**OPTICLOUD- AN ECOMMERCE VENTURE**

**DISSERTATION**

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE OF

**MASTER**

**IN**

**COMPUTER APPLICATIONS**

Under The Supervision Of

**Prof. AASIM ZAFAR**

By:-

**SANA**

Faculty No. 23CAMSA142

Enrollment No. GK1240



**DEPARTMENT OF COMPUTER SCIENCE**

**ALIGARH MUSLIM UNIVERSITY**

**ALIGARH(INDIA)**

**2024-2025**

# **Acknowledgement**

First of all, I acknowledge my gratitude to **Almighty God** for enlightening me with the power of knowledge. It is great pleasure to welcome this opportunity to extend my heartfelt thanks and a sense of gratitude to everybody who helped me throughout the successful completion of this project.

I am extremely grateful and remain indebted to my supervisor **Prof. Aasim Zafar Sir** , for being a source of inspiration and for his constant support in the Design, Implementation and Evaluation of the project.

I am thankful to them for constant constructive criticism and invaluable suggestions, which benefited me a lot while developing the project on **“OptiCloud- an ecommerce venture”**. They have been very co-operative and helping throughout this whole project work.

My thanks and appreciations also go to my friends and the people who have willingly helped me out with their abilities.

Finally, I gratefully acknowledge the support, encouragement & patience of my family. Thank You!

**SANA**

**23CAMSA142**

**GK - 1240**

# 

# 

TABLE OF CONTENT

**Certificate**……………………………………………………………………………………………...1

[**Acknowledgement** 2](#_Toc199337416)

[**Abstract** 6](#_Toc199337417)

[**CHAPTER-1:**](#_Toc199337419) [INTRODUCTION 8](#_Toc199337420)

[1.0 Introduction 10](#_Toc199337421)

[1.1 Purpose 10](#_Toc199337425)

[1.2 Document Convention 11](#_Toc199337426)

[1.3 Intended Audience and Reading Suggestions 11](#_Toc199337427)

[1.4 Product Scope 12](#_Toc199337428)

[**CHAPTER-2:**](#_Toc199337430) [PROBLEM FORMULATION 13](#_Toc199337431)

[2.0 Problems in Existing System: 13](#_Toc199337432)

[2.1 Various aspects of the problem: 14](#_Toc199337433)

[2.2 Benefits of system proposed 15](#_Toc199337434)

[**CHAPTER-3:**](#_Toc199337436) [SYSTEM ANALYSIS AND DESIGN 17](#_Toc199337437)

[3.0 Overall Description 17](#_Toc199337438)

[3.0.1 Product Functions 17](#_Toc199337439)

[3.0.2 User Classes and Characteristics 19](#_Toc199337440)

[3.0.3 Operating Environment 20](#_Toc199337441)

[3.0.4 Design and Implementation Constraints 20](#_Toc199337442)

[3.0.5 User Documentation 21](#_Toc199337443)

[3.0.6 Assumption and Dependencies 22](#_Toc199337444)

[**CHAPTER-4:**](#_Toc199337446) [SYSTEM IMPLEMENTATION 23](#_Toc199337447)

[4.0 External Interface Requirement 24](#_Toc199337448)

[4.1 System Features 27](#_Toc199337452)

[4.1.1 System Feature 27](#_Toc199337453)

[4.2 Other non-functional Requirement 31](#_Toc199337454)

[4.3 System testing 33](#_Toc199337460)

[**CHAPTER-5:**](#_Toc199337462) [RESULT AND DISCUSSION 36](#_Toc199337463)

[5.1 Interpretation of result 36](#_Toc199337464)

[5.2 Application area and scope of work 37](#_Toc199337465)

[**CHAPTER-6:**](#_Toc199337467) [CONCLUSION 39](#_Toc199337468)

[6.0 Product Scope 40](#_Toc199337469)

[6.1 Limitations 41](#_Toc199337470)

[**APPENDIX** 43](#_Toc199337472)

[**A.1 SDLC Model:** 44](#_Toc199337473)

[**A.2 Entity-Relationship Diagram (ERD)** 46](#_Toc199337474)

[**A.3 USE-CASE Diagram:** 47](#_Toc199337475)

[**A.4 Activity Diagram:** 49](#_Toc199337476)

[**A.5 Data Flow Diagram (DFD):** 51](#_Toc199337477)

[**A.6 Class Diagram:** 54](#_Toc199337478)

[**CODING** 57](#_Toc199337481)

[ pages/shop/ShopPage.jsx 59](#_Toc199337482)

[ search/Search.jsx 61](#_Toc199337483)

[ components/Login.jsx 62](#_Toc199337484)

[ user/OrderDetails.jsx 67](#_Toc199337485)

[ user/UserProfile.jsx 68](#_Toc199337486)

[ pages/dashboard/user/UserReviews.jsx 71](#_Toc199337487)

[ pages/dashboard/AdminDashboard.jsx 72](#_Toc199337488)

[ pages/dashboard/admin/AddProduct.jsx 73](#_Toc199337489)

[ pages/dashboard/admin /ManageProduct.jsx 74](#_Toc199337490)

[ pages/dashboard/admin/users/ManageUser.jsx 77](#_Toc199337491)

[ pages/dashboard/admin/users/UpdateUserModal.jsx 79](#_Toc199337492)

[ pages/dashboard/admin/manageOrders /ManageOrders.jsx 80](#_Toc199337493)

[ pages/dashboard/admin/manageOrders /UpdateOrderModal.jsx 82](#_Toc199337494)

[**SCREENSHOTS** 84](#_Toc199337496)

[◆ Home.jsx 85](#_Toc199337497)

[◆ ShopPage.jsx 86](#_Toc199337498)

[◆ search/Search.jsx 87](#_Toc199337499)

[◆ components/Login.jsx 87](#_Toc199337500)

[◆ user/dashboard/UserDMain.jsx 88](#_Toc199337501)

[◆ user/UserOrders.jsx 88](#_Toc199337502)

[◆ user/OrderDetails.jsx 89](#_Toc199337503)

[◆ user/UserProfile.jsx 89](#_Toc199337504)

[◆ user/UserReviews.jsx 90](#_Toc199337506)

[◆ dashboard/AdminDashboard.jsx 91](#_Toc199337507)

[◆ admin/AddProduct.jsx 91](#_Toc199337508)

[◆ admin/ManageProduct.jsx 92](#_Toc199337510)

[◆ admin/users/ManageUser.jsx 93](#_Toc199337511)

[◆ admin/users/UpdateUserModal.jsx 93](#_Toc199337512)

[◆ admin/manageOrders /ManageOrders.jsx 94](#_Toc199337513)

[◆ admin/manageOrders /UpdateOrderModal.jsx 94](#_Toc199337514)

# **Abstract**

OptiCloud: An E-commerce Venture is a full-fledged web-based e-commerce application developed to facilitate the seamless online sale and inventory management of optical lenses. Designed as part of an academic project, OptiCloud aims to bridge the gap between customers and optical retailers by providing a digital platform where a variety of lenses can be browsed, purchased, and reviewed, all from a single interface.

The platform is structured to cater to two primary user types: Customers and Administrators. Customers can register, browse through the product catalog, filter and search for specific lenses, add products to their shopping cart, place and track orders, make secure payments, and submit feedback or reviews. The system provides a smooth and intuitive interface to enhance user experience and satisfaction.

On the other hand, Administrators have access to a dedicated dashboard that allows them to manage core aspects of the platform, including updating product information, adjusting pricing, monitoring stock levels, processing orders. A key feature of OptiCloud is its integrated inventory management system, which ensures real-time tracking of product availability and stock adjustments after each transaction. This minimizes stockouts and enhances operational efficiency.

Technically, OptiCloud is built using the MERN stack, a modern and powerful set of technologies:

* MongoDB serves as the NoSQL database, storing structured data like user information, orders, product details, and inventory records.
* Express.js is used as the backend web framework, handling routing, middleware, and APIs.
* React.js powers the frontend, delivering a fast, responsive, and component-driven user interface.
* Node.js serves as the runtime environment for the server-side logic, enabling scalable and event-driven processing.

This technology stack ensures that OptiCloud is not only scalable and high-performing but also aligned with industry standards in full-stack web development.

In summary, OptiCloud exemplifies the effective integration of e-commerce functionality with inventory and administrative controls using cutting-edge technologies. It stands as a practical model for modern retail solutions and serves as a strong foundation for future enhancements such as AI-driven recommendations, SMS/email notifications, and multi-vendor support.

# 

# 

# 

# 

# 

# **CHAPTER-1:**

# INTRODUCTION

# Introduction

## **OptiCloud: An E-commerce Venture** is a web-based platform developed for the online sale of optical lenses and efficient inventory management. The system provides a smooth and responsive user experience for both customers and administrators.

## Customers can browse products, place orders, track deliveries, and leave reviews. Administrators can manage the product catalog, monitor stock levels, and handle orders through a dedicated dashboard. The integration of real-time inventory control enhances reliability and reduces stock-related issues.

## Created as part of an academic project, OptiCloud demonstrates the practical implementation of an e-commerce system that solves real-world retail challenges while offering scalability, performance, and ease of use.

## Purpose

The primary purpose of **OptiCloud** is to provide a convenient, efficient, and reliable platform for purchasing optical lenses online while streamlining inventory and order management for administrators. In today’s fast-paced digital environment, there is a growing demand for user-friendly e-commerce solutions that offer transparency, accessibility, and operational control.

OptiCloud aims to bridge the gap between optical product retailers and consumers by offering a centralized online system where customers can easily explore, compare, and order lenses without visiting physical stores. At the same time, it empowers administrators with the tools needed to manage product listings, monitor stock levels, and fulfill customer orders effectively.

By automating essential processes such as inventory tracking, order processing, and user interaction, OptiCloud reduces manual workload, minimizes errors, and enhances overall user satisfaction. This platform serves not only as a functional e-commerce website but also as an academic demonstration of how digital solutions can address real-world retail challenges in the optical industry.

## 1.2 Document Convention

This document uses Times font with a font size of 12 units throughout the document for normal text. Apart from the normal text, there are 3 different kinds of headings nested into each other with font sizes 16, 14 and 12 respectively. The use of italics for comments and bold to emphasize text has been done wherever necessary. Line spacing is of multiple types with 1.15 units and pages maintain 1” margins on all four sides. The normal text is justified.

The following are the list of conventions and acronyms to be used in this document and the project as well:

Admin: A login id representing a user with user administration privileges to the software

User: A general login id assigned to most users

Client: Intended users for the software

Layer: Represents a section of the project

User Interface Layer: The section of the assignment referring to what the user interacts with directly.

Application Logic Layer: The section of the assignment referring to the Web Server. This is where all computations are completed.

Data Storage Layer: The section of the assignment referring to where all data is recorded.

Data flow diagram: It shows the dataflow between the entities.

Use Case: A broad level diagram of the project showing a basic overview Boolean.

Interface: Something used to communicate across different mediums

## Intended Audience and Reading Suggestions

* First, it is anticipated that the SRS will be used by the application designers. Designers will use the information recorded here as the basis for creating the application's design.
* Second, the client for the project, is expected to review this document. The SRS will serve to establish a basis for agreement between the client and development team about the functionality to be provided by the application.
* Third, the application maintainers will review the document to clarify their understanding of what the application does.

## Product Scope

The purpose of this software development project is to create a new application called **Opticloud**. Opticloud is an e-commerce platform dedicated to selling various types of optical lenses. The client intends to launch a PC-based web application accessible via the internet, allowing customers to browse, order, and manage lenses easily, while also providing administrative and inventory management capabilities to staff members.

Key Capabilities of Opticloud:

* The application can be accessed via the internet on any PC or device, from any location.
* The application allows customers to register, browse lenses by category, view product details, and place orders online.
* The application enables customers to make secure payments and track the status of their orders.
* The application allows administrators to manage products, manage orders and manage user accounts, including customers.
* The application supports inventory managers in maintaining the product catalogue by adding, updating, or removing lens products.
* The system updates product stock levels automatically as orders are processed and fulfilled.
* The application generates analytics related to sales, inventory levels, and customer activities for administrative purposes.
* The application provides search and filter functionality for lenses based on attributes such as type, brand, price range, and availability.

# 

# **CHAPTER-2:**

# PROBLEM FORMULATION

## 2.0 Problems in Existing System:

The current traditional and fragmented processes in selling optical lenses and managing inventory face several limitations:

1. **Limited Online Presence:**A-One Opticals mainly sells through its physical store and has little or no modern online platform. Without a proper e-commerce website, it becomes hard to reach more customers, especially those who prefer shopping online. This limits sales, reduces convenience for buyers, and makes it harder for the business to compete with others that already sell online.
2. **Manual Inventory Management:** Inventory tracking is often done manually or using spreadsheets, leading to frequent errors, stockouts, and overstock situations.
3. **Inefficient Order Processing:** Without a centralized system, order placement, payment processing, and order tracking lack automation, causing delays and poor customer experience.
4. **Lack of Real-Time Stock Updates:** Customers and staff do not have access to real-time inventory data, which results in order cancellations or dissatisfaction due to unavailable products.
5. **Limited Administrative Control:** Existing systems often lack comprehensive tools for administrators and inventory managers to efficiently manage users, products, and sales reports.
6. **Poor Reporting Capabilities:** Businesses struggle to generate accurate and timely sales and inventory reports, limiting their ability to make informed decisions.
7. **Security Concerns:** Traditional systems may not incorporate modern security protocols, making customer and business data vulnerable.

## Various aspects of the problem:

1. **Customer Experience**

* Customers face difficulty in finding and purchasing lenses online due to limited or poorly designed e-commerce platforms.
* Lack of real-time product availability information leads to frustration when items are out of stock after ordering.
* Inadequate search and filtering options reduce ease of navigation and product discovery.
* Absence of order tracking causes uncertainty about delivery status.

1. **Inventory Management**

* Manual stock management leads to errors, inaccurate stock levels, and difficulty in timely restocking.
* Overstock or stockouts affect sales and customer satisfaction.
* Lack of integration between sales and inventory updates causes discrepancies.

1. **Operational Efficiency**

* Order processing is slow and often requires manual intervention, increasing turnaround time.
* Payment processing lacks automation or integration with secure payment gateways.
* Administrators and inventory managers lack centralized tools to efficiently manage products, users, and orders.

1. **Data and Reporting**

* Poor or non-existent reporting tools hinder tracking of sales performance and inventory trends.
* Lack of data analytics prevents informed business decisions and growth planning.

1. **Security and Reliability**

* Customer and business data may be exposed due to inadequate security measures.
* System downtime or failures in legacy or manual processes can result in loss of orders and revenue.

## Benefits of system proposed

The proposed **Opticloud** system addresses the limitations of existing manual and outdated processes by providing a modern, integrated, and user-friendly platform. Below are the key benefits:

**1. Enhanced Customer Convenience**

* Customers can browse, compare, and purchase lenses from anywhere, anytime.
* Real-time product availability and intuitive search/filter features improve the shopping experience.
* Order tracking keeps customers informed and increases satisfaction.

**2. Automated Inventory Management**

* Stock levels are automatically updated based on customer purchases.
* Alerts for low inventory help ensure timely restocking.
* Reduced human error in stock handling increases accuracy.

**3. Efficient Administrative Control**

* Admins can easily manage users, products, and orders from a centralized dashboard.
* Secure authentication ensures only authorized users can perform administrative tasks.

**4. Improved Operational Productivity**

* Automated order processing and payment handling reduce manual effort and turnaround time.
* Streamlined workflows improve service speed and accuracy.

**5. Reliable Reporting and Analytics**

* The system generates detailed sales, inventory, and user activity reports.
* Data insights help in planning marketing strategies, promotions, and stock control.

**6. Enhanced Data Security**

* Integration with secure payment gateways ensures safe transactions.
* User data is protected through encryption and authentication protocols.

**7. Scalability and Future Readiness**

* The system can easily scale to support more products, customers, and features.
* Designed to accommodate future upgrades like mobile apps, multi-language support, and AI-based recommendations.

**8. Cost Efficiency**

* Reduces the need for manual labor and paperwork.
* Lowers operational costs through automation and improved resource management.

# 

# 

# 

# **CHAPTER-3:**

# SYSTEM ANALYSIS AND DESIGN

# 

# 3.0 Overall Description

This section of the SRS should describe the general factors that affect 'the product and its requirements. It should be made clear that this section does not state specific requirements; it only makes those requirements easier to understand.

## Product Functions

* 1. **Customer Functions:**
* User Registration and Login: Allows new users to create accounts and existing users to securely log in.
* Browse and Search Products: Users can view lens products and filter/search by type, brand, price, or availability.
* View Product Details: Displays product descriptions, specifications, images, and price.
* Add to Cart: Enables users to add one or more items to a virtual shopping cart before checkout.
* Place Orders: Users can confirm purchases and provide delivery and payment information.
* Make Payments: Integration with payment gateways allows secure online payments.
* Track Orders: Customers can view the real-time status of their placed orders.
* Manage Profile: Users can update personal information, view past orders, and manage addresses.
* Add Reviews: Customers can provide feedback and rate products after purchase, helping others make informed decisions.
* View Analytics: Customers can view personal purchase history analytics such as total spent, most frequently bought products, and order frequency.
  1. **Administrator Functions:**
* User Management: Create, update, and deactivate user accounts as needed.
* Product Management: Add, edit, or remove lens products from the catalog.
* View Reports: Access sales, order, and user activity reports for analysis.
* Monitor Transactions: View payment confirmations and address issues related to failed or pending transactions.
* Role Assignment: Assign specific roles (e.g.,Admin) to staff accounts.
  1. **Inventory Manager Functions:**
* Add New Products: Enter new product information into the system, including images and descriptions.
* Update Inventory: Modify stock levels based on supply and demand.
* Remove Products: Discontinue products from the catalog when needed.
* Receive Stock Alerts: Get notifications when inventory falls below a predefined threshold.
  1. **System Functions**
* Authentication and Authorization: Ensures only authorized users access specific features based on their role.
* Data Validation: Checks input data for correctness before processing (e.g., price fields, email format).
* Secure Payment Integration: Supports encryption and secure protocols for online transactions.
* Responsive Web Interface: Ensures usability on both desktop and mobile browsers.

## 3.0.2 User Classes and Characteristics

The **Opticloud** system will cater to different types of users, each with specific roles, responsibilities, and levels of access. Understanding these user classes helps define functional requirements and system behavior.

**1. Customers**

**Description:**  
General users of the platform who browse, purchase, and review lens products.

**Characteristics:**

* May or may not be tech-savvy.
* Access the system via desktop or mobile browsers.
* Expect intuitive UI, fast loading, and secure transactions.
* Can be returning or first-time buyers.

**Responsibilities/Actions:**

* Register and log in.
* Browse and search for products.
* View product details and reviews.
* Add items to cart and make purchases.
* Track orders.
* Add product reviews.
* View personal analytics (e.g., order history, spending habits).

**2. Administrators**

**Description:**  
Users responsible for managing overall system settings, user accounts, product listings, and system reports.

**Characteristics:**

* Familiar with admin tools and basic data handling.
* Need access to secure, role-restricted features.
* May be IT staff or business managers.
* Require access to back-end product management tools.
* Work closely with the logistics or supply chain team.

**Responsibilities/Actions:**

* Create and manage user accounts (customers).
* Monitor system usage and transaction records.
* View and generate sales and user reports.
* Assign roles and permissions.
* Manage product listings.
* Update stock levels and pricing.
* Remove discontinued items.
* Receive stock alerts and handle restocking.

**3. Guest Users**

**Description:**  
Unregistered users who visit the platform to explore available products.

**Characteristics:**

* No login required.
* May be casual browsers or potential first-time customers.
* Prefer minimal barriers to exploring the platform.

**Responsibilities/Actions:**

* Browse and search for products.
* View product details.
* Register if they wish to make purchases.

## 3.0.3 Operating Environment

The OptiCloud operates in all famous browsers, for a model we are taking Microsoft Edge, Google Chrome, and Mozilla Firefox.

It can be accessed through any Operating system (Windows/Linux/Unix/Mac) as it is system independent. The system must have a web browser(Chrome/internet explorer/Firefox) and active internet connection to access the system.

## 3.0.4 Design and Implementation Constraints

1. **Platform**

* The application must be web-based, accessible via standard web browsers on both desktop and mobile devices.
* Responsive design is required for cross-device compatibility.

1. **Technology Stack:**

* Frontend: HTML, CSS, JS (optional: React/Bootstrap).
* Backend: Node.js/Django/Spring Boot.
* Database: MySQL/PostgreSQL.

1. **Security:**

* Must comply with standard **web security practices**, including HTTPS, password encryption, and secure API endpoints.
* Integration with secure and reliable **payment gateways** is mandatory.

1. **Performance:**

* The system should be optimized to handle a **moderate load of concurrent users** (as expected for an academic prototype).
* Page load times should be minimal, with efficient query handling for inventory and orders.

1. **Data Storage:**

* Use a centralized relational database.

## 3.0.5 User Documentation

This document outlines the procedures and functionalities available to various user roles within the **Opticloud** e-commerce platform. It serves as a user guide to ensure smooth navigation and effective use of the system.

**1. Customer User Guide:**

**a. Account Registration and Login**

* Navigate to the Opticloud homepage.
* Click “Register” to create a new account using a valid email and secure password.
* Use the “Login” option to access your account.

**b. Product Browsing and Search**

* Utilize the search bar or category filters to explore available lenses.
* View detailed product pages including specifications, images, and pricing.

**c. Shopping Cart and Checkout**

* Add selected items to the shopping cart.
* Proceed to checkout by entering shipping and payment details.
* Confirm the order to receive a transaction summary and email confirmation.

**d. Order Tracking**

* Access the “My Orders” section from your profile dashboard.
* Monitor real-time order status (e.g., Pending, Shipped, Delivered).

**e. Product Reviews**

* After receiving a product, navigate to the order history.
* Select “Leave a Review” to submit ratings and feedback.

