Day 2 MarketPlace Hackathon

1. Define Technical Requirements

1. Frontend Requirements

Goal: Ensure a seamless user experience across devices.

- **User-friendly Interface:** Prioritize usability by incorporating intuitive navigation and appealing visuals to facilitate easy browsing and shopping.

- **Responsive Design:** Use frameworks like Tailwind CSS or Bootstrap to ensure consistent performance on mobile, tablet, and desktop devices.

- **Essential Pages:** Define the structure and functionality for key pages:
  - **Home Page:** Highlight featured products, offers, and categories.
  - **Product Listing Page:** Enable filtering and sorting by categories, price, and ratings.
  - **Product Details Page:** Showcase product descriptions, reviews, images, and availability.
  - **Cart Page:** Provide an editable list of selected items.
  - **Checkout Page:** Offer a secure form for payment and shipping details.
  - **Order Confirmation Page:** Display a summary of the completed transaction.
  - **Shipments Details:** A shipment tracking system enables customers and administrators to monitor order status and delivery.
  - **Real Time Update:** Display shipment status (e.g., "Order Placed," "Shipped," "Out for Delivery," "Delivered").
  - **Tracking Interface:** Include a tracking number and link to the courier's tracking page.

---

2. Backend with Sanity CMS

Goal: Centralize data management for scalability and flexibility.

1. Product Schema

This schema will manage product-related data and includes essential details required to display, filter, and manage products in the marketplace.

- **Fields**:
  - **ProductID**: A unique identifier for each product.

    - **Name & Description**: Product title and detailed description.
    - **Category**: Defines the product category (e.g., Electronics, Apparel).
    - **Price**: The cost of the product.
    - **Stock Quantity**: Tracks inventory levels.
    - **Color & Size Options**: Lists available variations for customers.
    - **Ratings & Reviews**: Enables user feedback and ratings.
    - **Discount**: Represents an optional percentage discount.

2. Customer Schema

This schema captures information about the customers using the platform.

- **Fields**:
    - **CustomerID**: A unique identifier for each customer.
    - **Full Name & Contact Info**: Personal details like name, email, and phone.
    - **Address**: Shipping or billing addresses.
    - **Order History**: A record of past purchases (referencing the Orders schema).

3. Orders Schema

This schema manages details about customer orders.

- **Fields**:
    - **OrderID**: A unique identifier for each order.
    - **CustomerID**: References the customer placing the order.
    - **ProductID(s)**: A many-to-many relationship to track all products in an order.
    - **Order Date**: The date when the order was placed.
    - **Status**: Tracks the current status of the order (e.g., Pending, Shipped, Delivered).
    - **Total Amount**: The total cost of the order.

4. Payments Schema

This schema keeps track of payment details for each order.

- **Fields**:
    - **PaymentID**: A unique identifier for each payment.
    - **OrderID**: Links the payment to a specific order.
    - **Amount Paid**: The total amount paid by the customer.
    - **Payment Method**: The mode of payment (e.g., Credit Card, UPI, Wallet).
    - **Payment Status**: Indicates whether the payment was successful or pending.

5. Shipment Schema

This schema handles shipment logistics for each order.

- **Fields**:
  - **ShipmentID**: A unique identifier for each shipment.
  - **OrderID**: Links the shipment to a specific order.
  - **Courier Service**: The name of the courier company.
  - **Tracking Number**: The unique number to track shipment progress.
  - **Estimated Delivery Date**: Expected delivery time.
  - **Shipment Status**: Tracks progress (e.g., In Transit, Delivered).

Schema Relationships

1. **Orders → Customers**: Many-to-One relationship (Multiple orders can belong to one customer).
2. **Orders → Products**: Many-to-Many relationship (An order can have multiple products).
3. **Orders → Payments**: One-to-One relationship (Each order has one payment).
4. **Orders → Shipments**: One-to-One relationship (Each order has one shipment).

---

3. Third-Party APIs

Goal: Enhance functionality and streamline processes.

