DAY 2:

PLANING THE TECHNICAL FOUNDATION

Technical Aspects of the Project

1. Frontend

- **Technology Used:** Next.js
    - Next.js ensures a fast and dynamic user experience with server-side rendering (SSR) and static site generation (SSG).
    - Implements features like dynamic routing for seamless navigation between product pages, the cart, and checkout.
- **Styling Framework:** Tailwind CSS
    - Tailwind enables a modern, utility-first approach to styling, ensuring pixel-perfect and responsive design across all devices, from desktop to mobile.
    - Built-in responsiveness ensures seamless rendering across different breakpoints (e.g., 1920px, 1440px, 1152px, 768px, and 320px).

2. Backend

- **Technology Used:** Sanity CMS
    - **Product Management:** Sanity serves as the central database for managing product data, including categories, prices, images, and descriptions.
    - **Order Management:** Checkout and order details, including customer data and purchased items, are stored in Sanity for record-keeping.
    - **Shipment Tracking:** Updates for shipment statuses are saved in Sanity and fetched dynamically for user visibility.

3. External APIs

- **ShipEngine:**
  - Used for handling shipping functionalities such as:
    - Address validation to ensure accurate delivery locations.
    - Generating shipping labels and tracking numbers.
    - Displaying real-time shipment statuses to users.

- **Stripe:**
  - Integrated as the payment gateway to handle:
    - Secure payment processing for a seamless checkout experience.
    - Webhooks for real-time payment status updates.
    - Support for multiple payment methods like credit cards and digital wallets.

System Architecture for E-Commerce Platform

1. Marketplace Frontend (Next.js)

- The user interface where customers browse products, add items to the cart, and complete purchases.
- Communicates with backend services to fetch product data, manage carts, and handle orders.

2. Sanity CMS

- Acts as the central content hub for managing product listings, categories, and order data.
- Receives and stores order details after checkout.
- Sends product information to the frontend upon request.

3. Third-Party APIs

- **ShipEngine**: Provides real-time shipment tracking.
  - Sanity CMS sends order shipping details to ShipEngine.
  - The frontend fetches and displays tracking data from ShipEngine.
- **Stripe**: Handles secure payment processing.
  - Frontend sends payment requests to Stripe.

- Stripe confirms payment and sends the status to Sanity CMS and the frontend.

4. Data Flow

1. **Product Browsing**:
    a. Frontend requests product data from Sanity CMS.
    b. Sanity CMS responds with product listings and details.
2. **Order Placement**:
    a. User adds products to the cart and proceeds to checkout.
    b. Order details are sent to Sanity CMS for storage.
3. **Payment Processing**:
    a. Stripe processes payments securely and sends confirmation to the frontend and Sanity CMS.
4. **Shipment Tracking**:
    a. ShipEngine generates tracking data and provides updates to the frontend via API.

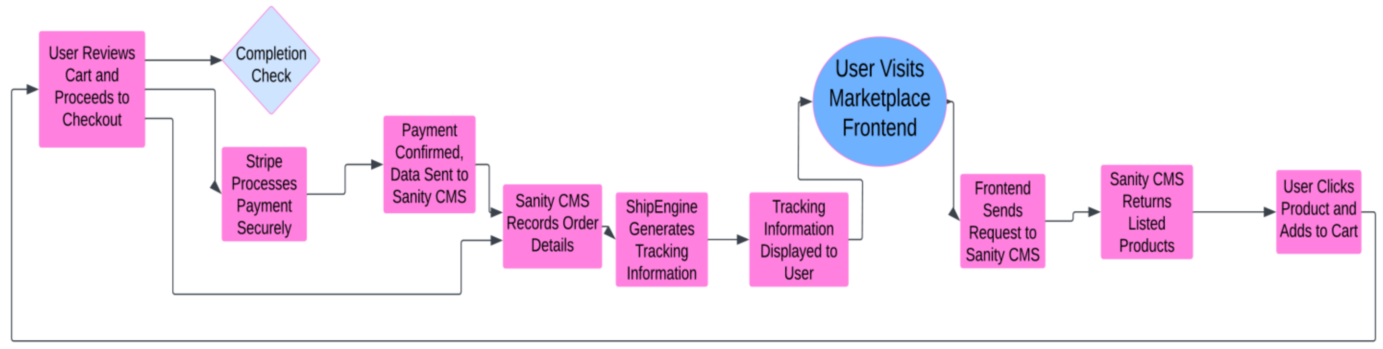
5. Key Technologies

- **Frontend**: Next.js
- **CMS**: Sanity CMS
- **Payment Gateway**: Stripe
- **Shipment Tracking**: ShipEngine
- **State Management**: React Context or Redux