**f. Customer Analytics**

* View purchase history, total spending, and frequently ordered products from dashboard.

**2. Administrator User Guide:**

**a. Access and Dashboard Overview**

* Log in using administrator credentials.
* Access the admin dashboard to monitor system activities.

**b. User and Role Management**

* Create, edit, or deactivate user accounts.
* Assign roles (e.g., customer, inventory manager) as needed.

**c. Product and Order Oversight**

* Manage product listings and oversee order fulfillment.
* Monitor transaction logs and user activity.

**d. Reporting**

* Generate detailed sales, inventory, and customer reports for business analysis and decision-making.

**e. Inventory Control**

* Add new lens products with detailed specifications and media.
* Update stock levels and product information regularly.

**f. Stock Monitoring and Alerts**

* Receive system-generated alerts for low inventory.
* Remove or archive discontinued products as necessary.

## 3.0.6 Assumption and Dependencies

The assumptions are: -

* The coding should be error free.
* The system should be user-friendly so that it is easy to use for the users.
* The information of all users, products and orders must be stored in a database that is accessible by the website.
* The system should have more storage capacity and provide fast access to the database.
* The system should provide a search facility and support quick transactions.
* Users may access from any computer that has Internet browsing capabilities.
* Users must have their correct usernames and passwords to enter into their online accounts and do actions.

The dependencies are: -

* The specific hardware and software due to which the product will run.
* Based on the listing requirements and specification, The project will be developed.
* The end users (admin) should have proper understanding of the product.
* The system should have the general report stored.
* The information of all the users must be stored in a database that is accessible.
* Any update regarding the product from inventory is to be recorded to the database and the data entered should be correct.

# 

# **CHAPTER-4:**

# SYSTEM IMPLEMENTATION

# 

# 

# 4.0 External Interface Requirement

This section contains the software requirements to a level of detail sufficient to enable designers to design the system and testers to test that system.

## **User Interface**

The user interfaces are to be made so simple for the model such that anybody can be comfortable in working with the system in just a few minutes. It includes:

GUI - The software provides a good graphical interface for the user and the administrator can operate on the system, performing the required task such as create, update, viewing the details of the book. It allows users to view quick reports Issued/Returned in between particular times. It provides stock criteria.

The **Opticloud** platform features a clean, responsive, and user-friendly interface designed to enhance usability and provide a seamless shopping experience. The interface is intuitive, allowing users to easily browse products, manage their accounts, and complete purchases with minimal effort.

Key UI elements include:

* **Navigation Bar** for quick access to home, product categories, cart, and user account.
* **Search and Filter Options** to help customers find specific lenses based on brand, type, or price.
* **Product Pages** with high-quality images, detailed descriptions, pricing, and customer reviews.
* **Shopping Cart and Checkout Flow** optimized for clarity and ease of use.
* **Admin and Inventory Dashboards** offering streamlined tools for managing users, products, and orders.

The UI is developed with modern web technologies (HTML, CSS, JavaScript, React) and is fully responsive across desktop and mobile devices, ensuring consistent functionality and appearance on all screens.

## **Hardware Interface**

It is a web-based application and therefore is system independent, It can be accessed via mobile phone/tablet via internet service enabled on that device through a given domain. Still a minimum hardware is required to run the application smoothly on a web browser that being:

**Processor**: - Intel Core 2 Duo T6600 @ 2.2GHz (2CPUs)

**Core 2** is a brand encompassing a range of Intel's consumer 64-bit x86-64 single-, dual-, and quad-core microprocessors based on the Core microarchitecture. The single- and dual-core models are single-die, whereas the quad-core models comprise two dies, each containing two cores, packaged in a multi-chip module. The introduction of Core 2 relegated the Pentium brand to the mid-range market, and reunified laptop and desktop CPU lines, which previously had been divided into the Pentium 4, Pentium D, and Pentium M brands.

**Hard Disk**: 50 GB

A hard disk is part of a unit, often called a “disk drive”, “hard drive”, or “hard disk drive”, that stores and provides relatively quick access to large amounts of data on an electromagnetically charged surface or a set of surfaces. A hard disk is really a set of stacked “disks”, each of which, like phonograph records, has data recorded electromagnetically in concentric circles or “tracks” on the disk. A “head” writes or reads the information on the tracks. Two heads, one on each side of a disk, read or write the data as the disk spins. A hard disk/drive unit comes with a set rotation speed varying from 4500 to 7200 the physical location can be identified with the cylinder, track, and sector locations, these are actually mapped to a Logical Block Address (LBA) that works with the larger address range on today’s hard disks.

**Random Access Memory**: 2 GB or More for Future

When people talk about computer memory, they usually mean the volatile RAM memory. Physically, this memory consists of some integrated circuit Chips (IC chips) either on the motherboard or on a small circuit board attached to the motherboard. A computer’s motherboard is designed in a manner that its memory capacity can be enhanced by adding more memory chips. Hence, if you decide to have more memory than your computer currently has, you can buy more memory chips, and plug them in the empty memory slots on the motherboard. This job is normally done by the service engineers.

**Monitor**: LCD or Laptop

A liquid crystal display (LCD) is a flat panel display, electronic visual display, or video display that uses the light modulating properties of liquid crystals (LCs). LCs do not emit light directly. It has a resolution of 1366 x 768 x 59 Hz

**Keyboard**: Multimedia

The multimedia keyboard is a kind of keyboard that gives you a superior level of desktop freedom to work anywhere, on almost any surface. The Multimedia Keyboard provides complete workspace freedom. Easily navigate to the web, check your email, control your audio, or access documents with the touch of a button on the multimedia keyboard. It features time saving multimedia and internet shortcut buttons including back, forward, stop, refresh, web, and search buttons for the internet as well as play controls, volume controls on Windows operating systems. The multimedia keyboard is compatible with almost all PC models.

Mouse: Optical

An optical mouse is an advanced computer-pointing device that uses a Light Emitting Diode (LED), an optical sensor, and Digital Signal Processing (DSP) in place of the traditional mouse ball and electromechanical transducer. Movement is detected by sensing changes in reflected light rather than by interpreting the motion of a rolling sphere. The optical mouse takes microscopic snapshots of the working surface at the rate of more than 1,000 images per second. If the mouse is moved then the image changes. The tiniest irregularities in the surface can produce images well enough for the sensor and DSP to generate usable movement data.

## 

## **Software Interface**

**Front End**: React, Redux

**React** (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library for building user interfaces based on UI components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies. React can be used as a base in the development of single-page, mobile, or server-rendered applications with frameworks like Next.js. However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

**Redux** is an [open-source](https://en.wikipedia.org/wiki/Open-source_software) [JavaScript library](https://en.wikipedia.org/wiki/JavaScript_library) for managing and centralizing application [state](https://en.wikipedia.org/wiki/State_(computer_science)). It is most commonly used with libraries such as [React](https://en.wikipedia.org/wiki/React_(web_framework)) or [Angular](https://en.wikipedia.org/wiki/Angular_(web_framework)) for building [user interfaces](https://en.wikipedia.org/wiki/User_interface). Similar to (and inspired by) Facebook's [Flux architecture](https://en.wikipedia.org/wiki/React_(JavaScript_library)#Unidirectional_data_flow), it was created by Dan Abramov and Andrew Clark. Since mid-2016, the primary maintainers are Mark Erikson and Tim Dorr.

**Back End**: - Nodejs - express / MongoDB

**Node.js** is an [open-source](https://en.wikipedia.org/wiki/Open-source_software) server environment. Node.js is [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) and runs on Windows, Linux, Unix, Mac OS, etc. Node.js is a [back-end](https://en.wikipedia.org/wiki/Front_end_and_back_end) [JavaScript](https://en.wikipedia.org/wiki/JavaScript) [runtime environment](https://en.wikipedia.org/wiki/Runtime_system). Node.js runs on a [JavaScript Engine](https://en.wikipedia.org/wiki/JavaScript_Engine) (i.e. [V8 engine](https://en.wikipedia.org/wiki/V8_(JavaScript_engine))) and executes JavaScript code outside a [web browser](https://en.wikipedia.org/wiki/Web_browser). **Node.js** lets developers use JavaScript to write command line tools and for [server-side scripting](https://en.wikipedia.org/wiki/Server-side_scripting). The functionality of running scripts server-side produces [dynamic web page](https://en.wikipedia.org/wiki/Dynamic_web_page) content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying [web-application](https://en.wikipedia.org/wiki/Web_application) development around a single programming language, rather than different languages for server-side and client-side scripts.

**Express.js**, or simply **Express**, is a [back end](https://en.wikipedia.org/wiki/Front_end_and_back_end) [web application framework](https://en.wikipedia.org/wiki/Web_application_framework) for building [REST](https://en.wikipedia.org/wiki/Representational_state_transfer)ful APIs with [Node.js](https://en.wikipedia.org/wiki/Node.js), released as [free and open-source software](https://en.wikipedia.org/wiki/Free_and_open-source_software) under the [MIT License](https://en.wikipedia.org/wiki/MIT_License). It is designed for building [web applications](https://en.wikipedia.org/wiki/Web_application) and [APIs](https://en.wikipedia.org/wiki/API). It has been called the [de facto standard](https://en.wikipedia.org/wiki/De_facto_standard) server framework for [Node.js](https://en.wikipedia.org/wiki/Node.js).

# 4.1 System Features

## 4.1.1 System Feature

The Opticloud platform is designed to be user-friendly and intuitive, suitable even for users with minimal technical skills. The system focuses on providing seamless navigation, efficient order processing, real-time inventory updates, and secure user management. This is a **medium-priority system** designed for academic evaluation and demonstration of real-world e-commerce operations.

**Stimulus/Response Sequences**

1. Welcome Page
2. Sign Up
3. Login
4. Customer Dashboard
5. Search Products
6. View Product Details
7. Add to Cart
8. Checkout & Payment
9. Add Product Reviews
10. View Analytics (customer history)
11. Manage Inventory (admin)
12. Generate Reports (admin)
13. Manage Orders (admin)
14. Manage Users (admin)
15. Log Out

**Functional Requirements-**

USER LOGIN-

Description: The system authenticates users based on registered credentials and grants access according to roles.

Functional Requirements:

* System must verify login credentials (email & password).
* Only valid users can access the platform.
* User roles (Customer, Admin, Inventory Manager) determine access privileges.
* Users can securely log out from the system.

Stimulus / Response:

* Stimulus: User initiates login.
* Response: System requests email and password.
* Stimulus: User enters credentials.
* Response: System validates against the database and grants or denies access.

SIGN UP NEW USER-

Description: Allows new customers to register for an account.

Functional Requirements:

* System must validate required fields (email, password, phone).
* Duplicate accounts (same email) must be rejected.
* System stores user information securely.

SEARCH PRODUCTS-

Description: Users can search lenses using filters like type, brand, or price.

Functional Requirements:

* System must accept search keywords and filters.
* System must return relevant products in a list view.
* Search must support partial matches (e.g., “contact” finds “contact lenses”).

VIEW PRODUCT DETAILS-

Description: Displays product images, description, price, and stock status.

Functional Requirements:

* System must fetch and display detailed info on selection.
* Customers should see ratings and reviews from other users.

ADD TO CART & CHECKOUT-

Description: Enables users to select items and proceed to checkout.

Functional Requirements:

* System must add selected products to a user-specific cart.
* System must update total price based on quantity.
* At checkout, the system must validate shipping and payment details.
* Orders must be stored and confirmation sent to the user.

ADD PRODUCT REVIEW-

Description: Customers can rate and review products after purchase.

Functional Requirements:

* Reviews must be linked to verified purchases.
* Users can submit text reviews and star ratings.
* Reviews are visible to other users on product pages.

VIEW CUSTOMER ANALYTICS-

Description: Customers can view their order history and purchase trends.

Functional Requirements:

* System must generate summaries of past orders.
* Spending patterns and order frequency must be displayed graphically or in tabular form.

INVENTORY MANAGEMENT (Admin/Inventory Manager)-

Description: Admins can manage stock and product listings.

Functional Requirements:

* Add/edit/delete lens products
* System must validate product info (name, type, price).
* System must alert for low stock.

Stimulus / Response:

* Stimulus: Admin selects “Add Product”.
* Response: System opens product entry form and validates inputs.

ORDER MANAGEMENT (Admin)-

Description: Allows admin to monitor, update, or cancel orders.

Functional Requirements:

* Admin can view all orders with filters (status, date, user).
* Admin can update delivery status or cancel orders with reason.

REPORT GENERATION (Admin)-

Description: Admins can generate analytical and operational reports.

Functional Requirements:

* Sales reports by product/date/customer.
* Inventory status reports.
* Export options (PDF/CSV) for reports.

LOG OUT-

Description: Allows users to securely exit their session.

Functional Requirements:

* System ends session and redirects to home or login page.

# 4.2 Other non-functional Requirement

## **Performance Requirements**

* **Static Requirements:**

Project College library management system is to support many users at a time. All necessary operations shall have been carried out with the help of the client and a server.

* **Dynamic Requirements:**

The Project Library management system has to avoid degradation of its performance by processing one or at most two requests at a time. If the user finishes his request and opens the other, then the previous request will be unloaded in order to not conflict with the database.

* **Logical Database Requirements:**

All the information should be stored in separate collections. These collections should be categorized and maintained in a logical manner. For example:- User information in User collection, Administrator information in Admin collection.

* **Design Constraints:**

Server and a Frontend build have to be deployed, and CORS has to be set accordingly. As the security feature included in the system, no one except the administrator and member can access the system.

## **Safety Requirements**

The system should be Safe in the sense that there should be no room for mistakes. Every activity/function of the project should be indefectible. The system should be maintainable in the sense that if any error occurs, it should be easily rectified, and the cost incurred in maintenance should be as low as possible.

## **Security Requirements**

The system is to be secure in the sense that nobody except the authenticated users can login and use the system.

## **Software Quality Attributes**

* **Reliability:**

The system should be reliable in the sense that there should be no room for mistakes. Every activity/ function of the project should be indefectible. If it moves out of normal operation mode, the requirement is to drop down the server and fix it as soon as possible and open it again. This emergency behavior shall not occur without reason.

* **Availability:**

The system is to be available as and when needed. Immediate feedback of the system's activities shall be communicated to the user by clearing the system and giving space and speed to their hospitality.

* **Portability:**

There is no portability requirement as far as our system is concerned because it is an online (server based) system so we can access it from anywhere through the internet connection. And we have to maintain the copy of stored data into our database.

* **Maintainability:**

The system should be maintainable in the sense that if any error occurs, it should be easily rectified, and the cost incurred in maintenance should be as low as possible.

* **Efficiency requirement:**

When a library management system will be implemented, librarians and users will easily access the library as searching and book transactions will be really fast.

* **Usability requirement:**

The system is designed for a user-friendly environment so that students and staff of the library can perform the various tasks easily and in an effective way.

## **Organizational Requirement**

* **Implementation Requirements:**

In implementing the whole system, it uses react in front end with node as a server side scripting language which will be used for database connectivity and the backend i.e. the database part is developed using mongoDB.

* **Delivery Requirements:**

The whole system is expected to be delivered in six months of time with a weekly evaluation by the project guide.

* **Technical Issue:**

In order to satisfy this requirement, the design should be simple and all the different interfaces should follow a standard template. There will be the possibility of changing colors and images, plus switching between interfaces with the minimum impact for the users.

# 4.3 System testing

Testing is the process of executing a program with the intent of finding an error. Testing begins at the module level and works “outward” towards the integration of the entire computer-based system.Different testing techniques are appropriate at different points of time.

A strategy for software testing must accommodate:

**Low Level Tests**: Low level tests are necessary to verify that a small code segment has been correctly implemented or not.

**High Level Tests:** High level tests that validate major system functions against customer requirements.

**Types of Testing**

The following 4 types of tests are performed:

1. Unit Testing

2. Functional Testing

3. System Testing

4. Acceptance Testing

**Unit Testing**

This is done to check syntax and logical errors in programs.At the time of preparation of technical specifications, unit test data is also prepared.The coding for that program is considered after verifying its output against this test data.

**Functional Testing**

This is done for each module and sub module present in the system.Functional testing serves as a means of the system, which confirms the original user requirement i.e. it checks whether the module does what it was supposed to do. Separate schedules were made for functional testing. It involves presentation of test data, writing of test cases, testing for conformance to test cases and preparation of bugs listing for non-conformance to test cases and preparation of bugs listing for non-conformities.

**System Testing**

System testing is done when the entire system has been fully integrated.The purpose of the system testing is to test how the different modules interact with each other and whether the entire system provides the functionality that was expected.The important and essential part of the system development phase, after designing and developing the software is system testing.It cannot be said that every program or system design is perfect and because of lack of communication between the user and the designer, some error is there in the software development.

System Testing consists of the following steps:

• Recovery Testing

• Security Testing

• Stress Testing

• Performance Testing

• User Acceptance Testing

**Acceptance Testing**

This is the final testing when the user is satisfied and finally accepts the software developed.

# 

# **CHAPTER-5:**

# RESULT AND DISCUSSION

# Interpretation of result

The implementation and testing of the Opticloud platform confirm that the system meets its primary objectives: enabling online sales of lenses, providing inventory management capabilities, and supporting both administrative and customer-side functionalities in an academic e-commerce context.

1. User Experience and Interface Effectiveness

During functional testing, users were able to:

* Seamlessly browse and filter products.
* Register and log in without errors.
* Add items to the cart and complete checkout.
* Submit reviews and access their order history.

This demonstrates that the user interface is intuitive and aligned with usability standards, especially for users with limited technical experience.

2. Functionality Coverage

All core modules — product catalog, search and filter, cart, checkout, review system, user dashboard and admin panels — performed according to the predefined functional requirements. Key workflows like sign-up/login, product management, and order processing functioned without critical issues.

3. System Performance

The system handled typical user flows with low response time and minimal latency during testing. Page transitions and database interactions (e.g., inventory updates, order placements) were executed efficiently in the test environment.

4. Inventory and Admin Control

Admin users were able to:

* Add, edit, and delete lens products.
* Track stock levels and update inventory in real-time.
* Generate analytics related to orders and inventory status.

This indicates that the platform successfully supports backend operations essential for managing an e-commerce business.

5. Security and Data Integrity

Basic validation and authorization mechanisms ensured that:

* Only registered users accessed the system.
* Sensitive operations (like order placement and admin updates) were protected against unauthorized access.
* Data entries (e.g., prices, stock quantity) were properly validated and stored.

6. Limitations Observed

* Third-party APIs used for payment were simulated or implemented in test mode, which limits assessment under real-world transaction volume.
* The current system supports only English and lacks multi-language support.

# 5.2 Application area and scope of work

Opticloud is an academic e-commerce solution specifically developed for the optical lens retail industry. Its application area includes:

* Online Retail of Optical Products: Enables the sale of various types of lenses, such as contact lenses, prescription lenses, and branded eyewear.
* Inventory Management: Supports real-time tracking and updating of stock levels to ensure efficient supply chain operations.
* Customer Interaction: Facilitates user engagement through product reviews, order history, and personalized analytics.
* Admin and Backend Operations: Allows administrators to manage products, monitor transactions, and generate performance reports.

This system is adaptable for use by:

* Local optical stores expanding into online sales
* Lens manufacturers seeking a direct-to-consumer platform
* Academic projects or prototypes related to retail and inventory management

Scope of Work

The scope of the Opticloud project includes:

* User Registration and Authentication: Secure login and role-based access for customers and admins.
* Product Catalog and Search: Browsing, searching, and filtering lenses by brand, type, and price.
* Shopping Cart and Checkout: Functional cart system with payment simulation and order confirmation.
* Customer Dashboard: Displays order history, allows review submission, and provides simple analytics.
* Admin Panel: Enables product management, order tracking, report generation, and user monitoring.
* Inventory System: Updates stock based on orders and admin inputs, with alerts for low inventory.
* Activity Logging: Tracks critical operations for accountability and debugging.

The project demonstrates the design, development, and deployment of a full-stack e-commerce web application, suitable for academic submission and real-world adaptation with minor enhancements.

# 

# **CHAPTER-6:**

# CONCLUSION

# 6.0 Product Scope

**Opticloud** is a custom-built e-commerce and inventory management system developed for **A-One Opticals**, a retail business that specializes in optical lenses and related products. The system is designed to support the business’s transition into digital commerce by providing an online platform where customers can browse and purchase products, while administrators can efficiently manage inventory and sales operations.

The primary objective of Opticloud is to enhance A-One Opticals’ business capabilities by expanding its customer reach beyond the physical store, increasing sales potential, and improving operational accuracy. The platform brings automation to daily business tasks and simplifies interactions between the business and its customers.

**Target Users**

* **Customers:** Can browse products, place orders, and track past purchases through a user-friendly interface.
* **Administrators (Store Staff):** Can manage product listings, update stock, and monitor order activity using a secure backend system.

**Core Capabilities**

Opticloud is built to provide the following core capabilities:

* **E-commerce Functionality:** Enables users to view product categories, make purchases, and receive updates on order status.
* **Inventory Management:** Maintains up-to-date stock records and helps prevent over-ordering or stockouts through automated tracking.
* **User Access Management:** Ensures secure login for customers and admin users with appropriate access rights.
* **Customer Engagement:** Supports basic analytics and product reviews, allowing customers to give feedback and view trends in their purchases.
* **Business Monitoring:** Allows administrators to track sales trends and inventory performance to make informed decisions.

**Scope Boundaries**

* The system is tailored **exclusively** for A-One Opticals and is not intended to support multiple vendors or stores.
* Payment gateway integration, delivery logistics, and mobile app development are considered **out of scope** for the current version but may be addressed in future phases.

**Expected Outcomes**

* Increased visibility and online presence for A-One Opticals.
* Reduced manual workload in inventory tracking and sales reporting.
* Improved customer satisfaction through a smooth and accessible online shopping experience.
* A fully functional, maintainable, and scalable application suitable for real-world deployment and academic demonstration.

