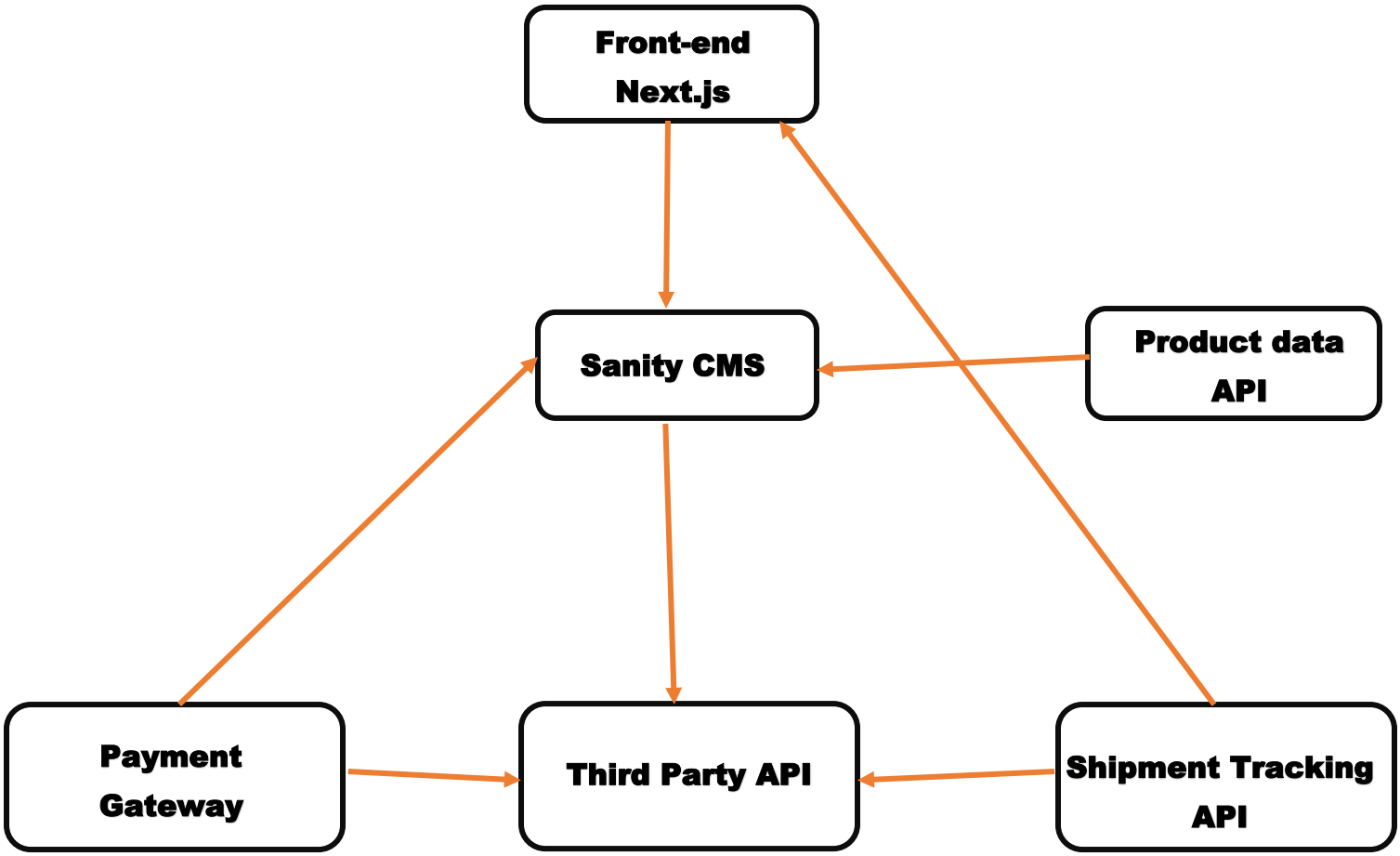
DAY 2 PLANNING THE TECHNICAL FOUNDATION

System Architecture



System Architecture

Here's the high-level system architecture diagram illustrating how the components interact:

1. **Frontend (Next.js):** The user interface for browsing products
2. **Sanity CMS:** Acts as the backend, managing data like product listings and user information.
3. **Product Data API:** Fetches product information from Sanity CMS for dynamic display on the frontend.
4. **Third-Party API:** Handles integrations like shipment tracking and payment processing.
5. **Shipment Tracking API:** Fetches real-time order delivery status.
6. **Payment Gateway:** Processes payments securely and sends payment confirmations back to Sanity CMS.

Key Workflow Steps:

1. **Product Browsing:**
   The frontend requests product listings via the Product Data API connected to Sanity CMS.
2. **Order Placement:**
   The user places an order; details are sent to Sanity CMS via an API request.
3. **Shipment Tracking:**
   Order shipment data is fetched through the Third-Party API and displayed on the frontend.

