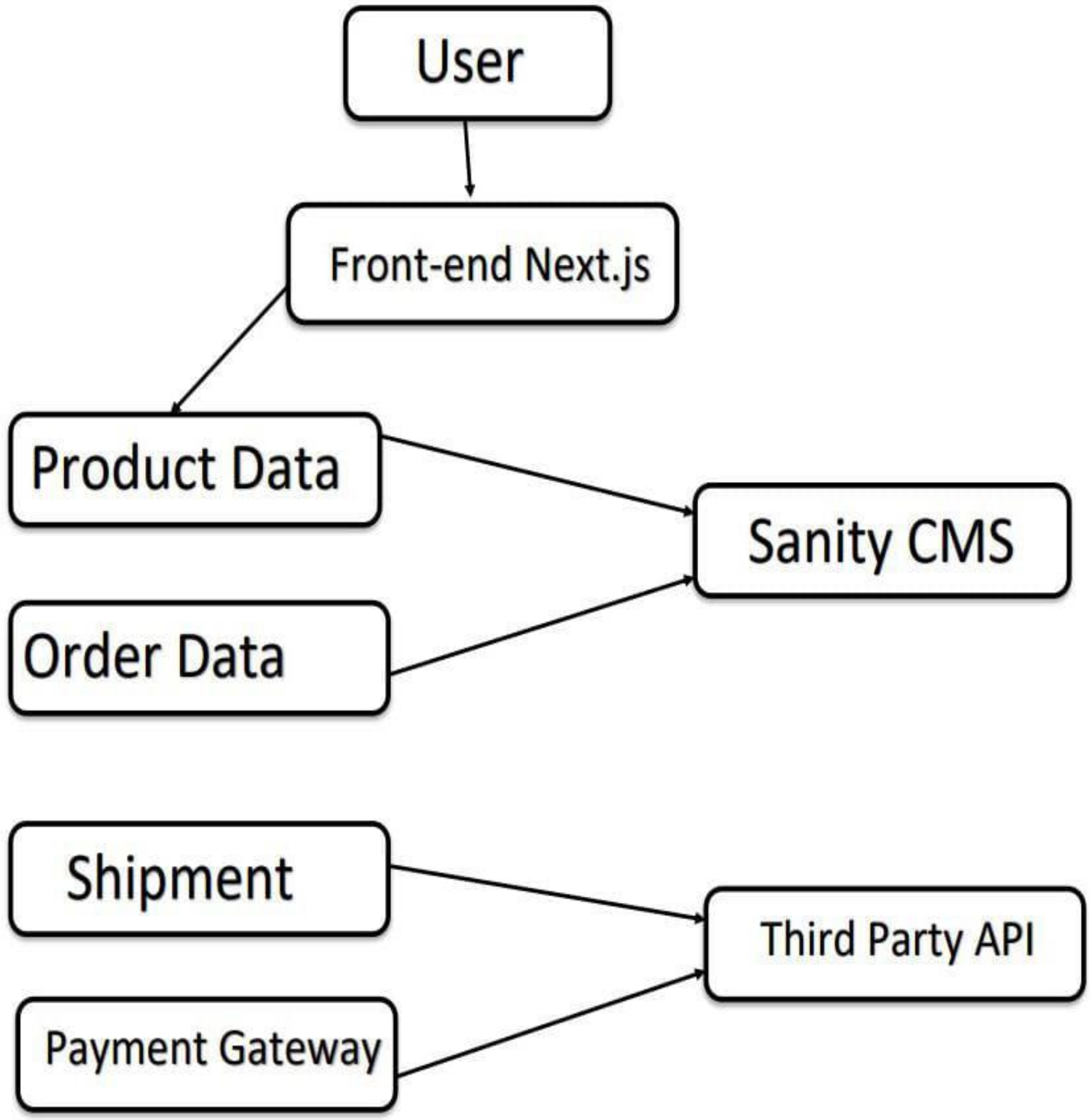
DAY 2 PLANNING THE TECHNICAL FOUNDATION

System Architecture



System Architecture

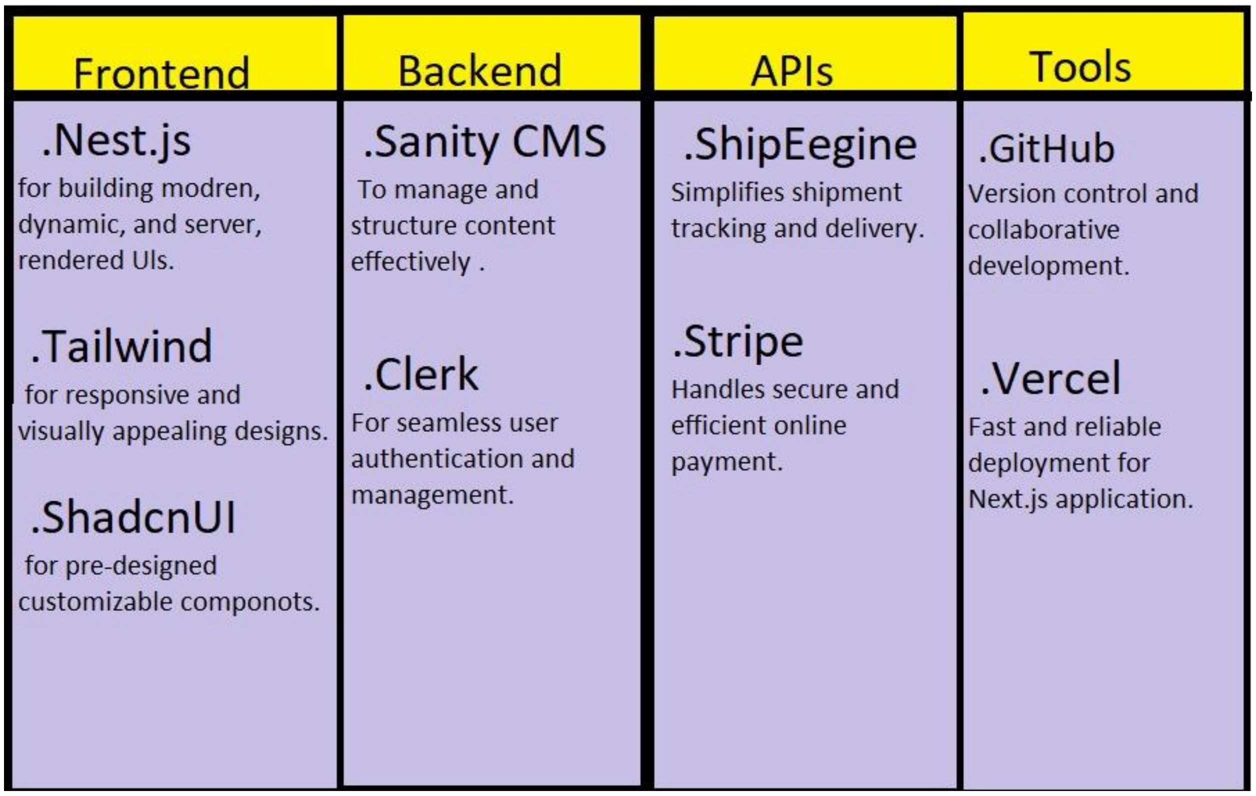
Here's the high-level system architecture diagram illustrating how the components interact:

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

Key Workflow Steps:

1. **Menu Browsing:**
   - The frontend requests menu items and promotions via the Product Data API connected to Sanity CMS.
2. **Order Placement:**
   - Users place an order; details like selected food items, delivery address, and payment status are sent to Sanity CMS via an API request.
3. **Real-Time Delivery Tracking:**
   - Delivery updates are fetched through the Third-Party API and displayed on the frontend for users.

Technologies



Frontend	Backend	APIs	Tools
.Nest.js for building modren, dynamic, and server, rendered UIs.	**.Sanity CMS** To manage and structure content effectively .	**.ShipEegine** Simplifies shipment tracking and delivery.	**.GitHub** Version control and collaborative development.
.Tailwind for responsive and visually appealing designs.	**.Clerk** For seamless user authentication and management.	**.Stripe** Handles secure and efficient online payment.	**.Vercel** Fast and reliable deployment for Next.js application.
.ShadcnUI for pre-designed customizable componots.			

Frontend

- **Next.js**: For building server-rendered, dynamic UIs for food browsing and order tracking.
- **Tailwind CSS**: For responsive and beautiful designs.
- **Shadcn/UI**: For customizable UI components, including food cards and order summaries.