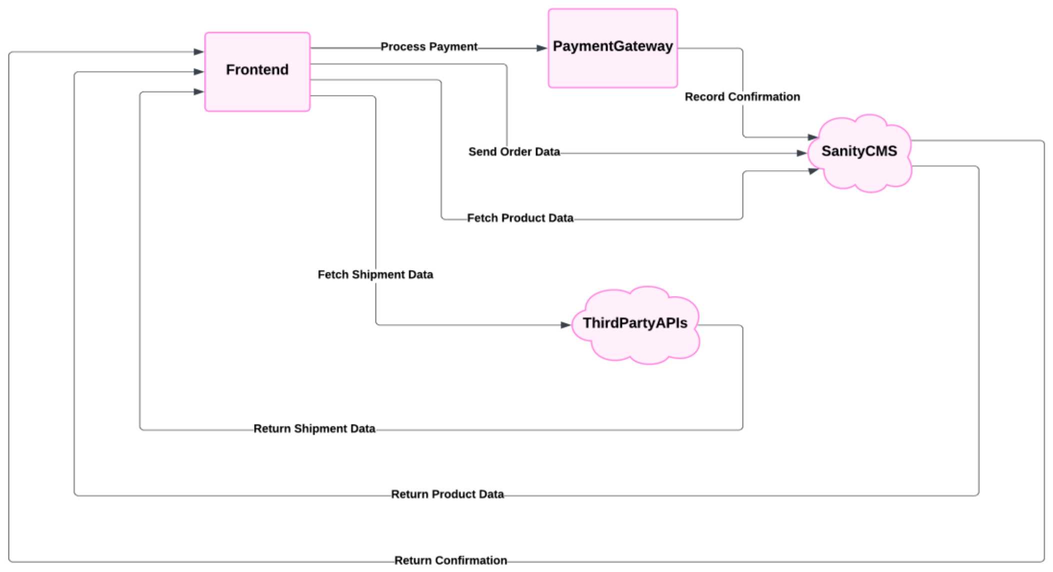
6. Security and Scalability

- Use secure API communication (e.g., HTTPS, token-based authentication).
- Design for scalability to handle increased traffic and data loads.

Data Flow Chart:


User Reviews Cart and Proceeds to Checkout
Completion Check
Stripe Processes Payment Securely
Payment Confirmed, Data Sent to Sanity CMS
Sanity CMS Records Order Details
ShipEngine Generates Tracking Information
Tracking Information Displayed to User
User Visits Marketplace Frontend
Frontend Sends Request to Sanity CMS
Sanity CMS Returns Listed Products
User Clicks Product and Adds to Cart

Architecture:



API Endpoints Chart

Endpoint	Method	Description
`/api/products`	GET	Fetch all product listings from Sanity CMS.
`/api/products/:id`	GET	Fetch details of a specific product.
`/api/cart`	POST	Add an item to the user's cart.
`/api/cart`	GET	Retrieve items in the user's cart.
`/api/cart/:id`	DELETE	Remove an item from the cart.
`/api/orders`	POST	Create a new order in Sanity CMS.
`/api/orders/:id`	GET	Fetch details of a specific order.
`/api/shipments`	POST	Send shipment data to ShipEngine.
`/api/shipments/:id`	GET	Retrieve tracking info for a shipment.
`/api/payment`	POST	Process payment through Stripe.
`/api/payment/confirm`	GET	Confirm the payment status.

Data Schema Design for E-Commerce Platform

Entities and Relationships

1. **Product**
    a. **Fields**:
        i. `id`: Unique identifier (UUID).
        ii. name: Name of the product.
        iii. `description`: Detailed description.
        iv. `price`: Price of the product.
        v. `category`: Reference to `Category` entity.
        vi. `images`: Array of image URLs.
        vii. `stock`: Quantity available.
    b. **Relationships**:
        i. Belongs to one `Category`.
2. **Category**

   a. **Fields**:
      i. `id`: Unique identifier (UUID).
      ii. `name`: Name of the category.
      iii. `description`: Description of the category.
   b. **Relationships**:
      i. Has many `Products`.

3. **User**
   a. **Fields**:
      i. `id`: Unique identifier (UUID).
      ii. `name`: Full name of the user.
      iii. `email`: Email address (unique).
      iv. `password`: Encrypted password.
      v. `address`: Array of addresses (for shipping and billing).
   b. **Relationships**:
      i. Has many `Orders`.
4. **Cart**
   a. **Fields**:
      i. `id`: Unique identifier (UUID).
      ii. `userId`: Reference to `User` entity.
      iii. `items`: Array of objects containing:
         1. `productId`: Reference to `Product` entity.
         2. `quantity`: Quantity of the product in the cart.
   b. **Relationships**:
      i. Belongs to one `User`.
5. **Order**
   a. **Fields**:
      i. `id`: Unique identifier (UUID).
      ii. `userId`: Reference to `User` entity.
      iii. `items`: Array of objects containing:
         1. `productId`: Reference to `Product` entity.
         2. `quantity`: Quantity of the product ordered.
      iv. `totalAmount`: Total amount for the order.
      v. `status`: Current status (e.g., pending, shipped, delivered).
      vi. `createdAt`: Timestamp of order creation.
   b. **Relationships**:
      i. Belongs to one `User`.
6. **Shipment**
   a. **Fields**:

        i. `id`: Unique identifier (UUID).
        ii. `orderId`: Reference to `Order` entity.
        iii. `trackingNumber`: Tracking ID from ShipEngine.
        iv. `carrier`: Shipping carrier name.
        v. `status`: Current shipment status.
    b. **Relationships**:
        i. Belongs to one `Order`.

7. **Payment**
    a. **Fields**:
        i. `id`: Unique identifier (UUID).
        ii. `orderId`: Reference to `Order` entity.
        iii. `paymentMethod`: Payment method (e.g., credit card, PayPal).
        iv. `amount`: Amount paid.
        v. `status`: Payment status (e.g., success, failed).
        vi. `transactionId`: Identifier from Stripe.
    b. **Relationships**:
        i. Belongs to one `Order`.

Entity Relationship Summary

- A `User` has many `Orders` and one `Cart`.
- An `Order` has many `Products` and one `Shipment`.
- A `Product` belongs to one `Category`.
- A `Shipment` and `Payment` are tied to one `Order`.