 Online Glass Purchase

**INTERNSHIP PROJECT** **REPORT**

***Submitted by***

**SARAVANA HARI A**

**(Register No.: 95192201087)**

***Third year student of***

**BACHELOR OF ENGINEERING**

**IN**

# COMPUTER SCIENCE AND ENGINEERING

**P.S.R. ENGINEERING COLLEGE, SIVAKASI – 626 140**

(An Autonomous Institution, Affiliated to Anna University, Chennai)

# MARCH 2025

# 

**BONAFIDE CERTIFICATE**

Certified that this project report **"**Online Glass Purchase**"** is the bonafide work of **SARAVANA HARI A (95192201087), "** who carried out the project work under my supervision.

|  |  |
| --- | --- |
| **SIGNATURE**  **Mrs.Arthi Venkatesh**  **EXTERNAL SUPERVISOR**  **Corporate Trainer,**  Evoriea Infotech Private Limited  Bangalore – 560076. | SIGNATURE **Mr. Mohamed Nawfal A**  **TEAM LEADER**  **Corporate Trainer – Head,**  Evoriea Infotech Private Limited  Bangalore – 560076. |

# 

# Submitted to the department for the internship report evaluation on \_\_\_\_\_\_\_\_\_\_.

# PROJECT COORDINATOR HOD/CSE

**ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to the following individuals and institutions who have contributed to the successful completion of the Weather App project.

**Evoriea Infotech Private Limited, Bengaluru:**

**Mrs.Arthi Venkatesh, Corporate Trainer,** for providing valuable guidance, mentorship, and constructive feedback throughout the project development process. Your expertise has been instrumental in shaping the project.

**P.S.R Engineering College, Sivakasi:**

We unreservedly express our indebtedness to our Managing Trustee and Correspondent **Thiru. R. SOLAISAMY** and Director **Er. S. VIGNESWARI ARUNKUMAR** for providing the needed resources and facilities.

It is our privilege to convey my sincere thanks to our respected Principal **Dr. J.S. SENTHIL KUMAAR M.E., Ph.D.,** for giving us permission to do this project in our organization.

We wish to extend our sincere thanks to our adored Head of the Department **Dr. A. RAMATHILAGAM M.E., Ph.D.,** Professorfor her motivation during this period of this internship project work.

Their contributions have significantly enhanced the overall quality and success of the "weather App" project.

**ABSTRACT**

The **Online Glass Purchase** application is a web-based platform developed using **Spring Boot**, providing a seamless and engaging e-commerce experience for customers looking to purchase eyeglasses and sunglasses online. The system offers a comprehensive catalog of various eyewear categories, including **prescription glasses, sunglasses, blue-light filtering glasses, and designer frames**, with filtering and sorting options based on **frame style, lens type, brand, and price**.The application integrates essential features such as **user authentication, order management, a shopping cart, and secure payment gateways**. It ensures a **responsive and intuitive user interface** using modern web technologies, offering smooth navigation and a user-friendly shopping experience. Admin functionalities include **inventory management, order tracking, and customer management**, enabling efficient operations and business growth.Spring Boot serves as the **core backend framework**, providing a solid foundation for **scalability, performance, and security**. The application is designed to be **easily maintainable and extendable**, with **RESTful APIs** for future integrations, such as **virtual try-on features, AI-powered recommendations, third-party payment systems, and customer support services**.In conclusion, the **Online Glass Purchase** system leverages **Spring Boot’s** capabilities to deliver a **robust, scalable, and user-focused platform** for selling eyeglasses and sunglasses online, ensuring a **delightful shopping experience for customers and efficient operations for the business**.

**TABLE OF CONTENTS**

**CHAPTER** **No TITLE**  **PAGE NO**

ABSTRACT 4

1 INTRODUCTION 7

1.1 OVERVIEW OF ONLINE GLASS PURCHASE

1.2 PURPOSE OF REPORT

1.3 TECHNOLOGIES USED

2SYSTEM ARCHITECTURE 8

2.1 OVERVIEW OF THE SYSTEM

2.2 SPRING BOOT ARCHITECTURE

2.3 DATABASE ARCHITECTURE

3 **BACKEND DEVELOPMENT WITH SPRING BOOT 11**

3.1 SETTING UP SPRING BOOT APPLICATION

3.2 REST API DEVELOPMENT

3.3 INTEGRATING MYSQL WITH SPRING DATA JPA

4 **FRONTEND INTEGRATION 13**

4.1 USING THYMELESF OR REACT ANGULAR

4.2 API COMMUNICATION BETWEEN BACKEND AND FRONT END

5FEATURES 15

5.1 USER FEATURES

5.2 ADMIN FEATURES

6 IMPLEMENTATION &CHALLENGES 17

**7** CONCLUSION 49

8 REFERENCE 50

|  |  |  |
| --- | --- | --- |
|  |  |  |

**CHAPTER 1**

**INTRODUCTION**

## ****Introduction to Java****

Java is a high-level, object-oriented programming language developed by **Sun Microsystems** (now owned by Oracle) in 1995. It is widely used for building platform-independent applications due to its **"Write Once, Run Anywhere" (WORA)** capability, enabled by the Java Virtual Machine (**JVM**).

**1.2 Purpose Of The Report**

The purpose of this report is to document the development and implementation of the Online Glass Purchase, an e-commerce platform built using Spring Boot. It provides a structured overview of the project's objectives, architecture, features, and technologies used. This report serves as a reference for developers, stakeholders, and businesses interested in understanding the system’s functionality, challenges, and future scope.

**1.3 Technologies Used**

* **Backend**: Spring Boot, Spring Security
* **Frontend**: HTML, CSS, JavaScript
* **Database**: MySQL
* **Build Tool**: Maven
* **Deployment**:AWS

**CHAPTER 2**

**SYSTEM ARCHITECTURE**

## 2.1 ****Overview of the System****

The **Online Glass Purchase** application is a web-based **e-commerce platform** designed to streamline the online purchase of **eyeglasses and sunglasses**. Customers can **browse, search, and filter** products based on categories, frame styles, lens types, and brands, add items to their **shopping cart**, and proceed with **secure checkout**. The system includes both **customer-facing and admin functionalities** to ensure smooth operations, **order management, and inventory tracking**.

The platform leverages modern technologies, including **Spring Boot for backend development, React/Angular for the frontend, and MySQL for database management**. Additionally, security measures such as **authentication, authorization, and secure payment gateways** are integrated to enhance **user trust and provide a seamless shopping experience**.

