Made by Kulsoom Adnan

Hackathon Day 02

Marketplace Technical Foundation

Furniro

Introduction :

Welcome to this presentation on the "Marketplace Technical Foundation." In this session, I will discuss how to plan and establish the technical foundation for our hackathon project. The key focus areas include defining technical requirements, designing system architecture, planning API requirements, This roadmap is tailored to ensure that we build a scalable and efficient marketplace platform.

1. Technical Requirements

The Technical Requirements are designed to set a strong foundation for the development of your e-commerce marketplace. This phase translates business goals into clear, actionable tasks that ensure both frontend and backend components work harmoniously to deliver a seamless user experience.

Frontend Resources:

1. Next.js

- **Purpose**: A powerful React framework for building fast and scalable web applications.
- **Usage:** Next.js will be used to create the entire frontend of the marketplace, including routing, dynamic rendering, and server-side rendering (SSR).
- **Why Next.js**: It enables fast performance, SEO optimization, and automatic code splitting for better user experience.

2. Tailwind CSS

- **Purpose**: A utility-first CSS framework for designing custom UIs without writing custom CSS.
- **Usage:** Tailwind CSS will be used to style the frontend components of the marketplace, ensuring a responsive and modern design.
- **Why Tailwind CSS**: It allows for rapid styling with a low learning curve and offers complete flexibility to customize designs.

3. ShadCN

- **Purpose:** A UI component library built with Tailwind CSS that offers pre-designed components.
- **Usage**: ShadCN will be used for UI elements like buttons, modals, forms, and input fields, speeding up the design and development process.
- **Why ShadCN**: It provides ready-to-use components that are fully customizable, ensuring consistency in design and speeding up development time.

4. React

- **Purpose**: A JavaScript library for building user interfaces.
- **Usage**: React will be the core library for creating dynamic components and managing the state within your e-commerce marketplace.
- **Why React**: React's component-based architecture makes it easier to create interactive UIs and manage state across the application.

Responsive Design:

The platform is designed with responsiveness in mind, ensuring it looks and functions perfectly across mobile and desktop devices.

- **Mobile First Approach**: The design is optimized for smaller screens, ensuring a smooth user experience on smartphones and tablets.
- **Flexible Layout:** The layout adjusts dynamically to different screen sizes, maintaining a consistent structure without compromising content.
- **Adaptive Navigation:** The navigation bar and menus adapt to different screen sizes, with mobile-friendly collapsible options for easy browsing.

Essential Pages:

- **Home Page:** Engaging hero section, key features, promotions, and categories.
- **Product Listing Page:** Display of product categories, filtering, and sorting options for easy browsing.
- **Product Details Page**: Rich product descriptions, high-quality images, specifications, and pricing details.
- **Cart Page:** Allows users to review, modify, and proceed to checkout with their selected items.
- **Checkout Page**: A streamlined, user-friendly process for entering payment and shipping details.
- **Order Confirmation Page**: A reassuring confirmation message with order summary and tracking information.

Backend (Sanity CMS):

Sanity CMS will act as the **central hub** for all content, enabling you to manage product information, customer data, and order records efficiently.

