PLANNING

THE

TECHNICAL FOUNDATION

HACKTHON : 2 TASK

Technical Requirements

1. Frontend Requirements (Next.js + Tailwind CSS)

The frontend will be made using Next.js (for server-side rendering and API routing) and Tailwind CSS (utility-first styling).

Pages and Features

- **Home Page (/)**

  Shows featured products, categories, and promotion banners.
  Hero section with CTA button
  Search bar to search for the product with a debounce to get optimized search results

- **Product Listing Page (/products)**

  Fetch all the products with a GET request from Sanity CMS and display on the page.
  Sorting and filtering by price, category, and stock availability.
  Pagination for big data with dynamic routing from Next.js
  Product Details Page (/products/[id])
  This will show product information such as product name, price, description, and images. Dynamically fetching data based on the product ID using Sanity CMS.
  Lightbox-enabled gallery to add multiple product images.
  Add to Cart button and state management through useReducer or Redux Toolkit

- **Cart Page (/cart)**

  showing all products in the cart, with quantity and breakdown of the total price.
  Create a cart context with the use of React Context API to handle global state.
  Option to increase and decrease quantities as well as delete an item
  Subtotal, shipping cost, and total amount

- **Checkout Page (/checkout)**

  Gather customer details by creating a controlled form
  Connect Stripe API to ensure safe transaction processing
  Real-time validation on fields like email and credit card numbers
  Order Confirmation Page (/order-confirmation)
  Success message along with the details of the order
  Unique Order ID by making use of Stripe's transaction ID.

- ## Login Page (/login)

INPUT FORM FOR E-MAIL AND PASSWORD.
VALIDATION :
REQUIRED FIELDS
INVALID INPUT, REAL-TIME FEEDBACK
LOGIN FLOW:
SEND A POST REQUEST TO /API/AUTH/LOGIN
SUCCESS: STORE JWT IN HTTP-ONLY COOKIES
REDIRECT TO THE DASHBOARD
LOGOUT BUTTON (COMPONENT)
LOG OUT FLOW :
SEND A POST REQUEST TO /API/AUTH/LOGOUT
CLEAR THE JWT COOKIE, REDIRECT TO /LOGIN.

- ## Admin Dashboard (/admin)

PROTECTED ROUTE: ONLY ACCESSIBLE WHEN A VALID JWT TOKEN IS PRESENTED.
FEATURES:
MANAGE PRODUCTS (ADD/EDIT/DELETE).
VIEW ORDERS WITH THEIR RESPECTIVE STATUSES (PENDING, SHIPPED, DELIVERED).INPUT VALIDATION ON THE CLIENT AND SERVER-SIDE.

- ## Front-end Security

FRONT-END SECURITY
JWT HANDLING: STORED IN HTTP-ONLY COOKIES
ROUTE PROTECTION: MIDDLEWARE-BASED ROUTE GUARDS
VALIDATION: FORM

- ## UI & UX Requirements:

RESPONSIVE DESIGN:
MOBILE-FIRST APPROACH WITH FULL RESPONSIVENESS ACROSS DEVICES.
SEO OPTIMIZATION: IMPLEMENT NEXT/HEAD FOR META TAGS AND OPEN GRAPH OPTIMIZATION.
ACCESSIBILITY:
ENSURE ARIA ROLES AND ALT TEXTS FOR BETTER USABILITY.
PERFORMANCE OPTIMIZATION:
IMAGE OPTIMIZATION USING NEXT/IMAGE.
SERVER-SIDE RENDERING (SSR) FOR SEO BENEFITS.
LAZY LOADING AND CODE SPLITTING FOR PERFORMANCE GAINS.

Frontend Libraries to Use:

- **Libraries:**

@SANITY/CLIENT

FETCHING AND SENDING DATA FROM SANITY

SHADCN

ACCESSIBLE MODERN UI COMPONENTS.

TAILWIND CSS:

UTILITY-FIRST CSS FRAMEWORK

LUCIDE-REACT

SVG ICONS
FORM HANDLING & VALIDATION

REACT_HOOK_FORM:

LIGHTWEIGHT FORM HANDLING.

ZOD:

SCHEMA VALIDATION USING REACT-HOOK-FORM
STATE MANAGEMENT

REACT CONTEXT API:

TO USE FOR THE APPLICATION'S GLOBAL STATE MANAGEMENT
HTTP REQUESTS & SECURITY

AXIOS:

FOR THE FRONTEND API REQUEST.
JSONWEBTOKEN (JWT-DECODE): USED FOR FRONTEND-SIDE
JWT DECODING.

