

Project Idea

Project Title:

E-commerce Website— A Full-Stack E-commerce Website

Objective

The objective of this project is to develop a full-stack e-commerce website that enables users to browse products, search, add items to a cart, purchase them securely, and receive order notifications, while providing administrators with tools to manage inventory, view sales analytics, and monitor customer activity.

The system will demonstrate clean code, layered architecture, and test-driven development principles while implementing both frontend and backend testing for complete reliability.

In-Scope Features

- User authentication (sign up, login, logout)
 - Product browsing and advanced search by category, price, and rating
 - Shopping cart and checkout system
 - Order tracking and email notifications
 - Admin dashboard for product and order management
 - Unit, integration, and end-to-end testing
-

Expected Outcomes

- A fully functioning web application (frontend + backend) deployed locally or online
 - Clean, modular code following SOLID principles
 - Unit, integration, and UI test cases demonstrating TDD practices
 - Architecture and design documentation (class, sequence, and entity diagrams)
 - Demonstration video and final report
-

Justification

We chose this idea because:

1. It represents a real-world, high-demand software system with multiple actors (user, admin).
 2. It includes core software engineering practices such as CRUD operations, layered architecture, and reusable components.
 3. It provides a strong foundation for testing — from unit tests (product model) to end-to-end user flows (checkout process).
 4. It's scalable, making it ideal to demonstrate design patterns and TDD principles.
-

Requirements

Functional Requirements

#	Functional Requirement	User Story
FR1	The system shall allow users to register, log in, and log out securely.	As a <i>customer</i> , I want to create an account so that I can securely access my orders and saved items.
FR2	The system shall allow users to browse and search for products by category, name, or price.	As a <i>customer</i> , I want to search for products easily so that I can find what I need quickly.
FR3	The system shall allow users to add or remove products from their shopping cart.	As a <i>customer</i> , I want to manage items in my cart so that I can decide what to buy.
FR4	The system shall allow users to place orders and view their order history.	As a <i>customer</i> , I want to place an order and view my order history so that I can track my purchases.
FR5	The system shall send order confirmation and notification emails after purchase.	As a <i>customer</i> , I want to receive notifications so that I can confirm my order was processed.
FR6	The system shall allow admins to manage products (add, edit, delete) and view sales analytics.	As an <i>admin</i> , I want to manage inventory and view sales reports so that I can make better business decisions.

Non-Functional Requirements

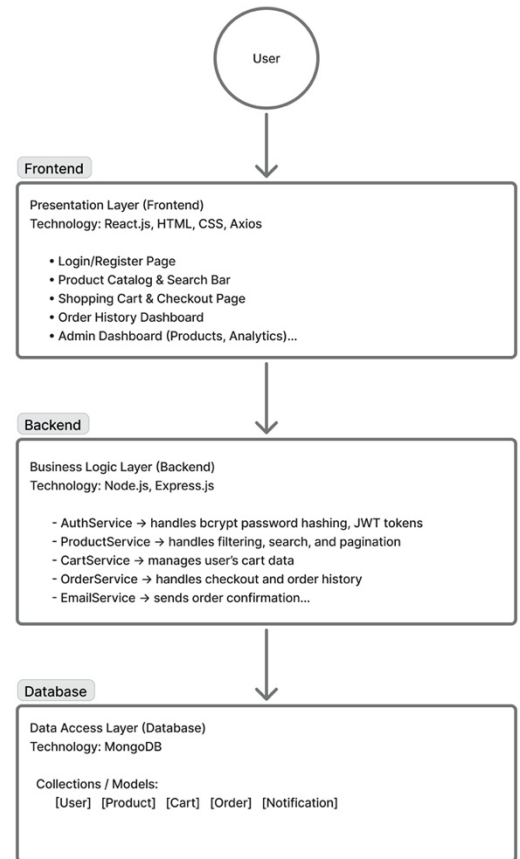
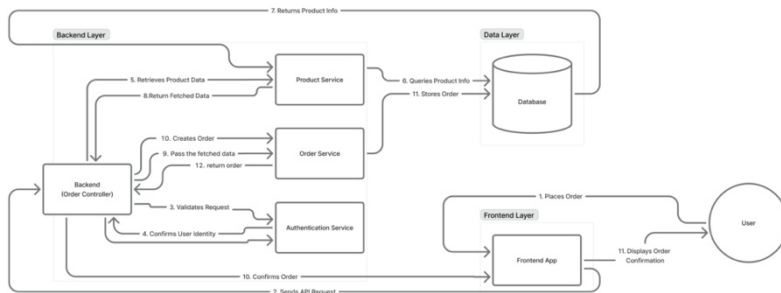
#	Non-Functional Requirement	Type
NFR1	The website shall load all pages within 3 seconds under standard network conditions.	Performance
NFR2	The system shall ensure 99% uptime when hosted.	Reliability
NFR3	All user passwords shall be stored using bcrypt encryption.	Security
NFR4	The system shall follow SOLID principles and maintain a clean, modular codebase.	Maintainability
NFR5	The system shall support concurrent user sessions without data inconsistency during checkout or cart updates.	Concurrency
NFR6	The system shall provide a responsive and accessible UI across desktop and mobile browsers.	Usability