## ****2.2 Spring Boot Architecture****

### ****Client Applications (Frontend Layer)****

* **Web App (React.js/Angular)** – Allows users to browse glasses, filter by frame type and lens category, add items to the cart, and place orders.
* **Mobile App (Flutter/React Native)** – Provides a seamless shopping experience on mobile devices.
* Communicates with the backend using **RESTful APIs**.

### ****API Gateway (Spring Cloud Gateway / Netflix Zuul)****

* Serves as a **single entry point** for all frontend requests.
* Routes API requests to respective **microservices**.
* Handles **authentication, rate limiting, and request filtering**.

### ****User Management Service****

* Manages **user registration, login, and authentication**.
* Uses **Spring Security with JWT and OAuth2** for secure authentication.
* Supports different **user roles** (Customer, Admin).

### ****Glass Catalog Service****

* Stores **eyeglass details, categories, pricing, and availability**.
* Uses **Spring Data JPA with MySQL/PostgreSQL** for database management.
* Supports **search and filtering** features.

### ****Shopping Cart Service****

* Manages **user carts**, allowing item additions and removals.
* Uses **Redis caching** for fast cart operations.
* Supports **session-based carts** for guest users.

### ****Order Management Service****

* Handles **order creation, tracking, and status updates**.
* Uses **Kafka/RabbitMQ** for **event-driven processing**.
* Implements **order validation before checkout**.

### ****Inventory Management Service****

* Tracks **eyeglass stock levels** and updates them automatically after purchases.
* Uses **Optimistic Locking** to prevent **overselling**.

### ****Notification Service****

* Sends **email and SMS notifications** for order confirmations, promotions, and updates.
* Uses **Spring Boot with Twilio, Firebase Cloud Messaging**.

### ****Review and Ratings Service****

* Manages customer **reviews and ratings** for glasses.
* Stores review data in **MongoDB** for **scalability**.

## ****2.3 Database Architecture****

The **Online Glass Purchase** system utilizes **different databases** for handling **structured and unstructured data efficiently**:

### ****Relational Database (MySQL/PostgreSQL)****

* Stores **users, orders, products, payments, and inventory details**.
* Uses **Spring Data JPA with Hibernate** for **Object-Relational Mapping (ORM)**.

### ****NoSQL Database (MongoDB/Elasticsearch)****

* Stores **customer reviews, logs, and search indexes**.
* Enhances **search functionality** for faster product discovery.

### ****Caching Layer (Redis)****

* Speeds up **cart operations** and frequently accessed data.
* Improves **performance and responsiveness** of the system.

**CHAPTER 3**

**BACKEND DEVELOPMENT WITH SPRING BOOT**

## ****Backend System Architecture****

The backend system is built using **Spring Boot Microservices Architecture**, ensuring **modularity, scalability, and maintainability** for an **Online Glass Purchase** platform.

### ****Key Components of Backend****

* **User Management Service** – Handles **authentication & authorization**.
* **Glass Catalog Service** – Manages **glasses, frame types, lens options, and categories**.
* **Shopping Cart Service** – Manages **user carts and checkout sessions**.
* **Order Management Service** – Handles **order creation, tracking, and status updates**.
* **Payment Service** – Integrates with **secure payment gateways (Stripe, PayPal, Razorpay)**.
* **Inventory Management Service** – Updates **stock levels dynamically**.
* **Notification Service** – Sends **email/SMS alerts for orders and promotions**.

## ****3.1 Setting Up Spring Boot Application****

Spring Boot simplifies backend development by providing an **easy-to-configure framework** for building scalable **e-commerce applications** like **Online Glass Purchase**. This section covers setting up a **Spring Boot project** with essential dependencies, configurations, and folder structures.

## ****3.2 REST API Development****

Developing a **REST API** is a crucial part of any **e-commerce platform** like **Online Glass Purchase**. With **Spring Boot**, we can efficiently build, secure, and deploy **RESTful services** to handle various operations such as **glass product management, user authentication, order processing, and payments**.

### ****This guide covers:****

**Setting Up a REST API** with Spring Boot  
 **Implementing CRUD Operations** for glass products  
 **User Authentication with JWT**  
 **Handling Orders and Payments**  
 **Securing APIs with Spring Security**

## ****3.3 Integrating MySQL with Spring Data JPA****

Integrating **MySQL with Spring Data JPA** is essential for an **e-commerce platform** like **Online Glass Purchase** to efficiently manage **products, users, and orders**. Spring Boot simplifies this integration, enabling seamless database interaction.

### ****This guide covers:****

**Setting Up MySQL in Spring Boot**  
 **Configuring Spring Data JPA**  
 **Creating Entities and Repositories**  
 **Performing CRUD Operations**

### ****Prerequisites:****

**MySQL Server** installed and running  
**Spring Boot Project** set up (via **Spring Initializr**)  
**Maven or Gradle** for dependency management  
**IDE** (IntelliJ, Eclipse, VS Code)  
**Postman** (for testing REST APIs)

**CHAPTER 4**

**4.FRONTEND INTEGRATION**

## ****4.1 Using Thymeleaf (Optional) or React/Angular for Frontend****

For the frontend of **Online Glass Purchase**, you have two main options:

### ****1. Thymeleaf (Server-Side Rendering - SSR)****

Integrated with **Spring Boot**  
Best for **simpler applications**  
Uses **HTML templates** with dynamic content  
Works well with **Spring Security**

### ****2. React/Angular (Client-Side Rendering - CSR)****

Separate **frontend & backend**  
More **interactive and responsive UI**  
Consumes **REST APIs** from Spring Boot  
Scalable for **large e-commerce applications**

### ****Which One Should You Choose?****

✔ **If you need a quick, simple UI → Use Thymeleaf.**  
✔ **If you want a modern, dynamic, scalable UI → Go with React or Angular.**

## ****Enable CORS in Spring Boot (For React/Angular Frontends)****

By default, browsers **block API calls from a different domain**. To allow the frontend (**React/Angular**) to communicate with the backend, **you must enable CORS** (**Cross-Origin Resource Sharing**) in Spring Boot.

## ****4.2 API Communication Between Backend and Frontend****

For **Online Glass Purchase**, the frontend (**React/Angular/Thymeleaf**) must efficiently communicate with the backend (**Spring Boot REST API**). This section explains:

**How the frontend requests data from the backend**  
 **Handling authentication (JWT) & error handling**  
 **CORS configuration for secure API access**

## ****API Communication Flow****

1️ The **frontend (React/Angular/Thymeleaf)** makes an HTTP request (**GET, POST, PUT, DELETE**).  
2️ The **Spring Boot backend** receives the request and processes it.  
3️ The **backend fetches data from MySQL** using **Spring Data JPA**.  
4️ The **backend sends a JSON response** back to the frontend.  
5️ The **frontend updates the UI** with the received data.

