Name: Noor-ul-Sehar

Roll No: 00349943

Slot: Friday 9 to 12

Hackathon Day 2:

Planning the Technical Foundation for Fashion E-Commerce Platform

---

Business Focus Recap

On **Day 1**, I laid the groundwork for my **Fashion E-Commerce Platform** by focusing on its foundation. Here's what I achieved:

1. **Defined the Mission:**
    - I identified the key challenges my platform tackles, such as:
        - Limited availability of authentic branded products like Nike, Adidas, and Puma in local stores.
        - High prices and lack of variety in traditional shopping experiences.
    - I outlined how my platform offers a better solution:
        - A **one-stop shop** for authentic branded apparel and footwear.
        - Competitive pricing, fast delivery, and a seamless shopping experience.
2. **Understood the Audience:**
    - I determined who my platform serves:
        - **Primary Audience:**
            - Fashion-conscious individuals who prefer branded clothing and footwear.
            - Sneaker enthusiasts (SNKRS fans) looking for exclusive or limited-edition releases.
            - Parents looking for branded kids' clothing and footwear.
        - **Secondary Audience:**
            - Gift shoppers looking for premium branded products for special occasions.

        - Resellers or collectors of branded sneakers and apparel.
3. **Highlighted What Makes My Platform Unique:**
    - I emphasized my platform's standout features:
        - **Authenticity:** Guaranteed genuine products from trusted brands.
        - **Wide Range:** A vast collection of branded apparel and footwear for all ages and genders.
        - **Competitive Pricing:** Affordable prices with regular discounts and offers.
        - **Exclusive Drops:** Early access to limited-edition SNKRS and other premium releases.
        - **User Experience:** A seamless, mobile-friendly shopping experience with features like wishlists, size guides, and personalized recommendations.
        - **Fast Delivery:** Quick shipping options for urgent needs.
4. **Created a Data Blueprint:**
    - I drafted a simple yet effective **data schema** to map out how key elements like **Products, Customers, Orders, and Payments** interact.

Day 2:

Transitioning to Technical Planning for Fashion E-Commerce Platform

1. Define Technical Requirements

Technical Requirements outline the technical specifications for building and maintaining a system, including frontend design, backend logic, and database management. They ensure alignment, reduce risks, and support scalable solutions.

My platform's technical requirements include a secure backend, responsive frontend, and API integration, which are given below:

Frontend Requirements:

- **Intuitive and User-Friendly Interface:**
    - Easy product exploration and seamless shopping experience.

- **Fully Responsive Design:**
  - Ensures seamless performance on mobile and desktop.

- **Key Pages:**
  - Homepage, Product Listings, Product Details, Cart, Checkout, and Order Confirmation.

Sanity CMS for Backend:

Sanity CMS will manage **Product Details, Customer Information, and Order Records**.

1. Manage Product Details

- **Purpose:**
  - Stores all information about the products available for sale.
- **Example Schema:**

```
export default {
  name: 'product',
  type: 'document',
  fields: [
    { name: 'name', type: 'string', title: 'Product Name' },
    { name: 'brand', type: 'reference', to: [{ type: 'brand' }], title: 'Brand' },
    { name: 'price', type: 'number', title: 'Price' },
    { name: 'stock', type: 'number', title: 'Stock' },
    { name: 'description', type: 'text', title: 'Description' },
    { name: 'image', type: 'image', title: 'Product Image' }
  ]
};
```

- **How it’s used:**
    - When a user searches for products, the frontend fetches data from this schema.

- **Example Data:**

```
{
  "name": "Nike Air Max",
  "brand": { "name": "Nike" },
  "price": 150,
  "stock": 50,
  "description": "Comfortable and stylish sneakers.",
  "image": "https://example.com/nike-air-max.jpg"
}
```

2. Customer Information

- **Purpose:**
    - Stores details about users who purchase products.
- **Example Schema:**

```
export default {
  name: 'customer',
  type: 'document',
  fields: [
    { name: 'name', type: 'string', title: 'Full Name' },
```

```
    { name: 'email', type: 'string', title: 'Email' },
    { name: 'phone', type: 'string', title: 'Phone Number' },
    { name: 'address', type: 'string', title: 'Address' }
  ]
};
```

- **How it’s used:**
  When a user signs up or places an order, their details are saved here.
- **Example Data**

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "phone": "+123456789",
  "address": "123 Main St, City"
  }
```

3. Track Order Records

- **Purpose:**
  - Stores all purchase transactions.
- **Example Schema:**

