HACKATHON DAY 2

PLANNING THE TECHNICAL FOUNDATION OF OUR E-COMMERCE WEBSITE

ROADMAP OF OUR E-COMMERCE WEBSITE

For our e-commerce website, we are using the following technologies:

Frontend:

- **Framework:** We have chosen Next.js for its ability to handle server-side rendering (SSR) and static site generation (SSG), ensuring a fast and responsive user experience.
- **Styling:** Tailwind CSS is being used for a modern, utility-first approach to styling.
- **State Management:** We're utilizing Context API for efficient state management.
- **Animations:** Framer Motion has been selected to add smooth and engaging animations.

Backend:

- **Framework:** We are using custom API routes in Next.js for server-side logic.
- **Database:** Sanity CMS is managing our content, such as products, orders, and categories.
- **Authentication:** NextAuth.js is handling secure login and user sessions.

Third-Party APIs:

- **Payment Gateway:** We've integrated Stripe for secure payment processing.
- **Shipping:** Shippo is being used for real-time shipment tracking and rate calculations.
- **Notifications:** SendGrid is used for sending email notifications, including order confirmations.

---

2. WEBSITE ARCHITECTURE

Overview

Our website architecture is designed to ensure seamless interaction between the user interface, backend API routes, Sanity CMS, and third-party integrations like Stripe and Shippo. We have structured it to provide an intuitive user experience and efficient backend operations.

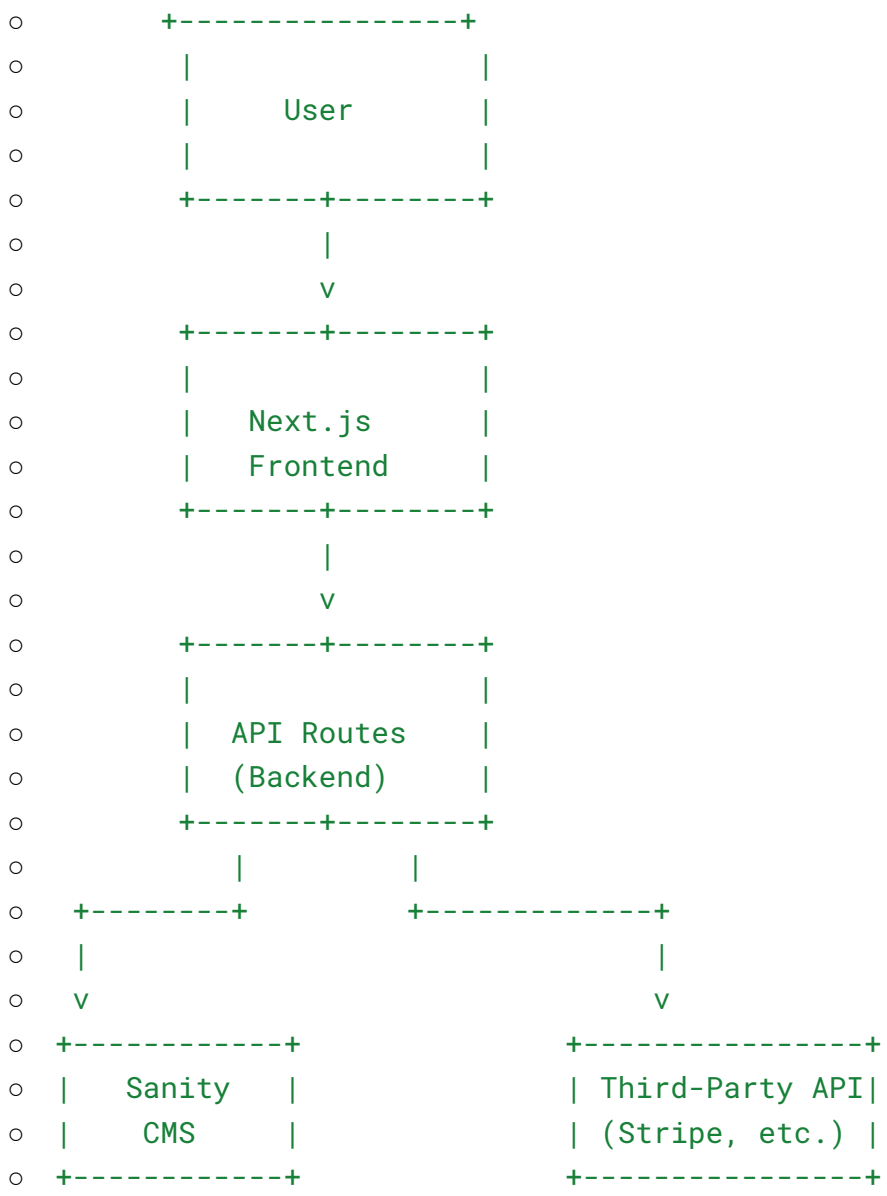
Data Flow Diagram

```
[User] -> [Next.js Frontend] -> [API Endpoints] -> [Sanity CMS | Third-Party APIs]
```