**CHAPTER 5**

# ****5.1 User Features****

## ****5.1.1 User Registration & Authentication****

🔹 **Sign Up**: Users can register with an **email and password**.  
🔹 **Login**: Secure authentication using **JWT (JSON Web Token)** or **Spring Security**.  
🔹 **Social Login (Optional)**: Login via **Google/Facebook**.  
🔹 **Forgot Password**: Reset password via **email verification**.

## ****5.1.2 User Dashboard****

🔹 **Profile Management**: Update **name, email, address, and password**.  
🔹 **Order History**: View past **glass orders**, track statuses, and re-order.  
🔹 **Wishlist**: Save favorite **glass products** for later.  
🔹 **Saved Addresses**: Manage **multiple shipping addresses**.

## ****5.1.3 Product Browsing & Search****

🔹 **Categories & Filters**: Browse **glasses** by **type (reading glasses, sunglasses, blue-light glasses, prescription glasses, etc.), frame material, brand, and price range**.  
🔹 **Search Bar**: Find **glass products** using keywords.  
🔹 **Sorting**: Sort by **popularity, price, latest arrivals, discounts**.  
🔹 **Product Details Page**: View **images, descriptions, specifications, lens options, and reviews**.

## ****5.1.4 Shopping Cart & Checkout****

🔹 **Add to Cart**: Add multiple **glass products** to the shopping cart.  
🔹 **Update Quantity**: Increase/decrease items in the cart.  
🔹 **Apply Coupons**: Use discount codes for special offers.  
🔹 **Secure Checkout**: Enter **shipping details** and proceed to **payment**.

## ****5.1.5 Payment Integration****

### ****Payment Methods:****

**Credit/Debit Cards** (**Visa, MasterCard, etc.**)  
 **UPI (Google Pay, PhonePe, Paytm)**  
 **Net Banking**  
 **Cash on Delivery (COD)**

### ****Payment Gateway:****

**Integrate with PayPal, Stripe, Razorpay, etc.**

## ****5.1.6 Order Management & Tracking****

🔹 **Order Confirmation**: Receive an **email/SMS** after placing an order.  
🔹 **Order Status Updates**: Track orders in **real-time** (**Pending → Shipped → Delivered**).  
🔹 **Estimated Delivery Date**: Display expected **delivery timelines**.  
🔹 **Cancel/Return Orders**: Request **order cancellations or returns**.

## ****5.1.7 Reviews & Ratings****

🔹 **Rate Products**: Users can give **1-5 star ratings**.  
🔹 **Write Reviews**: Share **experiences with uploaded images/videos**.  
🔹 **Verified Purchase Badge**: Only users who **bought the glass product** can review.

**5.1.8 Customer Support**

🔹 **Contact Support**: Reach support via **email, chat, or phone**.  
🔹 **FAQ Section**: Answers to common queries on **orders, payments, and returns**.  
🔹 **Live Chat (Optional)**: **AI chatbot or human agent support**.

## ****5.1.9 Notifications & Alerts****

🔹 **Email/SMS Notifications**: For **orders, promotions, and updates**.  
🔹 **Push Notifications (For Mobile Apps)**: **Flash sales, discounts, and order tracking updates**.

## ****5.1.10 Admin Dashboard (For Store Management)****

🔹 **Product Management**: Add, update, and delete **glass products**.  
🔹 **Order Management**: View and process **glass orders**.  
🔹 **User Management**: View registered users & manage accounts.  
🔹 **Reports & Analytics**: Sales performance, top-selling **glasses**, and customer trends.

**CHAPTER 6**

**6. IMPLEMENTATION &CHALLENGES**

# ****Implementation of Online E-Commerce Platform****

## ****Technology Stack****

* **Backend**: Spring Boot (**REST API, Spring Data JPA, Spring Security**)
* **Frontend**: React, Angular, or Thymeleaf
* **Database**: MySQL (**Spring Data JPA, Hibernate**)
* **Authentication**: JWT for secure login
* **Payment Integration**: Stripe, PayPal, Razorpay
* **Hosting**: AWS, Heroku, or DigitalOcean

## ****Step-by-Step Implementation****

### ****Step 1: Backend Development (Spring Boot)****

1️ **Set up Spring Boot Project** (**Spring Web, Spring Data JPA, MySQL, Lombok, Spring Security**)  
2️ **Create Database Schema** (**User, Glass Products, Cart, Order**)  
3️ **Develop REST APIs** for authentication, product catalog, cart, and orders  
4️ **Implement JWT Authentication** for secure login  
5️ **Handle Payment Integration** (**Stripe/PayPal**)  
6️ **Deploy Backend on Cloud** (**AWS, Heroku, etc.**)

### ****Step 2: Frontend Development (React/Angular/Thymeleaf)****

1️ **Set up React/Angular or Thymeleaf Project**  
2️ **Develop UI Components** (**Homepage, Product Page, Cart, Checkout**)  
3️ **Call REST APIs** using **Axios (React) or HttpClient (Angular)**  
4️ **Implement State Management** (**Redux for React, Services for Angular**)  
5️ **Integrate Payment Gateway in Checkout**  
6️ **Deploy Frontend on Cloud** (**Netlify, Vercel, Firebase, etc.**)

### ****Step 3: Database Integration (MySQL & JPA)****

1️ **Design Database Schema**  
2️ **Use Spring Data JPA & Hibernate** for data handling  
3️ **Optimize Queries** for performance  
4️ **Set up Indexing and Relationships** (**One-to-Many for Glass Products & Orders**)

### ****Step 4: Deployment & Monitoring****

1️ **Deploy Backend (Spring Boot) on AWS/Heroku**  
2️ **Deploy Frontend (React/Angular) on Vercel/Netlify**  
3️ **Set Up CI/CD Pipeline** (**GitHub Actions, Jenkins**)  
4️ **Monitor Logs & Performance** (**Prometheus, Grafana, ELK Stack**)

# ****Challenges & Solutions****

## ****Technical Challenges****

### ****1. API Communication & Performance Issues****

#### **Challenges:**

* Slow API responses, CORS errors, and inefficient data handling.

#### **Solutions:**

Enable CORS in **Spring Boot** for seamless frontend-backend communication.  
 Use **pagination and caching (Redis)** to optimize API responses.  
 Implement **GraphQL** for efficient data fetching.

### ****2. Secure Authentication & Authorization****

#### **Challenges:**

* Preventing unauthorized access, password leaks, and account takeovers.

#### **Solutions:**

Use **JWT-based authentication** with **Spring Security**.  
 Implement **OAuth2** for **Google/Facebook login**.  
 Enable **multi-factor authentication (MFA)** for enhanced security.