```
export default {
  name: 'order',
  type: 'document',
  fields: [
```

```
    { name: 'product', type: 'reference', to: [{ type: 'product' }], title: 'Product' },
    { name: 'customer', type: 'reference', to: [{ type: 'customer' }], title: 'Customer' },
    { name: 'quantity', type: 'number', title: 'Quantity' },
    { name: 'totalPrice', type: 'number', title: 'Total Price' },
    { name: 'status', type: 'string', title: 'Order Status' }
  ]
};
```

- **How it's used:**
  - When a user places an order, the details are saved here.
- **Example Data:**

```
{
  "product": { "name": "Nike Air Max" },
  "customer": { "name": "John Doe" },
  "quantity": 1,
  "totalPrice": 150,
  "status": "Confirmed"
}
```

Third-Party APIs

To enable **product browsing, payments, and shipment tracking**, I will integrate third-party APIs. Here's how they'll work:

A. User Sign-Up

- A new user creates an account, and their information is stored securely in Sanity CMS. They receive a confirmation message to verify their account.

B. Exploring Products

- Users browse through the list of available products. The frontend fetches product details (like name, price, and availability) from the **Product Data API** (powered by Sanity CMS). Products are displayed with filters for easy searching.

C. Placing an Order

- A user selects a product, chooses quantity, and provides their details. The frontend sends this information to Sanity CMS via the **Order API**, and the user gets a confirmation.

D. Making Real Payments

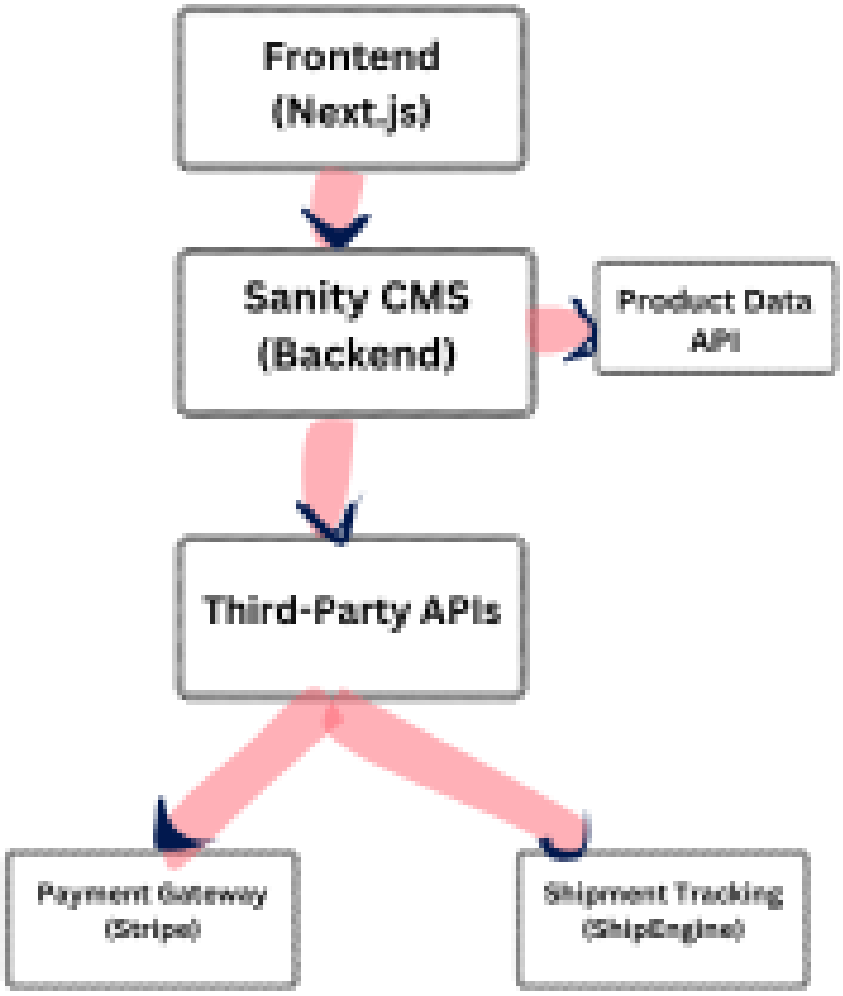
- The user enters their payment details (e.g., card number, amount). The frontend sends this data to the **Payment Gateway API** (e.g., Stripe) for processing. Once the payment is successful, the user receives a confirmation.

E. Shipment Tracking

- The frontend fetches real-time updates from the **Shipment Tracking API** (e.g., ShipEngine). The user can see the order's current status and estimated delivery time.

2. System Architecture Design

Here's the **system architecture** for my **Fashion E-Commerce Platform**, illustrating how the key components connect and interact:



How Components Connect

A. Frontend (Next.js) → Sanity CMS:

- The frontend pulls product and customer data from Sanity CMS.

B. Sanity CMS → Product Data API:

- Sanity CMS serves product details to the frontend through the Product Data API.

C. Sanity CMS → Third-Party API:

- Sanity CMS sends order details to third-party APIs for shipment tracking and payment processing.

D. Third-Party API → Shipment Tracking API:

- The Third-Party API retrieves real-time shipment updates for the user.

E. Third-Party API → Payment Gateway:

- The Third-Party API securely processes payments via the Payment Gateway.

3. Plan API Requirements

Based on my data schema, here are the **API endpoints** needed for my **Fashion E-Commerce Platform**:

A. Fetch All Products

- **Endpoint Name:** /products
- **Method:** GET
- **Description:** Fetch all available products from Sanity CMS.
- **Response Example:**

