COLLEGE CODE	: 8203
COLLEGE NAME	: A.V.C College Of Engineering
DEPARTMENT	: B.Tech-Information Technology
STUDENT NM_ID	: E425425876CCBA5BC5D442DCC749127F
ROLL NO	: 23IT83
DATE	: 08-09-2025

Completed the project named as Phase I
technology project name : E-Commerce Cart System

SUBMITTED BY,

NAME : Ragul R

MOBILE NO : 9940877476

Problem Statement :

In modern e-commerce, providing a seamless and reliable shopping experience is crucial for customer retention and sales conversion. However, online stores frequently face challenges with their shopping cart functionality, leading to user frustration and abandoned carts.

Key problems include:

- **Lack of Persistence**: Users lose their selected items if they accidentally close their browser, refresh the page, or switch devices. This is a direct result of the stateless nature of web protocols.
- **Inefficient Management**: Many systems offer clunky interfaces for updating item quantities or removing products, making the pre-purchase review process tedious for the user.
- **Insecure Data**: Without proper user session management, one user's cart data could potentially be exposed to another, creating a significant security and privacy risk.
- **Poor Performance**: Slow-loading carts or delayed updates can frustrate users and lead them to abandon their purchase entirely.

There is a clear need for a robust, backend-driven cart system that:

- Allows users to easily add, view, update, and remove items from their shopping cart.
- Persistently stores the cart's contents in a database (**MongoDB**) linked to a specific user.
- Secures user-specific cart data using modern authentication mechanisms like **Sessions or JWT**.
- Provides a fast, reliable, and user-friendly experience through a well-defined **REST API** built with **Node.js** and **Express**.

This solution aims to deliver a scalable and secure e-commerce cart API that serves as the backbone for a modern online shopping application.

Users & Stakeholders :

Users:

These are the individuals who will directly interact with the shopping cart functionality on an e-commerce website.

- **Online Shoppers**: Customers who browse the website and add products they intend to buy to their cart.
- **Guest Visitors**: First-time or unregistered users who want to add items to a temporary cart before deciding to create an account or check out.
- **Registered Customers**: Logged-in users who expect their cart to be saved and accessible across multiple devices and sessions.

Stakeholders:

These are the individuals or groups who have a vested interest in the project's success but may not use the cart system directly.

- **Project Development Team**: Responsible for designing, developing, testing, and deploying the cart API.
- **E-commerce Business Owner**: The primary stakeholder who wants a reliable system to maximize sales and customer satisfaction.
- **System Administrators / DevOps**: Ensure the server, database (MongoDB), and APIs are running smoothly, securely, and efficiently.
- **Marketing & Sales Team**: Rely on cart data (e.g., abandoned carts) to create retargeting campaigns and understand customer behavior.
- **Customer Support Team**: Assists users who face issues with their shopping carts or the checkout process.

User & Stakeholder Stories :

User Stories:

- **As an online shopper**, I want to add a product to my cart with a single click, so that I can continue browsing without interruption.
- **As a customer**, I want to view all the items in my cart on a single page, so that I can review my order before purchasing.
- **As a user**, I want to easily change the quantity of an item in my cart, so that I can buy more or fewer of a product as needed.
- **As a shopper**, I want to remove an item I no longer want from my cart, so that my order total is accurate.
- **As a registered customer**, I want my cart to be saved to my account, so that I can view it later from my phone or another computer.
- **As a user**, I want the checkout process to be straightforward, so I can complete my purchase quickly after reviewing my cart.

Stakeholder Stories:

- **As a developer**, I want to use a NoSQL database like MongoDB, so that I can store flexible cart data and scale the system easily.
- **As a developer**, I want to build RESTful API endpoints, so that the frontend team can easily integrate the cart functionality.

- **As a system administrator**, I want to secure the cart API using JWT or session tokens, so that each user's cart data remains private and protected.
- **As a business owner**, I want the cart to be cleared automatically after a successful purchase, so that our order and inventory management systems are accurate.

MVP (Minimum Viable Product) Features :

