINAL

### **TABLE OF CONTENTS**

|  |
| --- |
| **Introduction** |
| **Fuctional Requirments** |
| **Non- Fuctional Requirments** |
| **Tools and Technologies** |
| **Compatibility with System Environment** |
| **Challenges Faced** |
| **Conclusion** |
| **Implementation (Sprint 1,2,3)** |
| **Deployment issue error/fix report** |

### 

### **1. Introduction**

#### **1.1 Type of Project:**

This project is a **Web Application** developed using the **LAMP stack** (Linux, Apache, MySQL, PHP), following the **MVC architecture**.

#### **1.2 Purpose:**

This project is to create a simple yet functional product catalog where we can browse products, manage cart, and track order statuses. An admin interface is included to manage products and order statuses, which follows CRUD operation.

**1.3 Target Users:**

* **End User:**Customers who want to browse product cataglos, add iterms to their cart, and track order status will have separate account to deal with such operation. Their order status and other info will be updated real time in their dashboard.
* **Admin Panel:** Administrators who can mangae products, orders, and view order details.

### **2. Functional Requirements**

1. **Product Listing**: Allow display a catalog of products with images, descriptions, and prices.
2. **Product Filtering**: Allow users to filter products by category, price range, and availability.
3. **Product Details Page**: Show detailed information about a product when clicked.
4. **Add to Cart**: Users can add products to their shopping cart.
5. **View Cart**: Display all items in the shopping cart with product names, quantities, and total price.
6. **Update Cart**: Users can modify quantities or remove items from their cart.
7. **Order Confirmation**: Users can proceed to checkout, confirm the order, and provide shipping details.
8. **Order History**: Users can view a list of past orders.
9. **Order Status Tracking**: Users can check the status of their orders (e.g., "Pending", "Shipped").
10. **Admin Dashboard**: Admins can add, edit, remove products, and manage order statuse

### **3. Non-functional Requirements**

1. **Performance**: The website should not take too long to load. It should be user friendly, without any bugs, and easy to coordinate.
2. **Security**: Data input must be validated to avoid SQL injection and XSS vulnerabilities.
3. **Availability**: The system must be available 24/7, with minimal downtime for maintenance.
4. **Scalability**: The system should be able to handle increased traffic and products as the user base grows.

### **4. Tools and Technologies**

#### **4.1 Frontend:**

* **HTML5**, **CSS3**, **JavaScript** (for basic front-end interactions)
* **Bootstrap** for responsive design

#### **4.2 Backend:**

* **PHP** for server-side scripting and logic
* **MySQL** for database management
* **Apache** as the web server

#### **4.3 Development Environment:**

* **XAMPP** for local development (includes Apache, MySQL)
* **phpMyAdmin** for database management

### **5. Compatibility with System Environment**

The system will be developed and tested on the following environments:

* **Operating System**: Linux (Ubuntu) for development and deployment.
* **Web Server**: Apache.
* **Database**: MySQL.