```
[
  {
```

```
    "id": 1,
    "name": "Nike Air Max",
    "brand": "Nike",
    "price": 150,
    "stock": 50,
    "image": "https://example.com/nike-air-max.jpg"
  },
  {
    "id": 2,
    "name": "Adidas Ultraboost",
    "brand": "Adidas",
    "price": 180,
    "stock": 30,
    "image": "https://example.com/adidas-ultraboost.jpg"
  }
]
```


B. Create a New Order

- **Endpoint Name:** /orders
- **Method:** POST
- **Description:** Create a new order in Sanity CMS.
- **Payload:**


```
{
  "customerId": 123,
```

```
  "productId": 1,
  "quantity": 2,
  "totalPrice": 300,
  "paymentStatus": "Paid"
}
```


- **Response Example:**


```
  {
  "orderId": 456,
  "status": "Confirmed"
  }
```


C. Track Shipment Status

- **Endpoint Name:** `/shipment`
- **Method:** `GET`
- **Description:** Track the status of an order via a third-party API.
- **Response Example:**


```
{
  "shipmentId": 789,
  "orderId": 456,
  "status": "In Transit",
  "expectedDelivery": "2025-01-25"
}
```


D. Fetch Product Details

- **Endpoint Name:** `/product-details`
- **Method:** `GET`
- **Description:** Fetch detailed information about a specific product.
- **Response Example:**


```
{
  "id": 1,
  "name": "Nike Air Max",
  "brand": "Nike",
```

```
  "price": 150,
  "stock": 50,
  "description": "Comfortable and stylish sneakers.",
  "image": "https://example.com/nike-air-max.jpg"
}
```


E. Process Payment

- **Endpoint Name:** `/payments`
- **Method:** `POST`
- **Description:** Process payment for an order.
- **Payload:**


```
{
  "orderId": 456,
  "amount": 300,
  "paymentMethod": "Credit Card"
}
```


- **Response Example:**


```
{
  "paymentId": 789,
  "status": "Paid"
}
```


Why These APIs Are Important

- **Fetch All Products:** Allows users to browse available products.
- **Create a New Order:** Enables users to place orders seamlessly.
- **Track Shipment Status:** Provides real-time updates on order delivery.
- **Fetch Product Details:** Gives users detailed information about a product.
- **Process Payment:** Securely handles payment transactions.

Endpoint Name	Method	Description	Example Request/Response
/products	GET	Fetch all available products.	Request: GET /products Response: [{ "id": 1, "name": "Nike Air Max", ... }]
/orders	GET	Create a new order.	Request: POST /orders Response: { "orderId": 123, "status": "Confirmed" }
/shipment	GET	Track the status of an order.	Request: GET /shipment?orderId=123 Response: { "status": "In Transit", ... }
/product-details	GET	Fetch detailed information about a product.	Request: GET /product-details?id=1 Response: { "id": 1, "name": "Nike Air Max", ... }
/payments	POST	Process payment for an order.	Request: POST /payments Response: { "paymentId": 789, "status": "Paid" }

Conclusion

Over the past two days, I've successfully laid the foundation for my **Fashion E-Commerce Platform**, focusing on both **business goals** and **technical planning**. From

defining the mission and understanding the audience to creating a data schema and planning the technical architecture, every step has been carefully designed to ensure the platform is **user-friendly, scalable, and aligned with real-world needs**. With a clear roadmap in place, I'm excited to move forward and bring this vision to life!