HACKATHON DAY 2

PLANNING THE TECHNICAL FOUNDATION OF E-COMMERCE WEBSITE

ROAD MAP OF AN E-COMMERCE WEBSITE

1. Technology Stack

Frontend

- **Framework**: Next.js for server-side rendering (SSR) and static site generation (SSG).
- **Styling**: Tailwind CSS or Styled Components.
- **State Management**: Context API or Redux Toolkit.
- **Animations**: Framer Motion.

Backend

- **Backend Framework**: Custom API routes in Next.js or a separate Node.js server.
- **Database**: Sanity for CMS (product and content management).
- **Authentication**: NextAuth.js or Firebase Authentication.

Third-Party APIs

- **Payment Gateway**: Stripe or PayPal API.
- **Shipping**: Shippo or EasyPost API.
- **Notifications**: Twilio (SMS), SendGrid (email).

2. System Architecture

Overview

1. **Frontend (Client)**: Provides the user interface and communicates with backend APIs.
2. **Backend**: Manages business logic, authentication, and API endpoints.
3. **CMS (Sanity)**: Stores dynamic content like product details, blog posts, and metadata.
4. **Third-Party Services**: Handles payments, shipping, and notifications.

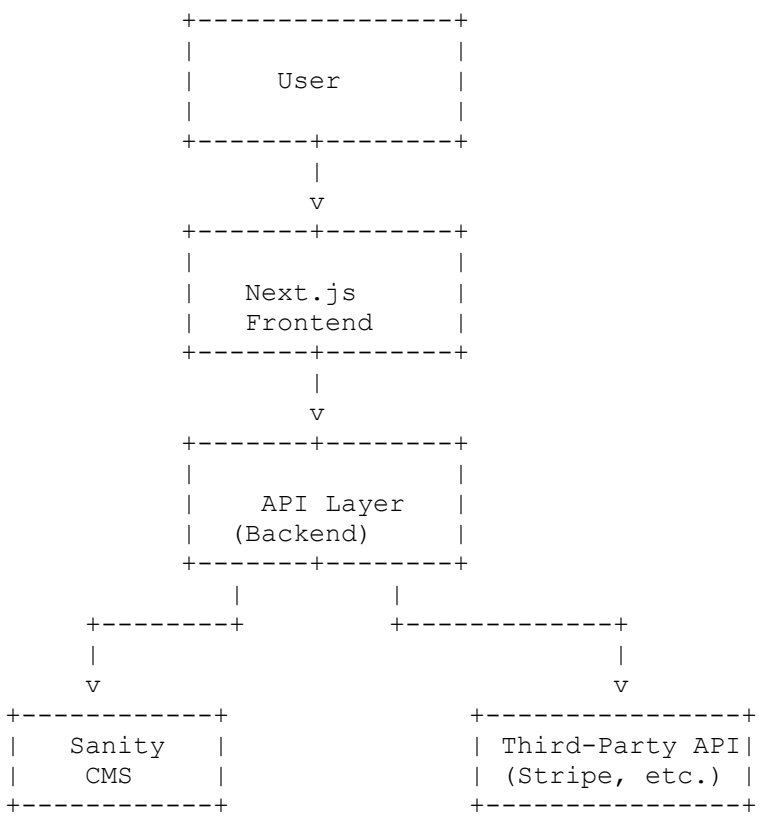
Flow

- User interacts with the frontend (Next.js).
- Next.js makes API calls to backend endpoints.
- Backend processes requests, interacts with Sanity and third-party APIs, and returns data.
- Frontend displays data dynamically.

Data Flow Diagram

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

Process Diagram



3. Features Breakdown

User Signup/Login

- **Process**:
    1. User submits the signup/login form.
    2. Backend authenticates using NextAuth.js or Firebase.
    3. JWT tokens are issued for authenticated sessions.

Product Listing

- Fetch product data from Sanity using GROQ queries.
- Display data with server-side rendering or incremental static regeneration (ISR).

Cart Management

- Store cart data in localStorage or a database for logged-in users.
- Sync cart data with the backend periodically.