2. BACKEND REQUIREMENTS (SANITY CMS).

Sanity CMS will be used as the headless CMS for content and order management.

- Product Schema

```
EXPORT DEFAULT {
  NAME: 'PRODUCT',
  TYPE: 'DOCUMENT',
  FIELDS: [
    { NAME: 'NAME', TYPE: 'STRING', TITLE: 'PRODUCT NAME' },
    { NAME: 'PRICE', TYPE: 'NUMBER', TITLE: 'PRICE' },
    { NAME: 'DESCRIPTION', TYPE: 'TEXT', TITLE: 'DESCRIPTION' },
    { NAME: 'CATEGORY', TYPE: 'STRING', TITLE: 'CATEGORY' },
    { NAME: 'STOCK', TYPE: 'NUMBER', TITLE: 'STOCK LEVEL' },
    { NAME: 'IMAGE', TYPE: 'IMAGE', TITLE: 'PRODUCT IMAGE' }
  ]
}
```

- Order Schema

```
EXPORT DEFAULT {
 NAME: 'ORDER',
 TYPE: 'DOCUMENT',
 FIELDS: [
  { NAME: 'CUSTOMERNAME', TYPE: 'STRING', TITLE: 'CUSTOMER NAME' },
  { NAME: 'CUSTOMEREMAIL', TYPE: 'STRING', TITLE: 'CUSTOMER EMAIL' },
  { NAME: 'ORDERDATE', TYPE: 'DATETIME', TITLE: 'ORDER DATE' },
  { NAME: 'PRODUCTS', TYPE: 'ARRAY', OF: [{TYPE: 'REFERENCE', TO: [{TYPE: 'PRODUCT'}]}] },
  { NAME: 'TOTALAMOUNT', TYPE: 'NUMBER', TITLE: 'TOTAL AMOUNT' },
  { NAME: 'PAYMENTSTATUS', TYPE: 'STRING', TITLE: 'PAYMENT STATUS' }
 ]
}
```

SANITY CMS ADMINISTRATION

REAL-TIME COLLABORATION: MULTIPLE CONTRIBUTORS CAN MANAGE PRODUCTS AND ORDERS.
API FIRST APPROACH: FETCH DATA USING SANITY API AT HTTPS://YOURPROJECT.API.SANITY.IO/V1/DATA/QUERY.
SECURE ACCESS: SANITY TOKENS SHOULD BE USED FOR READ/WRITE ACCESS IN THE ADMIN DASHBOARD.