Technologies

Frontend

- **Next.js**: For building server-rendered, dynamic UIs.
- **Tailwind CSS**: For responsive and beautiful designs.
- **Shadcn/UI**: For customizable UI components.

Backend

- **Sanity CMS**: To manage and structure content effectively.
- **Clerk**: For user authentication and management.

APIs

- **ShipEngine API**: For shipment tracking and delivery management.
- **Stripe API**: For secure and seamless payment processing.

Tools

- **GitHub**: For version control and collaboration.
- **Postman**: To test and document APIs.
- **Vercel**: For fast and reliable deployment.

API Endpoints

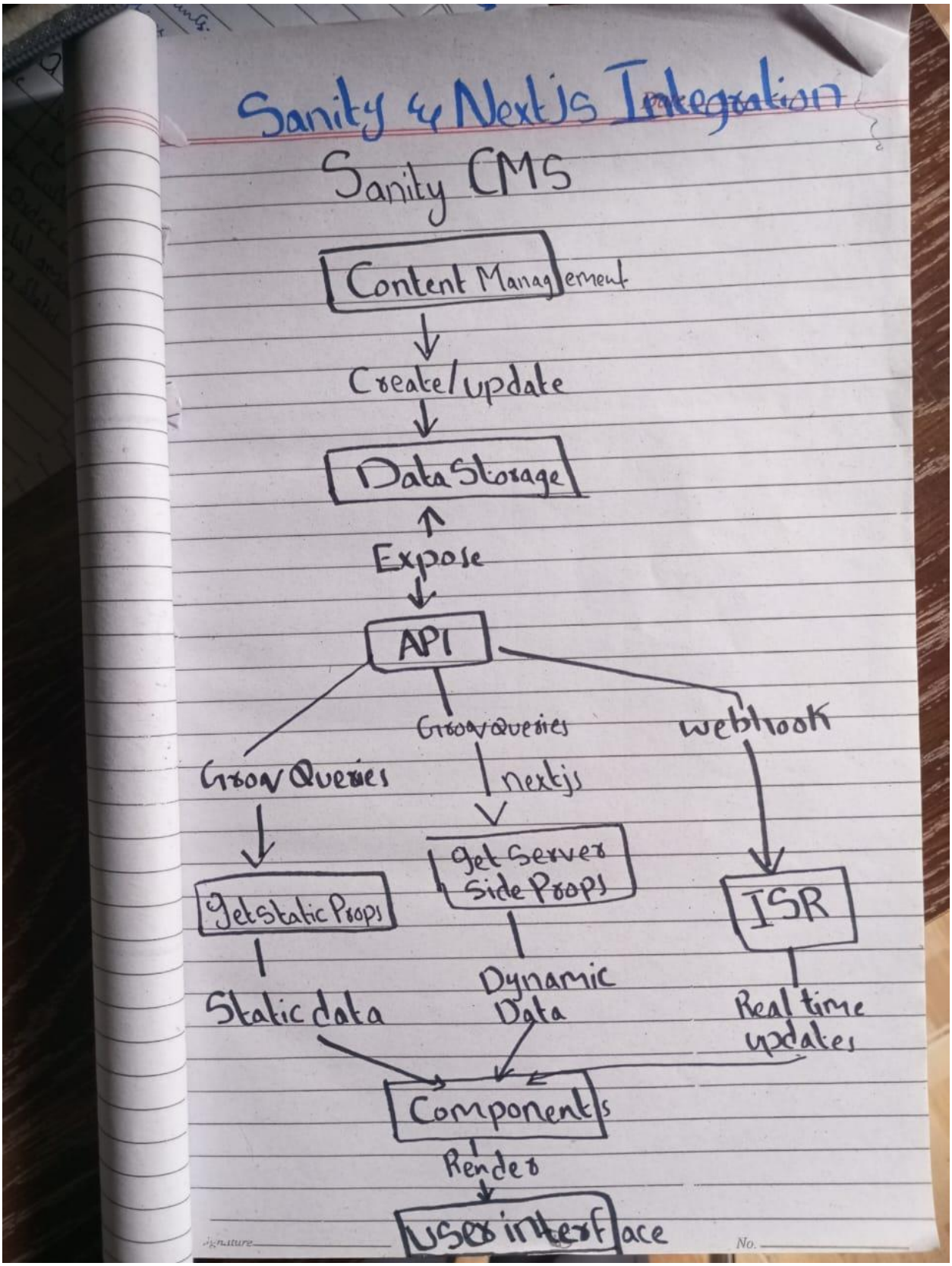
Here are the main API endpoints we'll be working with:

