DAY 2 PLANNING THE TECHNICAL FOUNDATION

GENERAL E_COMMERCE

Created By NADIA
nadiaawan400@gmail.com

DAY 2 PLANNING THE TECHNICAL FOUNDATION

Recap of Day 1: Business Focus

Primary Purpose:
Designed a versatile platform to offer diverse products with convenience, competitive pricing, and reliable delivery tailored to modern consumer needs.

Problem Solved:
Simplified fragmented shopping experiences and reduced delivery delays with
seamless navigation, swift fulfillment, and a user-eccentric design.

Target Audience:
Focused on time-conscious shoppers, families seeking convenience, and professionals looking for reliable delivery options.

Products and Services:
1. Categories: Furniture:
2. Added value: Subscription deliveries, exclusive discounts, and hassle free returns.

E-Commerce Data Schema:
1. **Core Entities:** Products, Customers, Orders, Payments, Shipment, and Delivery Zones.
2. **Key Relationships:** Integrated models for real-time tracking, customer order history, and dynamic delivery charges.

Marketplace Features:
1. Dynamic filters for products.
2. Real-time order tracking and flexible payment options.
3. AI-powered personalized recommendations and loyalty programs.Day 2 Activities: Transitioning to Technical Planning
1. Define Technical Requirements

This document outlines the technical planning phase for the e-commerce marketplace,
focusing on three key areas: fronted requirements, back-end integration using Sanity
CMS, and third-party API integrations.

1. Frontend Requirements

The front-end will deliver a seamless, user-friendly experience with the following
pages and features:

Essential Pages:

Homepage:

1. Highlights: Featured products, promotional banners, category shortcuts.
2. Call-to-Actions (CTA's): "Shop Now," "Browse Categories," "View Deals."

shop Section:

1. **Category Pages:**

Allow users to browse products by categories (e.g., Groceries, Electronics, Fashion).

2. **Product Listing Page:**

Displays products with:
Filters: Price, category, ratings, availability.
Sorting Options: Best Sellers, Price (Low to High), New Arrivals.

3. **Product Details Page:**

Key Features:
Product Title, Images, Description.
Price, Stock Availability, Discounts.
Color Family Selector and Size Options.Ratings and Reviews.
Questions about the Product (FAQ Section).
Add-to-Cart and Add-to-Wishlist functionality.
Recommendations for similar products.

cart Page:

1. Displays selected products with quantity and price breakdown.
2. Options to update quantities or remove items.

checkout Page:

1. Captures shipping details, payment method, and order summary.
2. Features for applying discount codes and selecting delivery preferences.

order Confirmation Page:

1. Displays order details, estimated delivery time, and shipment tracking.

About and Contact Pages:

1. Business details and a contact form for customer inquiries.

Technical Stack:

 Frameworks: React.js and Next.js for building dynamic and SEO-friendly
pages.
 Component Library: shadcn/ui for customizable, reusable components.
 Styling: Tailwind CSS for responsive and visually appealing design.

2. Backend with Sanity CMS

Sanity CMS will serve as the backend to manage dynamic data like products,
customers, and orders.

sanity Schema Design:

Products Schema:

o Fields:
 ProductID: Primary Key.
 Name, Description, Category, Price, Stock Quantity.
 Color Options, Size Options.
 Ratings, Reviews, and FAQs.
 Discount (if applicable).**Customer Schema:**

o Fields:
 CustomerID: Primary Key.
 Full Name, Email, Phone Number, Address.
 Order History, Loyalty Points (optional).

orders Schema:

O Fields:
 OrderID: Primary Key.
 CustomerID: Foreign Key.
 ProductID(s): Many-to-Many relationship.
 Order Date, Status (e.g., Pending, Shipped, Delivered).
 Total Amount.

payments Schema:

O Fields:
 PaymentID: Primary Key.