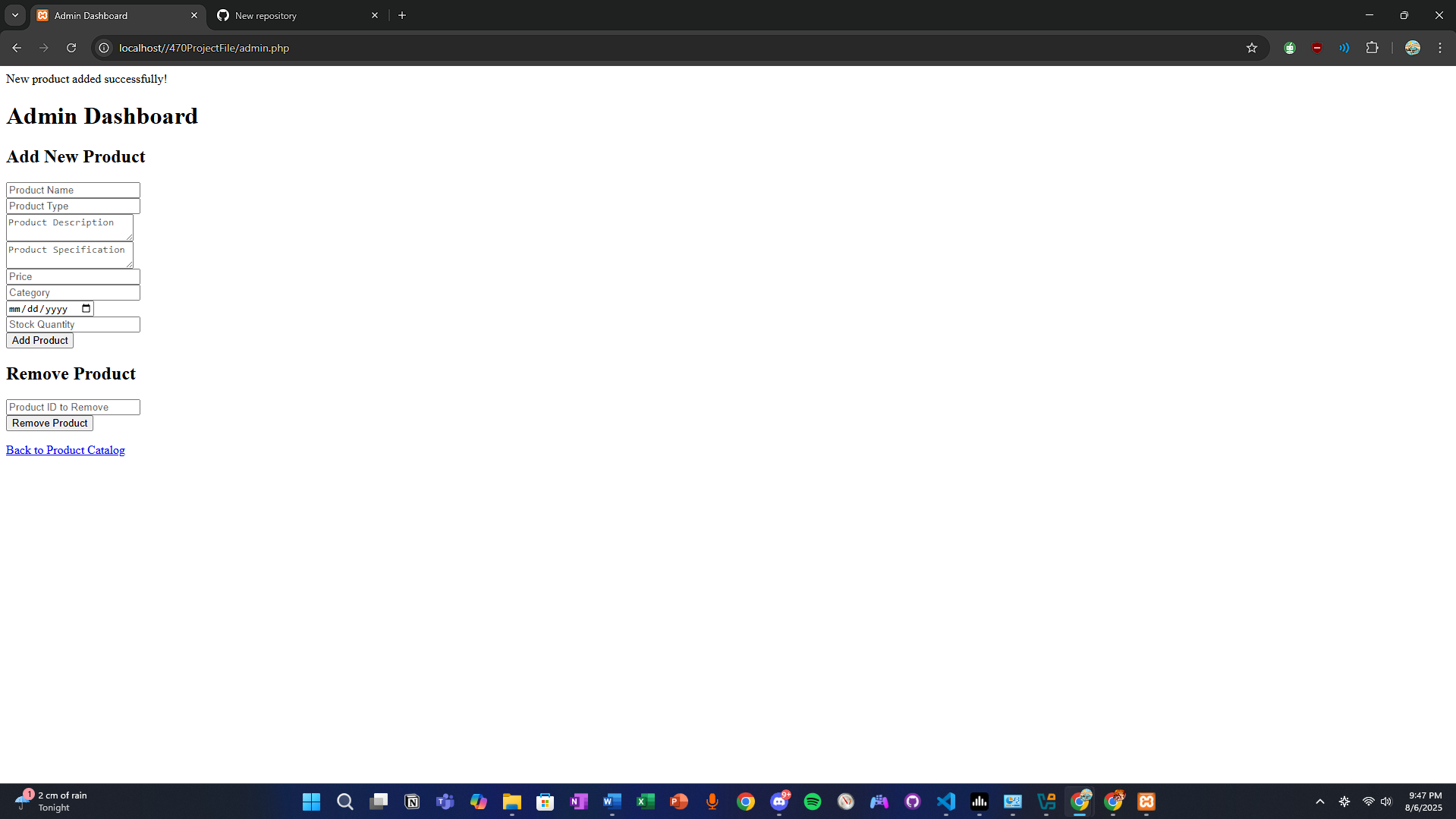
### **6. Challenges**

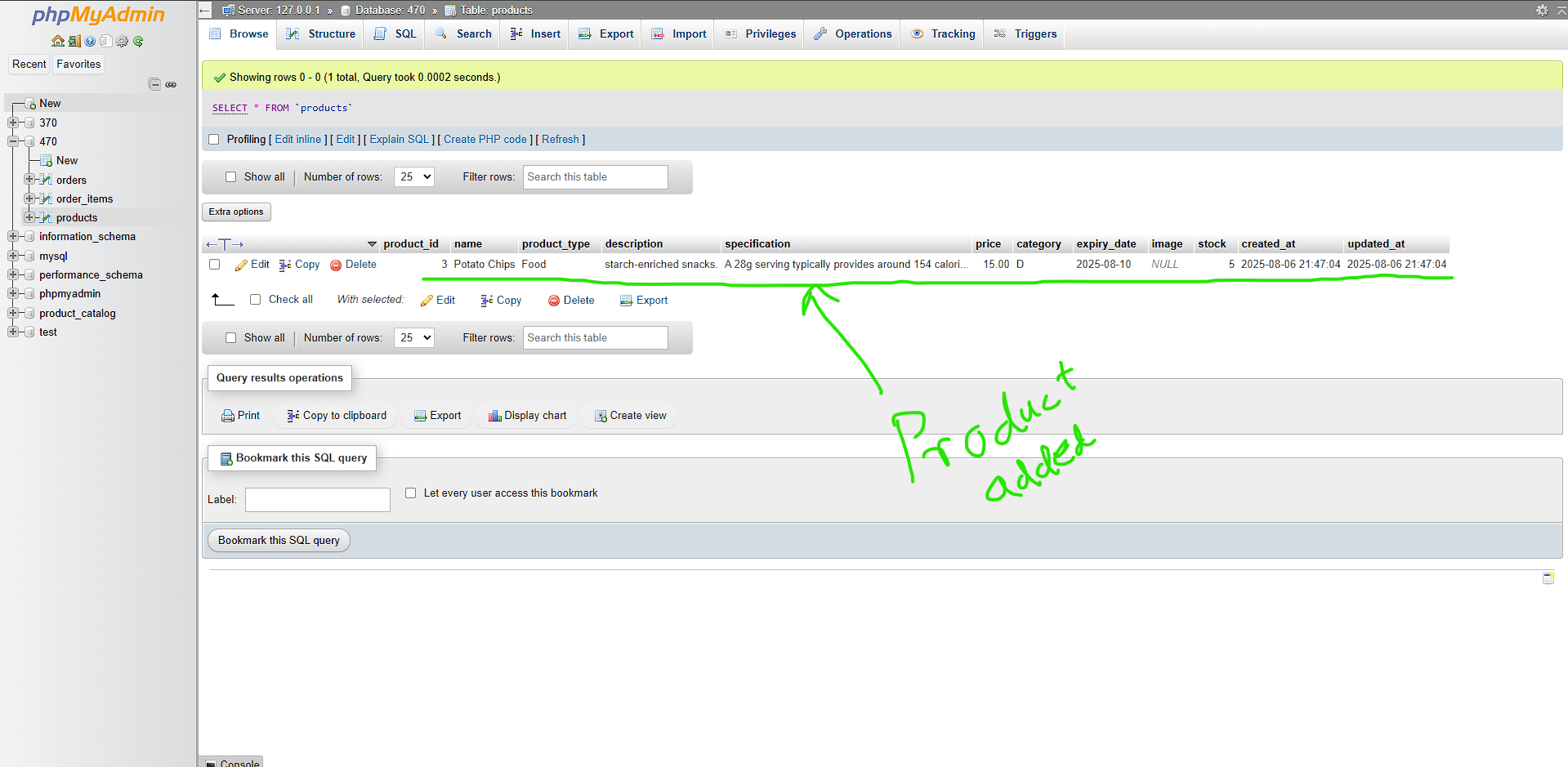
* **Image Handling**: Managing product images in a way that ensures quick loading and easy management.
* **Session Management**: Maintaining the state of the shopping cart for each user across different pages.
* **Security Concerns**: Ensuring the application is protected from common web vulnerabilities like SQL injection and XSS attacks.
* **Error Handle:** Deployment issue from linux was especially challenging as I had to manually set permissions and server to work locally from PC.

### **7. Conclusion**

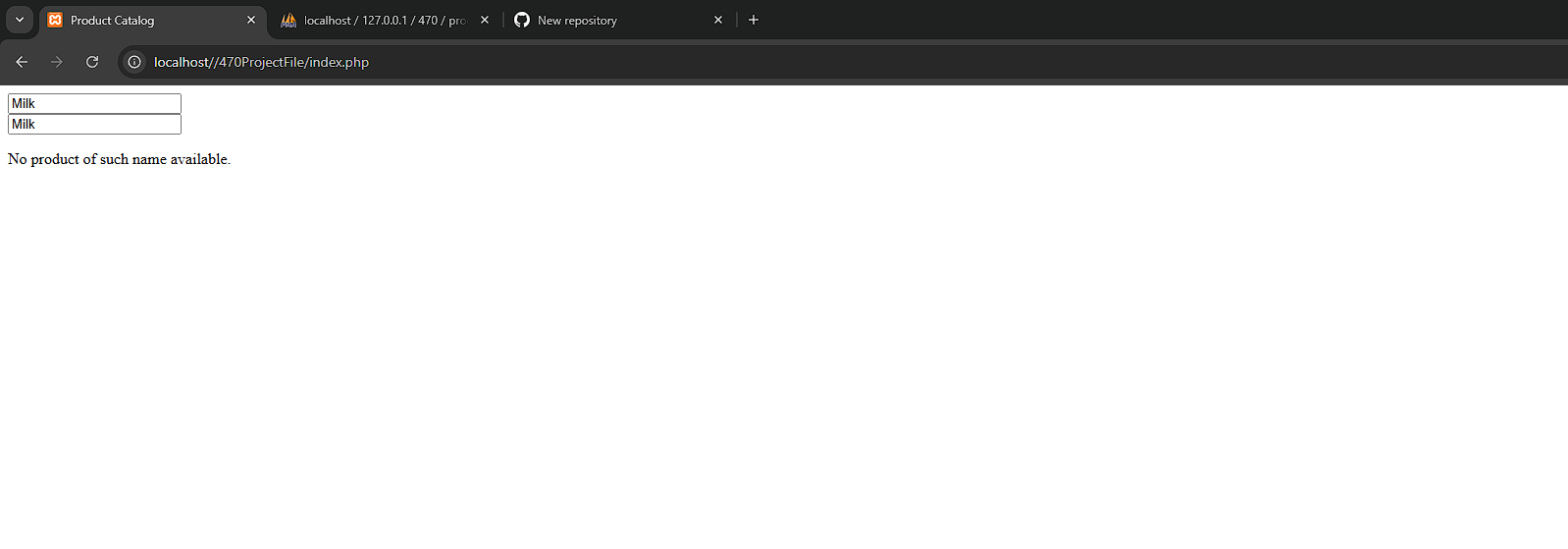
This project is designed to provide both functional and practical experience in web development using the LAMP stack. It will cover essential features like product listing, cart management, and order tracking while following the MVC architecture. The system will be scalable and secure, with future potential to add more features.