### ****3. Payment Gateway Integration****

#### **Challenges:**

* Ensuring smooth, secure, and error-free transactions.

#### **Solutions:**

Use trusted payment gateways like **PayPal, Stripe, Razorpay**.  
Implement **sandbox testing** before going live.  
Ensure compliance with **PCI DSS security standards**.

### ****4. Handling High Traffic & Scalability****

#### **Challenges:**

* The website may crash due to high user traffic during peak sales periods.

#### **Solutions:**

Deploy on **cloud platforms (AWS, GCP, Azure)** with **auto-scaling**.  
 Use **Load Balancing (NGINX, HAProxy)** to distribute traffic.  
 Optimize **database queries using indexing and caching**.

### ****5. Database Management & Performance****

#### **Challenges:**

* Slow queries, data inconsistencies, and poor performance.

#### **Solutions:**

Use **MySQL with indexing and normalization**.  
 Implement **Spring Data JPA** for optimized data handling.  
 Use **Redis caching** for frequently accessed data.

## ****Business Challenges & Solutions****

### ****1. Customer Retention & Abandoned Carts****

#### **Challenges:**

* Users add **glass products** to their cart but don’t complete the purchase.

#### **Solutions:**

Send **email/SMS reminders** for abandoned carts.  
Offer **discounts & limited-time deals** to encourage purchases.  
 Implement **AI-powered recommendations** based on user behavior.

### ****2. Competing with Large E-Commerce Platforms****

#### **Challenges**

* Competing with giants like **Amazon, Flipkart, and eyewear-specific online stores**.

#### **Solutions:**

Offer **unique glass designs, personalized customer service, and loyalty rewards**.  
 Provide **faster delivery options & hassle-free returns**.  
 Implement **AI-based personalized product recommendations**.

### ****3. Managing Product Returns & Refunds****

#### **Challenges:**

* High return rates can cause **financial losses**.

#### **Solutions:**

Implement a **clear return/refund policy**.  
 Use **AI to detect fraudulent returns**.  
 Offer **store credit instead of full refunds**.

**PROGRAM:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Shopping Cart</title>

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

</head>

<body>

<div class="container my-4">

<h2 class="text-center">Shopping Cart</h2>

<table class="table table-bordered">

<thead>

<tr>

<th>Image</th>

<th>Brand</th>

<th>Model</th>

<th>Price</th>

<th>Quantity</th>

<th>Total</th>

<th>Action</th>

</tr>

</thead>

<tbody id="cartItems"></tbody>

</table>

<h4>Total: $<span id="totalPrice">0.00</span></h4>

<a href="checkout.html" class="btn btn-primary">Proceed to Checkout</a>

</div>

<script>

**let** cart = JSON.parse(localStorage.getItem("cart")) || [];

**function** displayCart() {

**let** cartItems = document.getElementById("cartItems");

**let** totalPrice = 0;

cartItems.innerHTML = "";

cart.forEach((item, index) **=>** {

totalPrice += item.price \* item.quantity;

cartItems.innerHTML += `

<tr>

<td><img src="${item.imageUrl}" width="50"></td>

<td>${item.brand}</td>

<td>${item.model}</td>

<td>$${item.price}</td>

<td>${item.quantity}</td>

<td>$${(item.price \* item.quantity).toFixed(2)}</td>

<td><button class="btn btn-danger btn-sm" onclick="removeItem(${index})">Remove</button></td>

</tr>

`;

});

document.getElementById("totalPrice").innerText = totalPrice.toFixed(2);

}

**function** removeItem(index) {

cart.splice(index, 1);

localStorage.setItem("cart", JSON.stringify(cart));

displayCart();

}

window.onload = displayCart;

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Checkout</title>

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

</head>

<body>

<div class="container my-4">

<h2 class="text-center">Checkout</h2>

<form id="checkoutForm">

<div class="mb-3">

<label class="form-label">Full Name</label>

<input type="text" class="form-control" id="name" required>

</div>

<div class="mb-3">

<label class="form-label">Address</label>

<textarea class="form-control" id="address" required></textarea>

</div>

<div class="mb-3">

<label class="form-label">Payment Method</label>

<select class="form-control" id="paymentMethod">

<option value="Credit Card">Credit Card</option>

<option value="PayPal">PayPal</option>

</select>

</div>

<button type="submit" class="btn btn-success">Place Order</button>

</form>

</div>

<script>

document.getElementById("checkoutForm").addEventListener("submit", **function**(event) {

event.preventDefault();

localStorage.removeItem("cart");

alert("Order placed successfully!");

window.location.href = "index.html";

});

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Online Spectacles Shop</title>

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

<link rel="stylesheet" href="styles.css">

</head>

<body>

<!-- Navbar -->

<nav class="navbar navbar-expand-lg navbar-dark bg-primary">

<div class="container">

<a class="navbar-brand" href="#">Spectacles Shop</a>

<div class="navbar-nav ms-auto">

<a class="nav-link" href="index.html">Home</a>

<a class="nav-link" href="cart.html">🛒 Cart (<span id="cartCount">0</span>)</a>

<a class="nav-link" href="checkout.html">Checkout</a>

</div>

</div>

</nav>

<div class="container my-4">

<h2 class="text-center">Available Spectacles</h2>

<div class="row" id="productList"></div>

</div>

<script>

**let** cart = JSON.parse(localStorage.getItem("cart")) || [];

**function** fetchProducts() {

**let** products = [

{

id: 1,

brand: "Ray-Ban",

model: "Aviator",

price: 150,

imageUrl: "https://images.pexels.com/photos/701877/pexels-photo-701877.jpeg?auto=compress&cs=tinysrgb&w=600"

},

{

id: 2,

brand: "Oakley",

model: "Holbrook",

price: 120,

imageUrl: "https://images.pexels.com/photos/46710/pexels-photo-46710.jpeg?auto=compress&cs=tinysrgb&w=600"

},

{

id: 3,

brand: "Gucci",

model: "GG0061S",

price: 220,

imageUrl: "https://images.pexels.com/photos/735273/pexels-photo-735273.jpeg?auto=compress&cs=tinysrgb&w=600"

},

{

id: 4,

brand: "Prada",

model: "PR 17WS",

price: 180,

imageUrl: "https://images.pexels.com/photos/255305/pexels-photo-255305.jpeg?auto=compress&cs=tinysrgb&w=600"

},

{

id: 5,

brand: "Versace",

model: "VE 4391",

price: 200,

imageUrl: "https://images.pexels.com/photos/147641/pexels-photo-147641.jpeg?auto=compress&cs=tinysrgb&w=600"