# 6.1 Limitations

OptiCloud, while being a robust and purpose-built e-commerce and inventory system for A-One Opticals, has certain limitations that must be acknowledged for a comprehensive understanding of its scope and operational constraints:

**1. Limited Scalability**

OptiCloud is designed to serve a single business unit. It does not support multi-vendor or multi-store setups, making it less scalable for future expansion. If A-One Opticals decides to open new branches or operate across multiple regions, the current system would require significant upgrades to handle such complexity.

**2. Limited Customizability**

While OptiCloud meets the current operational requirements, its architecture is not modular enough to easily adapt to major changes such as switching product lines, implementing new promotional features, or integrating loyalty programs without development effort.

**3. Internet Dependency**

Being a cloud-based solution, the platform requires a stable internet connection to function effectively. In the event of network downtime or poor connectivity, access to inventory data, order processing, and customer services becomes inaccessible.

**4. Basic Role-Based Access Control**

The user roles are limited to customers and a single admin. There are no granular permission levels for employees, warehouse staff, or support agents, which could affect internal security and workflow management.

**5. Limited Analytical Insights**

OptiCloud currently offers only basic reporting features. It lacks advanced analytics like demand forecasting, customer behavior analysis, and real-time business intelligence that could provide strategic value to management.

**6. Restricted Payment Options**

The system supports a limited number of payment gateways. This may cause inconvenience for customers who prefer alternative methods such as digital wallets, net banking, or "buy now, pay later" options.

While OptiCloud effectively addresses the immediate operational needs of A-One Opticals, its current limitations could hinder growth, scalability, and adaptability in the long term. Future iterations should focus on enhancing scalability, mobile accessibility, third-party integrations, and advanced analytics.

# 

# 

# 

# 

# **APPENDIX**

## 

## **A.1 SDLC Model:**

Software engineering is a discipline that integrates processes, methods, and tools for the development of computer software. To solve actual problems in an industry setting, software engineers or a team of software engineers must incorporate a development strategy that encompasses, methods, and tools. This strategy is often referred to as a process model or a software engineering paradigm.

A number of different process models for the software engineering have been proposed, each exhibiting strengths and weaknesses, but all having a series of generic phases in common. Some of the commonly used software process models are:

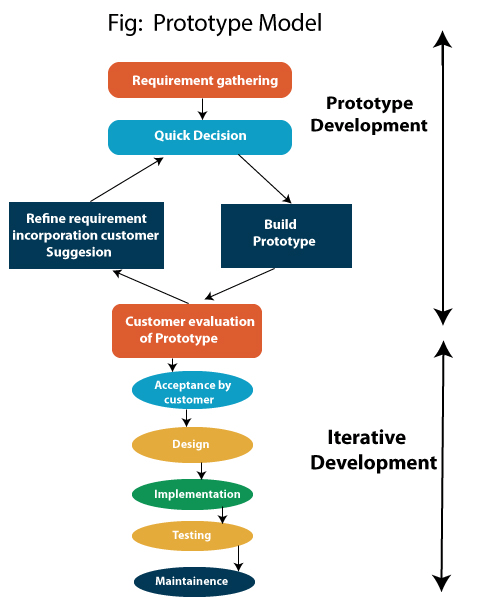
1. The linear sequential model
2. The prototyping model
3. The RAD model
4. The incremental model
5. The spiral model
6. The component assembly model
7. The concurrent development model
8. The formal methods model
9. Iterative Model

“I chose a prototyping model for the development of my system because to get the acceptance letter I had to show a working model of the system and document it and repeat the process until I had the ideal working system to deliver. So, in a prototyping model I design the system and code it and repeat the process because I don’t have enough data to complete the system in one go.”

**Prototyping Model:**

The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possibly exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software. In many instances, the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirement, the prototyping model may be employed.

Following is the pictorial representation of prototyping model:



**Prototyping Model Application:**

Prototype process model resembles the iterative enhancement model. The same phases are defined for the waterfall model that occurs here in a cyclical fashion. This model differs from the iterative enhancement model in the sense that this does not require a useful product at the end of each cycle. In evolutionary development, requirements are implemented by category rather than by priority.

## **A.2 Entity-Relationship Diagram (ERD)**

ERD stands for entity relationship diagram. People also call these types of diagrams ER diagrams and Entity Relationship Models. An ERD visualizes the relationships between entities like people, things, or concepts in a database. An ERD will also often visualize the attributes of these entities.

By defining the entities, their attributes, and showing the relationships between them, an ER diagram can illustrate the logical structure of databases. This is useful for engineers hoping to either document a database as it exists or sketch out a design of a new database.

**Symbols and Notation:**

An ER diagram has three main components: entities, relationships, and attributes connected by lines.

**Entities**: which are represented by rectangles. An entity is an object or concept about which you want to store information.

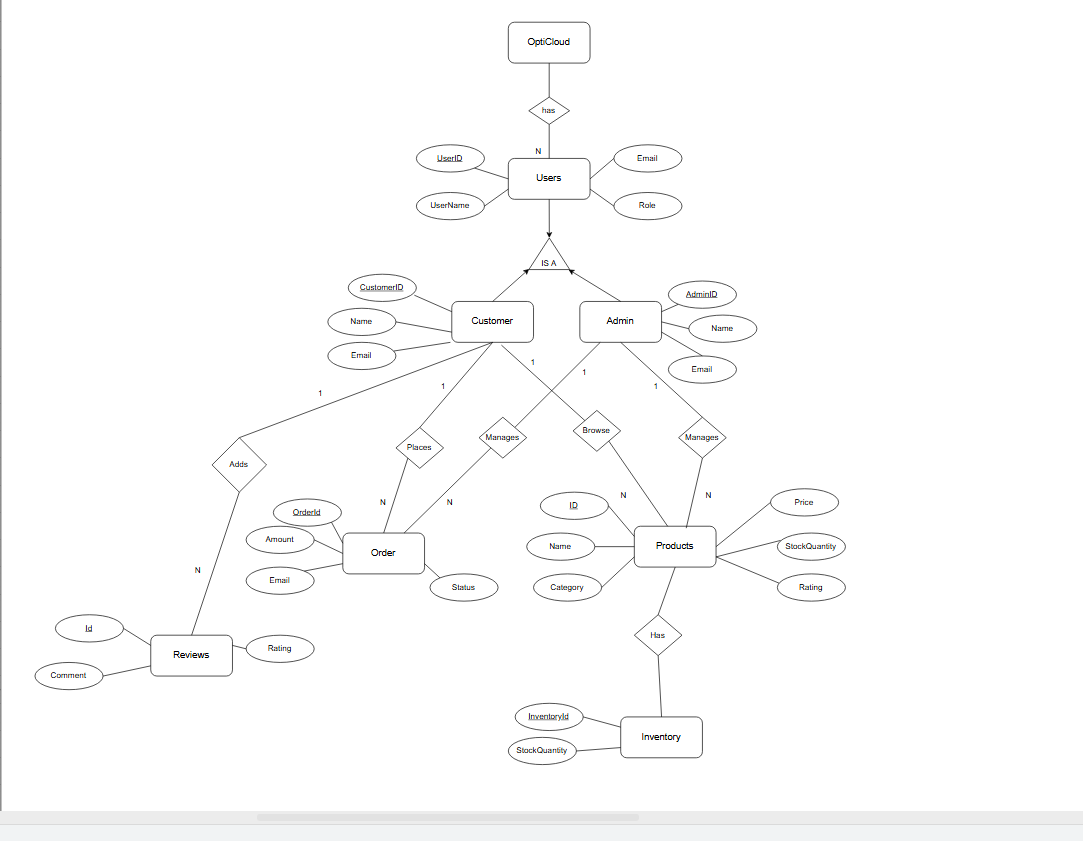
A weak entity is an entity that must be defined by a foreign key relationship with another entity as it cannot be uniquely identified by its own attributes alone.

**Relationships**: which are represented by diamond shapes, show how two entities share information in the database

**Attributes**: which are represented by ovals. A key attribute is the unique, distinguishing characteristic of the entity. For example, an employee's social security number might be the employee's key attribute.

**Cardinality:** specifies the numerical attribute of the relationship between entities. It can be one-to-one, many-to-one, or many-to-many.

**ER- Diagram:**

****

## **A.3 USE-CASE Diagram:**

UML is the modeling toolkit that you can use to build your diagrams. Use cases are represented with a labeled oval shape. Stick figures represent actors in the process, and the actor's participation in the system is modeled with a line between the actor and use case. To depict the system boundary, draw a box around the use case itself.

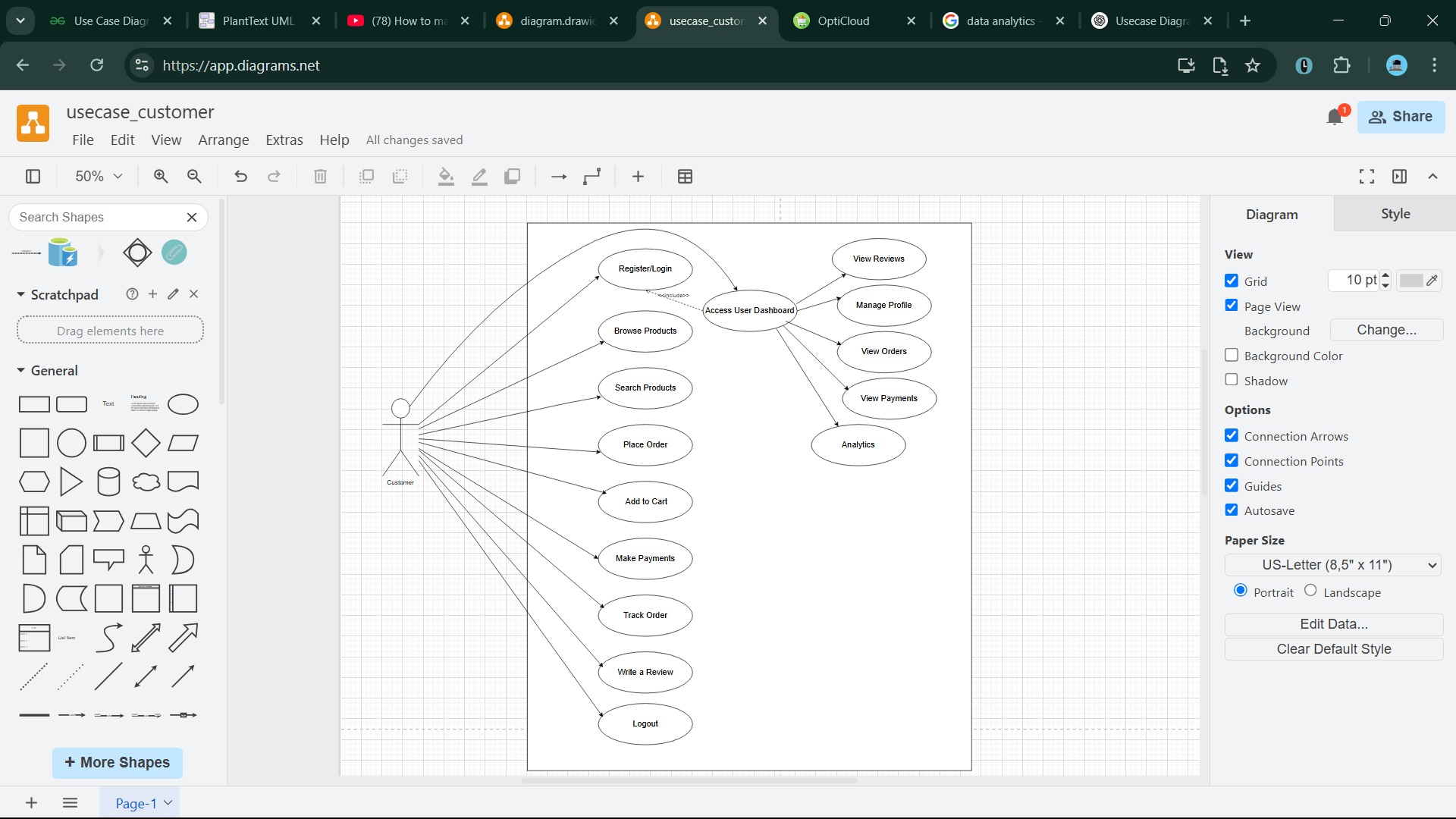
**Purpose:**

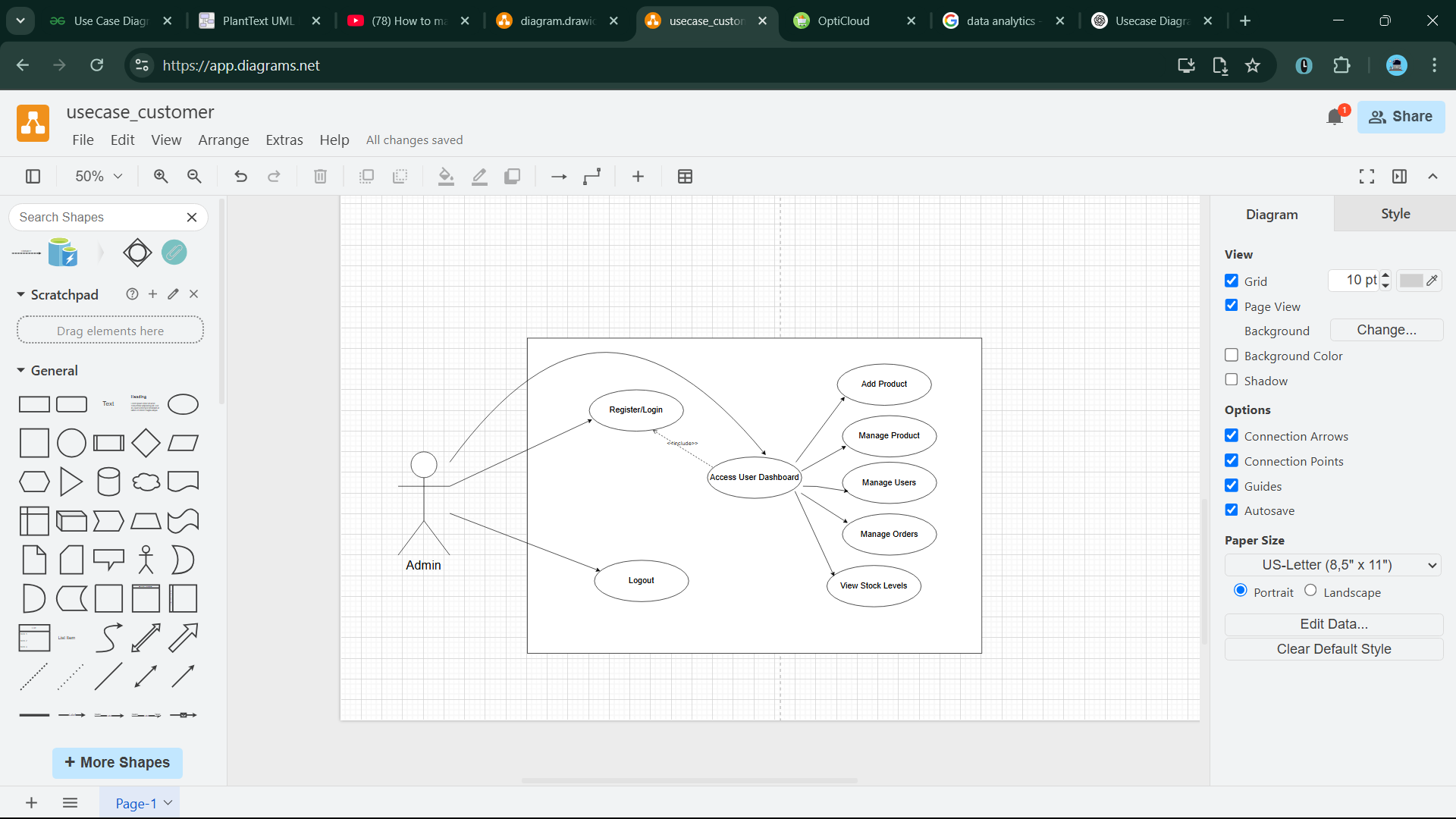
* Representing the goals of system-user interactions.
* Defining and organizing functional requirements in a system
* Specifying the context and requirements of a system
* Modeling the basic flow of events in a use case.

**Symbols and Notation:**

The notation for a use case diagram is pretty straightforward and doesn't involve as many types of symbols as other UML diagrams.

* Use cases: Horizontally shaped ovals that represent the different uses that a user might have.
* Actors: Stick figures that represent the people actually employing the use cases.
* Associations: A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.
* System boundary boxes: A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.
* Packages: A UML shape that allows you to put different elements into groups. Just as with component diagrams, these groupings are represented as file folders.
* Customer Use Case:



* Admin Use Case:

## **A.4 Activity Diagram:**

Activity diagram is another important behavioral diagram in UML diagram to describe dynamic aspects of the system. Activity diagram is essentially an advanced version of flowchart that models the flow from one activity to another activity. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc.

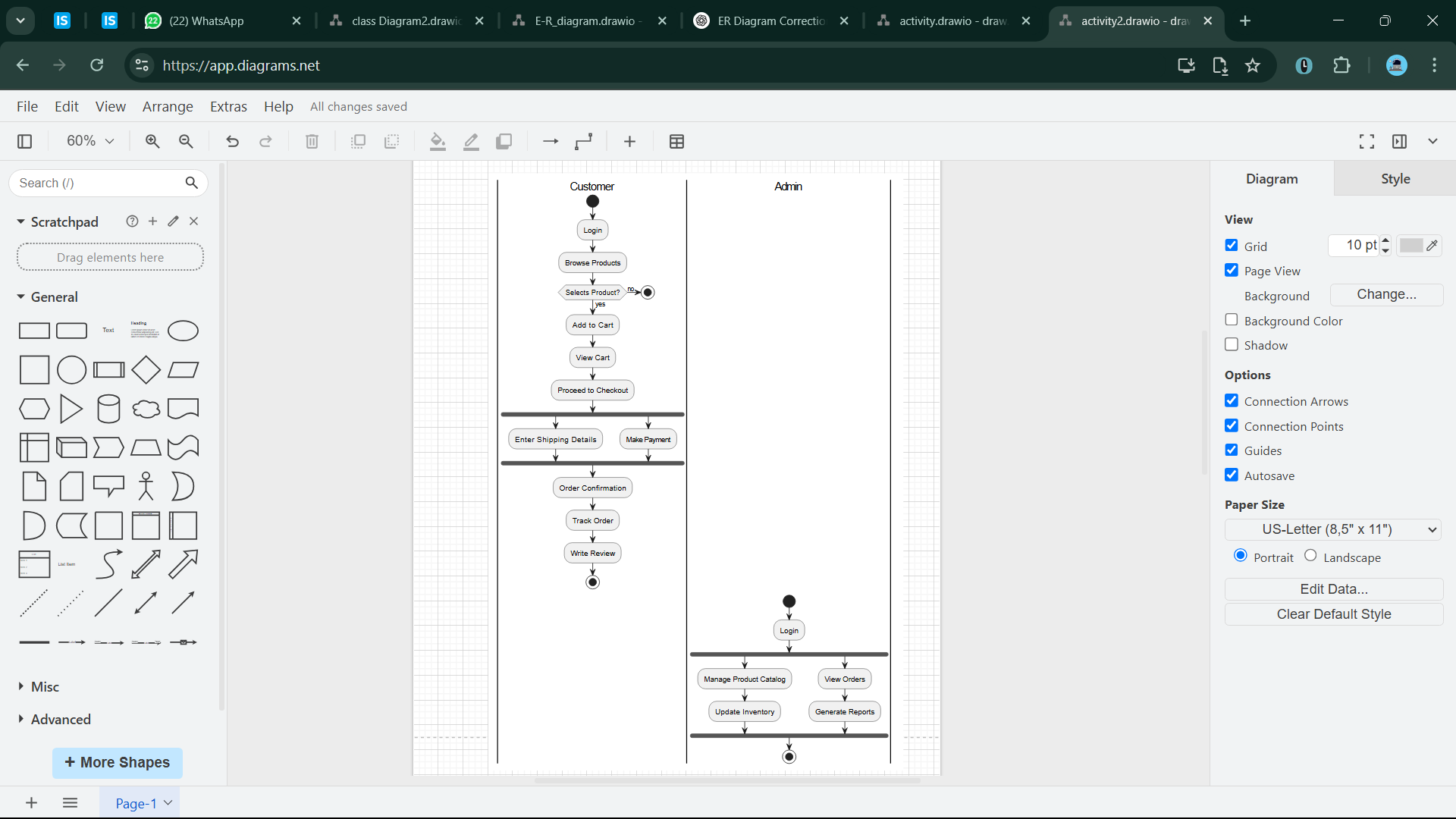
**Purpose:**

* Activity diagram is used to show message flow from one activity to another.
* Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system.
* The purpose of a Use Case is to just depict the functionality i.e. what the system does and not how it is done.

**Symbols and Notation:**

Activity diagrams are constructed from a limited number of shapes, connected with arrows. The most important shape types:

* Ellipses represent actions
* Diamonds represent decisions
* A black circle represents the start (initial node) of the workflow
* An encircled black circle represents the end (final node).



## **A.5 Data Flow Diagram (DFD):**

A data flow diagram (DFD) is a graphical or visual representation using a standardized set of symbols and notations to describe a business's operations through data movement. They are often elements of a formal methodology such as Structured Systems Analysis and Design Method (SSADM). Superficially, DFDs can resemble flow charts or Unified Modeling Language (UML), but they are not meant to represent details of software logic.

**Symbols and Notation:**

DFD notations and symbols vary according to the methodology model employed. Some organizations have adopted their own conventions, though this is not recommended.