System Architecture

Chosen Architecture: Layered Architecture (N-Tier)

The project will follow a Layered (3-tier) architecture:

1. **Presentation Layer (Frontend)**
2. **Business Logic Layer (Backend Services)**
3. **Data Access Layer (Database)**

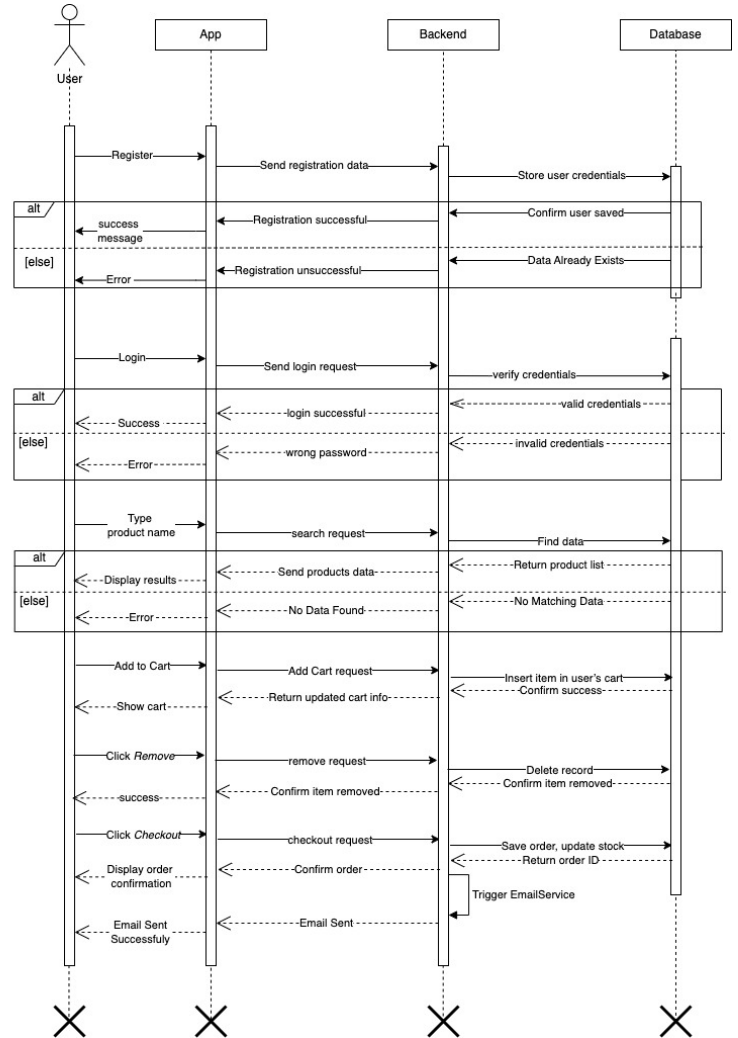
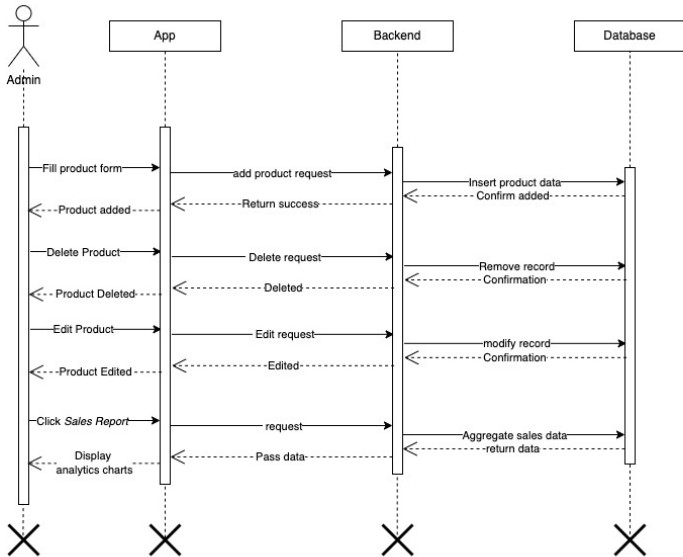