1. Integrate APIs for Payment Gateway

- **Features**:
  - Secure payments.
  - Support for multiple payment methods (credit cards, wallets, etc.).
  - Real-time transaction updates.
- **Integration**:
  - Use Stripe SDKs and APIs to manage payment flows.
  - Implement features such as payment intent creation, confirmation, and webhook-based event handling for status updates.

- **JazzCash**:
  - **Features**:
    - Widely used in Pakistan for online and mobile payments.
    - Supports bank transfers, mobile wallets, and direct payments.

        - **Integration**:
            - Use JazzCash API to enable secure transactions.
            - Provide options for mobile wallet and bank account payments.
- **EasyPaisa**:
    - **Features**:
        - Popular in Pakistan for seamless mobile payments.
        - Supports utility payments, online shopping, and P2P transfers.
    - **Integration**:
        - Integrate EasyPaisa API to offer diverse payment options.
        - Ensure compatibility with mobile wallets and online transactions.

2. Integrate APIs for Shipment Tracking

- **Ship Engine:**

- **Features**:
    - Multi-carrier support for shipping.
    - Real-time shipment tracking.
    - Shipping rate comparison for cost optimization.
- **Use Case**:
    - Generate efficient shipment labels and track shipments dynamically.

- **AfterShip**

- **Features**:
    - Real-time shipment tracking updates.
    - Customer notifications for shipment status.
- **Use Case**:
    - Provide live tracking information to customers via notifications or a dashboard.

- **EasyPost**

- **Features**:
    - Shipping label creation.
    - Rate calculation for various carriers.
    - Shipment tracking.
- **Use Case**:
    - Streamline backend processes for logistics and delivery management.

3. Additional APIs

- **Google Maps API**

- **Use Case**:
  - Validate customer addresses during checkout.
  - Map delivery zones to ensure service availability.

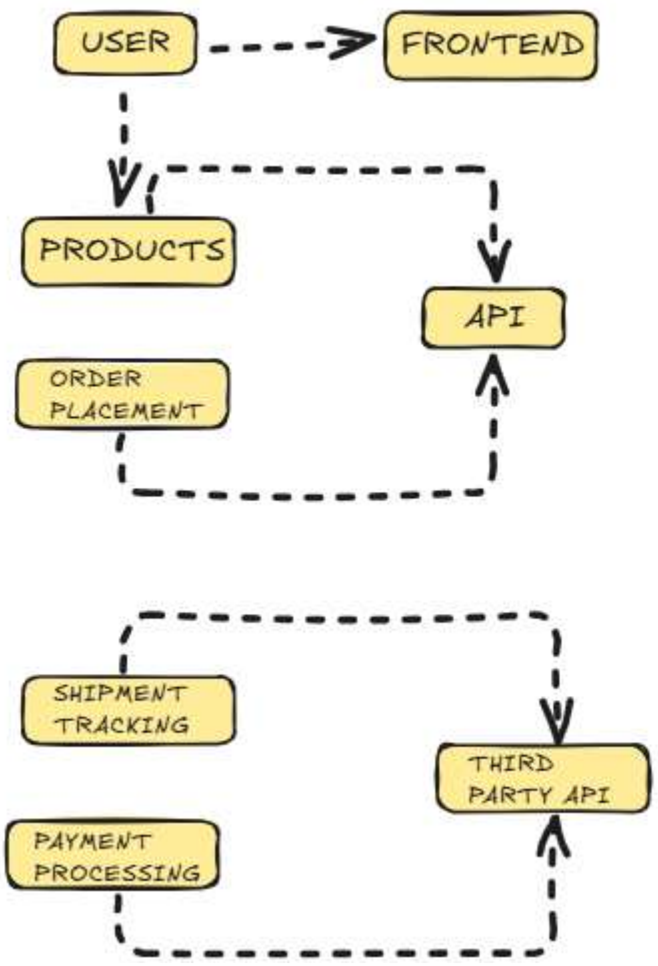
4. Notification APIs (Email/SMS)

- **Use Case**:
  - Send order confirmation messages.
  - Notify customers about delivery status changes using services like Twilio or SendGrid.