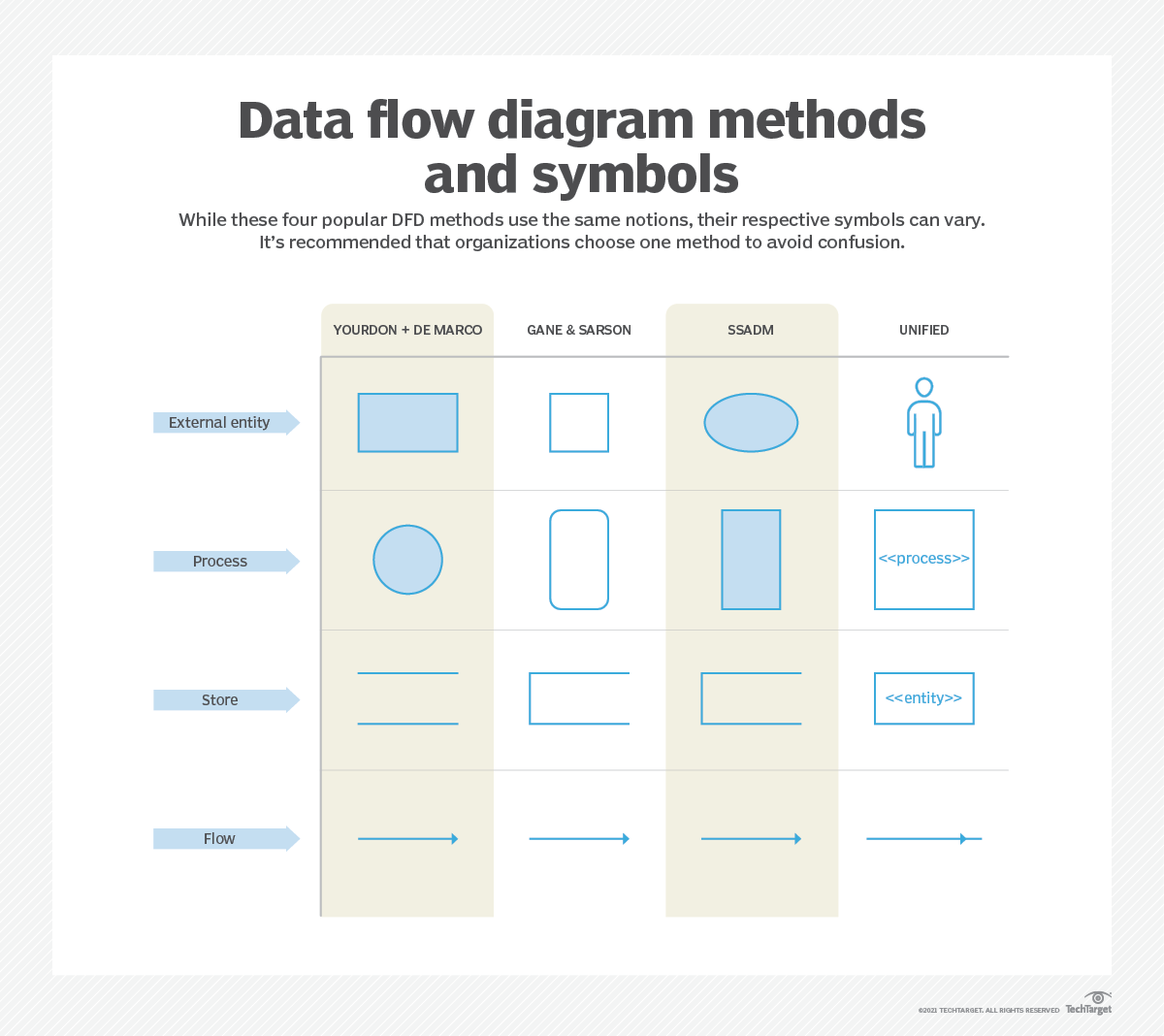
Different DFD notations include:

* 1. Gane and Sarson
  2. Yourdon and De Marco
  3. SSADM
  4. UML (commonly used to map software architecture, but can be used in DFDs)

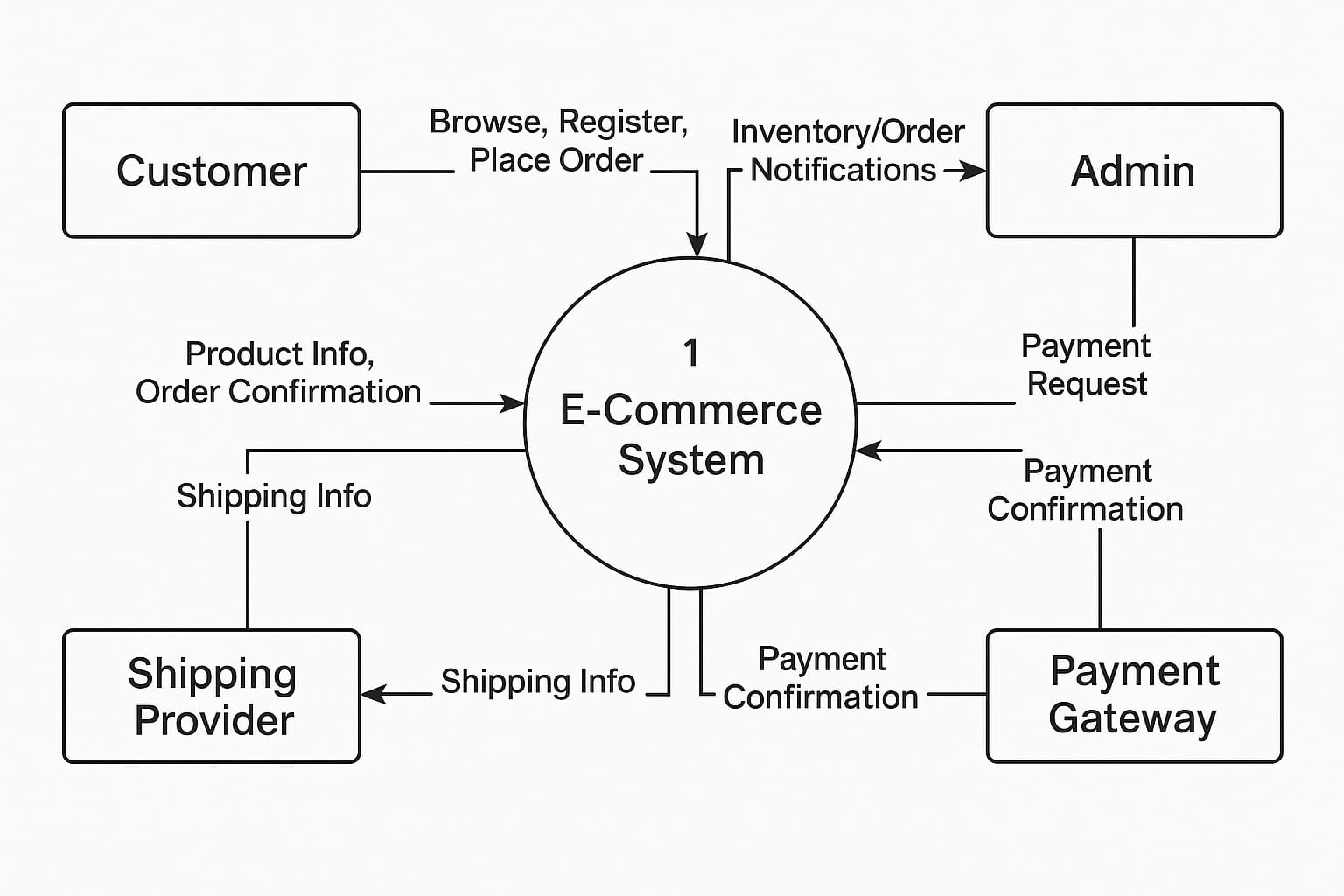
All DFD notions will represent the following:

* 1. External entities: information enters from or exits to the system being described
  2. Flows: define the movement of information to, from and within the system being described
  3. Stores: places where information is maintained or held, most often databases or database tables
  4. Processes: transform information

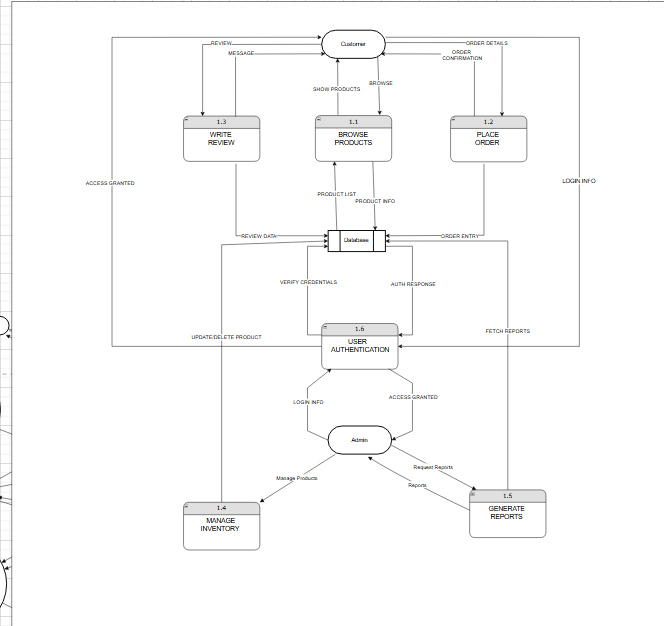
Different DFD methodologies use different symbol conventions. The differences and symbol rules are divergent enough to make it difficult for technologists to read the DFDs of methodologies they're not familiar with.Gane and Sarson notations will be used in this SRS.



* **Level-0 DFD :**

****

* **Level-1 DFD :**

****

## **A.6 Class Diagram:**

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and the constraints imposed on the system. The class diagrams are widely used in the modeling of object-oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

**Symbols and Notation:**

A class notation consists of three parts:

**Class Name**: The name of the class appears in the first partition.

**Class Attributes**: Attributes are shown in the second partition. The attribute type is shown after the colon. Attributes map onto member variables (data members) in code

**Class Operations**: (Methods)Operations are shown in the third partition. They are services the class provides. The return type of a method is shown after the colon at the end of the method signature. The return type of method parameters is shown after the colon following the parameter name. Operations map onto class methods in code.

# 

# 

# 

## 

# 

\

# 

# **CODING**

* pages/home/Home.jsx

import React from 'react'

import Banner from './Banner'

import Categories from './Categories'

const Home = () => {

return (

<>

<Banner/>

<Categories/>

</>

)

}

export default Home

* pages/home/Banner.jsx

import React from 'react'

import { Link } from 'react-router-dom'

import bannerImg from "../../assets/glass2.png"

const Banner = () => {

  return (

    <div className='section\_\_container header\_\_container'>

        <div className='header\_\_content z-30'>

        <h1 className="text-7xl font-bold">Discover and </h1>

            <h1 className="text-7xl font-bold">Customize your</h1>

            <h1 className="text-8xl font-bold">Perfect Lens!</h1>

            <div className="h-10"></div>

            <p>Explore our diverse range of high-quality optical lenses tailored for every need — from professional use. At OptiCloud, we combine innovation with customization to help you see better and live better. Dive into our smart selection tools and discover the lens that's just right for you.</p>

            <button className='btn'><Link to='/shop'>EXPLORE NOW</Link></button>

        </div>

        <div className='header\_\_image'>

            <img src={bannerImg} alt="banner image" />

        </div>

    </div> )}

* pages/home/Categories.jsx

import React from 'react'

import { Link } from 'react-router-dom'

import spherical from '../../assets/spherical.png'

import aspheric from '../../assets/aspheric.png'

import achro from '../../assets/achromatic.png'

import cylindrical from '../../assets/cylindrical.png'

import powell from '../../assets/Powell-prism.jpg'

import prism from '../../assets/prism.png'

const Categories = () => {

const categories = [

{name:'Spherical Singlet Lens',path:'Spherical Singlet Lens', image: spherical},

{name:'Aspheric Lens',path:'Aspheric Lens', image: aspheric},

{name:'Achromatic Lens',path:'Achromatic Lens', image: achro},

{name:'Cylindrical Lens',path:'Cylindrical Lens', image: cylindrical},

{name:'Powell Lens',path:'Powell Lens', image: powell},

{name:'Prism',path:'Prism', image: prism},

]

return (

<>

<section>

<div className="my-10 h-5 border-b-2 border-zinc-200 text-center text-2xl">

<span className="bg-white px-5 font-serif font-semibold" Featured Lenses </span>

</div>

<div className="product\_\_grid h-90">{

categories.map((category) => (

<Link key={category.name} to={`/categories/${category.path}`} className='categories\_\_card'>

<img src={category.image} alt={category.name} />

<h4 className="category-name">{category.name}</h4>

</Link> ))}

</div> </section>

</>

)}

export default Categories

# 

# pages/shop/ShopPage.jsx

import React, { useEffect, useState } from "react";

import productsData from "../../data/products.json";

import ProductCards from "./ProductCards";

import ShopFiltering from "./ShopFiltering";

import { useFetchAllProductsQuery } from "../../redux/features/products/productsApi";

const filters = {

categories: [

"all",

"Spherical Singlet Lens",

"Aspheric Lens",

"Achromatic Lens",

"Cylindrical Lens",

"Powell Lens",

"Prism", ],

priceRanges: [

{ label: "Under $50", min: 0, max: 50 },

{ label: "$50 - $100", min: 50, max: 100 },

{ label: "$100 - $200", min: 100, max: 200 },

{ label: "$200 and above", min: 200, max: Infinity },],};

const ShopPage = () => {

const [filtersState, setFiltersState] = useState({

category: "all",

priceRange: "", });

const [currentPage, setCurrentPage] = useState(1);

const [ProductsPerPage] = useState(8);

const { category, priceRange } = filtersState;

const [minPrice, maxPrice] = priceRange.split("-").map(Number);

const {

data: { products = [], totalPages, totalProducts } = {}, error, isLoading, } = useFetchAllProductsQuery({

category: category !== "all" ? category : "",

minPrice: isNaN(minPrice) ? "" : minPrice,

maxPrice: isNaN(maxPrice) ? "" : maxPrice,

page: currentPage,

limit: ProductsPerPage,

});

// clear the filters

const clearFilters = () => {

setFiltersState({

category: "all",

priceRange: "", });};

if (isLoading) return <div>Loading....</div>;

if (error) return <div>Error loading products.</div>;

const startProduct = (currentPage - 1) \* ProductsPerPage + 1;

const endProduct = startProduct + products.length - 1;

// handle page change

const handlePageChange = (pageNumber) => {

if (pageNumber > 0 && pageNumber <= totalPages) {

setCurrentPage(pageNumber);}};

return (

<>

<section className="section\_\_container bg-primary-light">

<h2 className="section\_\_header capitalize">Shop Page</h2>

<p className="section\_\_subheader">Discover the Hottest Picks: Elevate Your Style with Our Curated Collection of Trending Women's Fashion Products. </p>

</section>

<section className="section\_\_container">

<div className="flex flex-col md:flex-row md:gap-12 gap-8">

{/\* left side \*/}

<ShopFiltering

filters={filters}

filtersState={filtersState}

setFiltersState={setFiltersState}

clearFilters={clearFilters}

/>

{/\* right side \*/}

<div>

<h3 className="text-xl font-medium mb-4">

Showing {startProduct} to {endProduct} of {products.length}

{/\* Showing {startProduct} to {endProduct} of {totalProducts} products \*/}

</h3>

<ProductCards products={products} />

{/\* paginatin controls \*/}

<div className="mt-6 flex justify-center">

<button disabled={currentPage === 1} onClick={() => handlePageChange(currentPage - 1)} className="px-4 py-2 bg-gray-300 text-gray-700 rounded-md mr-2 > Previous </button>

{[...Array(totalPages)].map((\_, index) => (

<button

key={index}

onClick={() => handlePageChange(index + 1)}

className={`px-4 py-2 ${

currentPage === index + 1

? "bg-blue-500 text-white"

: "bg-gray-300 text-gray-700"} rounded-md mx-1 `}>

{index + 1} </button> ))}

<button

disabled={currentPage === totalPages}

onClick={() => handlePageChange(currentPage + 1)}

className="px-4 py-2 bg-gray-300 text-gray-700 rounded-md ml-2">

Next

</button>

</section>

</>);};

export default ShopPage;

# search/Search.jsx

import React, { useState } from 'react'

import productsData from "../../data/products.json"

import ProductCards from '../shop/ProductCards';

const Search = () => {

const [searchQuery, setSearchQuery] = useState('');

const [filteredProducts, setFilteredProducts] = useState(productsData);

const handleSearch = () => {

const query = searchQuery.toLowerCase();

const filtered = productsData.filter(product => product.name.toLowerCase().includes(query) || product.description.toLowerCase().includes(query));

setFilteredProducts(filtered); }

return (

<>

<section className='section\_\_container bg-primary-light'>

<h2 className='section\_\_header capitalize'>Search Products</h2>

<p className='section\_\_subheader'>Browse a diverse range of categories, from chic dresses to versatile accessories. Elevate your style today!</p>

</section>

<section className='section\_\_container'>

<div className='w-full mb-12 flex flex-col md:flex-row items-center justify-center gap-4'>

<input type="text" value={searchQuery} onChange={(e) => setSearchQuery(e.target.value)} className='search-bar w-full max-w-4xl p-2 border rounded' placeholder='Search for products...' />

<button onClick={handleSearch}className='search-button w-full md:w- auto py-2 px-8 bg-primary text-white rounded'>Search</button>

</div>

<ProductCards products={filteredProducts}/>

</section>

</>

)

}

export default Search

# components/Login.jsx

import React, { useState } from 'react'

import { useDispatch } from 'react-redux';

import { Link, useNavigate } from 'react-router-dom'

import { useLoginUserMutation } from '../redux/features/auth/authApi';

import { setUser } from '../redux/features/auth/authSlice';

const Login = () => {

const [message, setMessage] = useState('');

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const dispatch = useDispatch();

const [loginUser, {isLoading: loginLoading}] = useLoginUserMutation()

const navigate = useNavigate()

const handleLogin = async (e) => {

e.preventDefault();

const data = { email, password }

try {

const response = await loginUser(data).unwrap();

console.log(response)

const {token, user} = response;

dispatch(setUser({user}))

alert("Login successful");

navigate("/")

} catch (error) {

setMessage("Please provide a valid email and password") }}

return (

<section className='h-screen flex items-center justify-center'>

<div className='max-w-sm border shadow bg-white mx-auto p-8'>

<h2 className='text-2xl font-semibold pt-5'>Please Login</h2>

<form className='space-y-5 max-w-sm mx-auto pt-8' onSubmit={handleLogin}>

<input type="email" name="email" id="email"

onChange={(e) => setEmail(e.target.value)}

placeholder='Email Address' required

className='w-full bg-gray-100 focus:outline-none px-5 py-3 />

<input type="password" name="password" id="password"

onChange={(e) => setPassword(e.target.value)}

placeholder='Password' required

className='w-full bg-gray-100 focus:outline-none px-5 py-3' /> {

message && <p className='text-red-500'>{message}</p>}

<button type='button' onClick={handleLogin} className='w-full mt-5 bg- primary text-white hover:bg-indigo-500 font-medium py-3 rounded-md' >Login</button>

</form>

<p className='my-5 italic text-sm text-center'>Don't have an account?

