**MAIN PLAN**  
**Planning Stage**

Define Requirements: Understand the features and functionalities needed for the e-commerce website. This will include user roles, product categories, payment methods, etc.

Research: Investigate the technologies, libraries, and APIs you'll need. Given your proficiency in JavaScript and Node.js, Next.js, Tailwind, Redux, and Mongoose are excellent choices.

Action Plan: Create a detailed action plan, including a timeline and milestones.

**Development Stage**

Setup Environment: Configure your development environment, including databases, servers, and any other services.

Backend Development: Implement the server-side logic, database models, and APIs.

Frontend Development: Develop the user interface using Next.js and Tailwind. Integrate Redux for state management.

**Testing Stage**

Unit Tests: Write unit tests for individual components and functions.

Integration Tests: Test the interaction between different parts of the application.

**Deployment Stage**

Staging: Deploy the application to a staging environment for final testing.

Production: Once everything is tested and approved, deploy the application to the production environment.

**BREAKDOWN OF MAIN PLAN**

**Define Requirements**

**User Roles: Identify the different types of users who will interact with the application (e.g., customers, admins, vendors).**

**Features and Functionalities: List out the features you want to include. For an e-commerce site, this could be product search, user authentication, cart functionality, and payment processing.**

**Data Models: Define what kind of data you'll be storing and how it will be structured. This will help in setting up the database later.**

**APIs and Third-Party Services: Decide on any third-party services you'll need, such as payment gateways or email services.**

**UI/UX Design: Sketch out the basic design elements, keeping in mind the user experience.**

**Requirements Document for E-commerce Project**

**1. Introduction**

**Purpose: To build a full-stack e-commerce application.**

**Scope: The application will include features for browsing products, user authentication, cart management, and payment processing.**

**2. User Roles**

**Customer: Can browse products, add to cart, and make purchases.**

**Admin: Manages product listings, order processing, and user accounts.**

**Vendor: Optional role can list and manage their own products.**

**3. Features and Functionalities**

**Product Search: Users can search for products by name, category, or tags.**

**User Authentication: Secure login and registration process.**

**Cart Functionality: Users can add/remove products to/from their cart.**

**Payment Processing: Integration with a payment gateway for transactions.**

**Order Tracking: Users can track the status of their orders.**

**4. Data Models**

**User: Contains information like username, password, and order history.**

**Product: Includes details like name, description, price, and category.**

**Order: Captures details of transactions, including products, quantities, and payment status.**

**5. APIs and Third-Party Services**

**Payment Gateway: To handle transactions (e.g., Stripe).**

**Email Service: For sending order confirmations and notifications (e.g., SendGrid).**

**6. UI/UX Design**

**Layout: Responsive design compatible with various screen sizes.**

**Theme: Consistent color scheme and typography.**

**User Experience: Intuitive navigation and user-friendly interface.**

**7. Technology Stack**

**Frontend: Next.js, Tailwind, Redux**

**Backend: Node.js, Express.js, Mongoose**

**Database: MongoDB**

**8. Timeline and Milestones**

**To be determined based on the action plan.**

**Next Steps**

**Research: We'll investigate the specific libraries and packages that can be used for each feature. For example, we might consider using Passport.js for authentication.**

**Action Plan: We'll create a detailed action plan, breaking down each feature into smaller tasks and assigning deadlines. This will serve as our roadmap.**

**Environment Setup: Once the action plan is ready, we'll set up the development environment, including the database and any necessary third-party services.**

**1. Authentication**

**Google OAuth 2.0: Implement Google OAuth 2.0 and traditional email/password login using Passport.js.**

**Email and Password: Implement a secure login and registration system using Passport.js' Local strategy. This will include hashing passwords and possibly implementing two-factor authentication for added security.**

**2. Product Management**

**Implement product categorization, search, and filtering without external libraries.**

**3. Cart and Payment**

**Stripe: Since we've decided on Stripe, we'll investigate its Node.js SDK and how to integrate it for payment processing.**

**Cart Management: Research state management solutions for handling cart items. Redux Toolkit is a good fit here.**

**4. Email Notifications**

**SendGrid: Explore the SendGrid Node.js library for sending transactional emails like order confirmations.**

**5. UI/UX**

**Given that we're using Tailwind, we'll use Tailwind UI pre-designed, customizable components.**

**6. Testing**

**Use Jest for both front-end and back-end testing.**

**7. Deployment**

**Deploy both front-end and back-end on Vercel using Next.js' full-stack capabilities.**

**Development Stage: Next Steps**

**Environment Setup: Initialize the Next.js project and set up the development environment, including Git for version control.**

**Authentication: Start with the authentication module. Implement Google OAuth 2.0 and traditional email/password login using Passport.js.**

**Product Management: Develop the product categorization, search, and filtering features.**

**Cart & Payment: Implement the cart management using Redux Toolkit and integrate Stripe for payments.**

**Email Notifications: Set up SendGrid for sending transactional emails.**

**UI/UX: Integrate Tailwind UI components and customize them according to our needs.**

**Testing: Set up Jest for both front-end and back-end testing. Given the extended time, we can be thorough.**

**Deployment: Once all modules are developed and tested, proceed to deploy the application on Vercel.**

**Review and Iterate: After deployment, test the live application and make any necessary adjustments.**

**Development Phase Steps**

**Database Setup: Initialize the MongoDB database and create the necessary collections.**

**Backend API: Develop the backend API routes for user authentication, product management, and order processing.**

**Frontend Development: Create the front-end pages and components using Next.js and Tailwind CSS.**

**State Management: Implement state management using Redux Toolkit.**

**User Authentication: Integrate Passport.js for Google OAuth 2.0 and email/password authentication.**

**Payment Gateway: Integrate Stripe for handling transactions.**

**Email Notifications: Use SendGrid for sending order confirmation emails.**

**Testing: Write unit and integration tests using Jest.**

**Deployment: Deploy the application to Vercel.**

**Essential Collections for an E-Commerce Project:**

**Users: As you mentioned, this is a must-have. It should contain fields like userID, username, password, email, address, roles (admin, customer, etc.), and other personal details.**

