**Project Title: Multi-Vendor E-commerce Platform (Amazon Clone)**

**Technology Stack:**

* **Frontend:** React.js (with React Router DOM for routing, Redux Toolkit for state management, Axios for API calls, React Icons for iconography)
* **Backend:** Node.js (with Express.js for API, Mongoose for MongoDB ODM, JWT for authentication, bcryptjs for password hashing, Nodemailer for email services, Multer for file uploads, Cloudinary for image hosting, Stripe/Razorpay for payment gateway integration)
* **Database:** MongoDB (using MongoDB Atlas for cloud hosting)
* **Styling:** Tailwind CSS (for utility-first styling and responsiveness)

**Core Features & User Roles:**

**1. General User (Unauthenticated)**

* **Homepage:** Display featured products, categories, carousels, and potentially top-rated sellers.
* **Product Browse:**
  + Browse products by category, sub-category, brand.
  + Search products by keywords with auto-suggestions.
  + Filter products by price range, ratings, availability, seller.
  + Sort products by relevancy, price (low to high/high to low), new arrivals.
  + Product detail page with multiple images, description, specifications, seller information, customer reviews, and "Add to Cart" button.
* **Authentication:**
  + User registration (email, password, name, address).
  + User login (email/password).
  + Forgot password and reset password functionality (via email).

**2. Customer (Authenticated User)**

* **All General User features.**
* **Shopping Cart:**
  + Add/remove products to/from cart.
  + Update quantities in cart.
  + View cart summary (subtotal, shipping, total).
* **Checkout Process:**
  + Multi-step checkout: Shipping address, payment method, order review.
  + Guest checkout option (for faster checkout without creating an account).
  + Integration with a payment gateway (e.g., Stripe or Razorpay) for secure transactions.
  + Order confirmation page with order details.
* **User Profile:**
  + View and edit personal information (name, email, shipping addresses).
  + View order history with detailed order status (pending, processing, shipped, delivered).
  + Ability to leave product reviews and ratings.
  + Manage wish list (add/remove products).
* **Notifications:** Real-time notifications for order status updates, seller messages, etc.

**3. Seller (Vendor)**

* **Seller Registration & Approval:**
  + Sellers can register through a dedicated form on the website.
  + **Crucial:** Seller accounts must be approved by the main Admin before they can list products.
* **Seller Dashboard (Admin Panel for Sellers):**
  + **Overview:** Dashboard with sales summary, active listings, pending orders, revenue charts.
  + **Product Management:**
    - Add new products (name, description, images, categories, price, stock, variations like color/size, SKU).
    - Edit existing product details.
    - Delete products.
    - Manage product inventory/stock levels.
    - Upload multiple product images (Cloudinary integration).
  + **Order Management:**
    - View and manage their specific orders (new, pending, completed, cancelled).
    - Update order status (e.g., "processing," "shipped," "delivered").
    - Generate shipping labels (optional, but a good advanced feature).
  + **Sales Reports:** Basic charts and data showing their sales performance.
  + **Profile Management:** Edit seller profile, business details.
  + **Payout Information:** Configure payout methods (e.g., bank account details - ensure security).
  + **Customer Interaction:** Ability to respond to customer inquiries related to their products/orders.

**4. Main Admin (Website Owner)**

* **Admin Dashboard:**
  + **Comprehensive Overview:** Total users, total vendors, total products, total orders, overall revenue, charts for sales trends, user growth, vendor activity.
  + **User Management:**
    - View, edit, deactivate, or delete any customer account.
  + **Vendor Management:**
    - View all registered vendors.
    - Approve or reject new vendor registrations.
    - Deactivate or delete vendor accounts.
    - View vendor-specific products and orders.
  + **Product Management:**
    - View, edit, or delete *any* product listed on the platform.
    - Ability to feature products on the homepage.
  + **Order Management:**
    - View all orders across all vendors.
    - Filter orders by status, vendor, customer.
    - Ability to intervene or resolve disputes (optional but good for a marketplace).
  + **Category & Brand Management:**
    - Add, edit, delete product categories and sub-categories.
    - Add, edit, delete product brands.
  + **Settings:** Manage website-wide settings (e.g., shipping rates, payment gateway configurations, commission rates for sellers).
  + **Analytics & Reporting:** Detailed charts and graphs for various metrics (e.g., top-selling products, top-performing vendors, customer demographics, revenue breakdown by category/vendor).
  + **Reviews Management:** Moderate and manage customer reviews.