},

{

id: 6,

brand: "Tom Ford",

model: "TF 237",

price: 250,

imageUrl: "https://images.pexels.com/photos/1669595/pexels-photo-1669595.jpeg?auto=compress&cs=tinysrgb&w=600"

}

];

**let** productList = document.getElementById("productList");

productList.innerHTML = ""; // Clear previous data

products.forEach(product **=>** {

**let** productCard = `

<div class="col-md-4">

<div class="card mb-4">

<img src="${product.imageUrl}" class="card-img-top" alt="Product Image" onerror="this.onerror=null; this.src='https://via.placeholder.com/150'">

<div class="card-body">

<h5 class="card-title">${product.brand} - ${product.model}</h5>

<p class="card-text"><strong>Price: $${product.price}</strong></p>

<button class="btn btn-success" onclick="addToCart(${product.id}, '${product.brand}', '${product.model}', ${product.price}, '${product.imageUrl}')">Add to Cart</button>

</div>

</div>

</div>

`;

productList.innerHTML += productCard;

});

}

**function** addToCart(id, brand, model, price, imageUrl) {

**let** item = cart.find(product **=>** product.id === id);

if (item) {

item.quantity++;

} else {

cart.push({ id, brand, model, price, imageUrl, quantity: 1 });

}

localStorage.setItem("cart", JSON.stringify(cart));

updateCartCount();

}

**function** updateCartCount() {

document.getElementById("cartCount").innerText = cart.reduce((sum, item) **=>** sum + item.quantity, 0);

}

window.onload = **function**() {

fetchProducts();

updateCartCount();

};

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login</title>

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h2 class="text-center my-4">Login</h2>

<form id="loginForm">

<div class="mb-3">

<label class="form-label">Email</label>

<input type="email" id="email" class="form-control" required>

</div>

<div class="mb-3">

<label class="form-label">Password</label>

<input type="password" id="password" class="form-control" required>

</div>

<button type="submit" class="btn btn-success">Login</button>

</form>

<p class="mt-3">Don't have an account? <a href="signup.html">Sign Up</a></p>

</div>

<script>

document.getElementById("loginForm").addEventListener("submit", **function**(event) {

event.preventDefault();

**let** user = {

email: document.getElementById("email").value,

password: document.getElementById("password").value

};

fetch("http://localhost:8080/api/auth/login", {

method: "POST",

headers: {

"Content-Type": "application/json"

},

body: JSON.stringify(user)

})

.then(response **=>** response.json())

.then(data **=>** alert(data.message))

.catch(error **=>** console.error("Error:", error));

});

</script>

</body>

</html>

**async** **function** fetchSpecs() {

try {

**let** response = await fetch("http://localhost:8080/api/specs");

if (!response.ok) throw new Error("Failed to fetch data");

**let** data = await response.json();

**let** container = document.getElementById("specsContainer");

data.forEach(spec **=>** {

**let** card = document.createElement("div");

card.className = "col-md-4 mb-4";

card.innerHTML = `

<div class="card">

<img src="spectacles.jpg" class="card-img-top" alt="Spectacle">

<div class="card-body text-center">

<h5 class="card-title">${spec.brand} - ${spec.model}</h5>

<p class="card-text">Price: ₹${spec.price}</p>

<button class="btn btn-primary" onclick="addToCart(${spec.id}, '${spec.brand}', '${spec.model}', ${spec.price})">Add to Cart</button>

</div>

</div>

`;

container.appendChild(card);

});

} catch (error) {

console.error("Error:", error);

}

}

// Function to add items to the cart (stored in localStorage)

**function** addToCart(id, brand, model, price) {

**let** cart = JSON.parse(localStorage.getItem("cart")) || [];

**let** item = { id, brand, model, price };

cart.push(item);

localStorage.setItem("cart", JSON.stringify(cart));

updateCartCount();

alert("Item added to cart!");

}

// Function to update cart item count in the navbar

**function** updateCartCount() {

**let** cart = JSON.parse(localStorage.getItem("cart")) || [];

document.getElementById("cartCount").innerText = cart.length;

}

// Initialize cart count on page load

window.onload = **function**() {

fetchSpecs();

updateCartCount();

};

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Sign Up</title>

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<h2 class="text-center my-4">Sign Up</h2>

<form id="signupForm">

<div class="mb-3">

<label class="form-label">Name</label>

<input type="text" id="name" class="form-control" required>

</div>

<div class="mb-3">

<label class="form-label">Email</label>

<input type="email" id="email" class="form-control" required>

</div>

<div class="mb-3">

<label class="form-label">Password</label>

<input type="password" id="password" class="form-control" required>

</div>

<button type="submit" class="btn btn-primary">Sign Up</button>

</form>

<p class="mt-3">Already have an account? <a href="login.html">Login</a></p>

</div>

<script>

document.getElementById("signupForm").addEventListener("submit", **function**(event) {

event.preventDefault();

**let** user = {

name: document.getElementById("name").value,

email: document.getElementById("email").value,

password: document.getElementById("password").value

};

fetch("http://localhost:8080/api/auth/signup", {

method: "POST",

headers: {

"Content-Type": "application/json"

},

body: JSON.stringify(user)

})

.then(response **=>** response.json())

.then(data **=>** alert(data.message))

.catch(error **=>** console.error("Error:", error));

});

</script>

</body>

</html>

body {

font-family: Arial, sans-serif;

text-align: center;

margin: 20px;

}

table {

width: 50%;

margin: 20px auto;

border-collapse: collapse;

}

th, td {

border: 1px solid black;

padding: 10px;

text-align: center;

}

th {

background-color: #f2f2f2;

}

.checkout-btn {

background-color: #28a745;

color: white;

padding: 10px 20px;

border: none;

cursor: pointer;

font-size: 16px;

margin-top: 20px;

}

.checkout-btn:hover {

background-color: #218838;

}

body {

font-family: 'Poppins', sans-serif;

background-color: #f8f9fa;

}

.navbar {

background-color: #007bff;

}

.navbar-brand, .nav-link {

color: white !important;

}

.container {

margin-top: 30px;

}

.card {

border: none;

box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);

}

.card img {

height: 200px;

object-fit: cover;

}

.checkout-btn {

background-color: #28a745;

color: white;

padding: 10px 20px;

border: none;

cursor: pointer;

font-size: 16px;

margin-top: 20px;

border-radius: 5px;

}

.checkout-btn:hover {

background-color: #218838;

}

body {

background-color: #f8f9fa;

}

h1 {

text-align: center;

margin-top: 20px;

}

.card {

border-radius: 10px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

}

.card img {

height: 200px;

object-fit: cover;

}

.btn-primary {

width: 100%;}

package com.example.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration

public class SecurityConfig {

@Bean

public BCryptPasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

package com.example.controller;

import com.example.model.Order;

import com.example.repository.OrderRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/orders")