### **8. Sprint (1):**

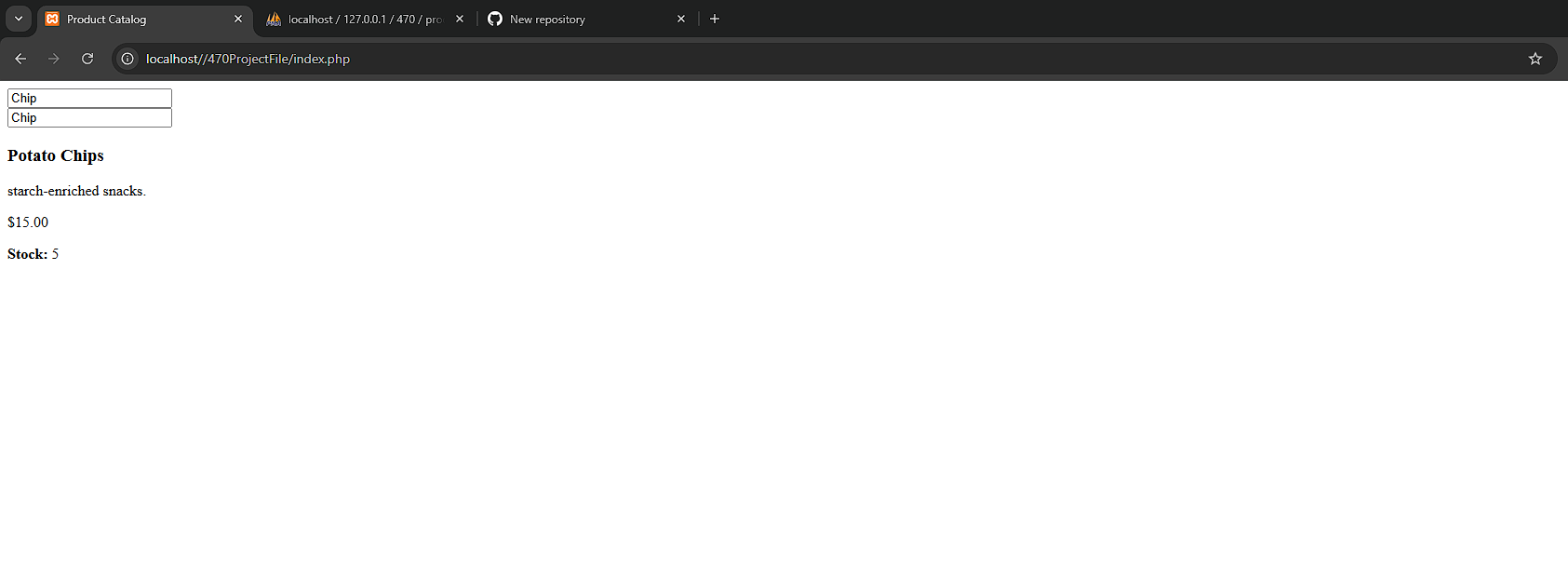




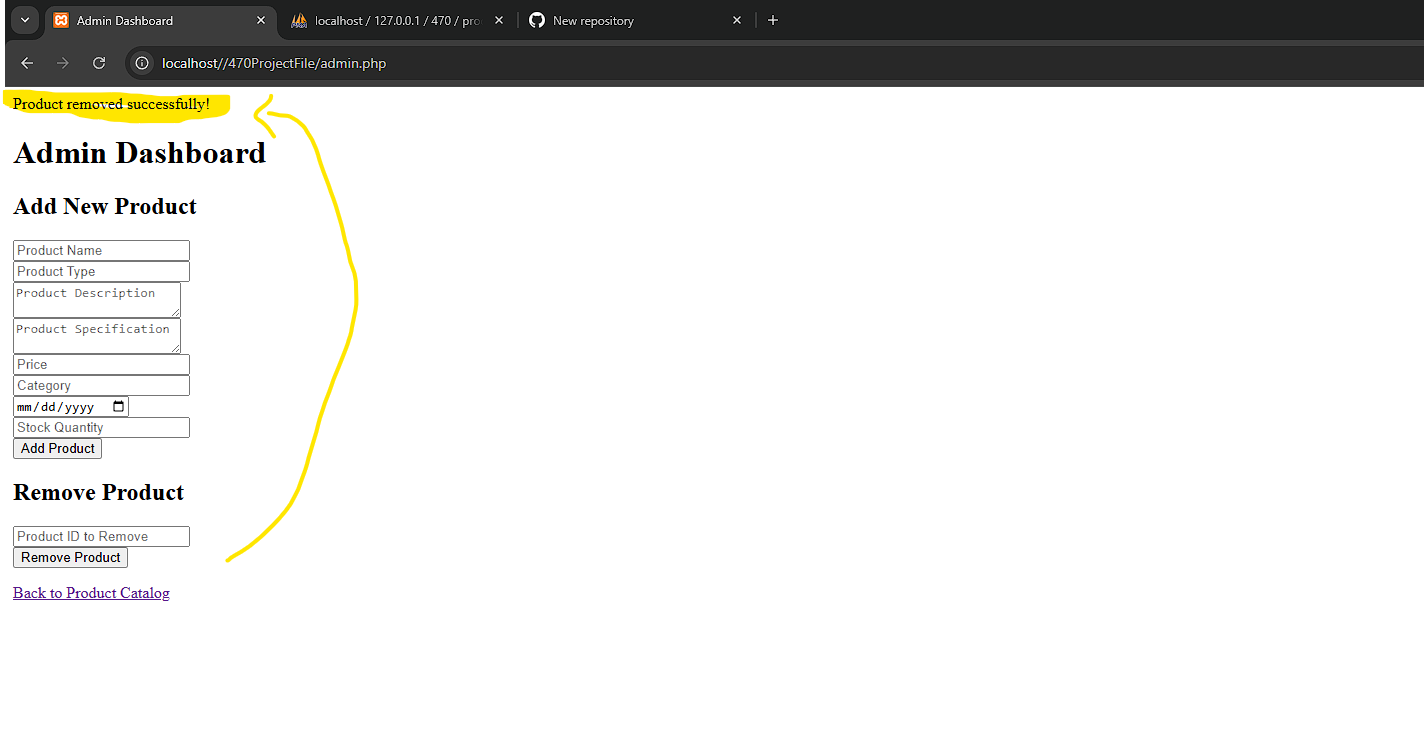
The above two screen shots shows product details being added into the database after entering information in admin page.



From user side, there is a search bar where users can filter out the name of the catalouge by fetching data from database. If product is not available or name doesn’t match, it pops up the message: “No product of such name available”