5. Ensure API

- **Data Coverage**:
  - Payment APIs should return transaction statuses (e.g., successful, pending) and payment details (e.g., amount, method used).
  - Shipment tracking APIs should provide real-time tracking data, delivery statuses, and estimated arrival times.
  - Address validation APIs (like Google Maps) should confirm that entered addresses are valid and serviceable.
- **Compatibility**:
  - The data structure returned by APIs must align with the frontend's requirements. For example, if the frontend expects a list of items, the API should return data in a compatible format (e.g., JSON).
- **Real-Time Updates**:
  - APIs like payment gateways and shipment trackers should allow real-time event updates, enabling the frontend to reflect changes dynamically (e.g., live shipment tracking, payment confirmation).
- **Ease of Use**:
  - APIs should have clear documentation and endpoints, making it easier to implement features like displaying payment success/failure, tracking shipments, and mapping delivery zones.

2. Design System Architecture



1. User Registration:

- **Step 1:** The user signs up on the website.
- **Step 2:** The registration details are securely stored in the **Sanity CMS** database via an API request.
- **Step 3:** A confirmation email or notification is sent back to the user, indicating successful registration.

2. Product Browsing:

- **Step 1:** The user browses product categories on the frontend of the marketplace.
- **Step 2:** The frontend sends a request to the **Product Data API** to fetch the product listings.
- **Step 3:** The **Product Data API** communicates with **Sanity CMS** to retrieve product information (such as title, description, images, and prices).
- **Step 4:** The product details are dynamically displayed on the website for the user to browse and interact with.

3. Order Placement:

- **Step 1:** The user selects the products they wish to purchase and adds them to the shopping cart.
- **Step 2:** The user proceeds to the checkout page and enters order details (e.g., shipping address, payment information).
- **Step 3:** The order details, including product information and user data, are sent to **Sanity CMS** via an API request for record-keeping.
- **Step 4:** The order is confirmed, and an order summary is sent to the user.

4. Shipment Tracking:

- **Step 1:** Once the order is placed, shipment tracking information is fetched in real-time from a third-party logistics API.
- **Step 2:** The **Shipping API** provides updates such as shipment status, expected delivery time, and tracking ID.
- **Step 3:** The real-time tracking information is displayed on the user's order details page, ensuring the user stays updated on their shipment status.

5. Payment Processing:

- **Step 1:** The user proceeds to make a payment via a secure **Payment Gateway** (e.g., Stripe, EasyPaisa etc.).
- **Step 2:** Payment details are encrypted and securely processed by the payment gateway.
- **Step 3:** After the transaction is successfully processed, a confirmation message is sent back to the user.
- **Step 4:** The payment status, transaction ID, and confirmation are recorded in **Sanity CMS**, ensuring the order and payment are properly linked.

3. PLAN API REQUIMENTS

Endpoint Name	Method	Description	Request Payload	Response Example
`/products`	`GET`	Fetch all available product details.	None	`{ "status": "success", "data": [ { "id": 101, "name": "Apple", "price": 1.99 } ] }`
`/products/{id}`	`GET`	Fetch details of a specific product by ID.	None	`{ "status": "success", "data": { "id": 101, "name": "Apple", "price": 1.99 } }`
`/categories/{id}`	`GET`	Fetch products within a specific category.	None	`{ "status": "success", "data": [ { "id": 101, "name": "Apple" } ] }`
`/cart/add`	`POST`	Add a product to the user's cart.	`{ "product_id": 101, "quantity": 2 }`	`{ "status": "success", "message": "Item added to cart." }`
`/cart`	`GET`	Retrieve the current state of the user's cart.	None	`{ "status": "success", "data": { "items": [ { "id": 101, "name": "Apple" } ] } }`
`/checkout`	`POST`	Process payment and place an order.	`{ "payment_method": "card", "shipping_address": { "address": "123 St" } }`	`{ "status": "success", "message": "Order placed successfully." }`
`/orders`	`POST`	Create a new order with customer and product details.	`{ "customer_info": { ... }, "products": [ ... ] }`	`{ "status": "success", "message": "Order created successfully.", "order_id": 123 }`
`/orders/{id}`	`GET`	Retrieve details of a specific order	None	`{ "status": "success", "data": { "order_id": 123, "status": "Shipped"}}`

