PRESENTATION

Day2_Planning_the_Technical_Foundation

HADIQA GOHAR

FRONTEND WORK FLOW

The user interacts with the website through the frontend built using Next.js and styled with Tailwind CSS. Upon visiting the homepage, they can search for products, view details, and add them to the cart. If the user is not logged in and tries to proceed to checkout, a notification prompts them to sign in or sign up. Once logged in, the user proceeds to payment using the Stripe API, which processes the transaction securely. After successful payment, an order ID is generated, and the user is directed to an order confirmation page.

BACKEND ARCHITECTURE

Sanity is used as the headless CMS to store and manage product details, customer information, and order records. Products are fetched from Sanity's API and displayed on the frontend using Next.js. When a user places an order, the backend (Sanity) stores the order details, including the product IDs, user information, and shipping address. Sanity’s API also manages the shipment tracking data. If any API call fails, error handling mechanisms are in place, and users are notified with retry options.

PAYMENT AND SHIPMENT TRACKING

The payment process is handled by Stripe, where users can securely enter their payment information, and the transaction is processed. After a successful payment, the order is stored in Sanity, and shipment details are provided to the user. Sanity also manages the order and shipment tracking data. In case of errors during payment or API calls, users are notified and can retry the transaction. This workflow ensures that users can place orders, track shipments, and securely manage payments.

SYSTEM ARCHITECTURE FLOWCHART:

```
[User]
  |
  v
[Frontend - Next.js]
  |                |
  v                v
[Sanity API (Product Data)]  ---> [Fetch Products]
  |                |
  v                v
[Product Page] <--- [Products from Sanity]
  |
  v
[Add to Cart] ---> [Frontend (Cart Management)]
  |
  v
[Proceed to Checkout] ---> [Login/Signup (Clark/Auth API)]
  |                 |
  v                 v
[Stripe Integration] ---> [Frontend (Send Payment Info)]
  |                 |
  v                 v
[Payment Processing] ---> [Stripe API (Secure Payment)]
  |                 |
  v                 v
[Payment Success] ---> [Sanity API (Store Order Details)]
  |                 |
  v                 v
[Order Confirmation] <--- [Order Info from Sanity API]
  |
  v
[Shipment Tracking] <--- [Sanity API (Track Shipment)]
```

TECHNICAL TOOLS AND LIBRARIES:

- Frontend:
  - Next.js: Framework for building the application.
  - Tailwind CSS: For responsive design and styling.
  - Redux: For managing global state (cart data, user data).
  - Stripe: For handling payments securely.
  - Clark/Auth: For authentication and session management.
  - Figma: For UI/UX design and prototyping.
- Backend:
  - Sanity: Headless CMS for managing product, user, and order data.
  - Node.js: Backend runtime for API and server-side functionality.

Graceful Error Handling:

If an error occurs, such as a failed payment, product fetch issue, or login failure, the system should display clear, user-friendly error messages. For instance, for payment failures, the user should see messages like "Payment failed, please try again" or "Card declined, please check your details". These messages should be concise and provide actionable next steps, such as "Retry" or "Change Payment Method". This prevents users from feeling frustrated or lost and encourages them to retry the action without leaving the site.

User Retention Strategies:

In case of a critical issue (e.g., API failure or an error in processing an order), users should not be left in a confusing state. Display a gentle, yet clear message like "We are experiencing technical difficulties. Please try again later." Additionally, implementing a retry mechanism allows the user to try the action again seamlessly. Ensuring the user experience is smooth by offering an easy way to recover from errors will reduce abandonment and improve retention. Additionally, always offer a contact support option for users who might need assistance, making them feel supported and valued.

ERROR HANDLING AND USER NOTIFICATIONS

MOCK API LINK:

The mock API for fetching product details is hosted at the following link:

API URL: https://677ed08294bde1c1252da4e6.mockapi.io/Template-0

This mock API provides a set of product data, such as the product name, description, price, and image, which can

MOCK API RESPONSE EXAMPLE:

```json
{
  "id": "1",
  "name": "Product Name",
  "description": "This is a detailed description of the product.",
  "price": 19.99,
  "image": "https://example.com/product-image.jpg"
}
```

PLAN API REQUIREMENTS:

- Endpoint: /product
- Method: GET
- Description : Fetch all product data
- Response: The API will return a JSON response containing product details such as the name, description, price, and image.

{ ”id” : 1 , “name” : “Product” , “price“ : 100 }

THANK YOU