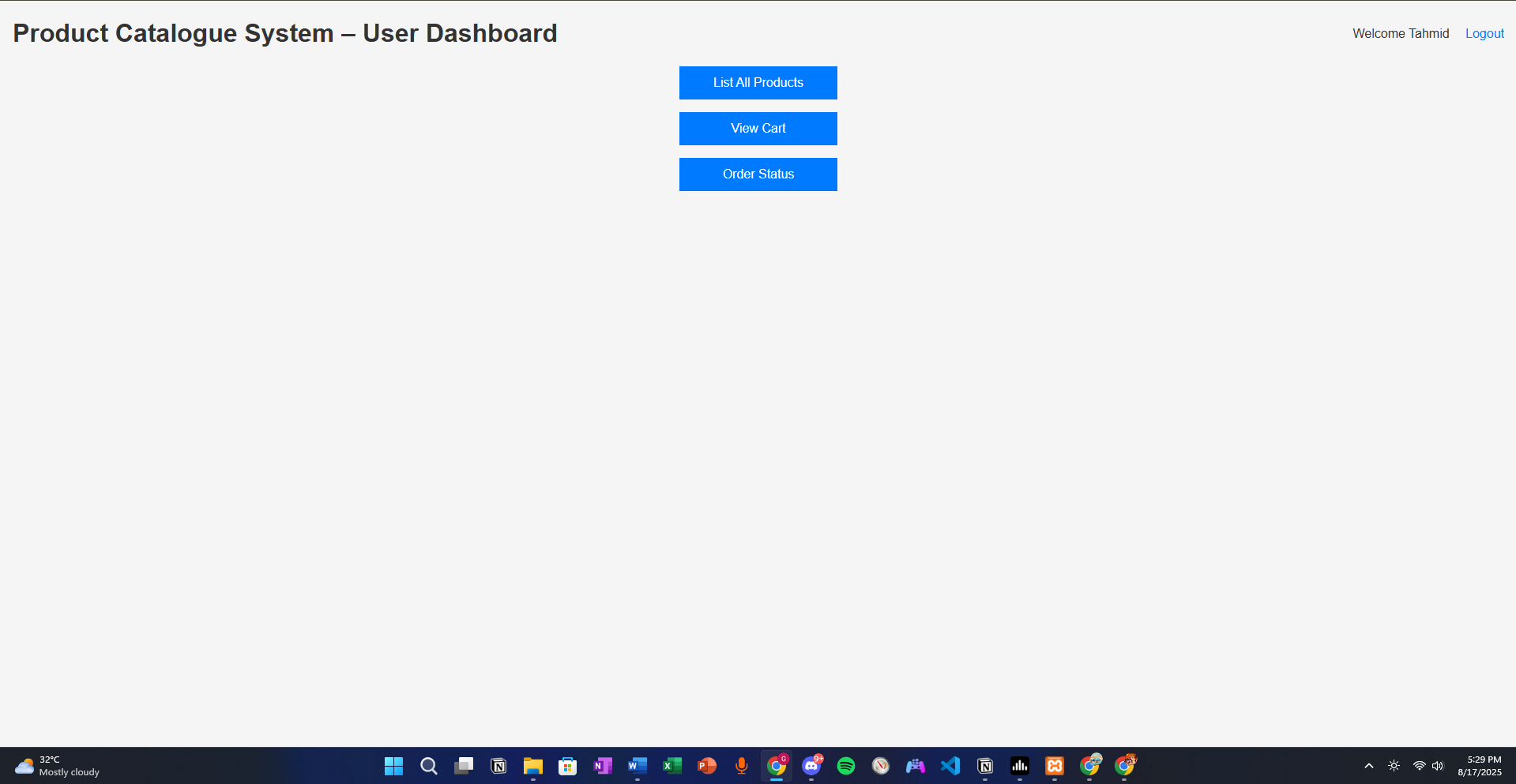


As in the above example, we added “Potato Chips” as soon as we entered chips, it shows the product details suggesting it was added from admin.



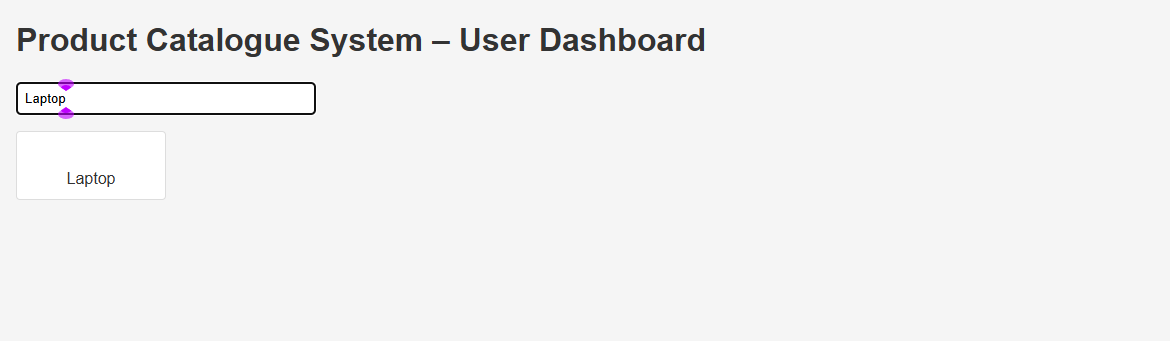
Product can be removed by admin by entering the correct Product id from data sql database and doing so pops the message “Product removed successfully”

### **Sprint 2:**

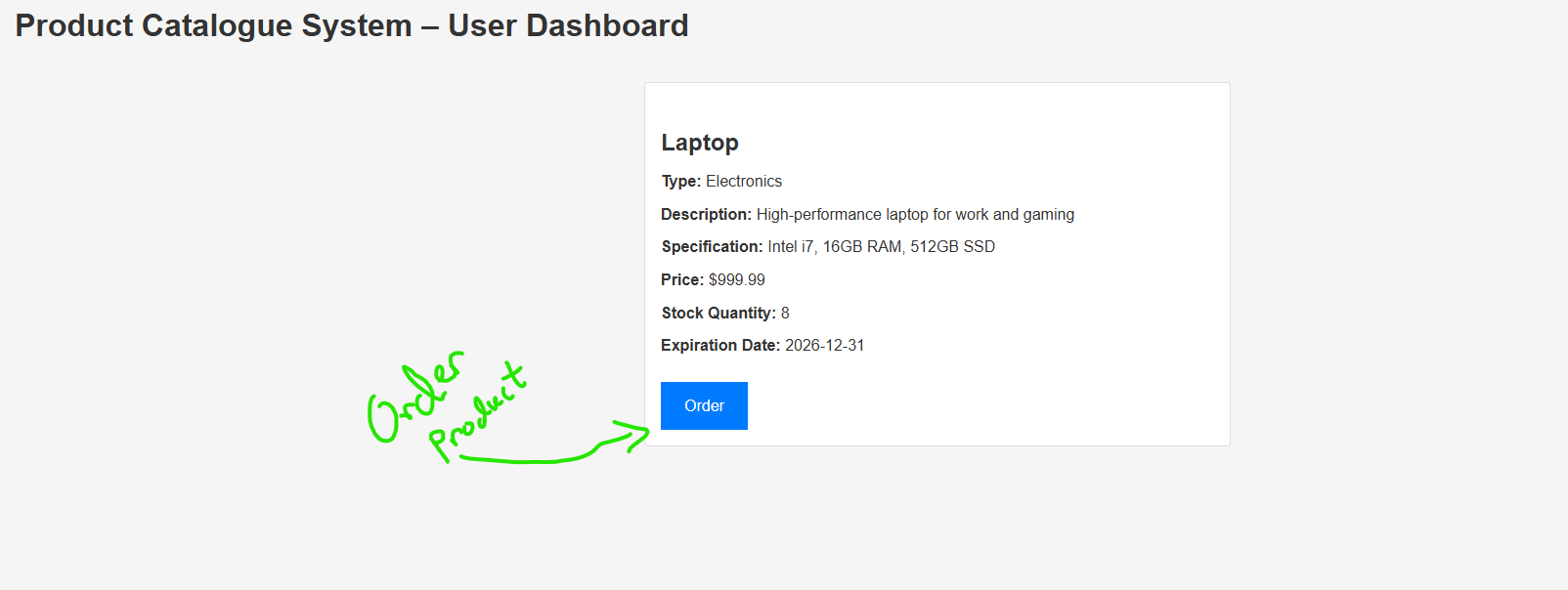


This is the user dashboard. These buttons will allow the user to prompt to the page to use the feature needed.