 OrderID: Foreign Key.
 Amount Paid, Payment Method (e.g., Credit Card, UPI, Wallet).
 Payment Status (e.g., Successful, Pending).

shipment Schema:

o Fields:
 ShipmentID: Primary Key.
 OrderID: Foreign Key.
 Courier Service, Tracking Number.
 Estimated Delivery Date, Shipment Status.

Implementation Steps:

1. Use **Sanity Studio** to design and test schemas.
2. Fetch and manipulate data on the frontend using **GROQ queries**.
3. Optimize schemas for scalability and future expansion.**3**

. Third-Party API Integrations

To provide critical marketplace functionality, integrate the following APIs:

Payment Gateways:

Stripe:

- Features: Secure payments, support for multiple payment methods, and

real-time transaction updates.

Integration: Use Stripe SDKs and APIs for seamless integration.

.

Shipment Tracking APIs:

ShipEngine:

o Features: Multi-carrier support, real-time tracking, and shipping rate comparison.
o Use Case: Efficient shipment label generation and tracking.

AfterShip:

o Features: Real-time shipment tracking and customer notifications.
o Use Case: Provide live tracking updates for customers.

EasyPost:

o Features: API for shipping label creation, rate calculation, and tracking.
o Use Case: Streamline the backend for logistics and delivery management.

Additional APIs:

1. **Google Maps API:**

o Use Case: Address validation and delivery zone mapping.

2. **Notification APIs (Email/SMS):**

o Use Case: Send order confirmations and delivery status updates.**Middleware Implementation:**

 Use **Node.js** and **Express.js** to handle API requests and process server-side
logic.

 Secure API endpoints using **JWT (JSON Web Tokens)**.

Development Pipeline

1. **Frontend Development:**

o Build responsive pages using React.js, Next.js, and Tailwind CSS.

2. **Backend Development:**

o Implement schemas in Sanity CMS and connect the frontend via APIs.

3. **API Integration:**

o Integrate payment and shipment APIs to ensure seamless functionality.

4. **Testing:**

o Conduct thorough testing for functionality, responsiveness, and security.

5. **Deployment:**

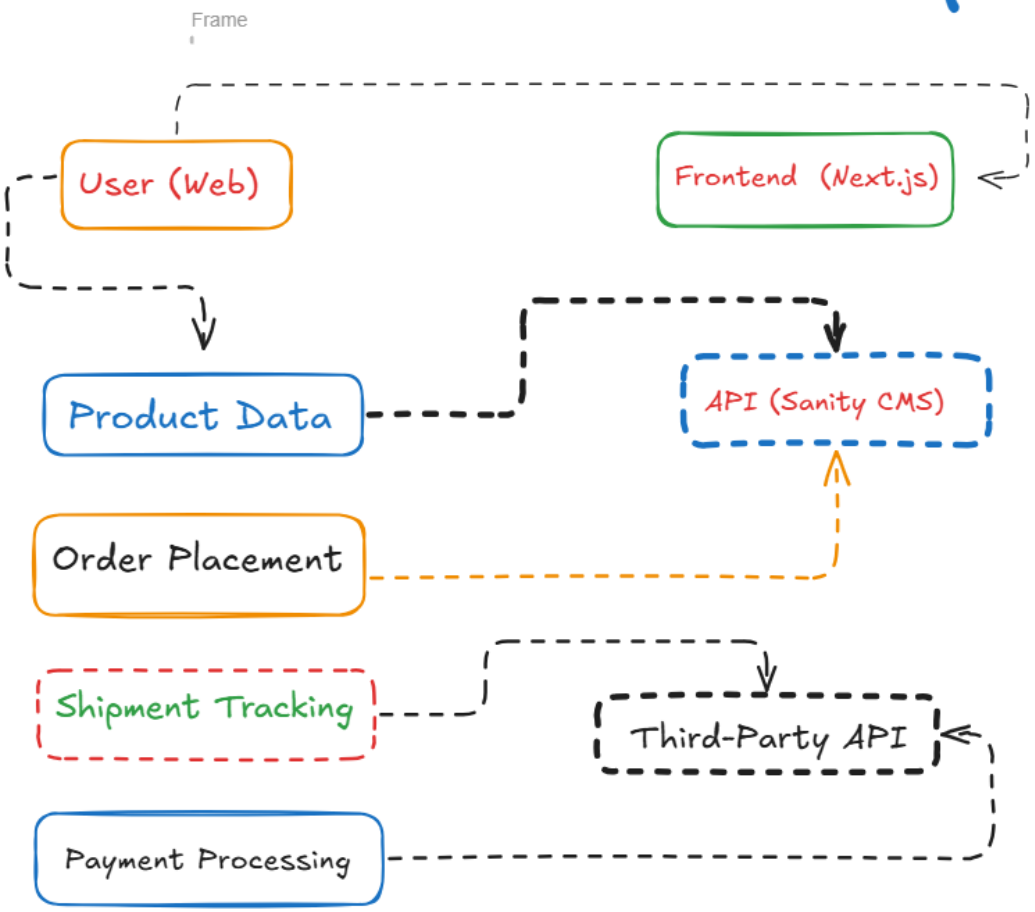
o Host the platform on **Vercel** (frontend) and **Heroku** or **AWS Lambda** (backend).