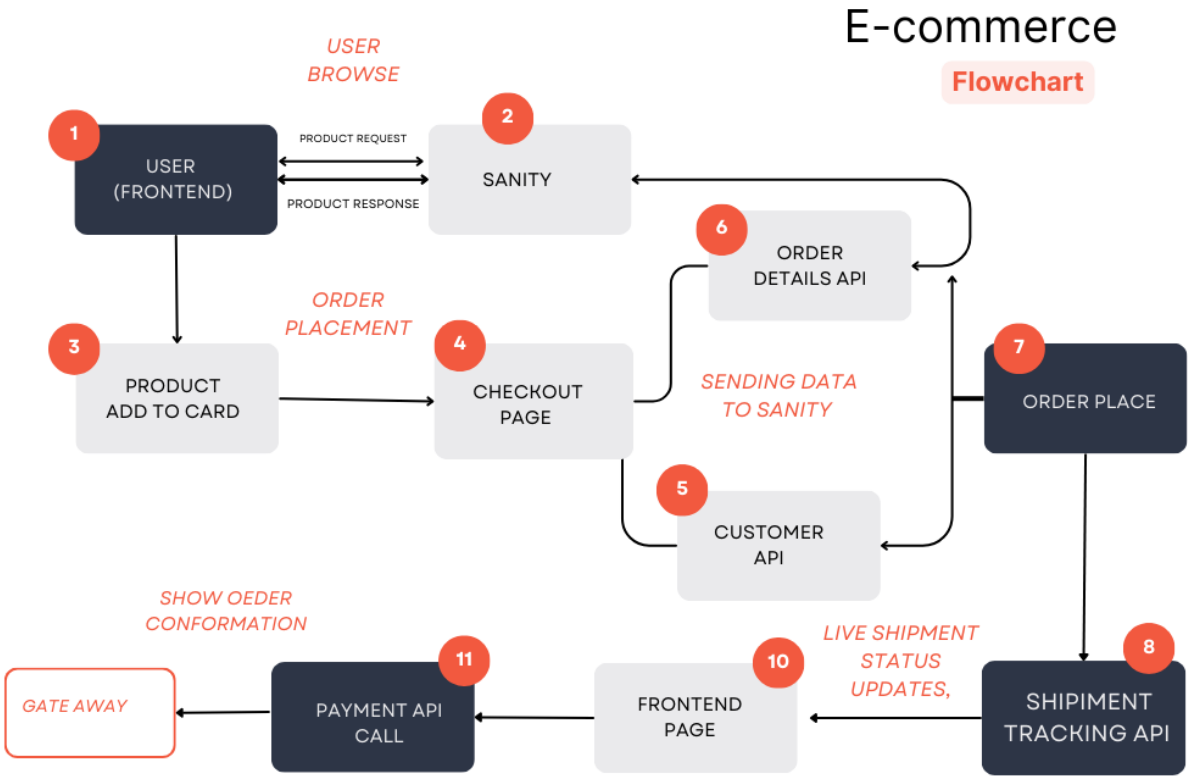
- **Product Data Management**: Store product details such as names, descriptions, prices, and stock information.
- **Customer Data Management**: Maintain user details like names, shipping addresses, and order history.
- **Order Management**: Keep track of every order placed, its status, and relevant customer information.

Usage: Sanity CMS will act as the backend for managing product data, customer details, order records, and more.

Third-Party APIs:

- **Shipment Tracking API**: Integrate a third-party API that provides real-time updates on shipment status and expected delivery times, enhancing customer experience**. ShipEngine** will be used to fetch real-time shipment tracking details, enabling users to track their orders directly from the marketplace.
- **Payment Gateway API:** Secure payment processing via popular gateways like Stripe or PayPal to handle customer transactions efficiently. **stripe** will be used to handle secure payment transactions during the checkout process, ensuring smooth and reliable processing of customer payments.

Design System Architecture



Data Flow Process:

1. **User Browsing**:
    - The user visits the frontend (built with Next.js) and browses the product listings.
    - The frontend requests product data from the **Sanity CMS** via the **Product Data API**, which dynamically displays the information on the site.
2. **Order Placement**:
    - Once the user adds products to their cart and proceeds to checkout, the order details are sent via an API request to **Sanity CMS** for storage and tracking.

3. **Shipment Tracking**:
    - After the order is placed, the **Shipment Tracking API** is called to fetch live shipment status updates, which are displayed to the user.
4. **Payment Processing**:
    - The user provides payment details through the **Payment Gateway API (Stripe)**. Once payment is processed, the user receives a confirmation, and the payment details are securely recorded in **Sanity CMS**.

5. Key Workflows

1. User Registration:

- User signs up and their details are stored in **Sanity CMS**.
- A confirmation message is sent to the user.

2. Product Browsing:

- User browses product categories.
- **Sanity CMS** fetches product data using the **Product Data API**, and the products are displayed dynamically on the frontend.

3. Order Placement:

- User adds products to the cart and proceeds to checkout.
- Order details are recorded in **Sanity CMS**.

4. Shipment Tracking:

- The **Shipment Tracking API** provides order status updates, which are displayed to the user in real-time.

---

6. Conclusion

This architecture provides a clear and scalable way to build a highly functional e-commerce platform using modern technologies like **Next.js**, **Sanity CMS**, and third-party services such as **Stripe** and **ShipEngine**. By integrating these components, we ensure a smooth experience for both the frontend user and the backend system, with seamless data flow and real-time updates.

API Endpoints

Fetch Products

- **Endpoint Name**: /products
- **Method**: GET
- **Description**: Retrieves all available product details from the database.
- **Response Example**:{

"id": 1,

"name": "Sofa Deluxe",

"price": 299.99,

"stock": 10,

"image": "sofa-deluxe.jpg"

}

- **Purpose**: This endpoint allows clients to fetch a list of all products, including essential details like ID, name, price, stock, and image, to display them on the storefront.