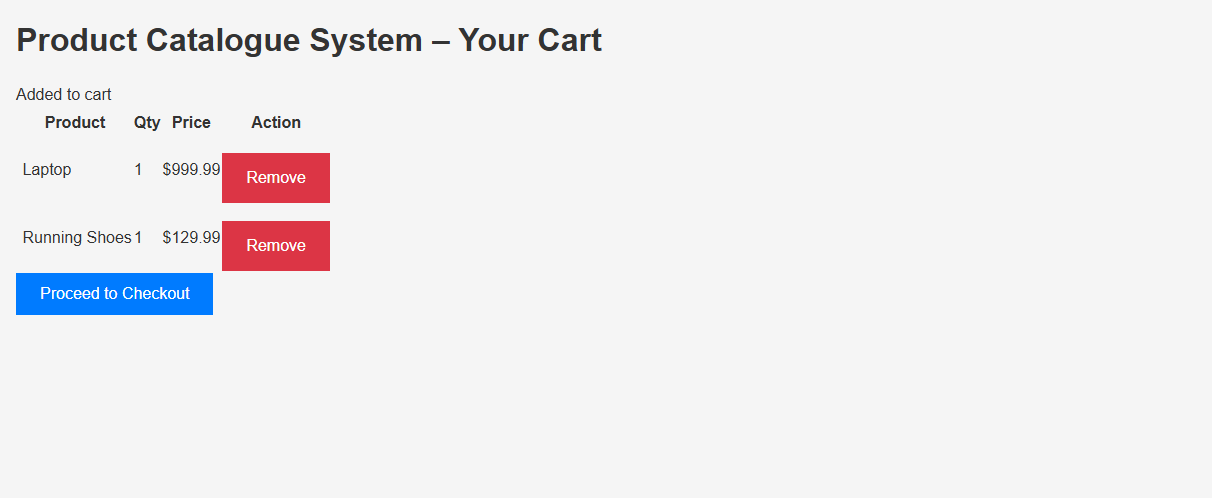
Feature 1: (Search and Filter)



All product cards are displayed in the HTML but they are set to hidden by default. A JavaScript listener on the search bar updates the visibility of cards in real time, showing only those whose names include the type text. As such, No page reload is needed --- filtering happens instantly as the user types.

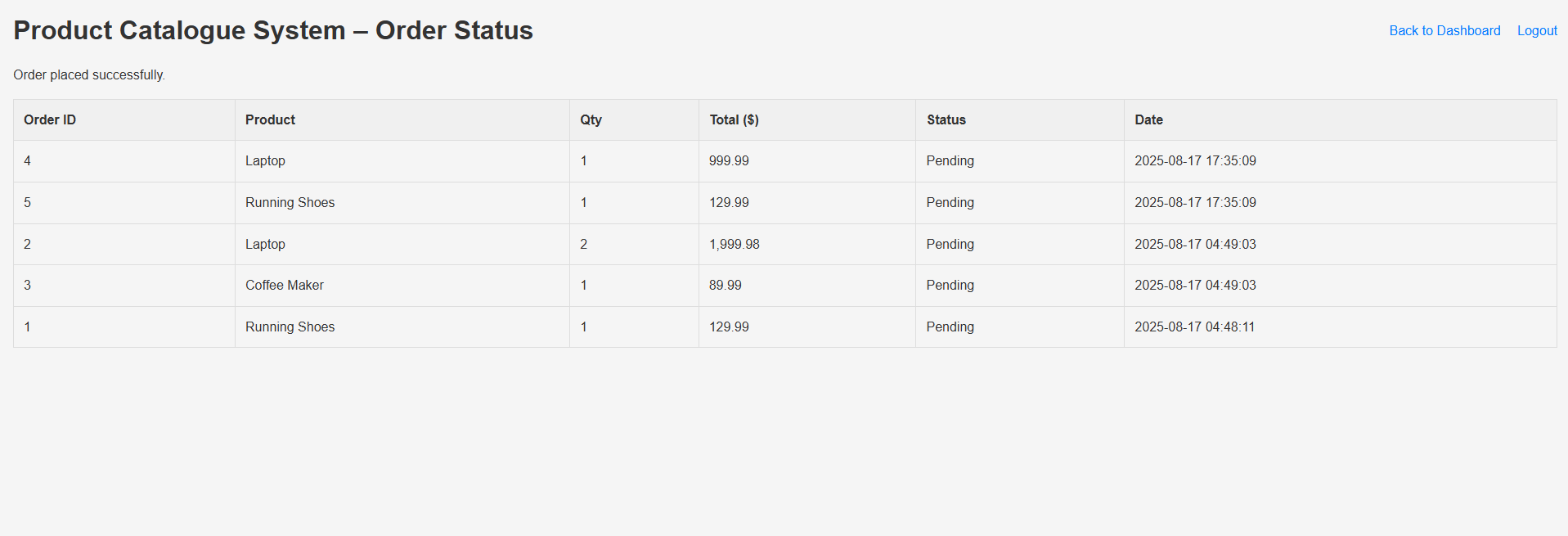


Feature 2 & 3: (View cart order, and Proceed to checkout/remove order)