3. THIRD PARTY API INTEGRATION

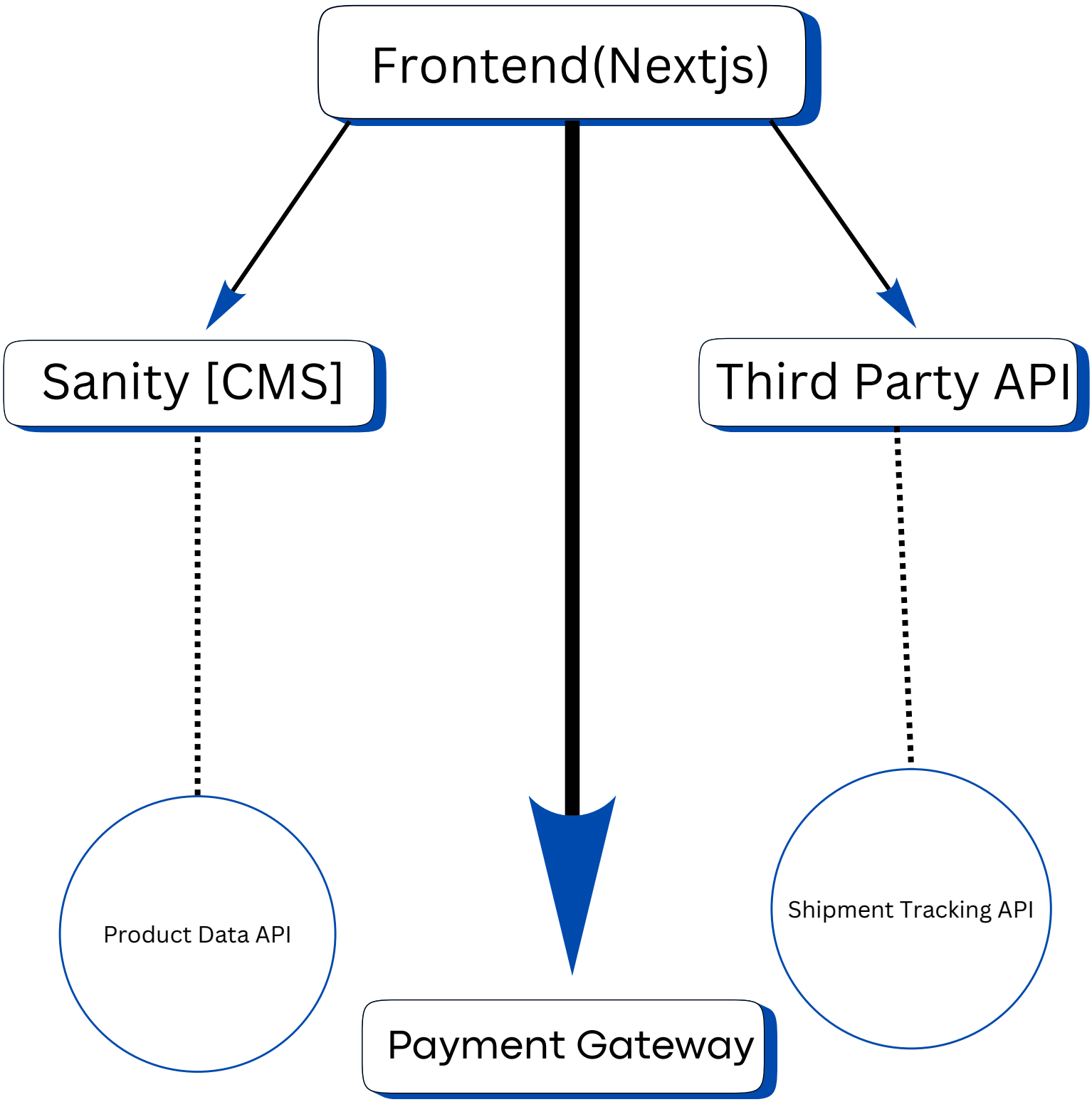
- ShipEngine API Integration:

INSTALL AXIOS: IF NOT ALREADY INSTALLED, USE NPM INSTALL AXIOS.
GET API KEY: SIGN UP ON SHIPENGINE TO OBTAIN YOUR API KEY.
MAKE REQUESTS: USE AXIOS TO MAKE REQUESTS TO SHIPENGINE'S API ENDPOINTS, SUCH AS FETCHING TRACKING INFORMATION.
HANDLE DATA: USE THE RESPONSE DATA (LIKE TRACKING INFO) IN YOUR APPLICATION.

- Stripe API Integration:

INSTALL STRIPE SDK: USE NPM INSTALL @STRIPE/STRIPE-JS FOR FRONTEND INTEGRATION.
GET API KEY: SIGN UP ON STRIPE AND OBTAIN YOUR API KEY.
CREATE PAYMENT INTENT: ON THE BACKEND, CREATE A PAYMENT INTENT TO MANAGE PAYMENTS.
HANDLE PAYMENT: USE STRIPE ELEMENTS ON THE FRONTEND TO SECURELY COLLECT PAYMENT DETAILS AND COMPLETE THE PAYMENT.

4. SYSTEM ARCHITURE DIAGRAM



5. API REQUIREMENTS

- GET Methods

Fetch Products:
Endpoint: /api/products
Method: GET
Description: Retrieve a list of products from the database.
Fetch Order Details:
Endpoint: /api/orders/:id
Method: GET
Description: Retrieve order details by order ID.
Fetch User Profile:
Endpoint: /api/users/:id
Method: GET
Description: Retrieve user profile by user ID.

- POST Methods

Add Product:
Endpoint: /api/products
Method: POST
Description: Add a new product to the store.
Create Order:
Endpoint: /api/orders
Method: POST
Description: Place a new order.
Register User:
Endpoint: /api/users
Method: POST
Description: Register a new user in the system.
These methods allow basic CRUD operations for managing products, orders, and user data in your eCommerce web app.

- PATCH & DELETE Method

PATCH /api/products/:id → Update product details
DELETE /api/products/:id → Delete a product
PATCH /api/orders/:id → Update order status
DELETE /api/orders/:id → Cancel/Delete an order
PATCH /api/users/:id → Update user profile
DELETE /api/users/:id → Delete a user (Admin only)
PATCH /api/cart/:id → Update cart item quantity
DELETE /api/cart/:id → Remove item from cart
DELETE /api/wishlist/:id → Remove item from wishlist