<Link to="/register" className='text-red-700 px-1 underline'>Register</Link> here.</p> </div>

</section>

)

}

export default Login

* user/dashboard/UserDMain.jsx

import React from 'react'

import { useSelector } from 'react-redux'

import { Bar } from "react-chartjs-2"

import {Chart as ChartJS, CategoryScale, LinearScale, BarElement, Title, Tooltip, Legend} from "chart.js"

import { useGetUserStatsQuery } from '../../../../redux/features/stats/statsApi';

import UserStats from './UserStats';

ChartJS.register(CategoryScale, LinearScale, BarElement, Title, Tooltip, Legend)

const UserDMain = () => {

const {user} = useSelector((state) => state.auth);

const {data: stats, error, isLoading} = useGetUserStatsQuery(user?.email)

console.log(stats)

if(isLoading) return <div className='text-center text-gray-500'>Loading...</div>

if(!stats) {

return <div className='text-center text-gray-500'>No data available.</div> }

const data = {

labels: ['Total Payments', 'Total Reviews', 'Total Purchased Products'],

datasets: [

{

label: 'User Stats',

data: [stats.totalPayments, stats.totalReviews \* 100, stats.totalPurchasedProducts \* 100],

backgroundColor: 'rgba(75, 192, 192, 0.2)',

borderColor:'rgba(75, 192, 192, 1)',

borderWidth: 1,

} ] }

const options= {

responsive: true,

plugins: {

legend: {

position: 'top', },

tooltip: {

callbacks: {

label: function (tooltipItem) {

return `${tooltipItem.label}: ${tooltipItem.raw}`

}}}}}

return (

<div className='p-6'>

<div>

<h1 className='text-2xl font-semibold mb-4'>User Dashboard</h1>

<p className='text-gray-500'>Hi, {user?.username}! Welcome to your user dashboard</p>

</div>

<UserStats stats={stats}/>

<div className='mb-6'>

<Bar data={data} options={options}/>

</div>

</div>)}

export default UserDMain

* user/UserOrders.jsx

import React from 'react'

import { useSelector } from 'react-redux'

import { useGetOrdersByEmailQuery } from '../../../redux/features/orders/orderApi';

import { Link } from 'react-router-dom';

const UserOrders = () => {

const {user} = useSelector((state) => state.auth);

const {data: orderdata, error, isLoading} = useGetOrdersByEmailQuery(user?.email);

const orders = orderdata?.orders;

console.log(orders)

if(isLoading) return <div>Loading...</div>

if(error) return <div>No order found!</div>

return (

<section className="py-1 bg-blueGray-50">

<div className="w-full mb-12 xl:mb-0 px-4 mx-auto">

<div className="relative flex flex-col min-w-0 break-words bg-white w-full mb-6 shadow-lg rounded ">

<div className="rounded-t mb-0 px-4 py-3 border-0">

<div className="flex flex-wrap items-center">

<div className="relative w-full px-4 max-w-full flex-grow flex-1">

<h3 className="font-semibold text-base text-blueGray-700">Your Orders</h3> </div>

<div className="relative w-full px-4 max-w-full flex-grow flex-1 text-right">

<button className="bg-indigo-500 text-white active:bg-indigo-600 text-xs font-bold uppercase px-3 py-1 rounded outline-none focus:outline-none mr-1 mb-1 ease-linear transition-all duration-150" type="button">See all</button>

</div>

</div>

</div>

<div className="block w-full overflow-x-auto">

<table className="items-center bg-transparent w-full border-collapse ">

<thead>

<tr>

<th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left"> #</th>

<th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left">Order ID </th>

<th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left">Date</th>

<th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left">Status</th>

<th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left">Total </th>

<th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left">View Order </th> </tr></thead>

<tbody> {

orders && orders.map((order, index) =>(

<tr key={index}>

<th className="border-t-0 px-6 align-middle border-l-0 border-r-0 text-xs whitespace-nowrap p-4 text-left text-blueGray-700 ">{index + 1}</th>

<td className="border-t-0 px-6 align-middle border-l-0 border-r-0 text-xs whitespace-nowrap p-4 "> {order?.orderId}</td>

<td className="border-t-0 px-6 align-center border-l-0 border-r-0 text-xs whitespace-nowrap p-4">{new Date(order?.createdAt).toLocaleDateString()}

</td><td className="border-t-0 px-6 align-middle border-l-0 border-r-0 text-xs whitespace-nowrap p-4">

<span className={`p-1 rounded

${order?.status === 'completed' ? 'bg-green-100 text-green-700' : order?.status === 'pending' ? 'bg-red-100 text-red-700' : order?.status === 'processing' ? 'bg-blue-100 text-blue-600' : 'bg-indigo-100 text-indigo-600'}`}>{order?.status}</span></td>

<td className="border-t-0 px-6 align-middle border-l-0 border-r-0 text-xs whitespace-nowrap p-4">{order?.amount}</td>

<td className="border-t-0 px-6 align-middle border-l-0 border-r-0 text-xs whitespace-nowrap p-4"><Link to={`/orders/${order?.\_id}`} className='underline hover:text-primary'>view order</Link></td>

</tr> ))}

</tbody>

</table>

</section>)}

export default UserOrders

## user/OrderDetails.jsx

import React from 'react'

import { useGetOrderByIdQuery } from '../../../redux/features/orders/orderApi';

import { useParams } from 'react-router-dom';

import TimelineStep from '../../../components/TimelineStep';

const OrderDetails = () => {

const { orderId } = useParams();

console.log(orderId)

const { data: order, error, isLoading } = useGetOrderByIdQuery(orderId);

if(isLoading) return <div>Loading...</div>

if(error) return <div>No orders!</div>

const isCompleted = (status) => {

const statuses = ["pending", "processing", "shipped", "completed"];

return statuses.indexOf(status) < statuses.indexOf(order.status)}

const isCurrent = (status) => order.status === status;

const steps = [

{ status: 'pending', label: 'Pending', description: 'Your order has been created and is awaiting processing.', icon: { iconName: 'time-line', bgColor: 'red-500', textColor: 'gray-800' }, },

{ status: 'processing', label: 'Processing', description: 'Your order is currently being processed.', icon: { iconName: 'loader-line', bgColor: 'yellow-800', textColor: 'yellow-800' },},

{ status: 'shipped', label: 'Shipped', description: 'Your order has been shipped.', icon: { iconName: 'truck-line', bgColor: 'blue-800', textColor: 'blue-800' }, },

{ status: 'completed', label: 'Completed', description: 'Your order has been successfully completed.', icon: { iconName: 'check-line', bgColor: 'green-800', textColor: 'green-900' },}, ];

return (

<section className='section\_\_container rounded p-6'>

<h2 className='text-2xl font-semibold mb-4'>Payment {order?.status}</h2>

<p className='mb-8'>Status: {order?.status}</p>

<ol className='sm:flex items-center relative'>

{ steps.map((step, index) => (

<TimelineStep

key={index}

step={step}

order={order}

isCompleted={isCompleted(step.status)}

isCurrent={isCurrent(step.status)}

isLastStep = {index === steps.length - 1}

icon={step.icon}

description={step.description}

/>))} </ol>

</section>)}

export default OrderDetails

## user/UserProfile.jsx

import React, { useEffect, useState } from 'react'

import { useDispatch, useSelector } from 'react-redux'

import { useEditProfileMutation } from '../../../redux/features/auth/authApi';

import avatarImg from '../../../assets/avatar.png'

import { setUser } from '../../../redux/features/auth/authSlice';

const UserProfile = () => {

const dispatch = useDispatch();

const { user } = useSelector((state) => state.auth);

const [editProfile, { isLoading, isError, error, isSuccess }] = useEditProfileMutation();

const [formData, setformData] = useState({

username: '',

profileImage: '',

bio: '',

profession: '',

userId: ''

});

const [isModalOpen, setIsModalOpen] = useState(false);

useEffect(() => {

if(user) {

setformData({

username: user?.username || '',

profileImage: user?.profileImage || '',

bio: user?.bio || '',

profession: user?.profession || '',

userId: user?.\_id || ''}) }}, [user])

const handleChange = (e) => {

setformData({

...formData,

[e.target.name]: e.target.value}) }

const handleSubmit =async (e) => {

e.preventDefault();

const updatedUser = {

username: formData.username,

profileImage: formData.profileImage,

bio: formData.bio,

profession: formData.profession,

userId: formData.userId}

try {

const response = await editProfile(updatedUser).unwrap();

console.log(response)

dispatch(setUser(response.user));

localStorage.setItem('user', JSON.stringify(response.user))

alert('Profile updated successfully!');

} catch (error) {

console.error("Failed to update profile", error) ;

alert("Failed to update profile. Please try again")

}

setIsModalOpen(false)

}

return (

<div className='container mx-auto p-6'>

<div className='bg-white shadow-md rounded-lg p-6'>

<div className='flex items-center mb-4'>

<img src={formData?.profileImage || avatarImg} alt="" className='w-32 h-32 object-cover rounded-full' />

<div className='ml-6'>

<h3 className='text-2xl font-semibold'>Username: {formData?.username || 'N/A'}</h3>

<p className='text-gray-700'>User Bio: {formData.bio || 'N/A'}</p>

<p className='text-gray-700'>Profession: {formData.profession || 'N/A'}</p>

</div>

<button

onClick={() => setIsModalOpen(true)}

className='ml-auto text-blue-500 hover:text-blue-700'>

<svg className="w-6 h-6" fill="none" stroke="currentColor" viewBox="0 0 24 24" xmlns="http://www.w3.org/2000/svg">

<path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="M11 3H4a1 1 0 00-1 1v14a1 1 0 001 1h7m2 0h7a1 1 0 001-1V4a1 1 0 00-1-1h-7m-2 0v14"></path>

</svg>

</button></div></div>

{/\* show modal \*/}

{

isModalOpen && (

<div className='fixed inset-0 bg-black bg-opacity-90 flex items-center justify-center z-50'>

<div className='bg-white p-6 rounded-lg md:w-96 max-w-xl mx-auto relative'>

<button onClick={() => setIsModalOpen(false)}

className='absolute top-2 right-2 text-gray-500 hover:text-gray-700'><i className="ri-close-line size-8 p-2 bg-black rounded-full"></i></button>

<h2 className='text-2xl font-bold mb-4'>Edit Profile</h2>

<form onSubmit={handleSubmit}>

<div className='mb-4'>

<label htmlFor="username" className='block text-sm font-medium text-gray-700 '>Username</label>

<input type="text" name='username' value={formData?.username}

onChange={handleChange}

placeholder='username'

className='mt-1 p-2 w-full border border-gray-300 rounded-md shadow-sm' required />

</div>

<div className='mb-4'>

<label htmlFor="profileImage" className='block text-sm font-medium text-gray-700 '>Profile Image Url</label>

<input type="text" name='profileImage' value={formData?.profileImage}

onChange={handleChange}

placeholder='profileImage url'

className='mt-1 p-2 w-full border border-gray-300 rounded-md shadow-sm' required />

</div>

<div className='mb-4'>

<label htmlFor="bio" className='block text-sm font-medium text-gray-700 '>Write Your Bio</label>

<textarea name="bio" row="3"

className='mt-1 p-2 w-full border border-gray-300 rounded-md shadow-sm' value={formData?.bio} onChange={handleChange} placeholder='add your bio' ></textarea> </div>

<div className='mb-4'>

<label htmlFor="profession" className='block text-sm font-medium text-gray-700 '>Profession</label>

<input type="text" name='profession' value={formData?.profession}

onChange={handleChange} placeholder='profession'

className='mt-1 p-2 w-full border border-gray-300 rounded-md shadow-sm' required /></div>

<button className={`mt-4 w-full bg-blue-500 text-white py-2 px-4 rounded-md ${isLoading ? 'opacity-50 cursor-not-allowed' : ''} `} type='submit'

disabled={isLoading} >{isLoading ? 'Saving...' : 'Save Changes'}</button>

{isError && <p className='mt-2 text-red-500'>Failed to update profile. Please try again</p>}

{isSuccess && <p className='mt-2 text-green-500'>Profile updated successfully!</p>}

</form>

</div>

</div>)}

</div>)}

export default UserProfile

## pages/dashboard/user/UserReviews.jsx

import React from 'react'

import { useSelector } from 'react-redux'

import { useGetReviewsByUserIdQuery } from '../../../redux/features/reviews/reviewsApi';

import { useNavigate } from 'react-router-dom';

const UserReviews = () => {

const {user} = useSelector((state) => state.auth);

const {data: reviews, error, isLoading} = useGetReviewsByUserIdQuery(user?.\_id);

const navigate = useNavigate()

if(isLoading) return <div>Loading...</div>

if(error) return <div>Failed to load reviews!</div>

const handleCardClick = () => {

navigate('/shop')}

return (

<div className='py-6'>

<h2 className='text-2xl font-bold mb-8"'>Your given Reviews</h2>

<div className='grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 mt-8 gap-6'>

{ reviews && reviews.map((review, index) => (

<div key={index} className='bg-white shadow-md rounded-lg p-4 border-gray-200 cursor-pointer hover:scale-105 transition-all duration-200'>

<p className='text-lg font-semibold mb-2'>Rating: {review?.rating}</p>

<p className='mb-2'><strong>Comment:</strong> {review?.comment}</p>

<p className='text-sm text-gray-500'><strong>ProductId:</strong> {review?.productId}</p>

<p className='text-sm text-gray-500'><strong>Date:</strong> {new Date(review?.createdAt).toLocaleDateString()}</p></div> ))}

<div onClick={handleCardClick} className='bg-gray-100 text-black flex items-center justify-center rounded-lg p-6 border cursor-pointer hover:bg-primary hover:text-white transition-all duration-200'>

<span>+</span>

<p>Add New Review</p>

</div>

</div>

</div>)}

export default UserReviews

## pages/dashboard/AdminDashboard.jsx

import { useLogoutUserMutation } from '../../redux/features/auth/authApi';

import { useDispatch } from 'react-redux';

import { Link, NavLink, useNavigate } from 'react-router-dom';

import { logout } from '../../redux/features/auth/authSlice';

const navItems = [

{ path: '/dashboard/admin', label: 'Dashboard' },

{ path: '/dashboard/add-product', label: 'Add Product' },

{ path: '/dashboard/manage-products', label: 'Manage Products' },

{ path: '/dashboard/users', label: 'Users' },

{ path: '/dashboard/manage-orders', label: 'Manage Orders' },

]

const AdminDashboard = () => {

const [logoutUser] = useLogoutUserMutation();

const dispatch = useDispatch();

const navigate = useNavigate()

const handleLogout = async () => {

try {

await logoutUser().unwrap();

dispatch(logout())

navigate('/')

} catch (error) {

console.error("Failed to log out", error) }}

return (

<div className='space-y-5 bg-white p-8 md:h-screen flex flex-col justify-between'>

<div>

<div className='nav\_\_logo'>

<Link to="/">OptiCloud<span>.</span></Link>

<p className='text-xs italic'>Admin dashboard</p> </div>

<hr className='mt-5' />

<ul className='space-y-5 pt-5'>{ navItems.map((item) =>(

<li key={item.path}> <NavLink className={ ({isActive}) => isActive ? "text-blue-600 font-bold" : 'text-black'} end to={item.path} > {item.label}</NavLink></li> )) } </ul></div>

<div className='mb-3'><hr className='mb-3'/>

<button onClick={handleLogout} className='text-white bg-primary font-medium px-5 py-1 rounded-sm'>Logout</button>

</div>

</div>)}

export default AdminDashboard

## pages/dashboard/admin/AddProduct.jsx

import React, { useState } from 'react';

import {useSelector } from 'react-redux';

import TextInput from './TextInput';

import SelectInput from './SelectInput';

import UploadImage from './UploadImage';

import { useAddProductMutation } from '../../../../redux/features/products/productsApi';

import { useNavigate } from 'react-router-dom';

const categories = [

{ label: 'Select Category', value: '' },

{ label: 'Spherical Singlet Lens', value: 'spherical' },

{ label: 'Aspheric Lens', value: 'aspheric' },

{ label: 'Achromatic Lens', value: 'achromatic' },

{ label: 'Cylindrical Lens', value: 'cylindrical' },

{ label: 'Powell Lens', value: 'powell' },

{ label: 'Prism', value: 'prism' },

];

const AddProduct = () => {

const { user } = useSelector((state) => state.auth);

const [product, setProduct] = useState({

name: '', category: '', rice: '',description: ''

});

const [image, setImage] = useState('');

const [AddProduct, {isLoading, error}] = useAddProductMutation()

const handleChange = (e) => {

const { name, value } = e.target;

setProduct({

...product,

[name]: value });};

const navigate = useNavigate()

const handleSubmit = async(e) => {

e.preventDefault();

if(!product.name || !product.category || !product.price || !product.description) {

alert('Please fill all the required fields');

return; }

try {

await AddProduct({...product, url: image, author: user?.\_id}).unwrap();

alert('Product added successfully');

setProduct({ name: '',

category: '', price: '', description: ''}) setImage('');

navigate("/shop")

} catch (error) {

console.log("Failed to submit product", error); }}

return (

<div className="container mx-auto mt-8">

<h2 className="text-2xl font-bold mb-6">Add New Product</h2>

<form onSubmit={handleSubmit} className="space-y-4">

<TextInput

label="Product Name"

name="name"

placeholder="Ex: Plano Concave"

value={product.name}

onChange={handleChange}/>

<SelectInput

label="Category"

name="category"

value={product.category}

onChange={handleChange}

options={categories}/>

<TextInput

label="Price"

name="price"

type="number"

placeholder="50"

value={product.price}

onChange={handleChange} />

<UploadImage

name="image" id="image" value={e => setImage(e.target.value)}

placeholder='Image' setImage={setImage}/> <div>

<label htmlFor="description" className='block text-sm font-medium text-gray-700'>Description</label>

<textarea name="description" id="description" className='add-product-InputCSS' value={product.description} placeholder='Write a product description'

onChange={handleChange}></textarea></div>

<div><button type='submit' className='add-product-btn'>Add Product</button>

</div>

</form>

);};

export default AddProduct;

## pages/dashboard/admin /ManageProduct.jsx

import React, { useState } from 'react'

import { useDeleteProductMutation, useFetchAllProductsQuery } from '../../../../redux/features/products/productsApi'

import { formatDate } from '../../../../utils/formateDate';

import { Link } from 'react-router-dom';

const ManageProduct = () => {

const [currentPage, setCurrentPage] = useState(1);

const [productsPerPage] = useState(12)

const { data: { products = [], totalPages, totalProducts } = {}, isLoading, error, refetch } = useFetchAllProductsQuery({

category: '',

// color: '',

minPrice: '',

maxPrice: '',

page: currentPage,

limit: productsPerPage,

})

// pagination

const startProduct = (currentPage - 1) \* productsPerPage + 1;

const endProduct = startProduct + products.length - 1;

const handlePageChange = (pageNumber) => {

if (pageNumber > 0 && pageNumber <= totalPages) {

setCurrentPage(pageNumber) }}

const [deleteProduct] = useDeleteProductMutation()

const handleDeleteProduct = async (id) => {

try {

const response = await deleteProduct(id).unwrap();

alert("Product deleted successfully")

await refetch()

} catch (error) {

console.error("Error deleting product", error) }}

return (

<>

{ isLoading && <div>Loading...</div>}

{ error && <div>Error loading products.</div>}

<section className="py-1 bg-blueGray-50">

<div className="w-full mb-12 xl:mb-0 px-4 mx-auto">

<div className="relative flex flex-col min-w-0 break-words bg-white w-full mb-6 shadow-lg rounded ">

<div className="rounded-t mb-0 px-4 py-3 border-0">

<div className="flex flex-wrap items-center">

<div className="relative w-full px-4 max-w-full flex-grow flex-1">

<h3 className="font-semibold text-base text-blueGray-700">All Products</h3> </div>

<div className="relative w-full px-4 max-w-full flex-grow flex-1 text-right">

<button className="bg-indigo-500 text-white active:bg-indigo-600 text-xs font-bold uppercase px-3 py-1 rounded outline-none focus:outline-none mr-1 mb-1 ease-linear transition-all duration-150" type="button">See all</button>

</div>

</div>

<h3 className='my-4 text-sm'>Showing {startProduct} to {endProduct} of {totalProducts} products</h3>

</div>

<div className="block w-full overflow-x-auto">

<table className="items-center bg-transparent w-full border-collapse ">

<thead>

<tr><th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left">No. </th>

<th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left">Product Name</th>

<th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left">Publishing date</th>

<th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left">Actions</th> </tr>

</thead>

<tbody>{products && products.map((product, index) => (

<tr key={index}>

<th className="border-t-0 px-6 align-middle border-l-0 border-r-0 text-xs whitespace-nowrap p-4 text-left text-blueGray-700 ">{index + 1}</th>

<td className="border-t-0 px-6 align-middle border-l-0 border-r-0 text-xs whitespace-nowrap p-4 ">

{product?.name}

</td><td className="border-t-0 px-6 align-center border-l-0 border-r-0 text-xs whitespace-nowrap p-4">{formatDate(product?.createdAt)} </td>

<td className="border-t-0 px-6 align-middle border-l-0 border-r-0 text-xs whitespace-nowrap p-4"><button onClick={() => handleDeleteProduct(product.\_id)} className='bg-red-600 text-white px-2 py-1'>Delete</button> </td> </tr>

)) }

</tbody>

</table>

</div>

</div>

</div>

{/\* pagination \*/}

<div className='mt-6 flex items-center justify-center'>

<button disabled={currentPage === 1} onClick={() => handlePageChange(currentPage - 1)} className='px-4 py-2 bg-gray-300 text-gray-700 rounded-md mr-2'>Previous</button>{ [...Array(totalPages)].map((\_, index) => (

<button onClick={() => handlePageChange(index + 1)} className={`px-4 py-2 ${currentPage === index + 1 ? 'bg-blue-500 text-white' : 'bg-gray-300 text-gray-700'} rounded-md mx-1`}>{index + 1}</button>))}

<button disabled={currentPage === totalPages} onClick={() => handlePageChange(currentPage + 1)} className='px-4 py-2 bg-gray-300 text-gray-700 rounded-md ml-2'>Next</button></div>

</section>

</>

)}

export default ManageProduct

## pages/dashboard/admin/users/ManageUser.jsx

import React, { useState } from 'react'

import { useDeleteUserMutation, useGetUserQuery } from '../../../../redux/features/auth/authApi'

import { Link } from 'react-router-dom';

import UpdateUserModal from './UpdateUserModal';

const ManageUser = () => {

const [isModalOpen, setIsModalOpen] = useState(false);

const [selectedUser, setSelectedUser] = useState(null)

const { data: users = [], error, isLoading, refetch } = useGetUserQuery();

console.log(users)

const [deleteUser] = useDeleteUserMutation()

const handleDelete = async (id) => {

try { const response = await deleteUser(id).unwrap();

alert("User deleted successfully!")

refetch();

} catch (error) {

console.error("Failed to delete user", error);

}}

const handleEdit = (user) => {

setSelectedUser(user)

setIsModalOpen(true) }

const handleCloseModal = () => {

setIsModalOpen(false)

setSelectedUser(null)

}

return (

<>

{ isLoading && <div>Loading...</div> }

{ error && <div>Error loading users data.</div>}

<section className="py-1 bg-blueGray-50">

<div className="w-full mb-12 xl:mb-0 px-4 mx-auto">

<div className="relative flex flex-col min-w-0 break-words bg-white w-full mb-6 shadow-lg rounded ">

<div className="rounded-t mb-0 px-4 py-3 border-0">

<div className="flex flex-wrap items-center">

<div className="relative w-full px-4 max-w-full flex-grow flex-1">

<h3 className="font-semibold text-base text-blueGray-700">All Users</h3> </div><div className="relative w-full px-4 max-w-full flex-grow flex-1 text-right"><button className="bg-indigo-500 text-white active:bg-indigo-600 text-xs font-bold uppercase px-3 py-1 rounded outline-none focus:outline-none mr-1 mb-1 ease-linear transition-all duration-150" type="button">See all</button>

</div>

</div>

</div>

<div className="block w-full overflow-x-auto">

<table className="items-center bg-transparent w-full border-collapse ">

<thead>

<tr><th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left">No. </th>

<th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left">User email </th>

<th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left">User role</th>

<th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left">Edit or manage</th>

<th className="px-6 bg-blueGray-50 text-blueGray-500 align-middle border border-solid border-blueGray-100 py-3 text-xs uppercase border-l-0 border-r-0 whitespace-nowrap font-semibold text-left">Actions</th></tr></thead>