Endpoint Name	Method	Description	Request Payload	Response Example
`/shipment`	`GET`	Retrieve real-time shipment tracking updates.	Query Parameter: `shipment_id=67890`	`{ "status": "success", "data": { "shipment_id": 67890, "status": "In Transit" } }`
`/homepage`	`GET`	Fetch homepage content, including featured items.	None	`{ "status": "success", "data": { "featured_products": [ ... ] } }`
`/user/register`	`POST`	Register a new user.	`{ "email": "user@example.com", "password": "1234" }`	`{ "status": "success", "message": "User registered successfully." }`
`/user/login`	`POST`	Authenticate a user and return a token.	`{ "email": "user@example.com", "password": "1234" }`	`{ "status": "success", "token": "abc123" }`
`/user/profile`	`GET`	Retrieve user profile details.	Authorization Header: `Bearer <token>`	`{ "status": "success", "data": { "name": "John Doe", "email": "user@example.com" } }`

4. WRITE TECHNICAL DOCUMENTATION

```
[Frontend Application] --> [Product Data API] --> [Sanity CMS]
                       --> [Order API]  --> [Sanity CMS]
                       --> [Shipment API]  --> [Third-Party Logistics API]
                       --> [Payment Gateway API]  --> [Payment Gateway Service]
```

1. System Architecture Overview:

- **Frontend:**
  - **Framework:**
    - React.js
  - **Responsibilities**:
    - Rendering dynamic user interfaces.
    - Handling client-side state management.
    - Managing client-side routing for seamless user navigation.

- **Backend**:
  - **Framework**:
    - Node.js with Express.js for API creation.
  - **Responsibilities:**
    - Managing business logic.
    - Handling API requests and responses.
    - Processing and validating data.

- **External APIs**:
    - Payment gateways, third-party integrations.

- **Deployment & Hosting:**
    - AWS, Vercel, or Netlify for deployment.

- **CMS:**
    - Sanity.io for managing dynamic content such as categories, product descriptions, and blogs.

2. Key WorkFlow

2.1 User Registration & Authentication

1. **User Signup**
    - Users register by providing their email, password, and profile details.
    - Data is validated and stored in the database.
2. **Login**
    - Users enter their credentials to obtain a JWT token, enabling secure session handling.
3. **Password Recovery**
    - Users reset passwords via a token-based recovery system.

2.2 Product Browsing & Filtering

1. Users view categories fetched from the CMS.
2. Clicking a category triggers the `/categories/{id}/products` API to display relevant products.
3. Users can filter products by attributes such as price, ratings, or availability.

2.3 Cart Management

1. Users add products to their cart via the `/cart/add` endpoint.
2. The cart updates dynamically, storing items in the database or local storage for guest users.
3. The cart is displayed using the `/cart` endpoint.

2.4 Checkout & Payment

1. The user proceeds to checkout, providing payment and shipping details.
2. The `/checkout` endpoint processes the payment and creates an order.

3. Users receive order confirmation via email.

---

3. Category-Specific Instructions

Groceries

- Ensure real-time stock updates using WebSockets or API calls.
- Implement expiry tracking for perishable items.

Fashion

- Enable size and color selection for products.
- Support a "virtual try-on" feature using AR/VR for an enhanced user experience.

Home Essentials

- Use bundle offers to encourage bulk purchases.
- Provide detailed specifications and care instructions.

Health & Wellness

- Include certifications and lab test reports for products.
- Highlight subscriptions for recurring orders (e.g., vitamins).

Electronics

- Showcase detailed product specs with comparison tools.
- Offer extended warranty and service plans.