Checkout

- Process payments via Stripe.
- Capture shipping details and calculate costs using the shipping API.
- Confirm order details and send notifications.

4. API Requirements

4.1 User Authentication

Endpoint: `/api/auth/signup`

- **Method**: POST
- **Description**: Registers a new user.
- **Request**:

```
{
  "name": "John Doe",
  "email": "john.doe@example.com",
  "password": "securepassword"
}
```

- **Response**:

```
{
  "message": "User registered successfully",
  "userId": "abc123"
}
```

4.2 Product List

Endpoint: `/api/products`

- **Method**: GET
- **Description**: Fetches all products.
- **Response**:

```
[
```

```
  {
    "id": "prod1",
    "name": "Product A",
    "price": 100.0,
    "image": "url_to_image",
    "description": "High-quality product"
  },
  {
    "id": "prod2",
    "name": "Product B",
    "price": 200.0,
    "image": "url_to_image",
    "description": "Another great product"
  }
]
```

4.3 Add to Cart

Endpoint: `/api/cart/add`

- **Method**: POST
- **Description**: Adds an item to the user's cart.
- **Request**:

```
{
  "userId": "abc123",
  "productId": "prod1",
  "quantity": 2
}
```

- **Response**:

```
{
  "message": "Item added to cart",
  "cartId": "cart123"
}
```

4.4 Checkout

Endpoint: `/api/checkout`

- **Method**: POST
- **Description**: Processes payment and confirms the order.
- **Request**:

```
{
  "userId": "abc123",
  "cartId": "cart123",
  "paymentMethod": "stripe",
  "shippingAddress": {
    "line1": "123 Main St",
```

```
    "city": "San Francisco",
    "state": "CA",
    "zip": "94105"
  }
}
```

- **Response**:

```
{
  "message": "Order placed successfully",
  "orderId": "order789"
}
```

5. Data Fetching Plan

Home Page

- Fetch featured products using ISR.

Product Details Page

- Use SSR for fetching individual product details from Sanity.

User Dashboard

- Fetch user-specific data (orders, cart) using client-side rendering (CSR).

6. Sanity Schema Definition

Example Schema: Product

```
import { defineField, defineType } from 'sanity';

export default defineType({
  name: 'product',
  title: 'Product',
  type: 'document',
  fields: [
    defineField({
      name: 'name',
      title: 'Name',
      type: 'string',
    }),
    defineField({
      name: 'price',
      title: 'Price',
      type: 'number',
    }),
    defineField({
```

```
      name: 'description',
      title: 'Description',
      type: 'text',
    }),
    defineField({
      name: 'image',
      title: 'Image',
      type: 'image',
      options: {
        hotspot: true,
      },
    }),
    defineField({
      name: 'category',
      title: 'Category',
      type: 'reference',
      to: [{ type: 'category' }],
    }),
  ],
});
```

Example Schema: Category

```
export default defineType({
  name: 'category',
  title: 'Category',
  type: 'document',
  fields: [
    defineField({
      name: 'name',
      title: 'Name',
      type: 'string',
    }),
    defineField({
      name: 'description',
      title: 'Description',
      type: 'text',
    }),
  ],
});
```

7. Technical Documentation

Setting Up the Project

1. **Install Dependencies**:

   ```
   npm install next react react-dom sanity @sanity/client @stripe/stripe-js
   ```
2. **Configure Environment Variables**:
   - `NEXT_PUBLIC_SANITY_PROJECT_ID`
   - `NEXT_PUBLIC_STRIPE_API_KEY`
   - `NEXTAUTH_SECRET`

Folder Structure

```
/project
├── /components
├── /pages
│   ├── /api
│   ├── index.tsx
│   ├── product/[id].tsx
├── /styles
├── /utils
├── /sanity
```

Deployment

- **Frontend**: Vercel.
- **Backend**: Hosted within Vercel or deployed as serverless functions.
- **Sanity**: Hosted on Sanity's platform.
- **Stripe Webhooks**: Use ngrok during development for local testing.

Monitoring

- Use tools like Sentry for error tracking and Postman for API testing.