Technical Execution

System Architecture

1. **Frontend:**
   - **Next.js + React**:
     - Next.js will be used for server-side rendering (SSR) and static site generation (SSG). React will handle the UI components and user interactions.
     - Next.js pages can fetch content from Sanity and display it on the e-commerce platform.
   - **API Calls**: The frontend will make API calls to the backend (Sanity and any external services) for product data, customer orders, authentication, etc.
2. **Backend:**
   - **Sanity**: Will be the headless CMS, storing product details, user reviews, and other dynamic content like blog posts, FAQs, etc. It will provide a rich set of APIs to interact with content in a structured manner.
   - **API Layer**: A Node.js server or serverless functions to manage business logic, authentication, order processing, and interactions with payment gateways..
3. **API Integration:**
   - APIs will be integrated for:
     - Product Data (via Sanity APIs)
     - Authentication
     - Order Management
     - Payment Gateway
     - Shipping APIs
     - Reviews and Ratings APIs

API Specification

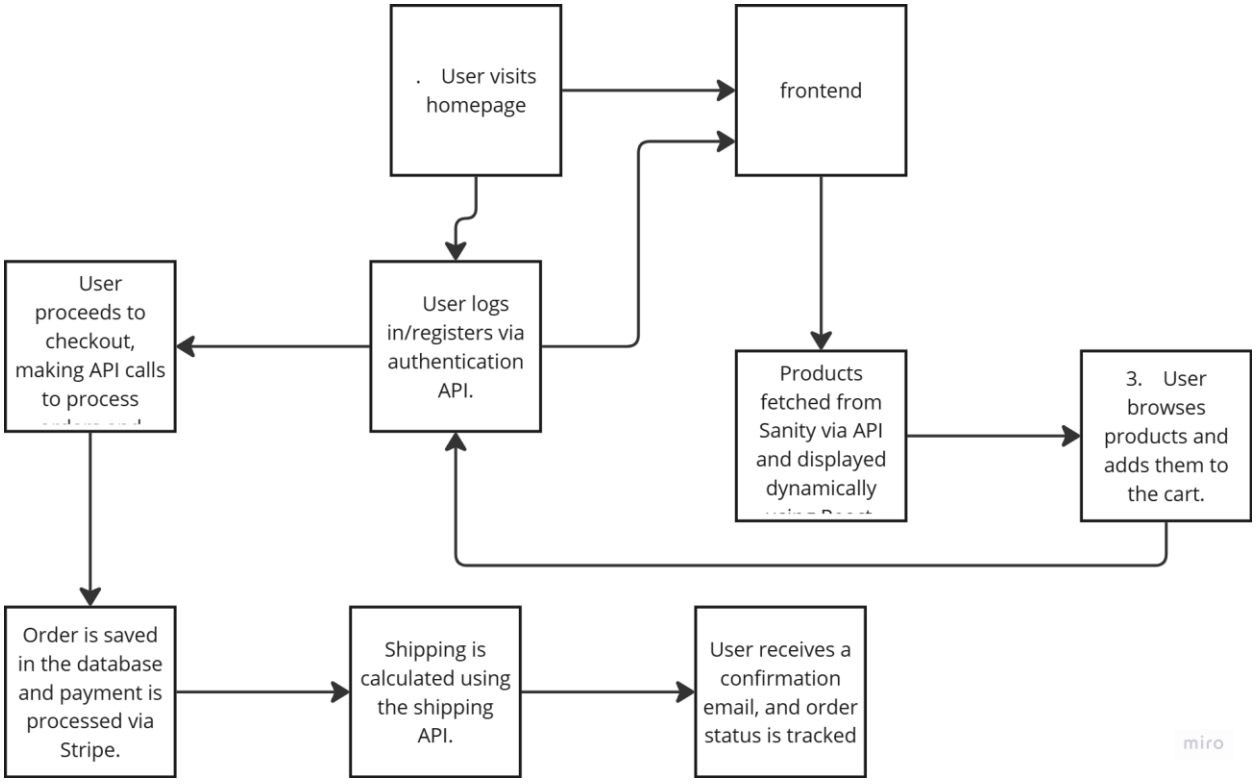
1. **Authentication API:**
   - `POST /api/auth/login`: User login (returns JWT)
   - `POST /api/auth/register`: User registration
   - `POST /api/auth/logout`: Logout user
2. **Product API:**
   - `GET /api/products`: Fetch all products
   - `GET /api/products/{id}`: Fetch product by ID
   - `POST /api/products`: Create new product (Admin only)
   - `PUT /api/products/{id}`: Update product details (Admin only)
   - `DELETE /api/products/{id}`: Delete product (Admin only)
3. **Order API:**
   - `POST /api/orders`: Place a new order