This detailed plan aligns technical implementation with business goals, ensuring a
scalable, user-centric, and efficient e-commerce platform.

2. Design System Architecture:

To visualize how these components interact, consider the following high-level architecture.

Data Flow Example



1.User Browsing:
A user visits the marketplace frontend to browse products.
The frontend requests product listings from the Product Data API.
2.Product Display:
The Product Data API fetches data from Sanity CMS.
Product details are displayed dynamically on the site. **3.Order Placement:**
When the user places an order, the order details are sent to Sanity CMS via an API
request.
The order is recorded in Sanity CMS.
4.Shipment Tracking:
Shipment tracking information is fetched through a Third-Party API.
Real-time tracking updates are displayed to the user.
5.Payment Processing:
Payment details are securely processed through the Payment Gateway.
A confirmation is sent back to the user and recorded in Sanity CMS.

By following this detailed plan, you can create a robust and scalable marketplace that
meets both business and technical requirements.

3. Plan API Requirements

Endpoint	HTTP Method	Description	Parameters	Response Example
/api/products	GET	Retrieve a list of products.	category, priceRange, sort, page	[{"id":1,"name":"Chair","price":50,"stock":20}, ...]
/api/products/:id	GET	Retrieve details of a specific product.	id (Path)	{"id":1,"name":"Chair","price":50,"stock":20}
/api/cart	GET	Retrieve the items in the user's cart.	Authorization Token	[{"productId":1,"quantity":2,"price":100}, ...]
/api/cart	POST	Add an item to the cart.	productId, quantity	{"message":"Item added to cart","cartId":123}
/api/cart/:id	DELETE	Remove an item from the cart.	id (Path)	{"message":"Item removed from cart"}
/api/orders	POST	Place an order.	Authorization Token, cartId	{"orderId":456,"status":"Processing"}
/api/orders	GET	Retrieve a list of user orders.	Authorization Token	[{"orderId":456,"status":"Processing"}, ...]
/api/orders/:id	GET	Retrieve details of a specific order.	id (Path)	{"orderId":456,"status":"Shipped", ...}
/api/users/register	POST	Register a new user.	name, email, password	{"message":"Registration successful"}
/api/users/login	POST	Authenticate a user and generate a token.	email, password	{"token":"xyz123", "userId":789}
/api/users/profile	GET	Retrieve user profile information.	Authorization Token	{"userId":789,"name":"John Doe","email":"john@example.com"}

Endpoint	HTTP Method	Description	Parameters	Response Example
/api/users/profile	PUT	Update user profile information.	Authorization Token, name, address	{"message":"Profile updated successfully"}
/api/categories	GET	Retrieve a list of product categories.	None	[{"id":1,"name":"Furniture"}, {"id":2,"name":"Decor"}]
/api/reviews/:productId	GET	Retrieve reviews for a specific product.	productId (Path)	[{"user":"Alice","rating":4,"comment":"Great!"}, ...]
/api/reviews/:productId	POST	Add a review for a specific product.	Authorization Token, rating, comment	{"message":"Review added successfully"}

.