<tbody>

{users && users.map((user, index) => (

<tr key={index}>

<th className="border-t-0 px-6 align-middle border-l-0 border-r-0 text-xs whitespace-nowrap p-4 text-left text-blueGray-700 ">{index + 1}</th>

<td className="border-t-0 px-6 align-middle border-l-0 border-r-0 text-xs whitespace-nowrap p-4 "> {user?.email || 'N/A'}</td>

<td className="border-t-0 px-6 align-center border-l-0 border-r-0 text-xs whitespace-nowrap p-4">

<span className={`rounded-full py-[2px] px-3 ${user?.role === "admin" ? "bg-indigo-500 text-white ": "bg-amber-300"}`} >{" "}{user?.role}</span> </td>

<td className="border-t-0 px-6 align-middle border-l-0 border-r-0 text-xs whitespace-nowrap p-4 cursor-pointer hover:text-primary">

<button onClick={() => handleEdit(user)} className='flex gap-1 items-center hover:text-red-500'> <i className='ri-edit-2-line'></i>Edit</button>

</td>

<td className="border-t-0 px-6 align-middle border-l-0 border-r-0 text-xs whitespace-nowrap p-4"><button onClick={() => handleDelete(user?.\_id)}

className='bg-red-600 text-white px-2 py-1'>Delete</button>

</td></tr>))}

</tbody>

</table>

</div>

</div>

</div>

</section>

{ isModalOpen && <UpdateUserModal user={selectedUser} onClose={handleCloseModal} onRoleUpdate={refetch} />

}

</>)}

export default ManageUser

## pages/dashboard/admin/users/UpdateUserModal.jsx

import React, { useState } from 'react'

import { useUpdateUerRoleMutation } from '../../../../redux/features/auth/authApi';

const UpdateUserModal = ({ user, onClose, onRoleUpdate }) => {

const [role, setRole] = useState(user.role);

const [updateUerRole] = useUpdateUerRoleMutation()

const handleUpdateRole = async () => {

try {

await updateUerRole({ userId: user?.\_id, role }).unwrap();

alert('Updated role successfully!')

onRoleUpdate();

onClose();

} catch (error) {

console.error("Failed to update user role", error); }}

return (

<div className='fixed inset-0 flex items-center justify-center bg-black bg-opacity-80'>

<div className='bg-white p-4 rounded shadow-lg w-1/3'>

<h2 className='text-xl mb-4'>Edit User Role</h2>

<div className='mb-4 space-y-4'>

<label className='block text-sm font-medium text-gray-700'>Email</label>

<input type="email"

value={user?.email}

readOnly

className='mt-1 bg-gray-100 block w-full shadow-sm sm:text-sm border-gray-300 rounded-md py-2.5 px-5 focus:outline-none'/>

</div>

<div className='mb-4 space-y-4'>

<label className='block text-sm font-medium text-gray-700'>Role</label>

<select value={role} onChange={(e) => setRole(e.target.value)}

className='block w-full shadow-sm sm:text-sm bg-gray-100 border-gray-300 rounded-md py-2.5 px-5 focus:outline-none' >

<option value="user">User</option>

<option value="admin">Admin</option>

</select>

</div>

<div className='flex justify-end pt-5'>

<button onClick={onClose}

className="bg-primary text-white px-4 py-2 rounded mr-2">Cancel</button>

<button onClick={handleUpdateRole} className="bg-indigo-500 text-white px-4 py-2 rounded">Save</button>

</div>

</div>

</div> )}

export default UpdateUserModal

## pages/dashboard/admin/manageOrders /ManageOrders.jsx

import React, { useState } from 'react'

import { useDeleteOrderMutation, useGetAllOrdersQuery } from '../../../../redux/features/orders/orderApi'

import { formatDate } from '../../../../utils/formateDate';

import { Link } from 'react-router-dom';

import UpdateOrderModal from './UpdateOrderModal';

const ManageOrders = () => {

const { data: orders, error, isLoading, refetch } = useGetAllOrdersQuery();

const [selectedOrder, setSelectedOrder] = useState(null);

const [isModalOpen, setIsModalOpen] = useState(false);

const [deleteOrder] = useDeleteOrderMutation();

const handleEditOrder = (order) => {

setSelectedOrder(order);

setIsModalOpen(true);

}

const handleCloseModal = () => {

setIsModalOpen(false);

setSelectedOrder(null);

}

const handleDeleteOder = async (orderId) => {

try {

await deleteOrder(orderId).unwrap();

alert("Order deleted successfully");

refetch();

} catch (error) {

console.error("Failed to delete order:", err);

}}

if (isLoading) return <div>Loading....</div>

if (error) return <div>Something went wrong!</div>

return (

<div className='section\_\_container p-6'>

<h2 className='text-2xl font-semibold mb-4'>Manage Orders</h2>

<table className='min-w-full bg-white border border-gray-200 rounded-lg'>

<thead className='bg-gray-100'>

<tr>

<th className='py-3 px-4 border-b'>Order Id</th>

<th className='py-3 px-4 border-b'>Customer</th>

<th className='py-3 px-4 border-b'>Status</th>

<th className='py-3 px-4 border-b'>Date</th>

<th className='py-3 px-4 border-b'>Actions</th>

</tr>

</thead>

<tbody>

{

orders && orders.map((order, index) => (

<tr key={index}>

<td className='py-3 px-4 border-b'>{order?.orderId}</td>

<td className='py-3 px-4 border-b'>{order?.email}</td>

<td className='py-3 px-4 border-b'>

<span className={`inline-block px-3 py-1 text-xs text-white rounded-full ${getStatusColor(order?.status)}`}>{order?.status}</span>

</td>

<td className='py-3 px-4 border-b'>{formatDate(order?.updatedAt)}</td>

<td className='py-3 px-4 border-b flex items-center space-x-4'>

{/\* <Link to='#' className="text-blue-500 hover:underline">View</Link> \*/}

<button className="text-green-500 hover:underline" onClick={() => handleEditOrder(order)}>Edit</button>

<button className="text-red-500 hover:underline" onClick={() => handleDeleteOder(order?.\_id)}>Delete</button>

</td>

</tr>

)) }

</tbody>

</table>

{/\* update order modal \*/}

{

selectedOrder && (

<UpdateOrderModal

order={selectedOrder}

isOpen={isModalOpen}

onClose={handleCloseModal}

/>)}

</div>

)}

const getStatusColor = (status) => {

switch (status) {

case 'pending':

return 'bg-yellow-500';

case 'processing':

return 'bg-blue-500';

case 'shipped':

return 'bg-green-500';

case 'completed':

return 'bg-gray-500';

default:

return 'bg-gray-300';

}};

export default ManageOrders

## pages/dashboard/admin/manageOrders /UpdateOrderModal.jsx

import React, { useState } from 'react'

import { useUpdateOrderStatusMutation } from '../../../../redux/features/orders/orderApi';

const UpdateOrderModal = ({order, isOpen, onClose}) => {

const [status, setStatus] = useState(order?.status);

const [updateOrderStatus, {isLoading, error}] = useUpdateOrderStatusMutation();

const handleUpdateOrderStatus = async ()=> {

try {

await updateOrderStatus({id: order?.\_id, status})

onClose();

} catch (error) {

console.error("Failed to update order status:", err);}

if(!isOpen) return null;

return (

<div className="fixed inset-0 flex items-center justify-center bg-black bg-opacity-80">

<div className="bg-white p-6 rounded-lg shadow-lg max-w-sm w-full">

<h2 className="text-xl font-semibold mb-4">Update Order Status</h2>

<div className="mb-4">

<label className="block text-gray-700 mb-2" htmlFor="status">Status</label>

<select id="status"