@CrossOrigin(origins = "http://localhost:5500") // Adjust if needed

public class OrderController {

@Autowired

private OrderRepository orderRepository;

@PostMapping

public Order placeOrder(@RequestBody Order order) {

return orderRepository.save(order);

}

@GetMapping

public List<Order> getAllOrders() {

return orderRepository.findAll();

}}

package com.example.controller;

import com.example.model.Product;

import com.example.repository.ProductRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/products")

@CrossOrigin(origins = "http://localhost:5500") // Adjust if needed

public class ProductController {

@Autowired

private ProductRepository productRepository;

@GetMapping

public List<Product> getAllProducts() {

return productRepository.findAll();

}

@PostMapping

public Product addProduct(@RequestBody Product product) {

return productRepository.save(product);

}

}

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import com.example.model.Specs;

import com.example.service.SpecsService;

import java.util.List;

@RestController

@RequestMapping("/api/specs")

@CrossOrigin("\*")

public class SpecsController {

@Autowired

private SpecsService service;

@GetMapping

public List<Specs> getAll() {

return service.getAllSpecs();

}

@PostMapping

public Specs addSpecs(@RequestBody Specs specs) {

return service.addSpecs(specs);

}

}

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class OnlineSpecsApplication {

public static void main(String[] args) {

SpringApplication.run(OnlineSpecsApplication.class, args);

}

}

**package** com.example.model;

**import** jakarta.persistence.\*;

@Entity

@Table(name = "users")

**public** **class** User {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long id;

**private** String name;

**private** String email;

**private** String password;

**public** User() {}

**public** User(String name, String email, String password) {

**this**.name = name;

**this**.email = email;

**this**.password = password;

}

// Getters and Setters

}

package com.example.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.example.model.Order;

import com.example.repository.OrderRepository;

import java.util.List;

@Service

public class OrderService {

@Autowired

private OrderRepository orderRepository;

public Order placeOrder(Order order) {

return orderRepository.save(order);

}

public List<Order> getAllOrders() {

return orderRepository.findAll();

}

}

package com.example.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.example.model.Specs;

import com.example.repository.SpecsRepository;

import java.util.List;

@Service

public class SpecsService {

@Autowired

private SpecsRepository repository;

public List<Specs> getAllSpecs() {

return repository.findAll();