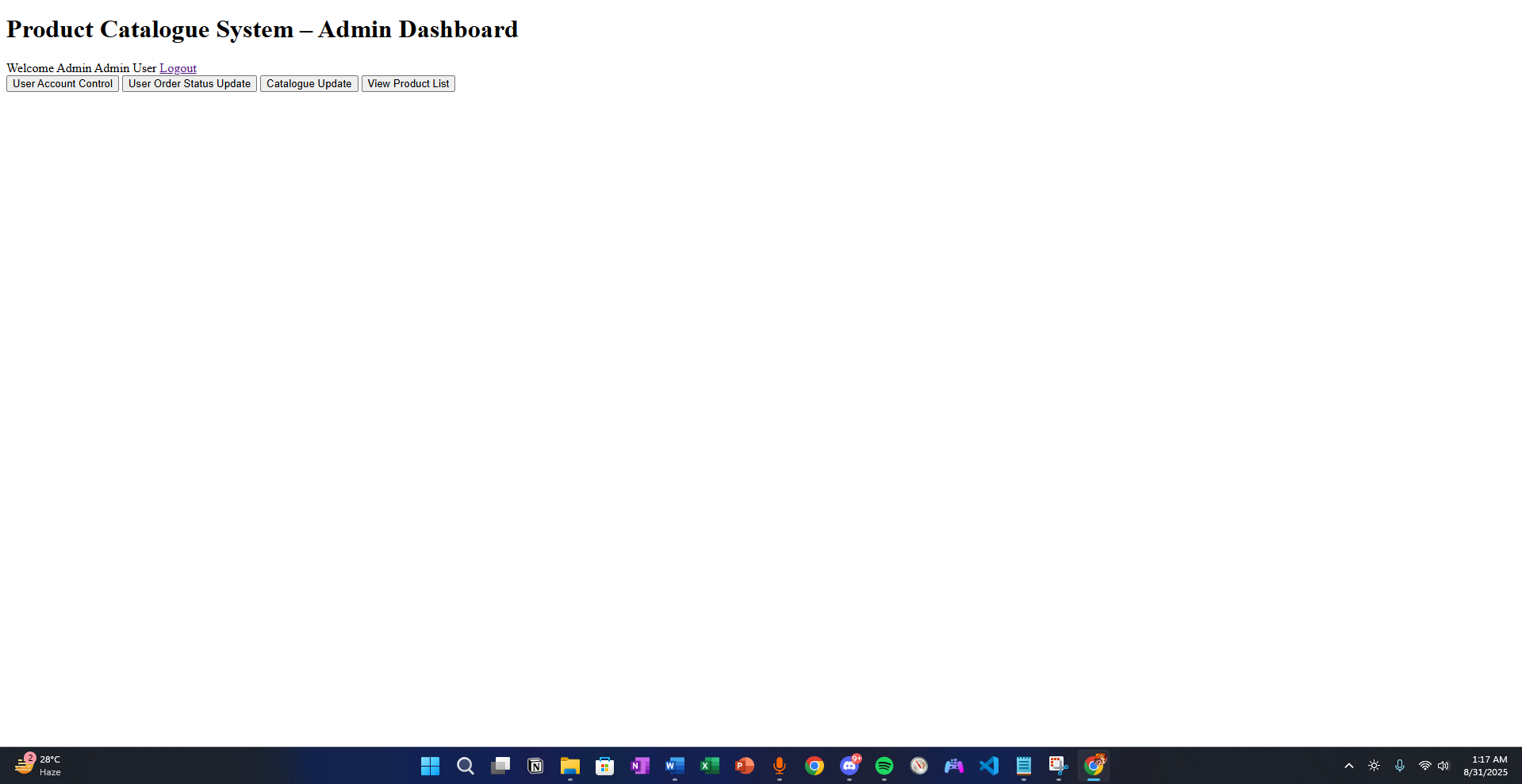
(For view car): Cart items are fetched for the user from the database. Each item is displayed in a tble with product name, quantity, and price. A “remove” button is included that deletes the item and refreshes the cart.

Feature 4: (Order status)

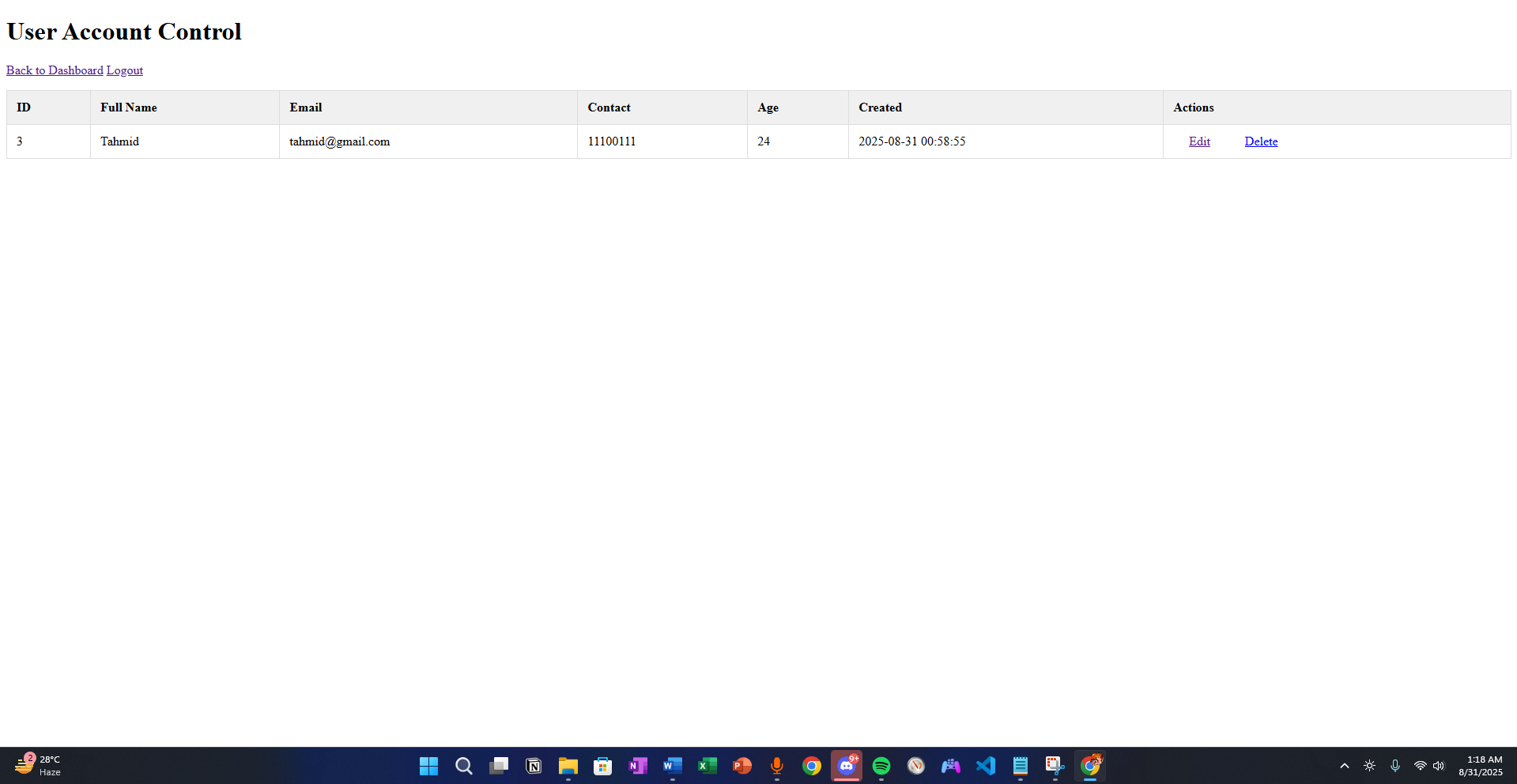


(For order status): All orders belonging to the user are retrieved to the user from the database. This items are shown in a table with order ID, product info, quantity, status etc. Status updates reflect order progress, which changes in real time if changed from admin side.