---

2. Create Order

- **Endpoint Name**: /orders
- **Method**: POST
- **Description**: Creates a new order in the database with customer information, product details, and payment status.
- **Payload Example**:{

"customerInfo": {

"name": "John Doe",

"email": "john.doe@example.com",

"address": "123 Main Street"

},

"productDetails": [

{ "id": 1, "quantity": 2 },

{ "id": 3, "quantity": 1 }

],

"paymentStatus": "Paid"

}

- **Response Example**:


```
{
  "orderId": "abc123",
  "status": "Order Created",
  "createdAt": "2025-01-17T12:00:00Z"
}
```


- **Purpose**: This endpoint facilitates order creation, capturing customer and product information, while also recording payment status for processing and tracking.

---

3. Track Shipment

- **Endpoint Name**: /shipment
- **Method**: GET
- **Description**: Tracks the status of a shipment using a third-party API.
- **Query Parameter**:
    - shipmentId: The ID of the shipment to track.
- **Response Example**:


```
{
  "shipmentId": "xyz456",
  "orderId": "abc123",
  "status": "In Transit",
  "expectedDeliveryDate": "2025-01-20"
}
```


- **Purpose**: This endpoint allows users to track their orders in real-time, providing updates on shipment status and estimated delivery dates.

4. Customer Management

- **Endpoint Name**: /customers
    - **Method**: POST
    - **Description**: Registers a new customer in the database.
    - **Payload Example**:

```
{
  "name": "Jane Doe",
  "email": "jane.doe@example.com",
  "password": "securePassword123",
  "address": "456 Elm Street"
}
{
  "customerId": "c123",
  "status": "Customer Registered Successfully"
}
```

    - **Purpose**: This endpoint enables customer registration, storing user details for order placement and account management.

Brief Overview

1. **Purpose of Endpoints**:
    - /products: Fetches product catalog details for the storefront.
    - /orders: Handles order placement with customer and product information.
    - /shipment: Tracks order shipment status with live updates.
    - /customers: Manages customer accounts, enabling registration and information retrieval.
2. **Workflow Alignment**:
    - These APIs are designed to facilitate seamless customer interactions in the marketplace, from browsing products to order placement and shipment tracking.

Technical Roadmap

Day 2: Cart Functionality

- **Objective**: Implement the shopping cart functionality.
- **Tasks**:
    - Set up state management for the cart using:
        - **React Context API**, or
        - A library like Zustand or Redux (if needed for complex state).
    - Add the ability to:
        - Add products to the cart.
        - Update quantities in the cart.
        - Remove products from the cart.
    - Display the cart summary (total price and item count).

Day 3: Checkout Process

- **Objective**: Integrate the payment and checkout flow.
- **Tasks**:
    - **Stripe Integration**:
        - Set up a Stripe account and API keys.
        - Implement the /api/checkout route to create Stripe payment sessions.
        - Redirect users to Stripe for payment processing.
    - Update the UI to:
        - Validate the cart before proceeding to checkout.
        - Show payment success or failure status.
    - Save the order data to **Sanity CMS** after a successful payment.

Day 4: Shipment Tracking

- **Objective**: Implement shipment tracking functionality.
- **Tasks**:
    - Integrate **ShipEngine API**:
        - Configure the API for tracking shipments.
        - Implement a /api/shipment route to fetch shipment details.
    - Update the UI to:
        - Display tracking information for completed orders.
        - Allow users to view order and shipment status on a "My Orders" page.

Day 5: Testing and Debugging

- **Objective**: Ensure all functionality works smoothly and the app is error-free.
- **Tasks**:
    - Test the following flows:
        - Product browsing and cart functionality.
        - Checkout and payment processing.
        - Order confirmation and shipment tracking.
    - Handle edge cases:
        - Empty cart checkout.
        - Invalid payment attempts.
        - API response errors.
    - Debug and fix any issues.

Day 6: Optimization

- **Objective**: Optimize performance and ensure the app is responsive.
- **Tasks**:

- **Performance**:
    - Use lazy loading for images and API requests.
    - Optimize API responses for faster loading.
- **Accessibility**:
    - Ensure the app is keyboard-navigable.
    - Add alt text to images and aria labels for buttons.
- **Responsiveness**:
    - Test and refine the UI for various screen sizes.

Day 7: Deployment

- **Objective**: Deploy the application to a live environment.
- **Tasks**:
    - Deploy the app on **Vercel**.
    - Configure environment variables for API keys (Sanity, Stripe, ShipEngine).
    - Perform a final round of testing on the live environment.
    - Share the live URL for feedback or user testing.

Post-Deployment: Optional Enhancements

If time allows, consider adding these enhancements:

1. **Wishlist**: Allow users to save favorite products for later.
2. **Order History**: Implement a user account system for viewing past orders.
3. **Reviews & Ratings**: Enable users to leave reviews for products.