Justification for Layered Architecture. Why Suitable?

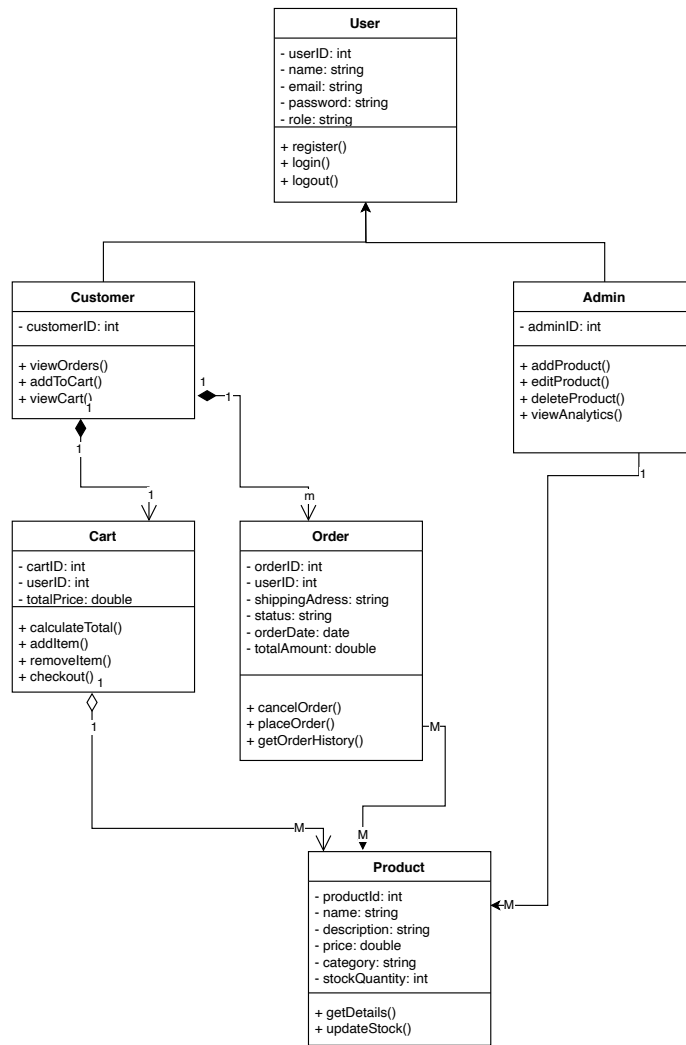
- **Separation of Concerns:** Each layer handles a distinct responsibility (UI, business logic, data).
- **Maintainability:** Easier to update one layer (e.g., UI) without affecting others.
- **Testability:** Layers can be tested independently — crucial for TDD.
- **Scalability:** Supports future upgrades, e.g., moving to microservices later.
- **Reusability:** Logic and models can be reused across multiple interfaces (admin vs customer).

Architecture Diagram UML

1. Sequence Diagram



2. Class Diagram



3. Entity-Relationship Diagram (ERD)