- `GET /api/orders/{userId}`: Fetch orders by user
- `PUT /api/orders/{orderId}`: Update order status
- `DELETE /api/orders/{orderId}`: Cancel order

4. **Payment API (Stripe):**
   - `POST /api/payment/checkout`: Handle payment processing
   - `GET /api/payment/status`: Check payment status
5. **Shipping API** (if integrating a third-party service like USPS, DHL):
   - `GET /api/shipping/calculate`: Calculate shipping cost based on user's cart and location

Workflow Diagram

1. **User Journey**:
   1. User visits homepage (Next.js).
   2. Products fetched from Sanity via API and displayed dynamically using React.
   3. User browses products and adds them to the cart.
   4. User logs in/registers via authentication API.
   5. User proceeds to checkout, making API calls to process orders and payment.
   6. Order is saved in the database and payment is processed via Stripe.
   7. Shipping is calculated using the shipping API.
   8. User receives a confirmation email, and order status is tracked.



Final Deployment:

- Deploy **Frontend** on **Vercel**..

- Use **Sanity** for content management and database.

1. Product Listing (Fetching Products):

- **Client Request:**
  - The frontend (React/Next.js) sends a request to the API.
  - API Endpoint: `/api/products`
- **API Logic:**
  - Fetch product data from Sanity CMS
- **Response:**
  - The product list is returned to the client.

---

2. Product Details:

- **Client Request:**
  - When the user clicks on a product, a request is sent to fetch detailed data.
  - API Endpoint: `/api/products/[id]`
- **API Logic:**
  - Fetch the specific product's details from Sanity CMS using its `id`.
- **Response:**
  - Return detailed product information to the frontend.

---

3. Cart Management:

- **Client-Side Handling:**
  - When a user adds a product to the cart, it is managed in local state (e.g., React State, Redux).
- **Optional Server-Side Handling:**
  - If cart data needs to be saved on the backend, use the API Endpoint `/api/cart`.

---

4. Checkout Process:

- **Client Request:**
  - The user initiates the checkout process, sending a request with order details.
  - API Endpoint: `/api/checkout`

- **API Logic:**
  - Save the order data to Sanity CMS or another database.
  - Integrate with a payment gateway (e.g., Stripe or PayPal) to process the payment.
- **Response:**
  - Return payment status (success or failure) and order confirmation details.

---

5. Order History:

- **Client Request:**
  - The user requests their order history.
  - API Endpoint: `/api/orders`
- **API Logic:**
  - Retrieve the list of orders associated with the user from Sanity CMS.
- **Response:**
  - Return the order history to the client.

---

6. Admin Features:

- **Endpoints:**
  - `/api/admin/products` for managing products.
  - `/api/admin/orders` for managing orders.
- **API Logic:**
  - Provide functionality to add, update, or delete products.
  - Allow the admin to view and manage customer orders.
- **Backend Management:**
  - Use Sanity Studio as the admin panel for content management.

API Flow Diagram

1. **Client (React/Next.js):** Sends requests to Next.js API routes.
2. **Next.js API Routes:** Handles the logic and acts as a bridge between the client and the backend services.
3. **Sanity CMS:** Stores and fetches data such as products, orders, and user information.
4. **Payment Gateway (e.g., Stripe):** Processes payments securely.
5. **Database (Sanity Data Store):** Stores the final data, such as user orders and product information.