4. Write Technical Documentation

Technical Documentation for eCommerce System:

This documentation provides a comprehensive guide to the eCommerce system's
architecture, workflows, API endpoints, and Sanity CMS schema examples. It aims to
assist developers, project managers, and stakeholders in understanding and
implementing the platform effectively.**1. System Architecture Overview**

The eCommerce platform is built with a modular architecture to ensure scalability,
maintainability, and performance.

Frontend:

o **Framework**: React.js / Next.js for fast and responsive UI.

o **Styling**: TailwindCSS or Material-UI for component-based styling.

o **State Management**: Redux or Context API for seamless state handling.

Backend:

o **Framework**: Node.js with Express.js for API creation.

o **Database**: MongoDB for a scalable, NoSQL solution to store products, users, and orders.

o **Authentication**: JSON Web Tokens (JWT) for secure user authentication and authorization.

CMS:

o **Sanity.io**: For managing dynamic content such as categories, product descriptions, and blogs.

Deployment:

o **Hosting**: AWS, Vercel, or Netlify for deployment.

o **CI/CD**: GitHub Actions or Jenkins for automated deployment pipelines.

2. Key Workflows

2.1 User Registration & Authentication

1. **User Signup**: Users register by providing their email, password, and profile details. Data is stored in the database after validation.
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 Endpoints**

Authentication

1. /api/users/register – Register a new user.
2. /api/users/login – Authenticate user and return a JWT token.

Categories

1. /api/categories – Retrieve all categories.

2. /api/categories/{id}/products – Retrieve products under a specific category.

Products

1. /api/products/{id} – Retrieve detailed information about a product.

Cart

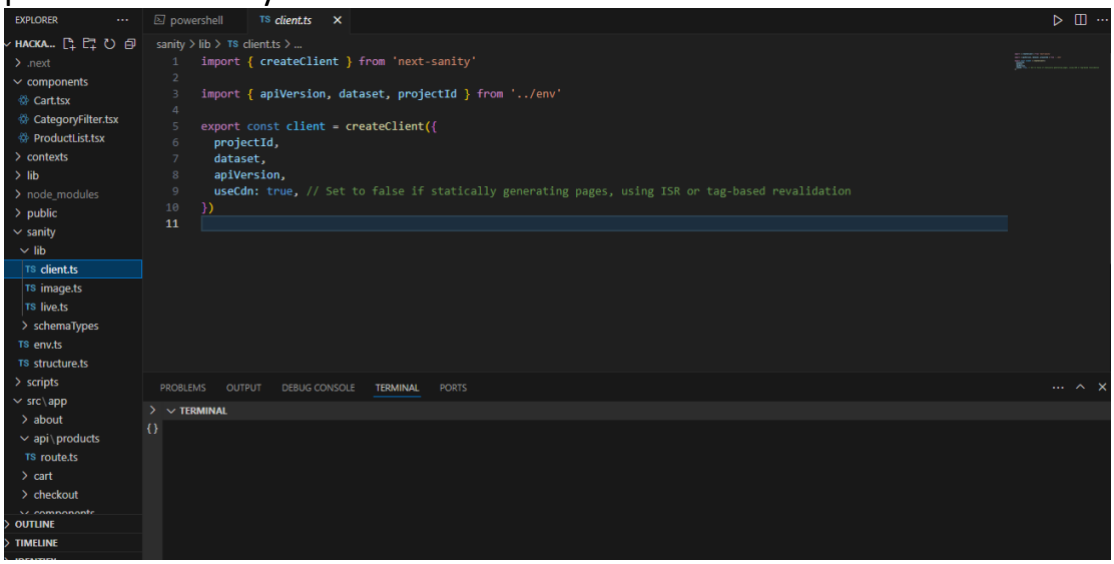
1. /api/cart/add – Add a product to the cart.
2. /api/cart – Retrieve cart details.