value={status}

onChange={(e) => setStatus(e.target.value)}

className="border border-gray-300 p-2 rounded w-full">

<option value="pending">Pending</option>

<option value="processing">Processing</option>

<option value="shipped">Shipped</option>

<option value="completed">Completed</option>

</select></div>

{error && <p className="text-red-500 mb-4">Failed to update status.</p>}

<div className="flex justify-end space-x-2">

<button

onClick={onClose}

className="bg-gray-300 text-gray-800 px-4 py-2 rounded">Cancel</button>

<button

onClick={handleUpdateOrderStatus}

disabled={isLoading}

className="bg-blue-500 text-white px-4 py-2 rounded">

{isLoading ? 'Updating...' : 'Update'}

</button>

</div>

</div>

</div>

)}

export default UpdateOrderModal

# 

# 

# 

# **SCREENSHOTS**

# 

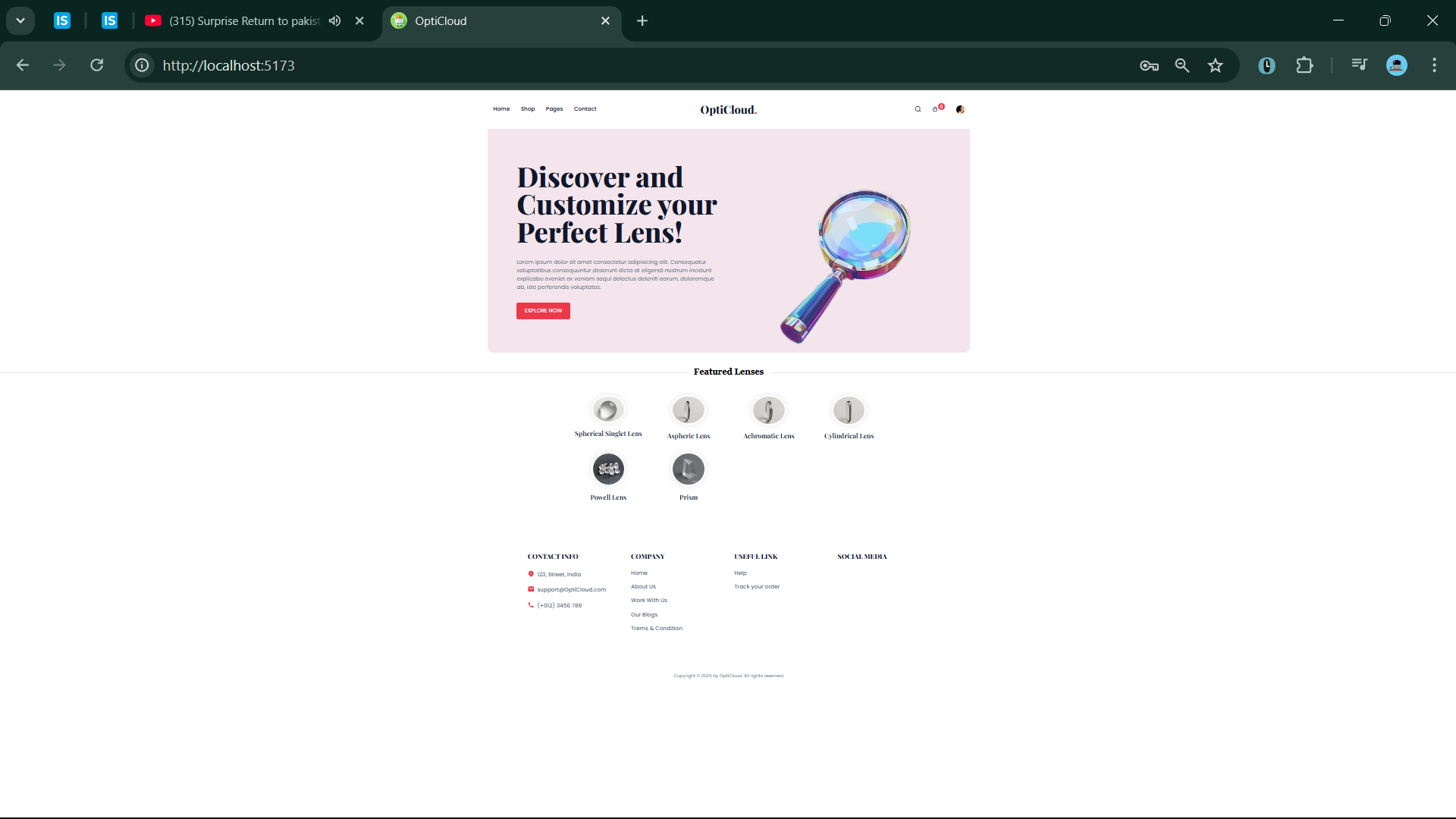
# 

# 

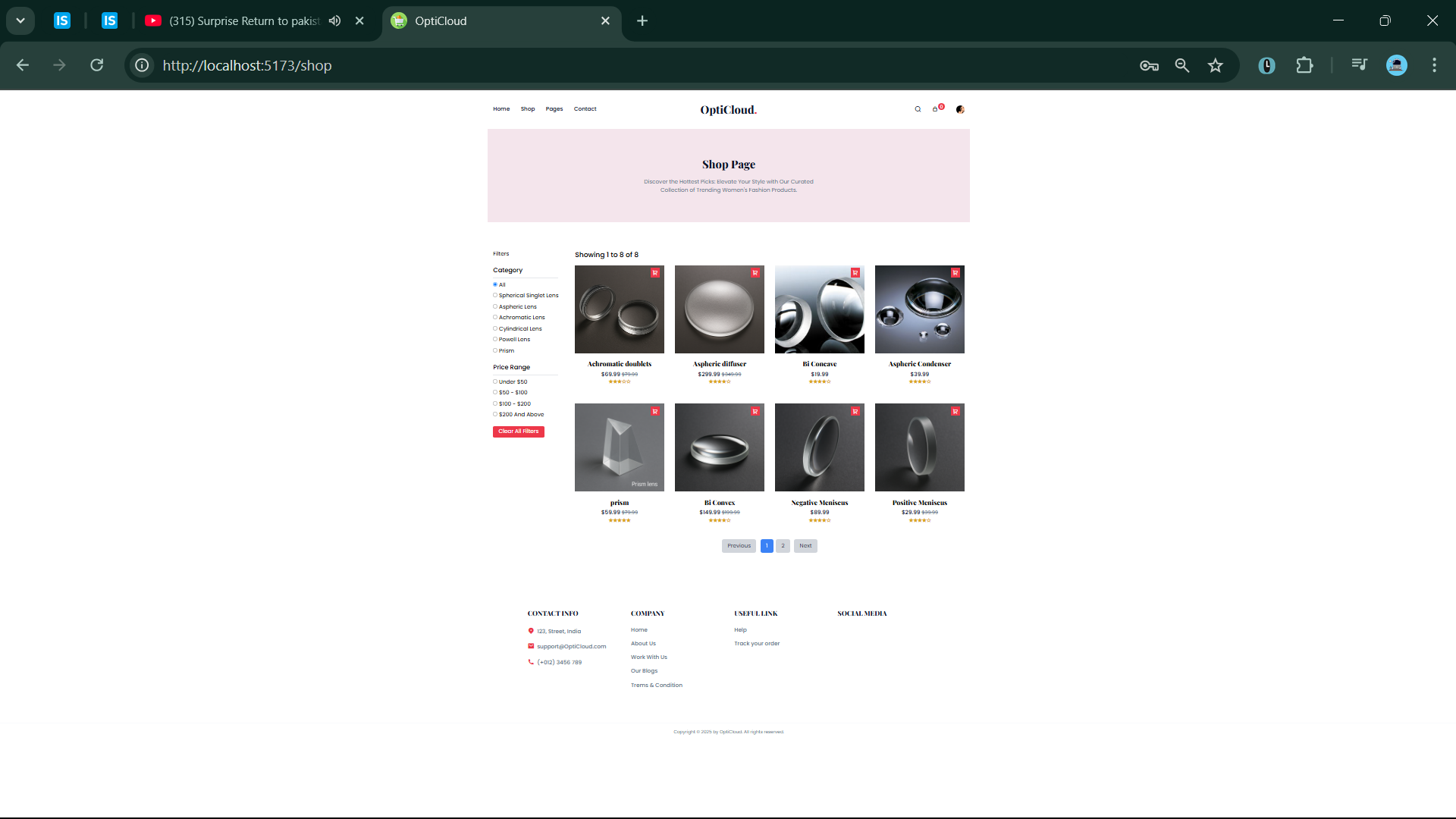
# 

# 

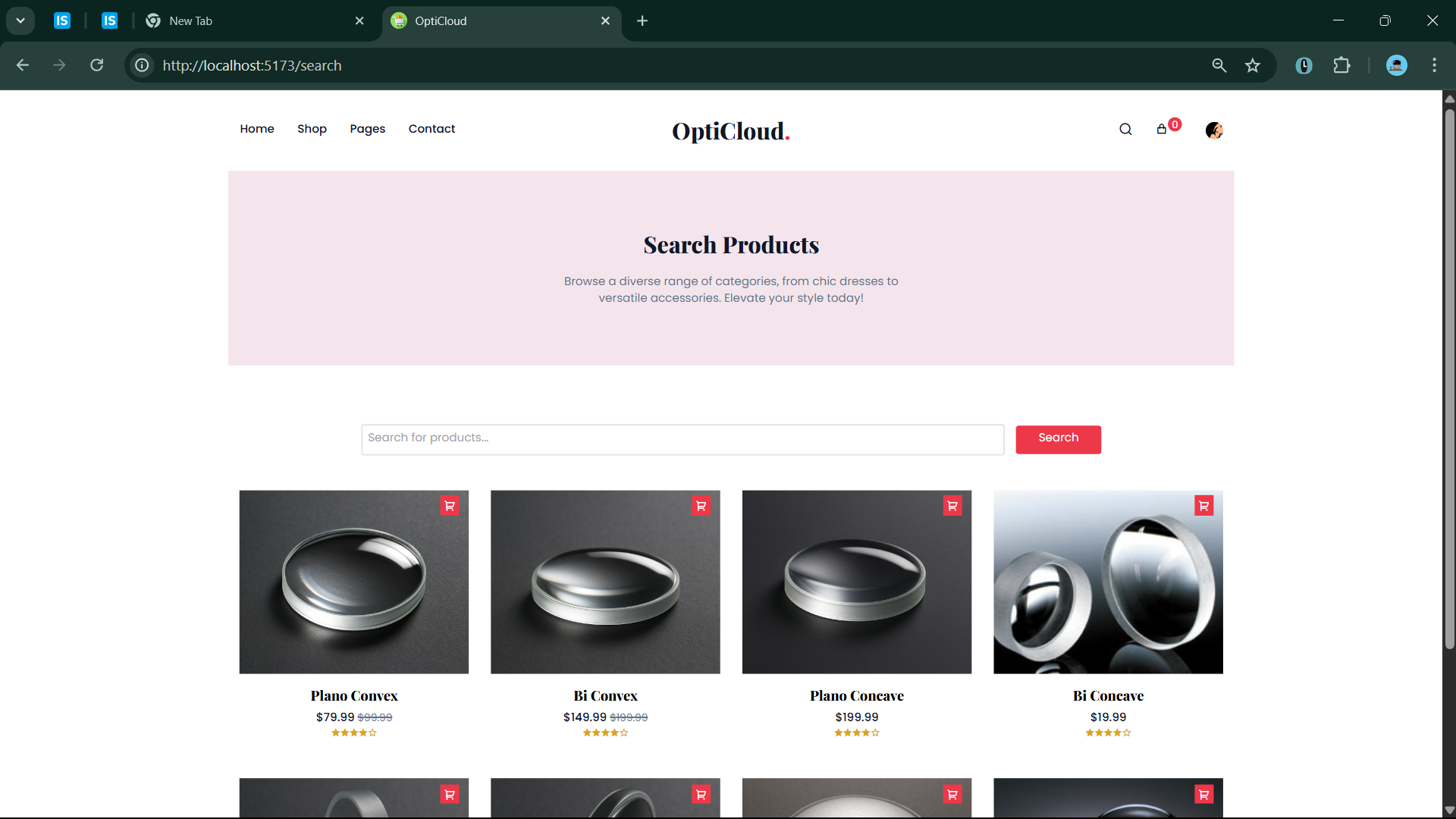