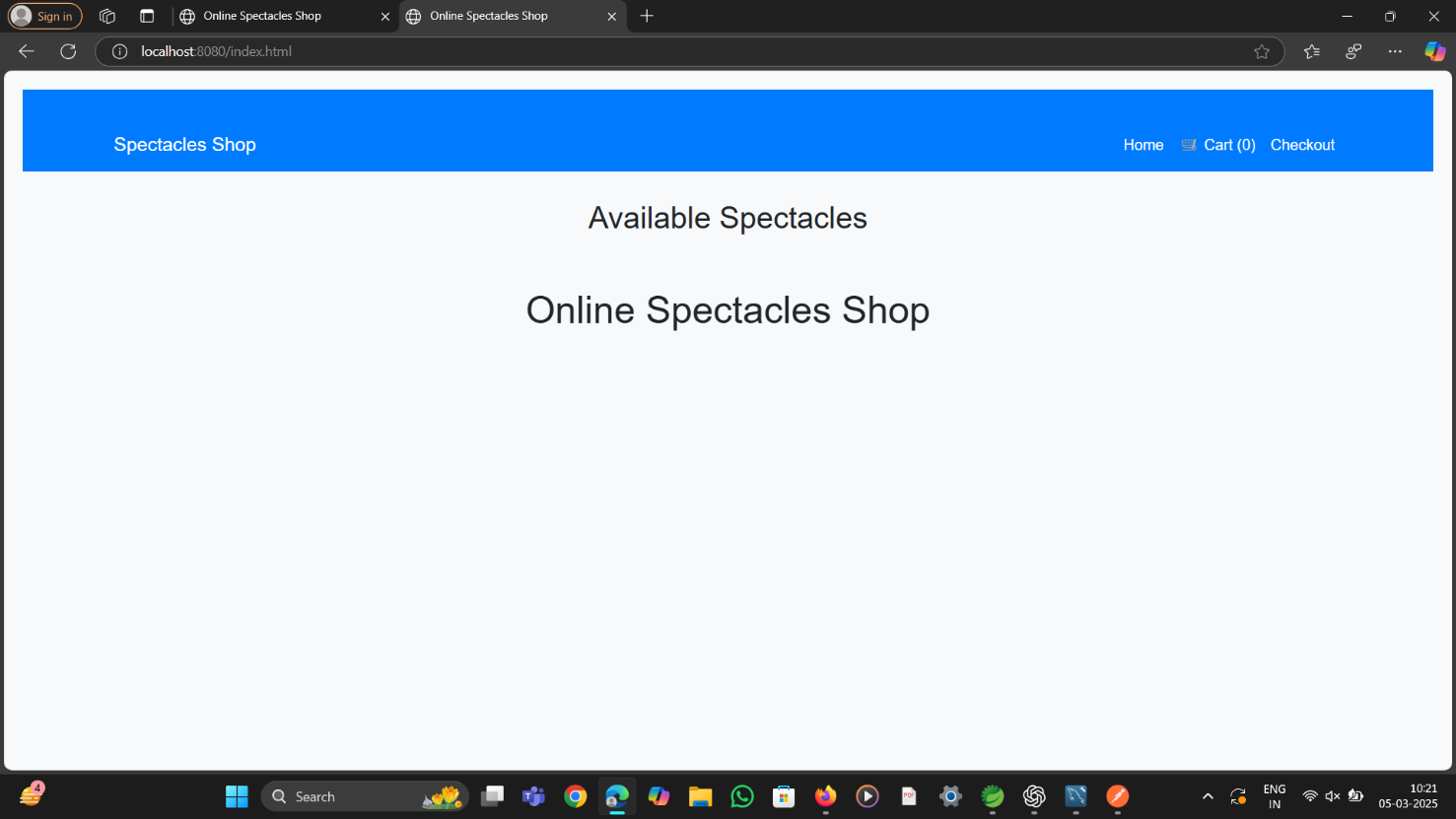
}

public Specs addSpecs(Specs specs) {

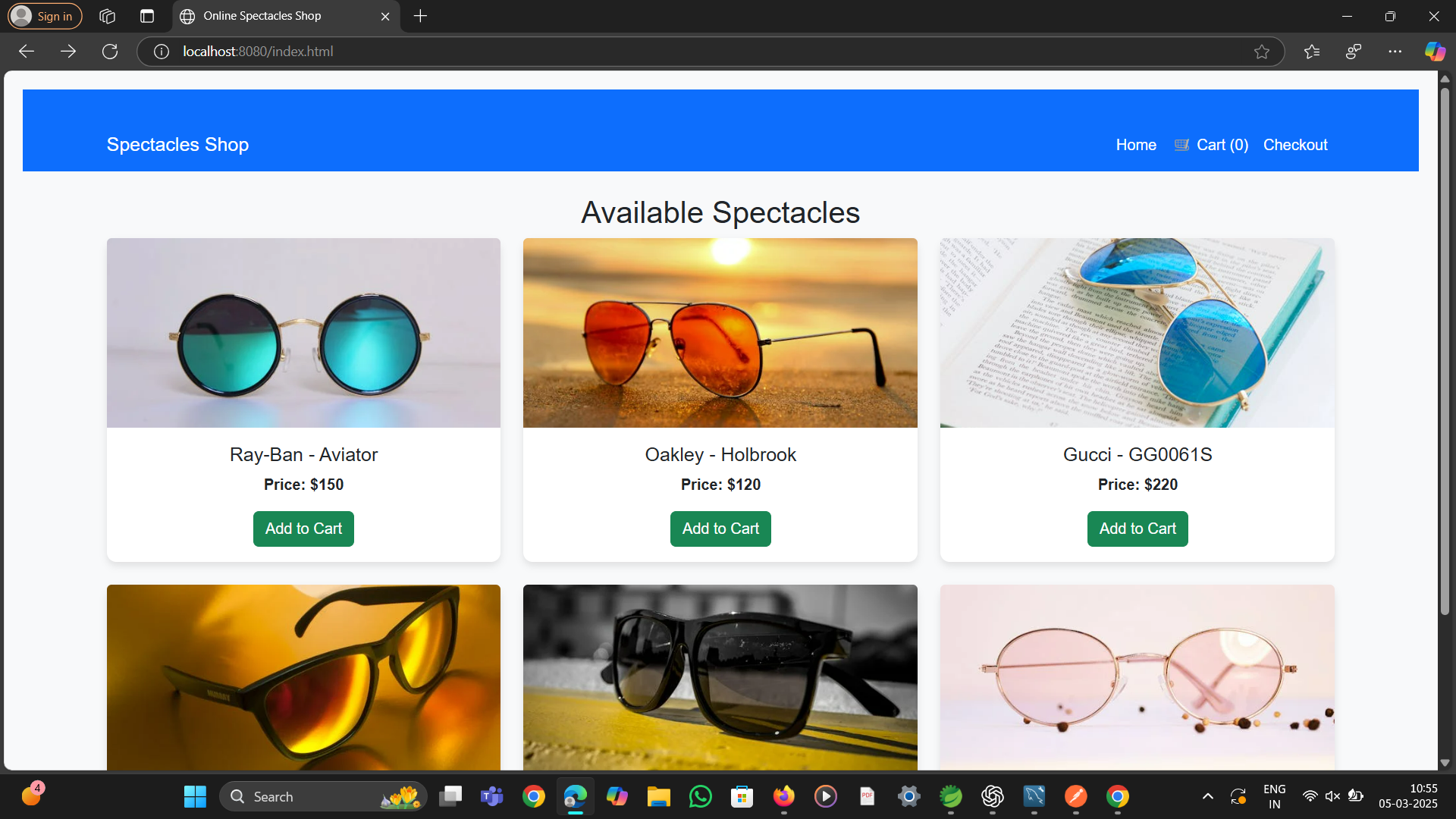
return repository.save(specs);