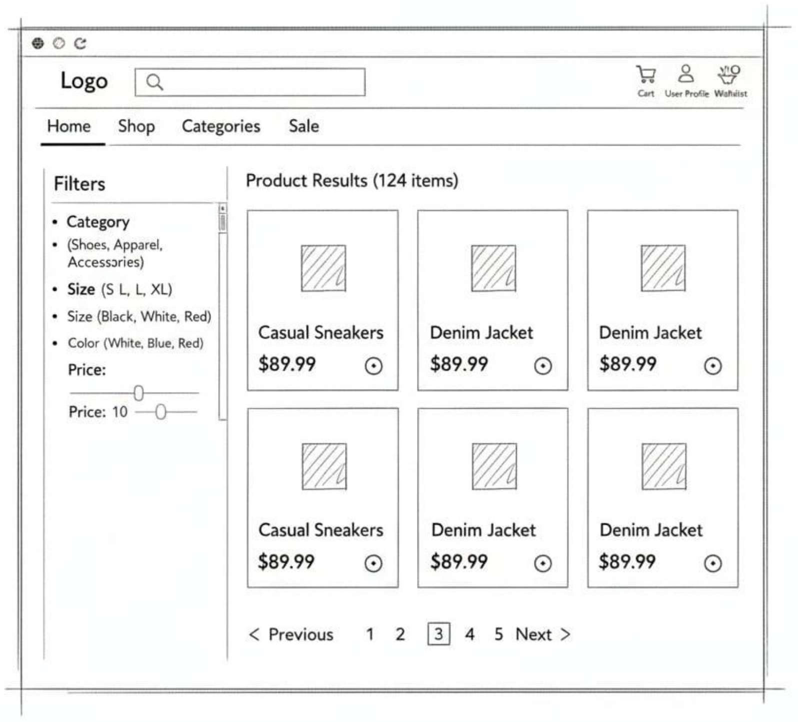
1. **Add Item to Cart**: Users can add a specific product (with a given quantity) to their personal cart.
2. **View Cart**: Users can retrieve the full contents of their cart, including product details, quantities, and price subtotals.
3. **Update Item Quantity**: Users can increase or decrease the quantity of an item already in their cart.
4. **Remove Item from Cart**: Users can completely remove an item from their cart.
5. **Persistent Storage**: The cart's state is saved in a MongoDB database, linked to a user ID.
6. **User Identification**: The system uses session tokens or JWT to identify the user and associate them with their correct cart.
7. **Basic Checkout**: A checkout endpoint that processes the order (mockup), finalizes the purchase, and clears the user's cart.

Wireframes & API Endpoint List :

Mid-Fidelity Wireframes

Cart Page Wireframe: This view shows the list of items, quantities, and options to update the cart or proceed to checkout.

Checkout Page Wireframe: A simplified view for the final step of the purchasing process.



API Endpoint List

Endpoint	Method	Description	Request Body (JSON)	Response (JSON)
/cart	POST	Add a product to the cart. If the item already exists, its quantity is updated.	{ "productId": "...", "quantity": 1 }	The updated user cart object.
/cart	GET	Retrieve the current user's cart contents.	(None)	The user's cart object with all items.
/cart/:itemId	PUT	Update the quantity of a specific item in the cart.	{ "quantity": 3 }	The updated user cart object.

Endpoint	Method	Description	Request Body (JSON)	Response (JSON)
/cart/:itemId	DELETE	Remove a specific item from the cart.	(None)	A confirmation message and the updated cart.
/checkout	POST	Process the order and clear the user's cart upon success.	{ "paymentDetails": "..." }	{ "status": "success", "orderId": "..." }

Acceptance Criteria :

Cart Management

- **Given** a user selects a product, **when** they make a POST request to /cart with a productId and quantity, **then** the item must be added to their cart in the database.
- **Given** a user is on the cart page, **when** a GET request is made to /cart, **then** the system must return all items, quantities, and prices associated with that user.
- **Given** an item is in the cart, **when** the user makes a PUT request to /cart/:itemId with a new quantity, **then** the item's quantity must be updated in the database.
- **Given** an item is in the cart, **when** the user makes a DELETE request to /cart/:itemId, **then** the item must be removed completely from their cart.

Data Persistence & Security

- **Given** a registered user adds items to their cart, **when** they log out and log back in, **then** their cart items must still be present.
- **Given** a user is interacting with the cart API, **when** a request is made, **then** it must be authenticated using a valid session token or JWT to ensure data privacy.
- **Given** an unauthenticated user, **when** they try to access a cart endpoint, **then** the system should return a 401 Unauthorized error.

Checkout Process

- **Given** a user has items in their cart, **when** they make a POST request to /checkout, **then** an order should be created (mocked) and a success message returned.
- **Given** a checkout was successful, **when** the user views their cart again, **then** the cart must be empty.

Error Handling

- **Given** a user tries to add a product with an invalid productId, **when** a POST request is made to /cart, **then** the API must return a 404 Not Found error with a clear message.
- **Given** an API or database service is down, **when** a user performs any cart action, **then** the API must return a 503 Service Unavailable error with a user-friendly message.

This document should provide a solid foundation for your project. Let me know if you need any section expanded!

Now that you have the complete Phase 1 documentation, a logical next step would be to define the database structure. Would you like me to outline the MongoDB schemas for the products, users, and carts collections?