Endpoint	Method	What it does
`/api/create-order`	POST	Creates a new order when someone buys something
`/api/orders`	GET	Fetches all orders (admin stuff)
`/api/shipengine/create-label`	POST	Makes a shipping label for orders
`/api/shipengine/get-rates`	GET	Checks shipping costs
`/api/shipengine/track-shipment`	GET	Tracks where a shipment is
`/api/track-orders`	GET	Lets users see all of their orders
`/api/send/confirmation-email`	POST	Sends an email to confirm an order
`/api/reviews/[productId]`	POST	Adds a review for a product
`/api/reviews/[productId]`	GET	Fetches reviews for a product

Sanity and Next JS Interaction

Sanity CMS is integrated into the Next.js application to provide dynamic and flexible content management. The interaction works as follows:

Sanity & Next.js Integration



1. **Data Management in Sanity:** All the e-commerce data (e.g., products, orders, customers) is stored and managed in Sanity's content studio.
2. **Fetching Data:** Next.js fetches data from Sanity using GROQ queries via Sanity's API endpoints. These queries retrieve structured content, ensuring high flexibility.
3. **Server-side Rendering (SSR):** For dynamic pages like product detail pages, order details page.
4. **Static Site Generation (SSG):** Pages like home and category pages are pre-rendered at build time.
5. **Real-time Updates:** Sanity's webhooks notify the application of changes, enabling immediate updates without manual rebuilds.
6. **Rendering Components:** The fetched data is passed to React components for rendering dynamic and interactive user interfaces.

This seamless integration ensures a robust and scalable e-commerce platform.

Schemas

We're using Sanity CMS, so our schemas are pretty straightforward. Here's a sneak peek at our product schema:

Product

Field	Type	Description
`id`	String	Unique identifier for the product.
`name`	String	Name of the product.
`price`	Number	Current selling price.
`stock`	Number	Quantity available in stock.
`description`	String	Detailed description of the product.
`images`	Array[String]	List of image URLs for the product.
`category`	Reference	Associated category of the product.
`createdAt`	Date	Product creation date.
`slug`	String	URL-friendly identifier for the product.
`rating`	Number	Average customer rating.
`originalPrice`	Number	Original price before any discounts.
`colors`	Array[String]	Available colors for the product.
`sizes`	Array[String]	Available sizes for the product.
`tags`	Array[String]	Searchable tags associated with the product.
`isNewArrival`	Boolean	Flag indicating if the product is a new arrival.
`isTopSelling`	Boolean	Flag indicating if the product is a top-seller.
`productDetails`	Object	Additional details about the product.
`faqs`	Array[Object]	Frequently asked questions about the product.

Customer

Field	Type	Description
`customerId`	String	Unique identifier for the customer.
`name`	String	Customer's full name.
`email`	String	Email address of the customer.
`phone`	String	Contact phone number.
`address`	String	Residential address.
`city`	String	City of residence.
`state`	String	State of residence.
`zipCode`	String	ZIP or postal code.

Order

Field	Type	Description
`orderId`	String	Unique identifier for the order.
`customer`	Reference	Customer who placed the order.
`items`	Array[Object]	List of items in the order.
`totalAmount`	Number	Total cost of the order.
`status`	String	Current order status.
`shipping`	Object	Shipping details.
`createdAt`	Date	Order creation date.
`updatedAt`	Date	Last update date for the order.

Shipment

Field	Type	Description
`shipmentId`	String	Unique identifier for the shipment.
`order`	Reference	Associated order.
`carrier`	String	Shipping carrier name.
`status`	String	Current status of the shipment.
`trackingId`	String	Tracking ID for the shipment.
`estimatedDeliveryDate`	Date	Expected delivery date.
`actualDeliveryDate`	Date	Actual delivery date.
`shippingLabel`	String	URL of the shipping label.
`createdAt`	Date	Shipment creation date.
`updatedAt`	Date	Last update date for the shipment.

Category

Field	Type	Description
`name`	String	Name of the category.
`slug`	String	URL-friendly identifier.

Coupon

Field	Type	Description
`code`	String	Unique coupon code.
`discountType`	String	Type of discount (e.g., percentage, flat).
`discountValue`	Number	Value of the discount.
`minPurchase`	Number	Minimum purchase amount required.
`expiryDate`	Date	Expiration date of the coupon.
`isActive`	Boolean	Indicates if the coupon is active.

Conclusion

Day 2 of the hackathon focused on laying the technical foundation for our project, transitioning from business ideation to actionable technical planning. By defining technical requirements, designing a robust system architecture, planning API integrations, and drafting technical documentation, we established a clear roadmap for implementation. Collaborative discussions allowed the team to refine ideas, ensuring alignment with industry best practices and project goals.

This day's efforts have set the stage for a seamless development process in the coming days, with a well-structured plan to guide execution. As we move forward, our focus will shift to coding, testing, and delivering a functional and scalable solution. By adhering to the submission guidelines and maintaining a collaborative spirit, we are confident in achieving a successful project outcome.