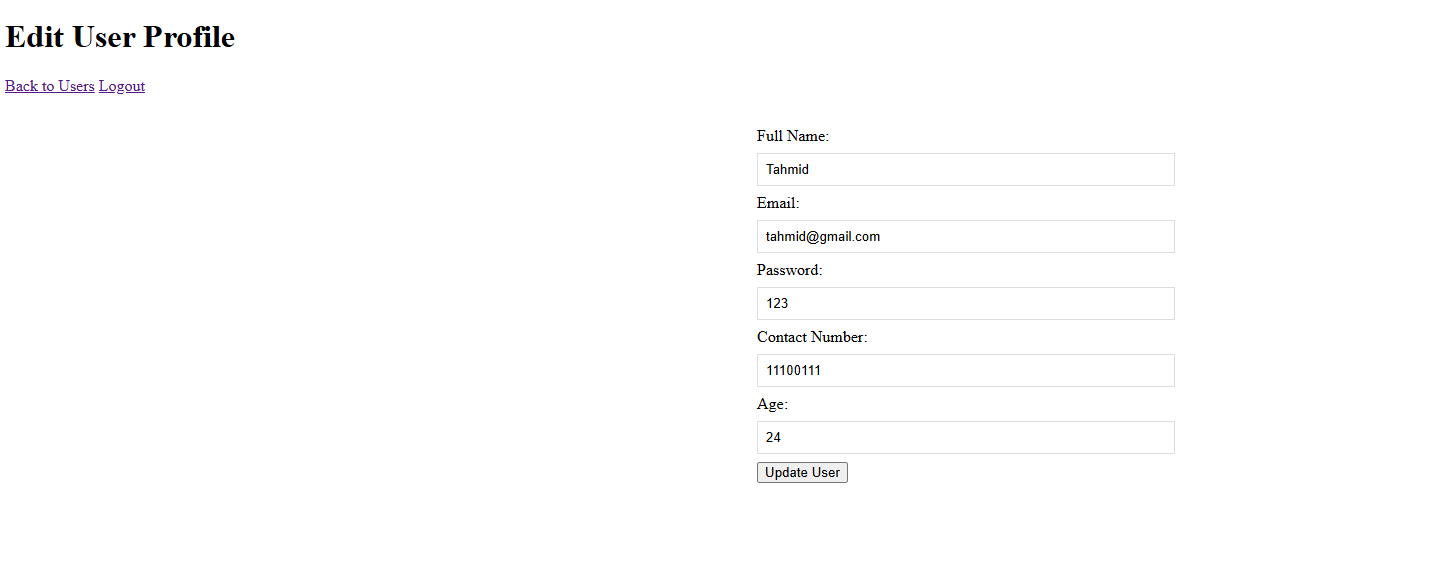
### **Sprint 3: (Final features: admin panel)**



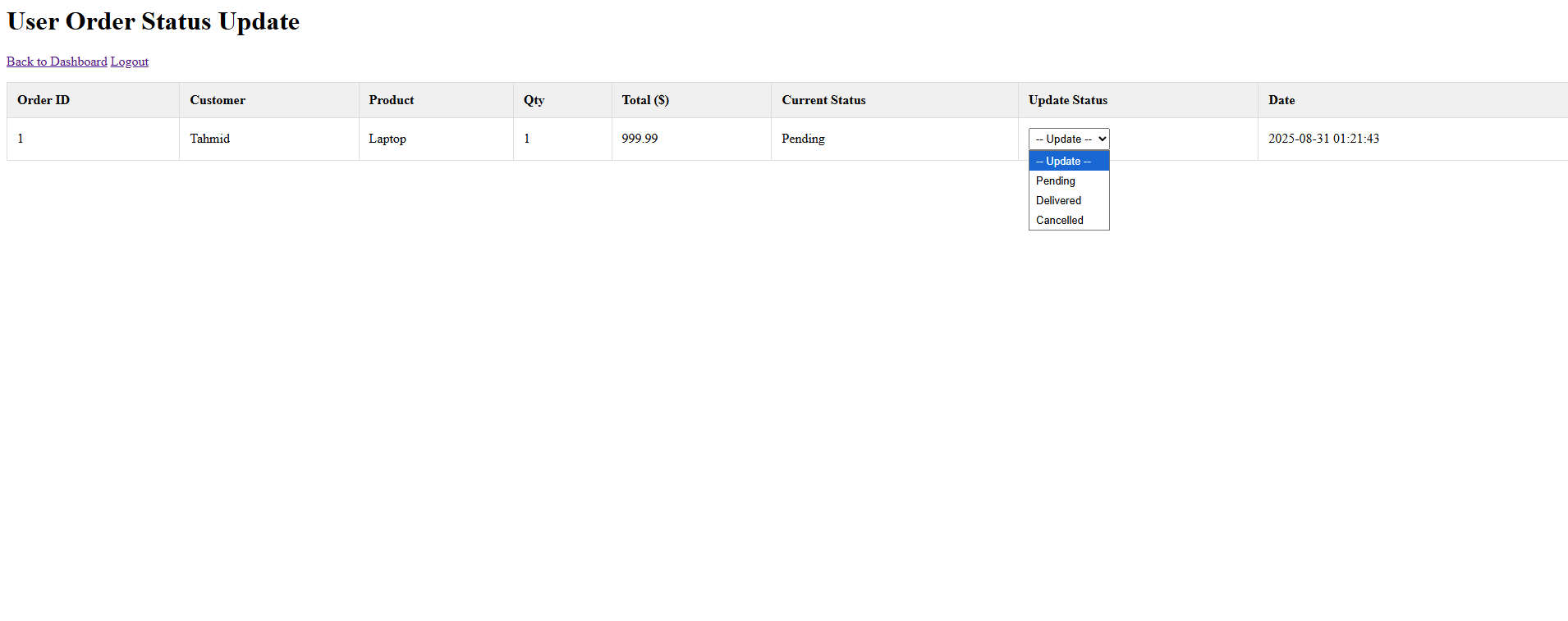
In admin dashboard, we have 4 options to interact with. Each redirects us to a their respective workspace.

