**Step 1: Project Description**

**Main Purpose:**  
The main purpose of my e-commerce app, ShopEase, is to provide a user-friendly platform where customers can browse products, add items to their cart, and complete secure transactions online.

**Key Functionalities:**

* User registration and authentication (login, registration, JWT-based security)
* Product catalog with categories, prices, and search/filtering
* Add-to-cart and remove-from-cart features
* Cart updates and total calculation in real-time
* Cart state persists across sessions using browser local storage
* Responsive and accessible UI design

**Technologies & Frameworks Used:**

* **Blazor WebAssembly (WASM)** for the client-side frontend
* **ASP.NET Core (.NET 8)** for the backend API
* **MySQL** for the relational database
* **Blazored.LocalStorage** for cart state management in the browser
* **JWT (JSON Web Token)** authentication for secure user sessions
* **BCrypt.Net** for secure password hashing
* **Bootstrap/CSS** for responsive, accessible UI styling

**Step 2: Development Process Analysis**

**Major Challenges:**

* **Integrating secure authentication:** Setting up JWT authentication and ensuring tokens were properly validated and included in all requests was complex.
* **Protecting against security threats:** Preventing SQL injection and XSS required careful attention to input validation and output encoding.
* **Persisting cart state:** Ensuring the cart remained intact across browser refreshes and sessions needed robust local storage logic.

**How I Overcame Challenges:**

* Consulted documentation, forums, and experimented with test tokens and claims to debug authentication.
* Used parameterized queries and regex-based input sanitization to secure all database operations.
* Leveraged Blazored.LocalStorage for a simple and effective state persistence solution.

**Ensuring Code Quality & Efficiency:**

* Used clear separation of concerns with repositories, services, and DTOs.
* Consistently used async/await for smooth UI and performant API calls.
* Modularized Blazor components and used dependency injection for scalability and maintainability.

**Role of Microsoft Copilot:**

* Copilot provided code suggestions, highlighted potential vulnerabilities, and improved the efficiency of code writing.
* It helped refactor and debug challenging sections, especially in authentication and state management logic.

**Step 3: Review of Key Features**

**Business Logic Implementation:**

* **Product and Cart Classes:**  
  Products are represented as C# classes with properties for ID, name, price, and category.  
  The Cart class maintains a list of products, supports add/remove operations, and calculates totals.

**Front-End Development:**

* **Blazor Components:**  
  Product catalog, cart, and authentication forms are Blazor components that interact with backend APIs via injected services.  
  State updates in the UI in real-time using data-binding and events.

**UI/UX Enhancements:**

* Used Bootstrap and custom CSS for a responsive layout across devices.
* Implemented accessibility features: ARIA labels, keyboard navigation, and semantic HTML tags for screen readers.

**Security Implementation:**

* All passwords are securely hashed using BCrypt before storing in the database.
* JWT tokens ensure secure authentication and authorized API access.
* Input validation and sanitization on the server-side prevent SQL injection and XSS attacks.
* Used parameterized SQL queries throughout.

**State Management:**

* Shopping cart data is saved and retrieved from browser local storage using Blazored.LocalStorage.
* This ensures the cart persists after page reloads and browser restarts, providing a seamless user experience.

*End of Submission*