}

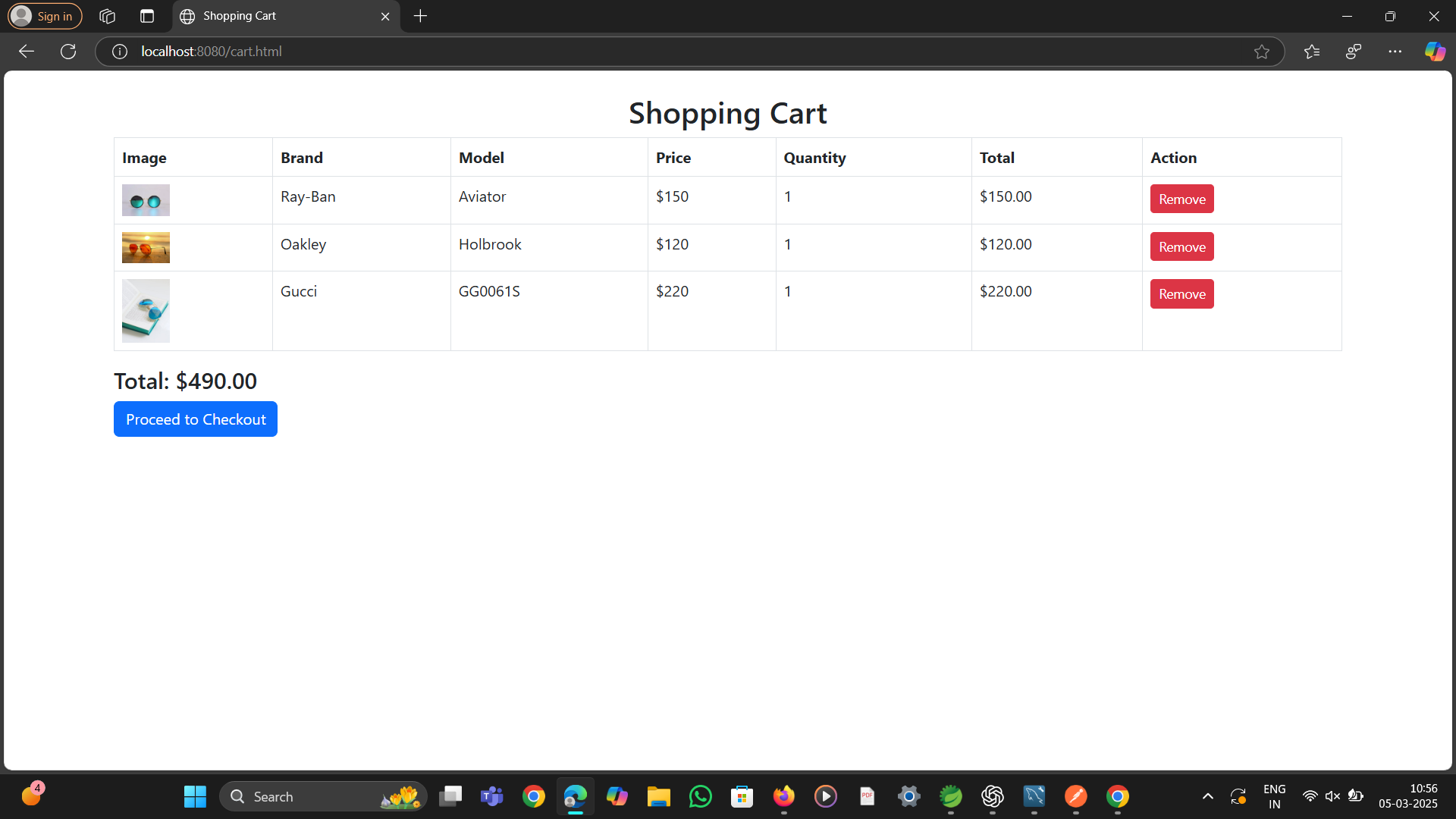
**HOME PAGE :**



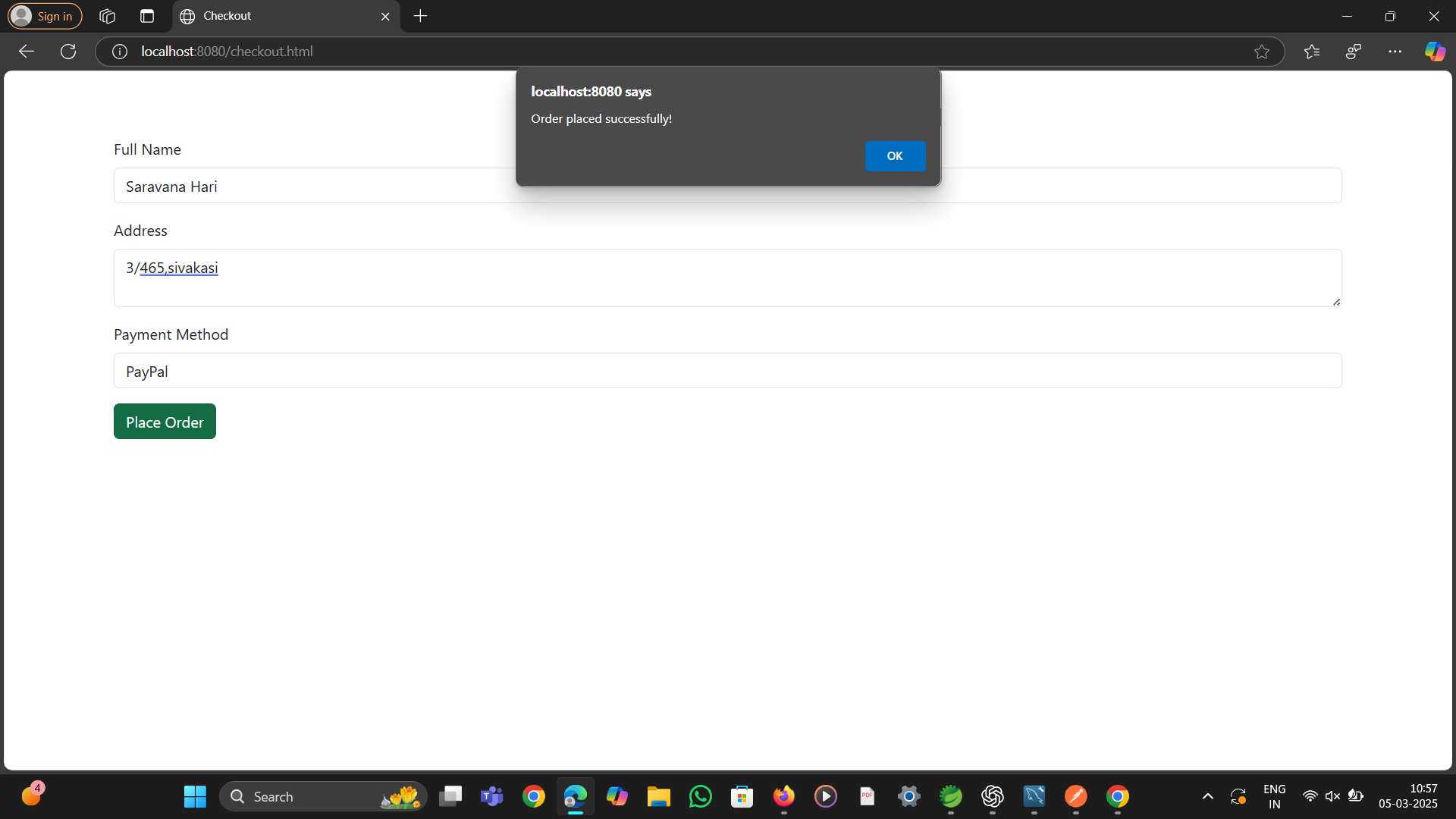
**STORE PAGE:**



**ADD TO CART PAGE**:



**CHECK OUT PAGE:**

****

**CHAPTER 7**

### ****Conclusion****

Developing an **Online Glass Purchase** platform requires a combination of modern technologies, strategic planning, and user-centric features to deliver a smooth and secure shopping experience. By leveraging **Spring Boot (Backend), React/Angular/Thymeleaf (Frontend), and MySQL (Database),** the platform ensures **scalability, security, and high performance** while meeting both business and customer needs.

### ****Key Takeaways:****

1️. **Robust Backend with Spring Boot** – Manages API interactions, user authentication, and glass product data efficiently.  
2️. **Interactive Frontend with React/Angular/Thymeleaf** – Provides a visually appealing and user-friendly shopping experience.  
3️. **Secure Transactions & Payment Gateway Integration** – Ensures smooth and safe online payments for purchasing eyewear.  
4️. **Scalable & Optimized Database** – Enables real-time inventory tracking, seamless order processing, and data consistency.  
5️. **User-Centric Features** – Wishlist, order tracking, virtual try-on (if implemented), product reviews, and personalized recommendations enhance customer satisfaction.  
6️. **Efficient Order Fulfillment & Logistics** – Ensures faster deliveries, smooth inventory management, and excellent customer service.

**CHAPTER 8**

### ****REFERENCE****

H. Zhan, B. Shi, and A. C. Kot, **"Street-to-Shop Glass Retrieval Using a Multi-Scale Viewpoint-Invariant Triplet Network,"** 2017 IEEE International Conference on Image Processing (ICIP), Beijing, China, 2017, pp. 1102-1106. DOI: 10.1109/ICIP.2017.8296452.

**Keywords:** Glasses, Proposal Generation, Training, Visualization, Image Edge Detection, Robustness, Glass Retrieval, Viewpoint Invariance, Triplet Network, Street-to-Shop Matching, Feature Embedding.