Interaction Flow

1. Users interact with our **frontend** to browse products, manage their cart, and complete purchases.
2. The **backend** processes these interactions, handles user authentication, and communicates with Sanity CMS and third-party APIs.
3. **Sanity CMS** stores our product, category, and order data dynamically.
4. Third-party APIs like Stripe and Shippo handle payments and shipping.

Process Diagram



---

3. FEATURES BREAKDOWN

User Signup/Login

- We allow users to register and log in securely.
- Authentication is managed through NextAuth.js, which issues tokens to ensure session security.

Product Listing

- Products are dynamically fetched from our Sanity CMS using GROQ queries.
- We ensure the data is rendered efficiently using server-side rendering (SSR) or incremental static regeneration (ISR).

Cart Management

- For guest users, their cart data is stored in localStorage.
- For logged-in users, the cart is synced with our backend, ensuring their data is persistent and accessible across devices.

Checkout

- We process payments securely through Stripe.
- Shipping costs and delivery times are calculated in real-time using Shippo APIs.
- Email notifications are sent via SendGrid once the order is placed successfully.

---

4. API REQUIREMENTS

Authentication API

- **Endpoint:** `/api/auth/signup`
- **Method:** POST
- **Description:** Registers new users.

Products API

- **Endpoint:** `/api/products`
- **Method:** GET
- **Description:** Fetches product details from our Sanity CMS.

Cart API

- **Endpoint:** `/api/cart/add`
- **Method:** POST
- **Description:** Adds items to the user's cart.

Checkout API

- **Endpoint:** `/api/checkout`
- **Method:** POST
- **Description:** Processes payments and finalizes orders.

---

5. DATA FETCHING PLAN

- **Home Page:** We fetch featured products using incremental static regeneration (ISR) to keep data fresh without sacrificing performance.
- **Product Details Page:** Server-side rendering (SSR) is used to deliver up-to-date product details.
- **User Dashboard:** Client-side rendering (CSR) is used for personalized data like orders and cart items.

---

6. SANITY CMS SCHEMAS

Product Schema:

- Fields:
    - Name (string)
    - Price (number)
    - Description (text)
    - Image (image with hotspot support)
    - Category (reference to the Category Schema)

Category Schema:

- Fields:
    - Name (string)
    - Description (text)

Order Schema:

- Fields:
    - Customer Name (string)
    - Products (array of references to the Product Schema)
    - Total Price (number)
    - Status (enum: pending, shipped, completed)

Customer Schema:

- Fields:
    - Name (string)
    - Email (string)
    - Address (object: street, city, state, zip)
    - Orders (array of references to the Order Schema)

---

7. FOLDER STRUCTURE

```
/project
├── /components          # Reusable UI components
├── /pages
│   ├── /api             # API endpoints
│   ├── index.tsx        # Home page
│   ├── product/[id].tsx # Product details page
│   ├── cart.tsx         # Cart page
│   ├── checkout.tsx     # Checkout page
├── /styles              # Tailwind CSS files
├── /utils               # Utility functions
├── /sanity              # Sanity CMS schemas
├── /public              # Static assets
```

---

8. TECHNICAL DOCUMENTATION SUMMARY

Frontend:

We developed the frontend using Next.js, leveraging its SSR, SSG, and dynamic routing capabilities.

CMS:

Sanity CMS is used to manage all content, including products, categories, and orders.

Third-Party Integrations:

- **Stripe:** For secure payment processing.
- **Shippo:** For calculating shipping rates and tracking orders.

API Endpoints:

Our website includes APIs for authentication, product management, cart operations, checkout, and order processing.

Data Flow:

We dynamically fetch data using Next.js to ensure a seamless user experience. Transactions and updates are efficiently managed through API calls.