## Home.jsx



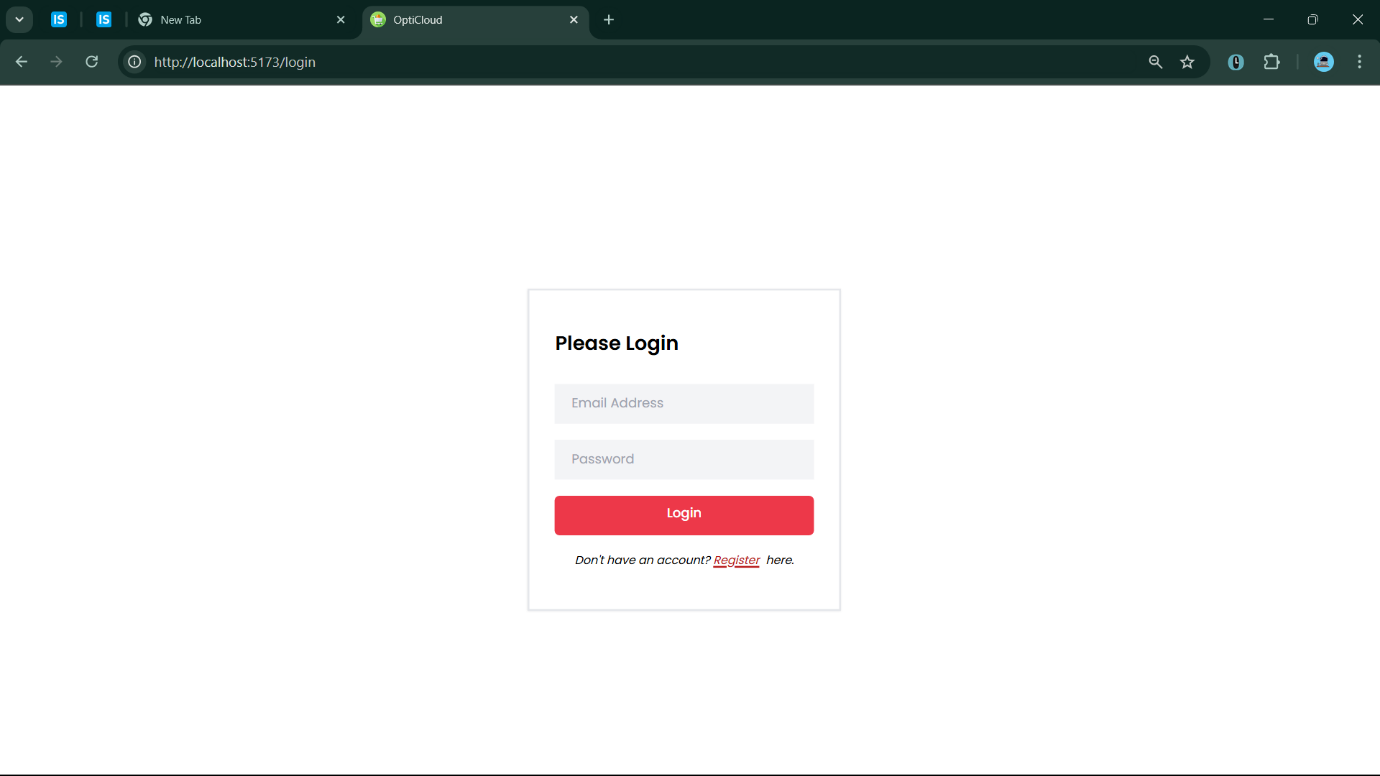
## ShopPage.jsx



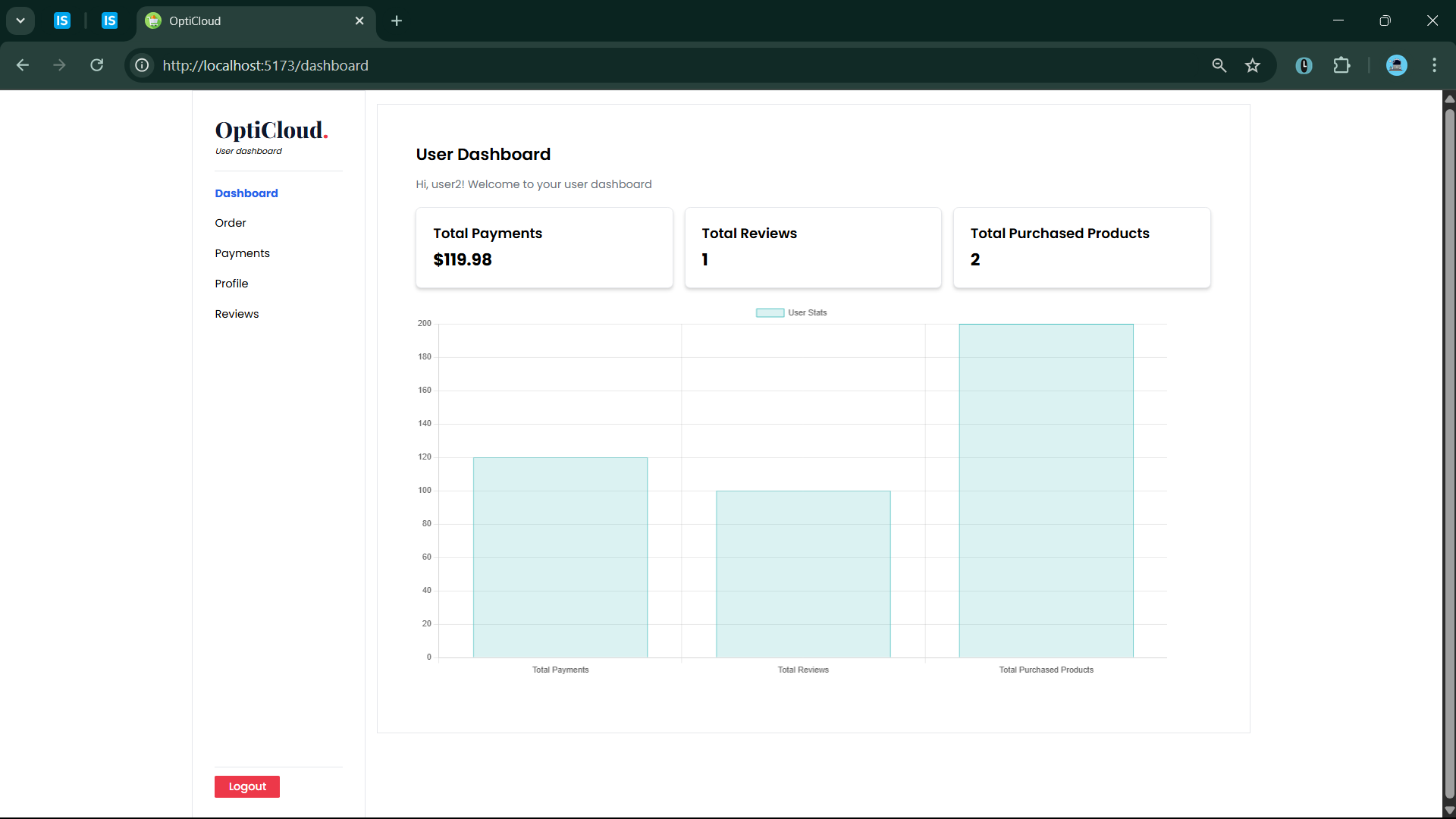
## search/Search.jsx



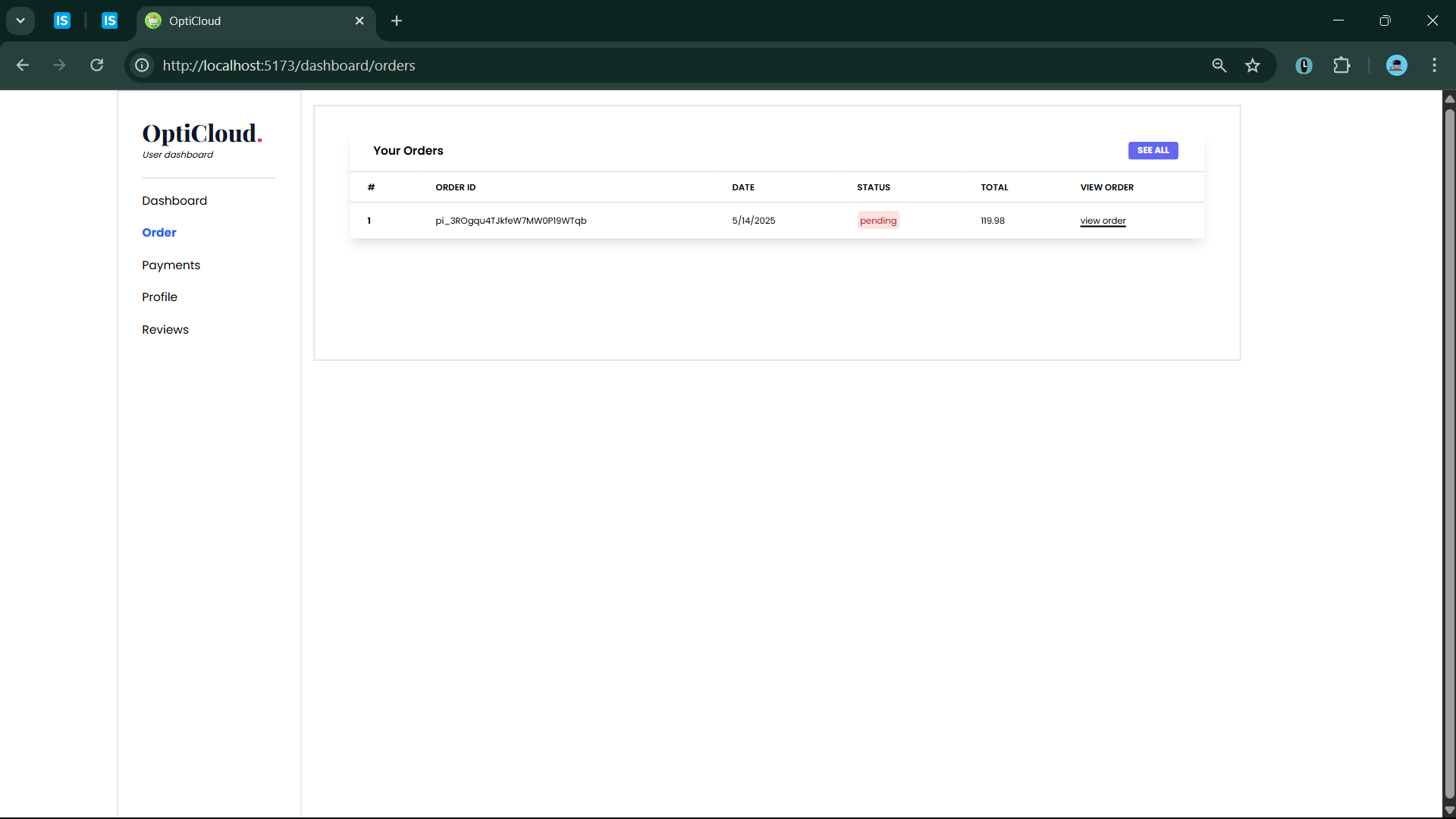
## components/Login.jsx



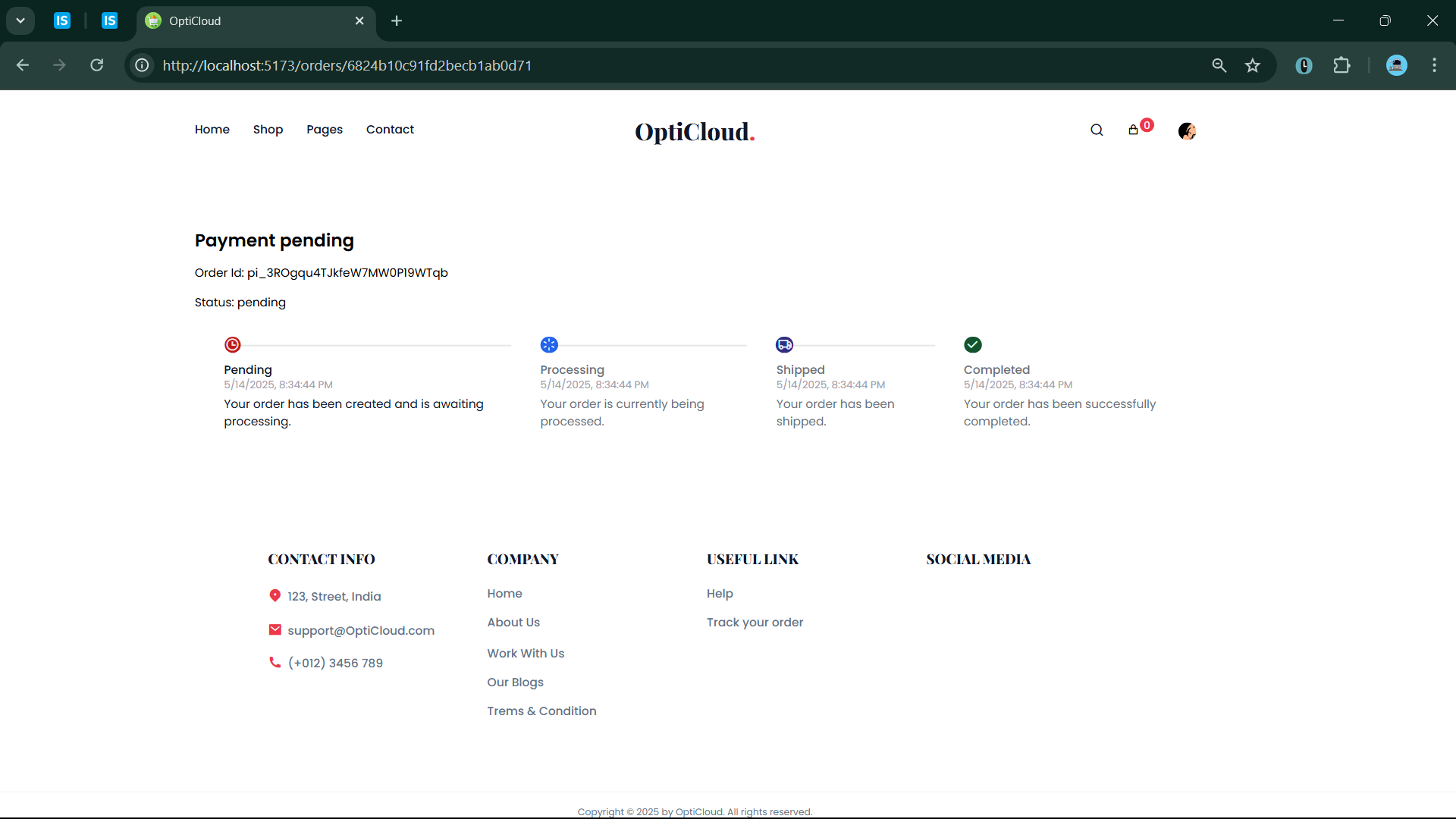
## user/dashboard/UserDMain.jsx



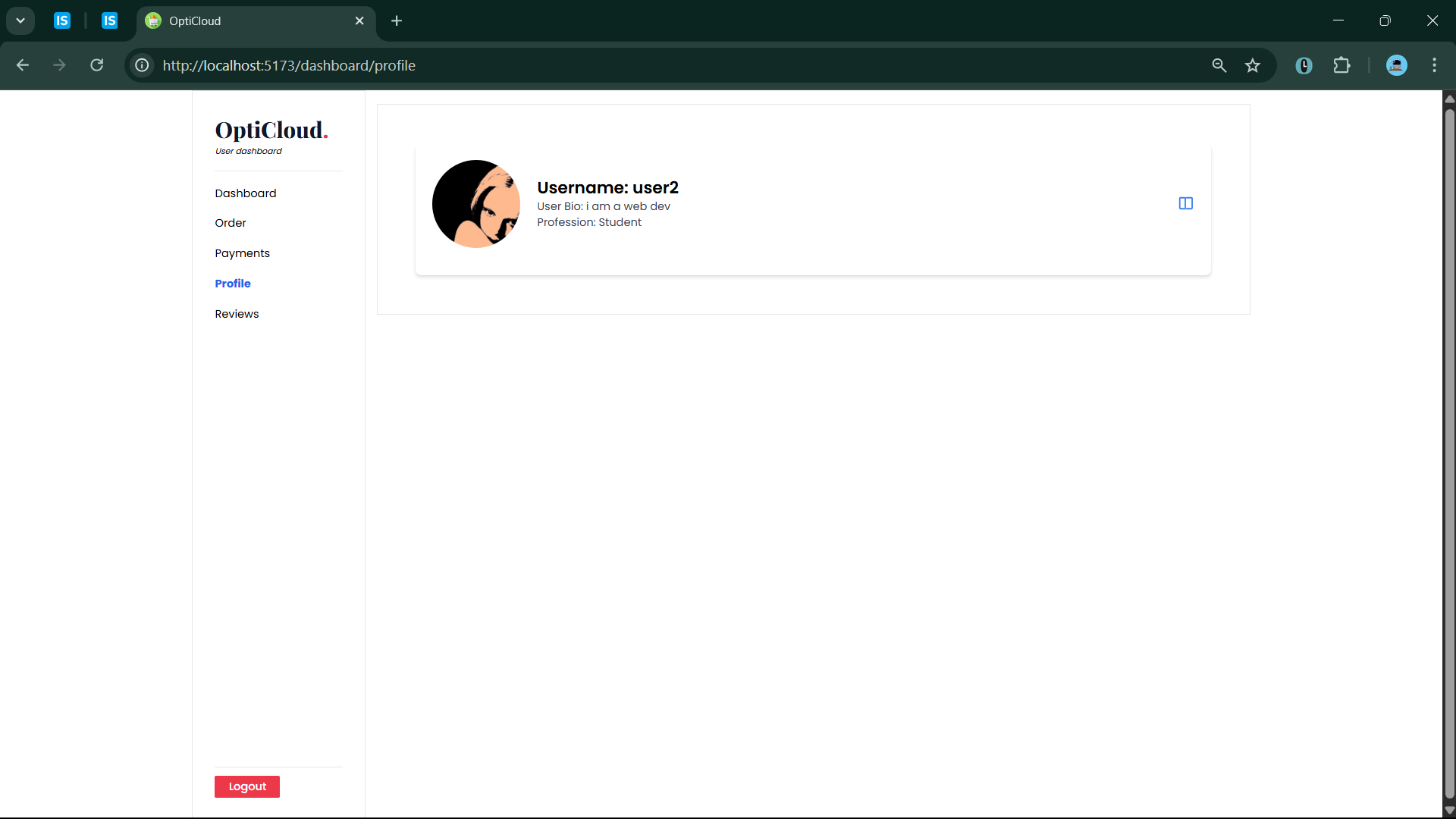
## user/UserOrders.jsx



## user/OrderDetails.jsx

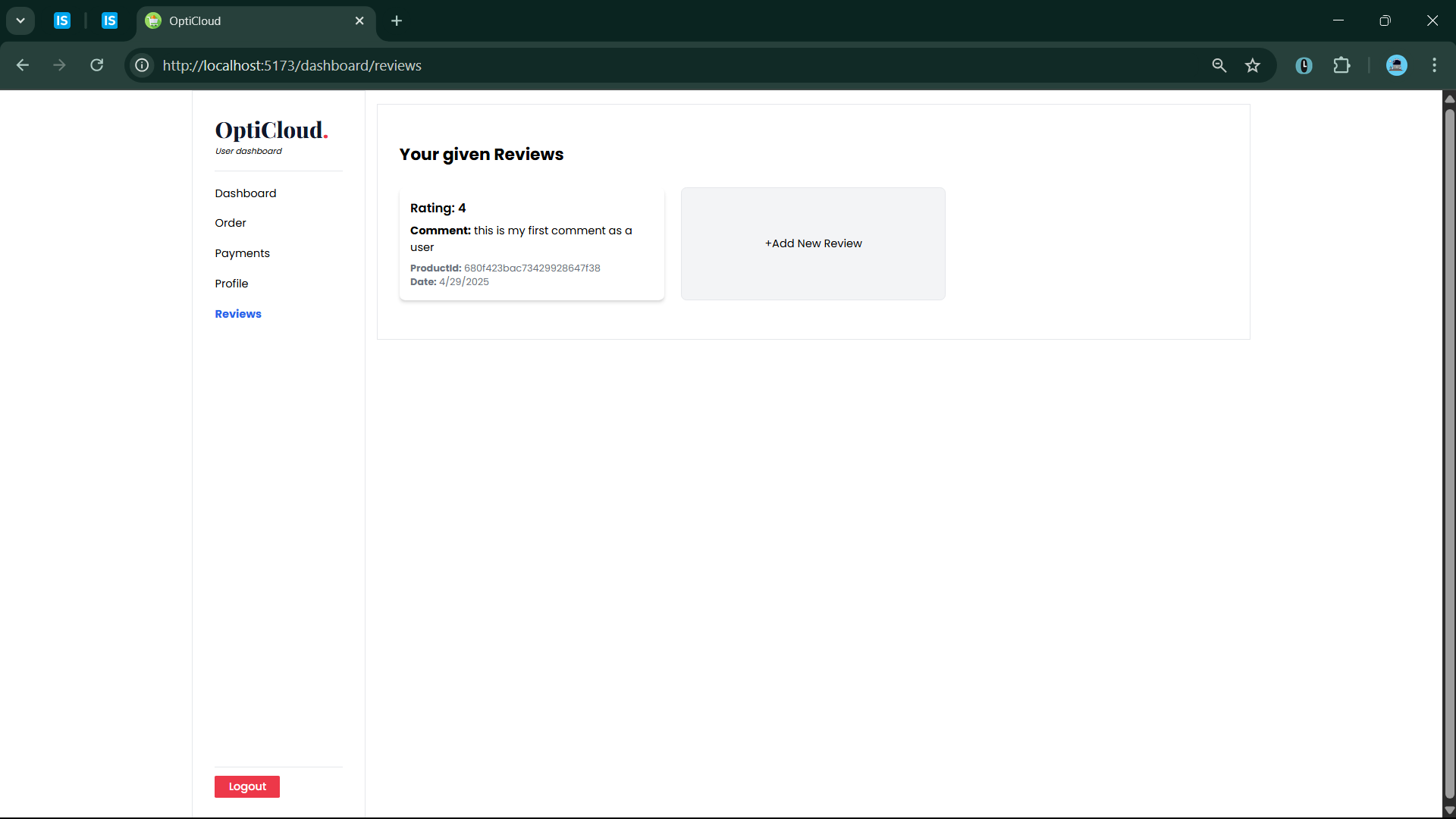


## user/UserProfile.jsx

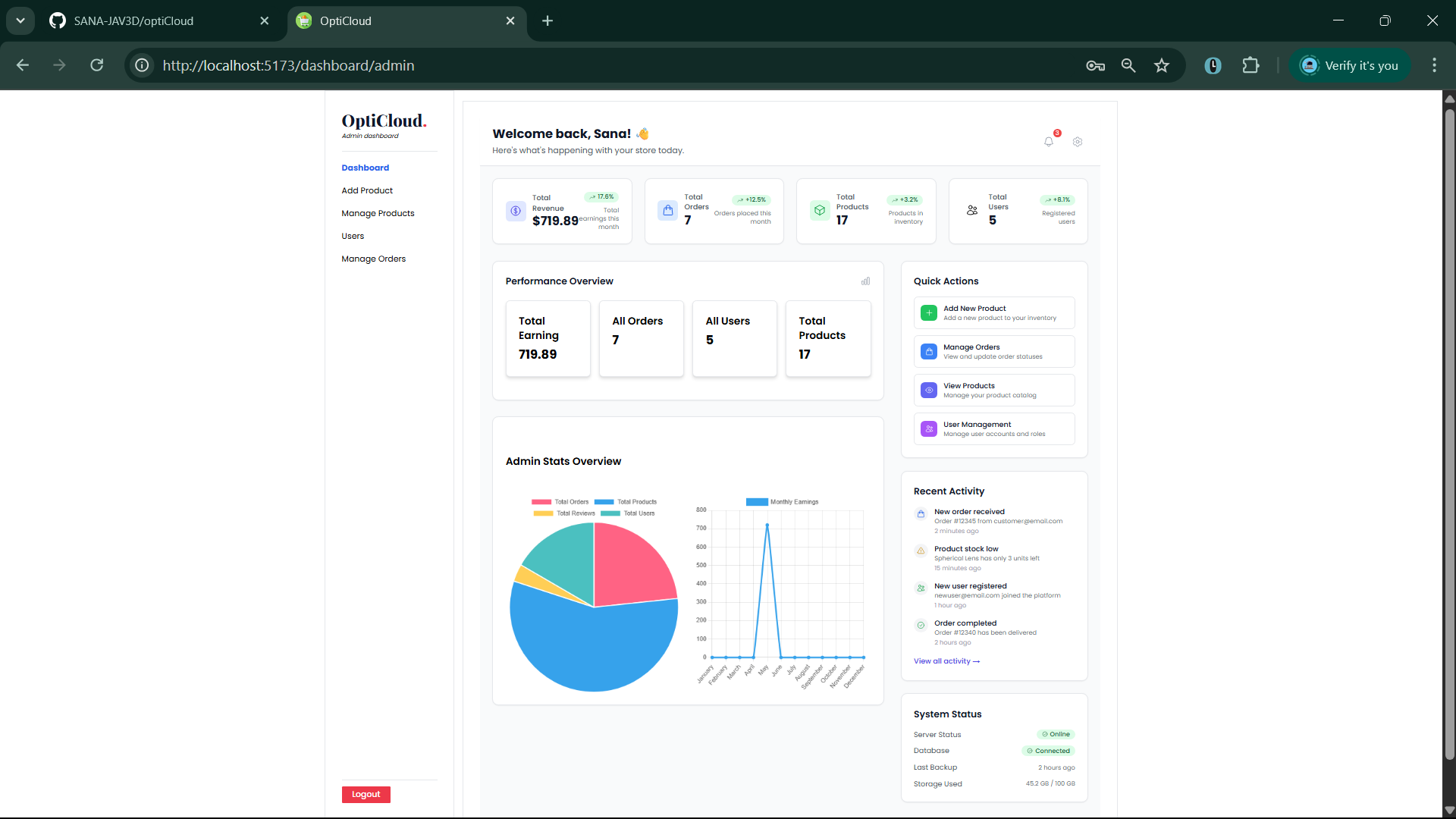


## 

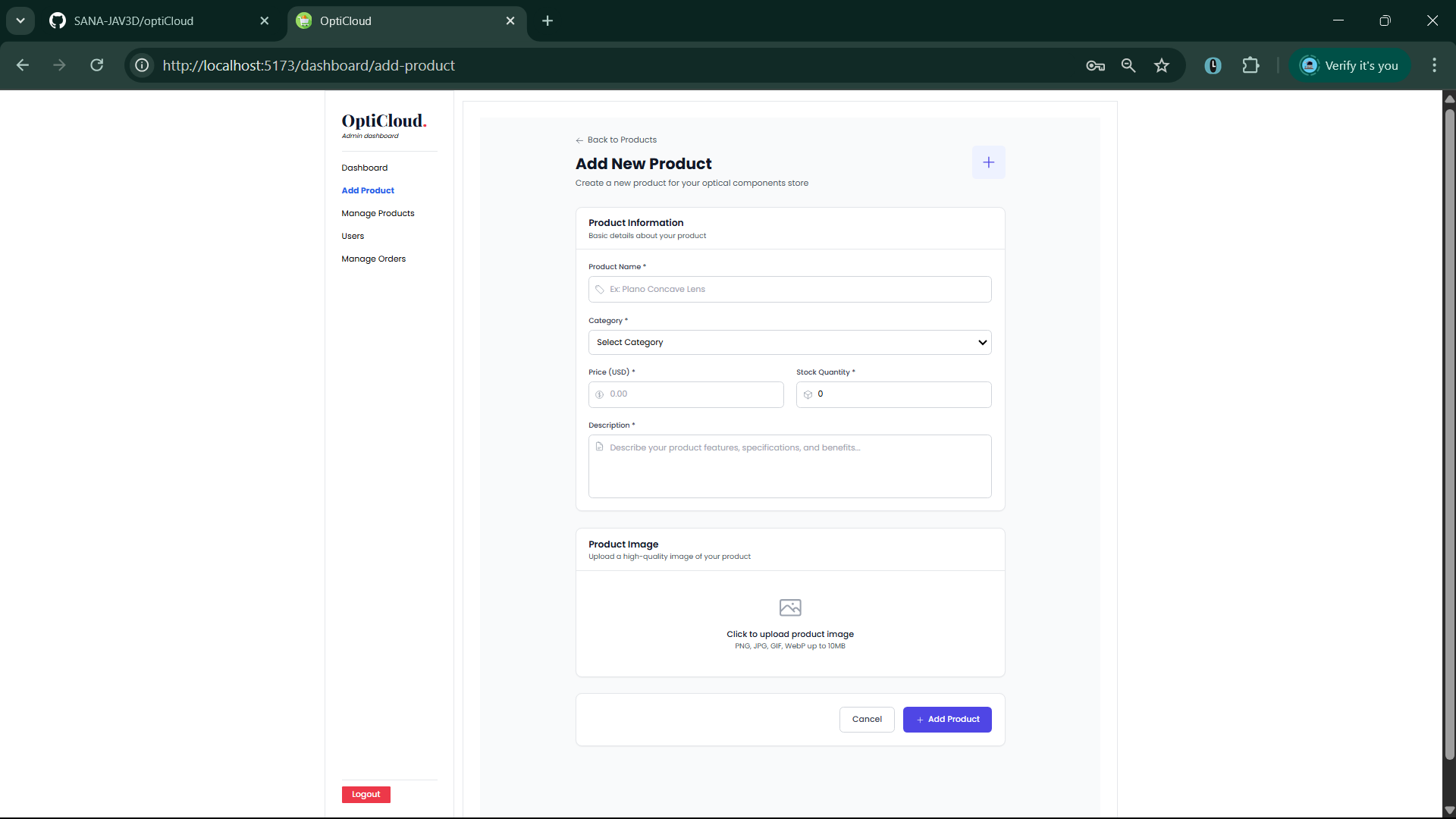
## user/UserReviews.jsx



## dashboard/AdminDashboard.jsx

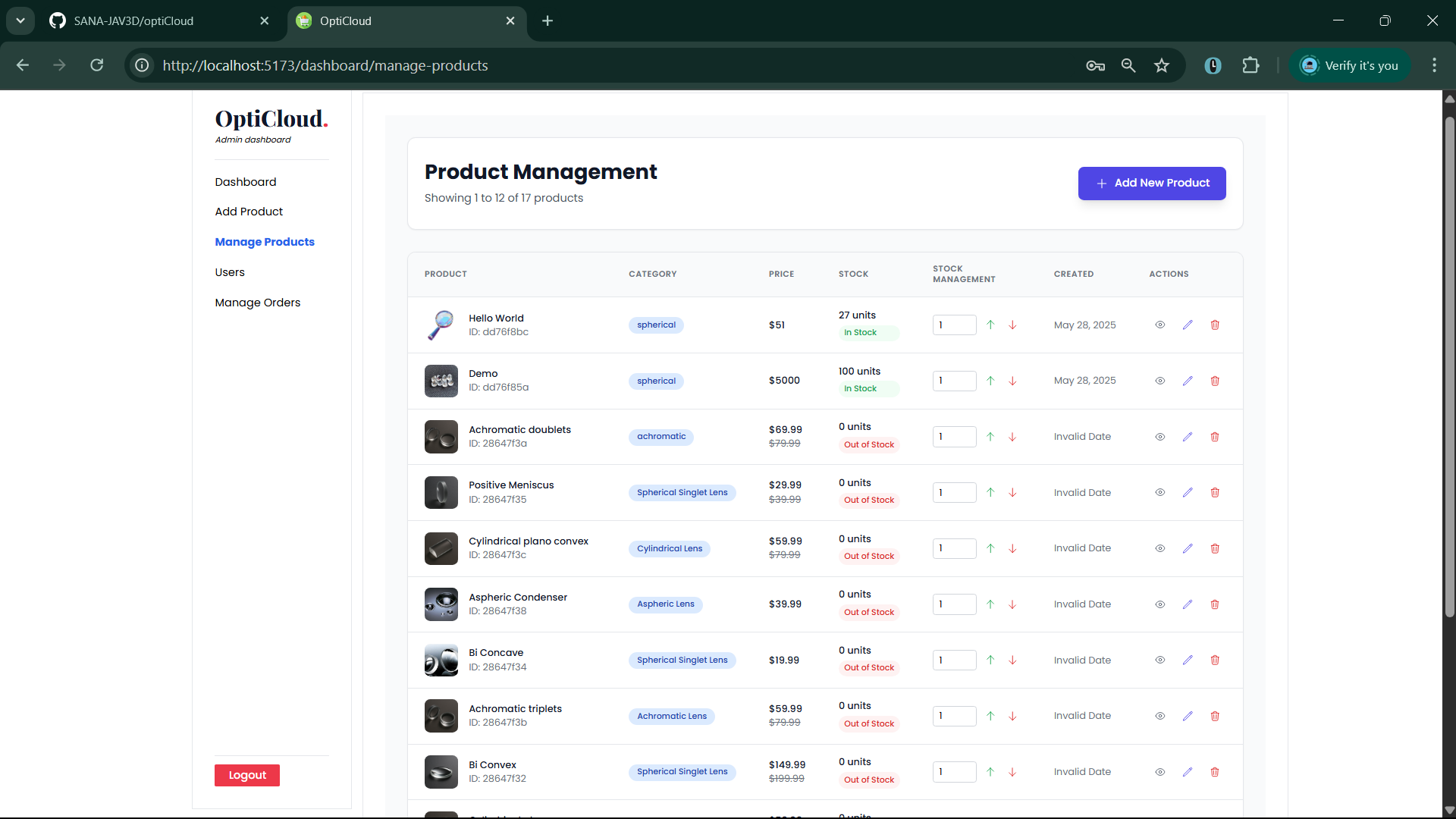


## admin/AddProduct.jsx

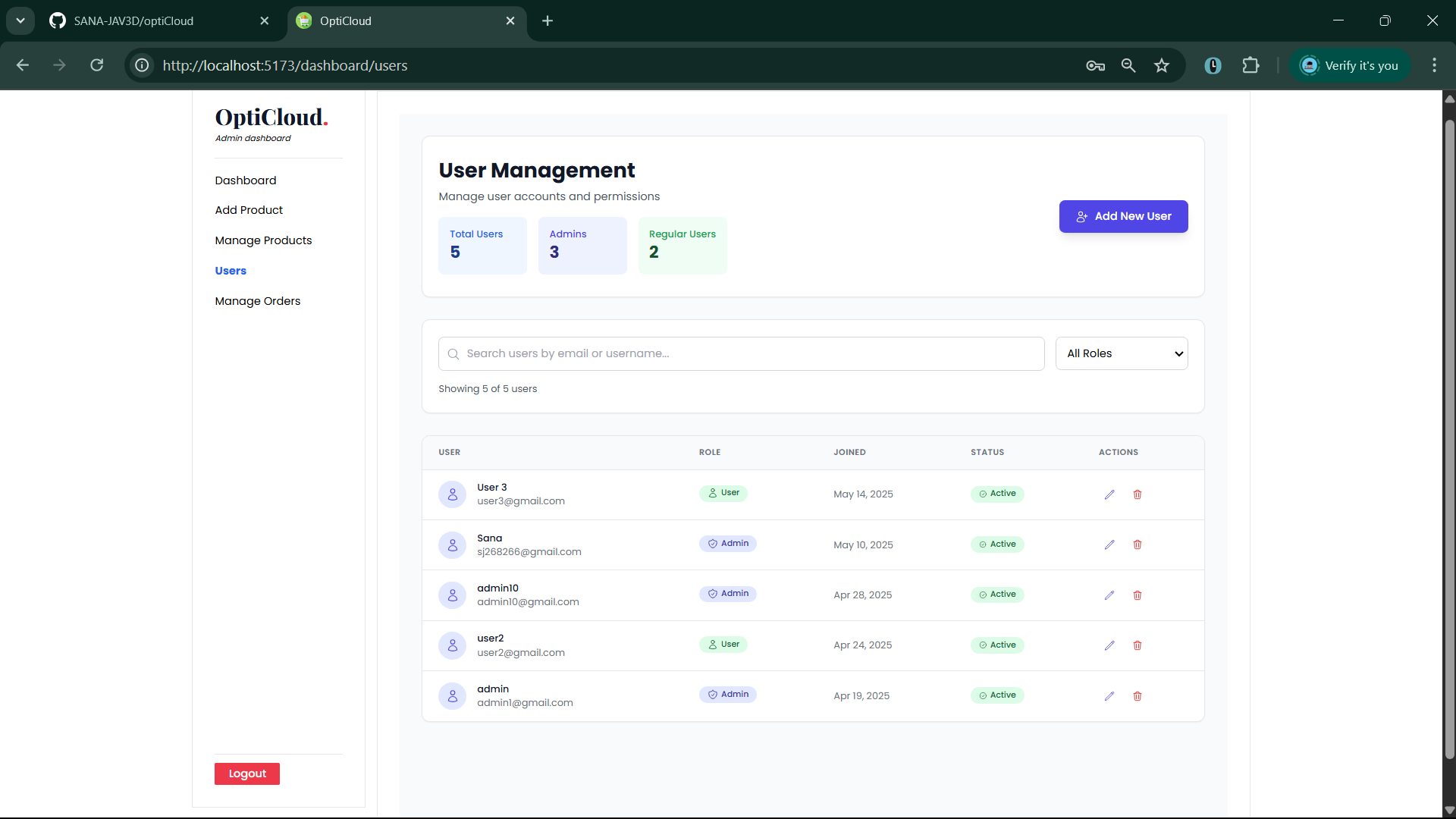


## 

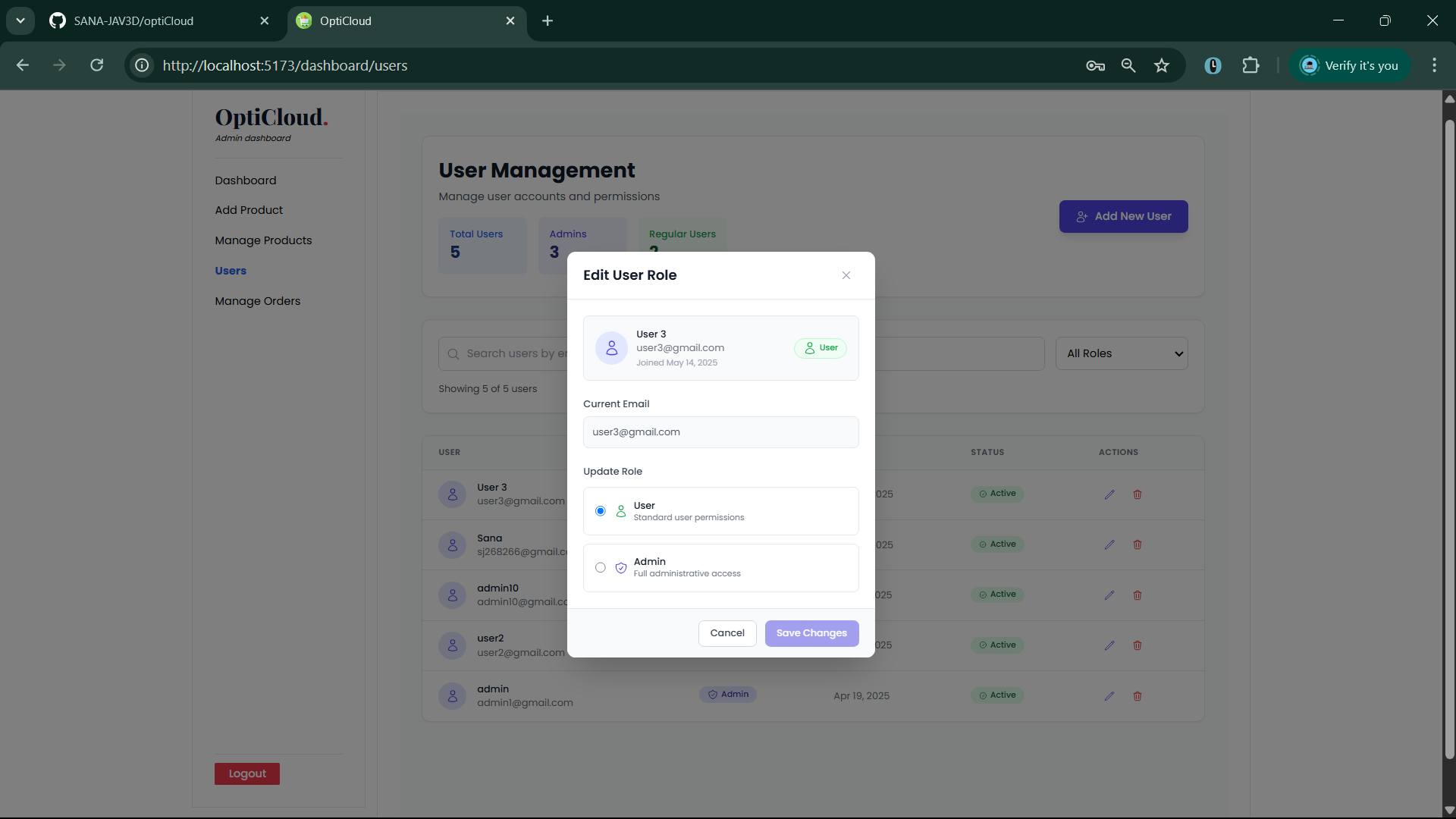
## admin/ManageProduct.jsx



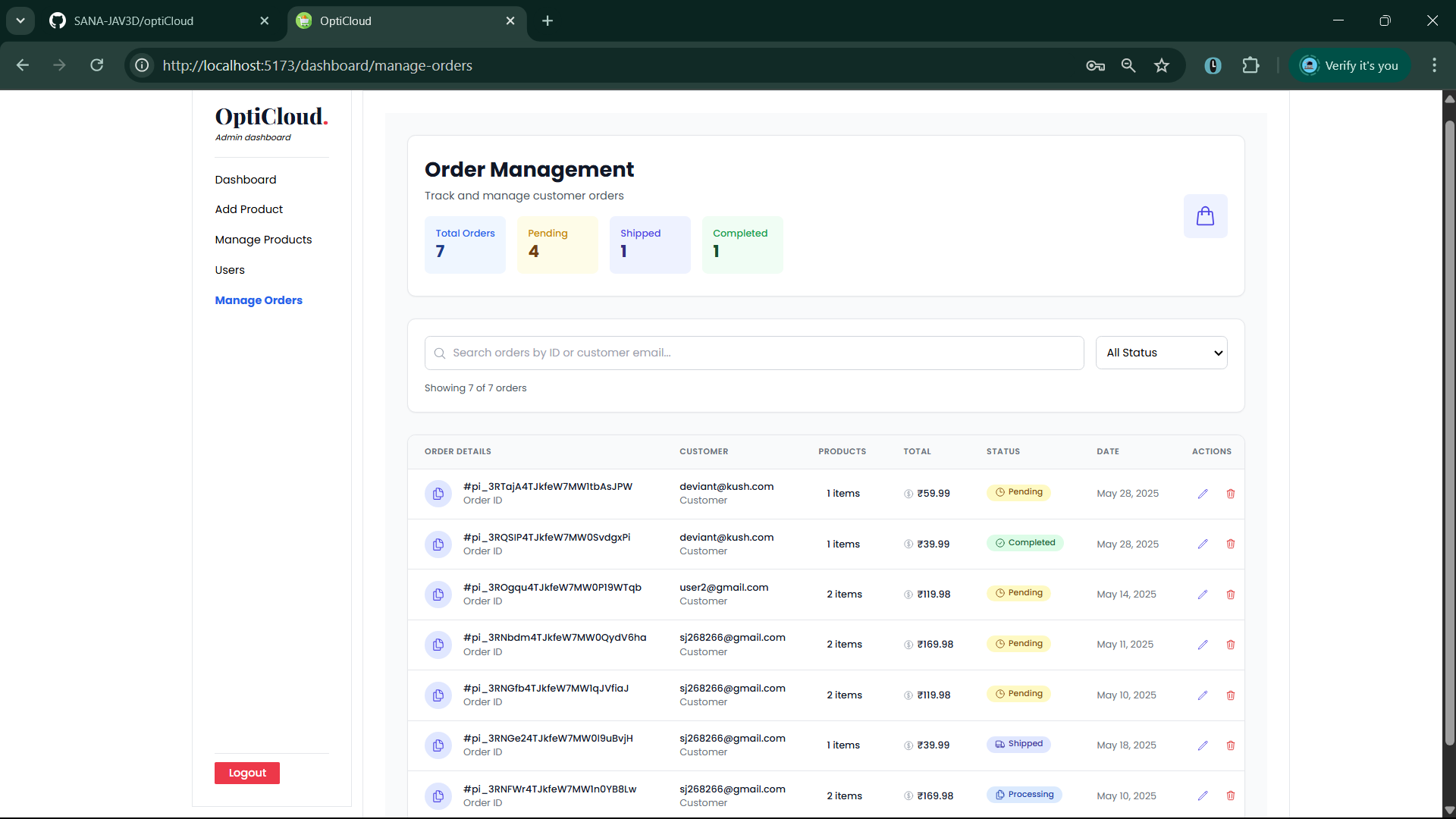
## admin/users/ManageUser.jsx



## admin/users/UpdateUserModal.jsx



## admin/manageOrders /ManageOrders.jsx



## admin/manageOrders /UpdateOrderModal.jsx