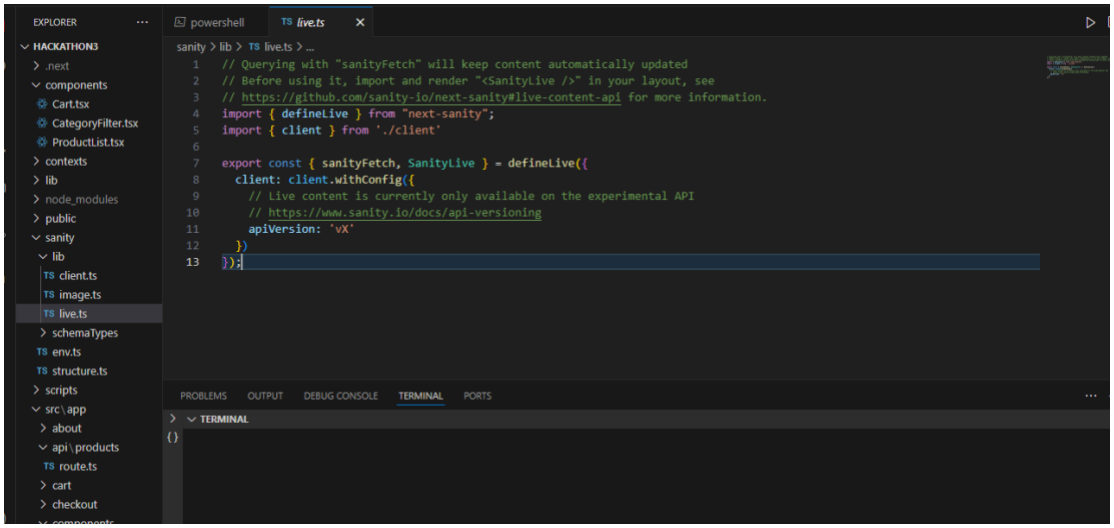
Orders

1. /api/checkout – Process payment and create an order.

5. Sanity Schema Example

Here's an example schema for managing product categories and products in Sanity.io:





5. Collaborate and Refine

1. Feedback Integration

Bro, I love how you brought in the idea of collecting feedback from stakeholders and end-users—it's super user-focused, which is key.

Suggestion: Let's get specific here. Maybe outline how we're collecting feedback, like through surveys, regular check-ins, or testing sessions. Also, defining when we'll do this (e.g., end of every sprint) can make it more structured.

Example:

"Let's set up biweekly feedback surveys for end-users and hold stakeholder review meetings at the end of every sprint to ensure continuous improvement."

---

2. Code Reviews

You nailed it by emphasizing peer reviews. Code quality is everything, bhai.

Suggestion: What if we make this smoother by using a standardized checklist or tool for code reviews? Like GitHub pull requests with a checklist for readability, security, and performance? We could also rotate review leads so everyone learns from each other.

Example:

"Let's make code reviews consistent by using a pull request template on GitHub with checks for readability, edge cases, and adherence to standards."

---

3. Iterative Testing

Solid point on testing multiple layers—unit, integration, UI. Respect for mentioning all the key areas.

Suggestion: What if we automate as much as possible? We could use Jest for unit tests, Postman/Newman for API tests, and Cypress for UI. This way, we save time and avoid manual errors. What do you think?

Example:

"Automating tests will help big time. Jest for units, Postman for APIs, and Cypress for UI testing should give us robust coverage."

---

4. Documentation Updates

Bro, you're spot on about keeping the docs updated. It's the kind of thing we all tend to forget, but it's so crucial.

Suggestion: Let's make it someone's responsibility to own this, or maybe tie it into sprint rituals. A changelog or version history could also make it easier to track updates.