**Technical Requirements & Architecture:**

* **RESTful API:** Develop a robust RESTful API using Node.js/Express.js for all functionalities (user authentication, product management, order processing, etc.).
* **Database Schema (MongoDB):**
  + **Users Collection:** \_id, name, email, password, role (customer, seller, admin), address (array of objects), cart (array of product references/details), wishlist (array of product references), createdAt, updatedAt.
  + **Sellers Collection:** \_id, userId (reference to Users collection), shopName, description, address, phone, email, isApproved (boolean), payoutDetails, createdAt, updatedAt.
  + **Products Collection:** \_id, name, description, price, images (array of URLs), category (reference to Categories collection), brand, stock, seller (reference to Sellers collection), ratings (average rating), numOfReviews, reviews (array of objects: userId, rating, comment), variants (e.g., size, color, with corresponding stock/price adjustments), SKU, isFeatured (boolean), createdAt, updatedAt.
  + **Categories Collection:** \_id, name, slug, parentCategory (self-reference for sub-categories).
  + **Orders Collection:** \_id, user (reference to Users collection), seller (reference to Sellers collection), orderItems (array of objects: product (reference), quantity, price), shippingAddress, paymentMethod, paymentResult (status, transaction ID), itemsPrice, taxPrice, shippingPrice, totalPrice, orderStatus (pending, processing, shipped, delivered, cancelled), deliveredAt, createdAt, updatedAt.
* **Authentication & Authorization:**
  + JWT (JSON Web Tokens) for secure user authentication.
  + Role-based access control (RBAC) to restrict access to certain routes/features based on user roles (customer, seller, admin).
  + Password hashing using bcryptjs.
* **Error Handling:** Implement comprehensive error handling for API endpoints and client-side interactions.
* **Image Uploads:** Utilize Multer for handling image uploads on the backend and Cloudinary for storing and serving images efficiently.
* **Payment Gateway:** Integrate a secure payment gateway (Stripe or Razorpay) for processing online payments. Handle webhooks for payment success/failure.
* **Email Services:** Use Nodemailer for sending transactional emails (e.g., order confirmations, password resets, seller approval notifications).
* **Real-time Features (Optional but highly recommended):**
  + Live order status updates for customers and sellers (using Socket.IO).
  + Chat functionality between customers and sellers (using Socket.IO).
* **Responsiveness:** Ensure the entire application is fully responsive and works seamlessly across various devices (desktop, tablet, mobile) using Tailwind CSS.
* **User Interface/User Experience (UI/UX):**
  + Clean, intuitive, and modern design inspired by Amazon.
  + Clear navigation for customers, sellers, and admin.
  + Consistent styling using Tailwind CSS.
  + Loading indicators, error messages, and success notifications for user feedback.
  + Accessibility considerations.

**Project Structure (Suggestions):**

**Backend (Node.js/Express):**

* server.js (main entry point)
* config/db.js (database connection)
* middleware/authMiddleware.js (JWT authentication, role authorization)
* models/ (User, Seller, Product, Category, Order models)
* routes/ (userRoutes, sellerRoutes, productRoutes, orderRoutes, adminRoutes)
* controllers/ (userController, sellerController, productController, orderController, adminController)
* utils/ (errorHandler, sendEmail, generateToken)

**Frontend (React.js):**

* src/
  + App.js (main component, routing)
  + index.js
  + components/common/ (Header, Footer, Navbar, Buttons, Modals)
  + components/auth/ (Login, Register, ForgotPassword)
  + components/products/ (ProductCard, ProductDetails, ProductList, Filters)
  + components/cart/ (Cart, CheckoutSteps)
  + components/orders/ (OrderDetails, OrderHistory)
  + components/seller/ (SellerDashboard, SellerProducts, SellerOrders)
  + components/admin/ (AdminDashboard, UserManagement, VendorManagement, ProductManagement, OrderManagement, AnalyticsCharts)
  + pages/ (Home, ProductPage, CartPage, CheckoutPage, ProfilePage, SellerDashboardPage, AdminDashboardPage)
  + redux/ (store.js, reducers, actions for authentication, products, cart, orders, sellers, admin)
  + api/ (axios instances, API utility functions)
  + styles/ (tailwind.css and any custom CSS if needed)
  + assets/ (images, SVGs)

**Deliverables:**

* **Complete MERN Stack Application:** Fully functional frontend and backend.
* **Clear Code Structure:** Well-organized, modular, and commented code.
* **API Documentation:** (Optional, but highly beneficial) Swagger or Postman collection.
* **Deployment Instructions:** Clear steps on how to set up and run the application locally and potentially deploy it.

**Key Considerations for LLM Generation:**

* **Modular Design:** Emphasize generating code in a modular fashion, separating concerns (components, reducers, actions, API calls).
* **Error Handling & Validation:** Request robust input validation on both frontend and backend, and comprehensive error handling.
* **Security Best Practices:** Stress the importance of secure password handling, JWT best practices, and preventing common web vulnerabilities (XSS, CSRF).
* **Scalability:** While a basic implementation is fine, hint at considerations for future scalability (e.g., efficient database queries, proper indexing).
* **Readability and Maintainability:** Ask for clean, readable code with appropriate comments where necessary.
* **Tailwind CSS Integration:** Ensure all styling is done using Tailwind CSS utility classes, promoting a consistent and efficient design system.
* **Real-world Data:** Suggest using mock data or a seeding script for initial setup and testing.
* **User Experience (UX):** Remind the LLM to consider user flows and provide a smooth, intuitive experience.