4. API EndPoint Documentation

- USER

Endpoint	Method	Purpose	Payload Example	Response Example
`/api/users/register`	POST	Register a new user.	`{ "email": "example@example.com", "password": "password123", "profileDetails": {} }`	`201 Created`

`/api/users/login`	POST	Authenticate user and return a JWT token.	`{ "email": "example@example.com", "password": "password123" }`	`{ "token": "jwt_token_string" }`

- ## Categories

Endpoint	Method	Purpose	Payload Example	Response Example
`/api/categories`	GET	Retrieve all categories.	N/A	`[ { "id": 1, "name": "Electronics" }, { "id": 2, "name": "Books" } ]`
`/api/categories/{id}/products`	GET	Retrieve products under a specific category.	N/A	`[ { "id": 101, "name": "Laptop", "price": 999.99 }, { "id": 102, "name": "Phone", "price": 599.99 } ]`

- ## Products

Endpoint	Method	Purpose	Payload Example	Response Example
`/api/products/{id}`	GET	Retrieve detailed information about a product.	N/A	`{ "id": 101, "name": "Laptop", "details": { "brand": "BrandX", "specs": { "RAM": "16GB" } } }`

- ## Cart

Endpoint	Method	Purpose	Payload Example	Response Example
`/api/cart/add`	POST	Add a product to the cart.	`{ "productId": 101, "quantity": 2 }`	`200 OK`
`/api/cart`	GET	Retrieve cart details.	N/A	`[ { "productId": 101, "quantity": 2 }, { "productId": 102, "quantity": 1 } ]`

- ### Orders

Endpoint	Method	Purpose	Payload Example	Response Example
`/api/checkout`	POST	Process payment and create an order.	`{ "paymentDetails": {}, "shippingDetails": {} }`	`201 Created`

5. SANITY SCHEMA EXAMPLE

EXAMPLE:

```
export default {
 name: 'product',
 title: 'Product',
 type: 'document',
 fields: [
  {
   name: 'name',
   title: 'Product Name',
   type: 'string',
   description: 'The name of the product.',
   validation: Rule => Rule.required().min(3).max(100),
  },
  {
   name: 'slug',
   title: 'Slug',
   type: 'slug',
   description: 'URL-friendly identifier for the product.',
  options: {
    source: 'name',
    maxLength: 96,
  },
  validation: Rule => Rule.required(),
},
{
  name: 'image',
  title: 'Product Image',
  type: 'image',
  options: {
    hotspot: true,
  },
  fields: [
    {
      name: 'alt',
      title: 'Alt Text',
      type: 'string',
      description: 'Alternative text for accessibility and SEO.',
    },
  ],
},
{
  name: 'price',
  title: 'Price',
  type: 'number',
  description: 'The price of the product.',
  validation: Rule => Rule.required().min(0),
},
{
  name: 'description',
  title: 'Description',
  type: 'text',
  description: 'Detailed description of the product.',
  validation: Rule => Rule.max(500),
},
{
  name: 'categories',
  title: 'Categories',
  type: 'array',
  of: [{ type: 'reference', to: [{ type: 'category' }] }],
  description: 'Categories this product belongs to.',
},
{
  name: 'stock',
  title: 'Stock',
  type: 'number',
  description: 'The quantity of the product in stock.',
  validation: Rule => Rule.integer().min(0),
},
{
  name: 'sku',
  title: 'SKU',
  type: 'string',
  description: 'Stock Keeping Unit for inventory tracking.',
},
{
  name: 'isFeatured',
      title: 'Featured Product',
      type: 'boolean',
      description: 'Mark as a featured product.',
    },
    {
      name: 'releaseDate',
      title: 'Release Date',
      type: 'datetime',
      description: 'The release date of the product.',
    },
  ],
  preview: {
    select: {
      title: 'name',
      media: 'image',
      subtitle: 'price',
    },
  },
};
```

• Collaborate and Refine

- **Group Discussions:**
  Use tools like Slack or Google Meet for brainstorming and focus on innovative system and API designs.
- **Peer Review:**
  Share plans for feedback and review peers' work to refine designs.
- **Version Control:**
  Track changes and collaborate using GitHub with clear commit messages.
- **Divide and Conquer:**
  Collaborate on frameworks but ensure unique individual submissions.

- **Submission Requirements:**
  Submissions must reflect individual understanding and originality.