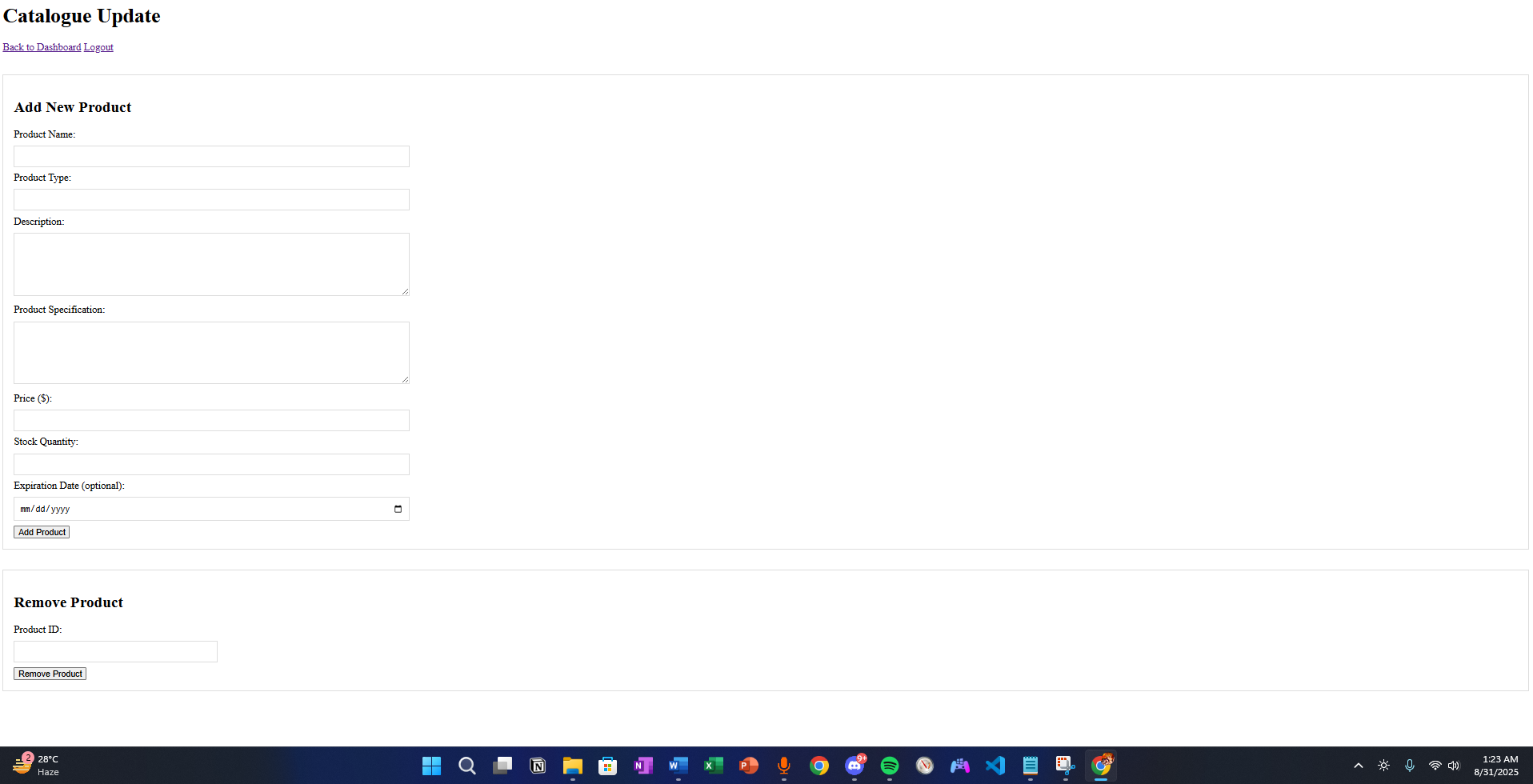


User control workspace shows list of all users who registered an account. There is the option to edit them (their profile info) or permanently delete their account from the database.

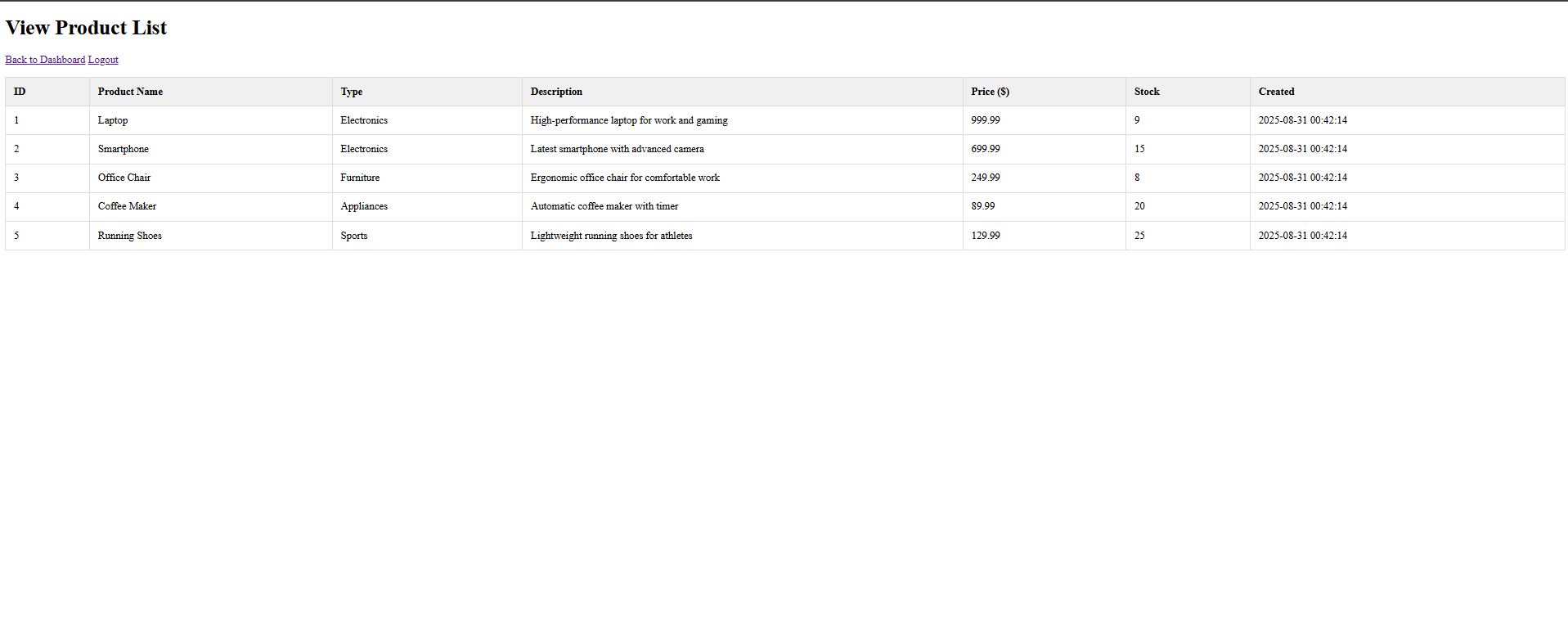


We can edit any of the above info, which will get updated in the “users” table of the database.



In user order status, we will not only be able to see list of all orders from user side, but also the user who ordered plus update their order’s status.

This page allows the admin to add product and their descriptions into the catalouge alongside the feature to remove any product by using their id. Each product added in catalouge will have a unique id which can be found from “view product” list in dashboard page of admin.



This is the “Product list” mentioned earlier. Whatever we add/remove from the database will be reflected here.

### **9. Deployment report : Fulfiling LAMP Stack criteria.**

During deployment on the WSL 2 Ubuntu environment, several challenges were encountered:

1. **Folder Structure & Apache Configuration**  
   The initial deployment failed because the Apache virtual host pointed incorrectly to the *project root* rather than the ***public*** directory. Adjusting the virtual host **DocumentRoot** to the **public**folder and enabling **.htaccess** overrides solved routing issues.
2. **Apache mod\_rewrite & .htaccess**  
   Apache’s ***mod\_rewrite*** module was disabled by default, causing URL rewriting failures. Enabling ***mod\_rewrite*** and creating a *proper .htaccess* file with correct rewrite rules ensured routing to the front controller index.php worked properly.
3. **Database Import & Connection**  
   The MySQL server initially pointed to an old or empty database—the schema and sample data were missing in WSL’s MySQL instance, leading to “table not found” errors. Dropping the existing incomplete database and freshly importing the provided SQL schema resolved these errors. The old database was found to be from a tested XAMPP environment, which was unreadable by WSL 2.
4. **PHP Include Paths**  
   Moving the web root to the public directory introduced incorrect relative paths in PHP includes and requires. Updating the scripts to correct \_\_DIR\_\_ relative paths fixed all autoload and class loading errors.
5. **PHP Error Reporting**  
   Initial silent failures were traced to disabled PHP error display and missing error log configurations. Enabling error display and setting up error logging exposed hidden fatal errors, aiding debugging. This was done by manually by updating ***php.ini***  turning on default\_errors, log\_errors, error\_log set to php.log from etc/php in ubuntu.
6. **File Permissions**  
   Correct file ownership and permissions for www-data ensured Apache could read and serve files securely without permission issues. (Using sudo chown commands)
7. **Service Management**  
   Understanding manual start/stop/restart commands for Apache and MySQL and WSL instance management allowed reliable control over development and demonstration uptime.