**Products: This collection will store all the information about the products you're selling. Fields might include productID, name, description, price, stockQuantity, categoryID, images, ratings, etc.**

**Categories: To categorize products. Fields could be categoryID, name, description, and perhaps a parentCategoryID for nested categories.**

**Orders: This will track all customer orders. Fields could include orderID, userID, totalAmount, status (processing, shipped, etc.), paymentMethod, orderDate, and an array of productIDs with quantities.**

**Cart: To hold items that a user is considering purchasing. Fields might include cartID, userID, and an array of productIDs with quantities.**

**Reviews: For customer reviews of products. Fields could include reviewID, userID, productID, rating, comment, and datePosted.**

**Payment: To store payment methods and transactions. Fields could be paymentID, userID, orderID, paymentMethod, transactionID, status, and date.**

**Shipping: To manage shipping details. Fields could include shippingID, orderID, userID, status, trackingNumber, and estimatedArrival.**

**Wishlist: To store products that a user wishes to buy in the future. Fields might include wishlistID, userID, and an array of productIDs.**

**Coupons: To manage discount coupons. Fields could include couponID, code, description, discountAmount, validFrom, and validTo.**

**Logs: For auditing and debugging. Fields could include logID, action, userID, date, and details.**

**Notifications: To manage notifications for users. Fields could include notificationID, userID, message, status, and date.**

**User Authentication: Implement routes for user registration, login, and token verification. This is crucial as many other routes will require user authentication.**

**User Profile: After authentication, you can work on routes for fetching and updating the user profile.**

**Products: Implement CRUD (Create, Read, Update, Delete) operations for products. This will be one of the core functionalities.**

**Cart: Once you have products, you can implement the cart functionality. This would include adding items to the cart, updating quantities, and removing items.**

**Orders: After the cart is functional, you can work on placing orders, which would involve creating new orders, updating their status, and listing orders for a user.**

**Reviews: Implement routes for adding, updating, and deleting product reviews.**

**Wishlist: Implement routes for adding and removing items from the wishlist.**

**Payment: Implement routes for handling payments, which could be triggered when an order is placed.**

**Shipping: Implement routes for updating and tracking shipping information.**

**Optional Features: If you decide to add coupons, logs, or notifications later, they can be implemented last.**

**Backend API Development**

**User Authentication**

**You've already implemented routes for user registration and login. Next, you could add routes for token verification, password reset, and email verification.**

**User Profile**

**Create routes to fetch and update user profile information. This could include changing the password, updating the email, and editing personal details.**

**Products**

**Implement CRUD operations for products. This will involve creating new products, reading product information, updating existing products, and deleting products.**

**Cart**

**Implement routes for cart operations. This includes adding items to the cart, updating item quantities, and removing items from the cart.**

**Orders**

**Create routes for order management. This involves placing new orders, updating order status, and fetching order history for a user.**

**Reviews**

**Implement routes for product reviews. This includes adding new reviews, updating existing reviews, and deleting reviews.**

**Wishlist**

**Create routes for managing the wishlist. This involves adding items to the wishlist and removing items from it.**

**Payment**

**Implement routes for payment processing. This could involve integrating with Stripe to handle transactions.**

**Shipping**

**Create routes for shipping management. This involves updating shipping status and tracking information.**

**Optional Features**

**If you decide to implement coupons, logs, or notifications, these can be added last. Create routes for managing these features.**

**Frontend Development**

**Create the necessary pages and components using Next.js and Tailwind CSS. Make sure to integrate them with your backend API.**

**State Management**

**Implement state management using Redux Toolkit. This will help you manage the application's state more efficiently.**

**Payment Gateway**

**Integrate Stripe for handling transactions. This will involve both frontend and backend work.**

**Email Notifications**

**Use SendGrid to send order confirmation emails and other notifications.**

**Testing**

**Write unit and integration tests using Jest to ensure that your application is robust and works as expected.**

**Deployment**

**Once everything is tested and ready, deploy your application to Vercel.**

**Step 1: Initial GET Request to /api/auth/signin**

**Create a new request in Postman.**

**Set the request type to GET.**

**Enter the URL http://localhost:<Your\_Port>/api/auth/signin.**

**Send the request.**

**Step 2: Check Cookies and Script Output**

**After sending the GET request, go to the "Test Results" tab to see the output of the script you added to the collection. You should see the csrfToken and sessionToken values printed.**

**The script should also set these tokens as environment variables in Postman.**

**Step 3: POST Request to /api/auth/callback/credentials**

**Create a new POST request in Postman.**

**Set the URL to http://localhost:<Your\_Port>/api/auth/callback/credentials.**

**In the "Body" tab, select x-www-form-urlencoded.**

**Add a key-value pair where the key is csrfToken and the value is {{csrfToken}}. This will use the environment variable that the script set.**

**Step 4: Send the POST Request**

**Send the POST request.**

**If everything is set up correctly, you should be authenticated, and a session should be established. Check the "Cookies" tab in Postman to confirm that you have received a session cookie.**

**Step 5: Test Other Routes**

**Now that you have the sessionToken, you can use it to authenticate other routes. For example, to test the GET session route:**

**Create a new GET request in Postman.**

**Set the URL to http://localhost:<Your\_Port>/api/session.**

**In the "Headers" tab, add a new key-value pair: Authorization: Bearer {{sessionToken}}.**

**Send the request. If everything is set up correctly, you should receive the session information in the response.**