Backend

- **Sanity CMS**: To manage menu items, restaurant data, and user orders 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

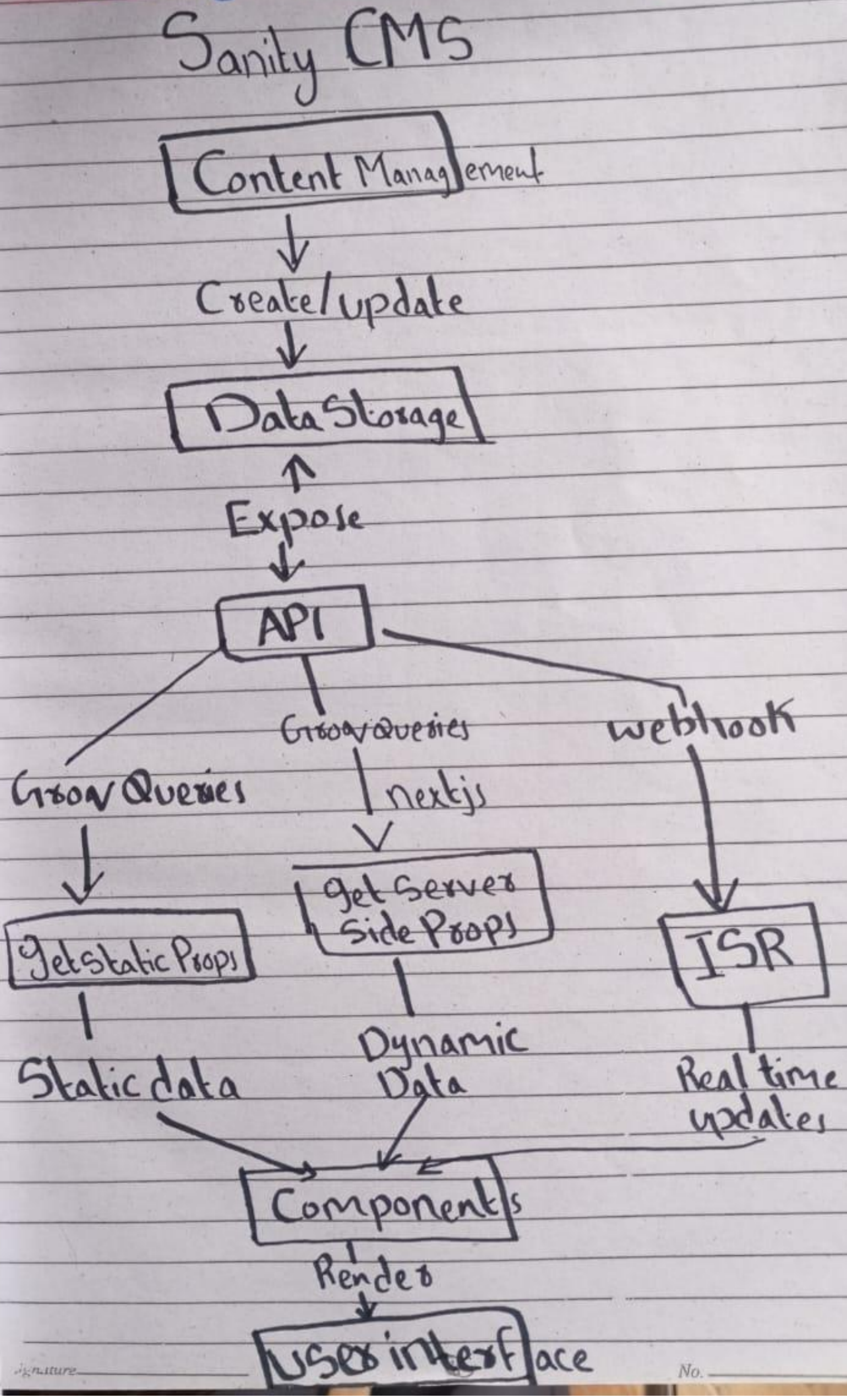
Here are the main API endpoints we'll be working with:

Endpoint	Method	Description
`/api/create-order`	POST	Creates a new order when a user places an order.
`/api/orders`	GET	Fetches all orders (admin purposes).
`/api/shipengine/create-label`	POST	Generates a delivery label for the order.
`/api/shipengine/get-rates`	GET	Calculates delivery costs.
`/api/shipengine/track-shipment`	GET	Tracks delivery status in real-time.
`/api/track-orders`	GET	Lets users track their current orders.
`/api/send/confirmation-email`	POST	Sends a confirmation email for the order.
`/api/reviews/[restaurantId]`	POST	Adds a review for a restaurant or food item.
`/api/reviews/[restaurantId]`	GET	Fetches reviews for a restaurant or food item.

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



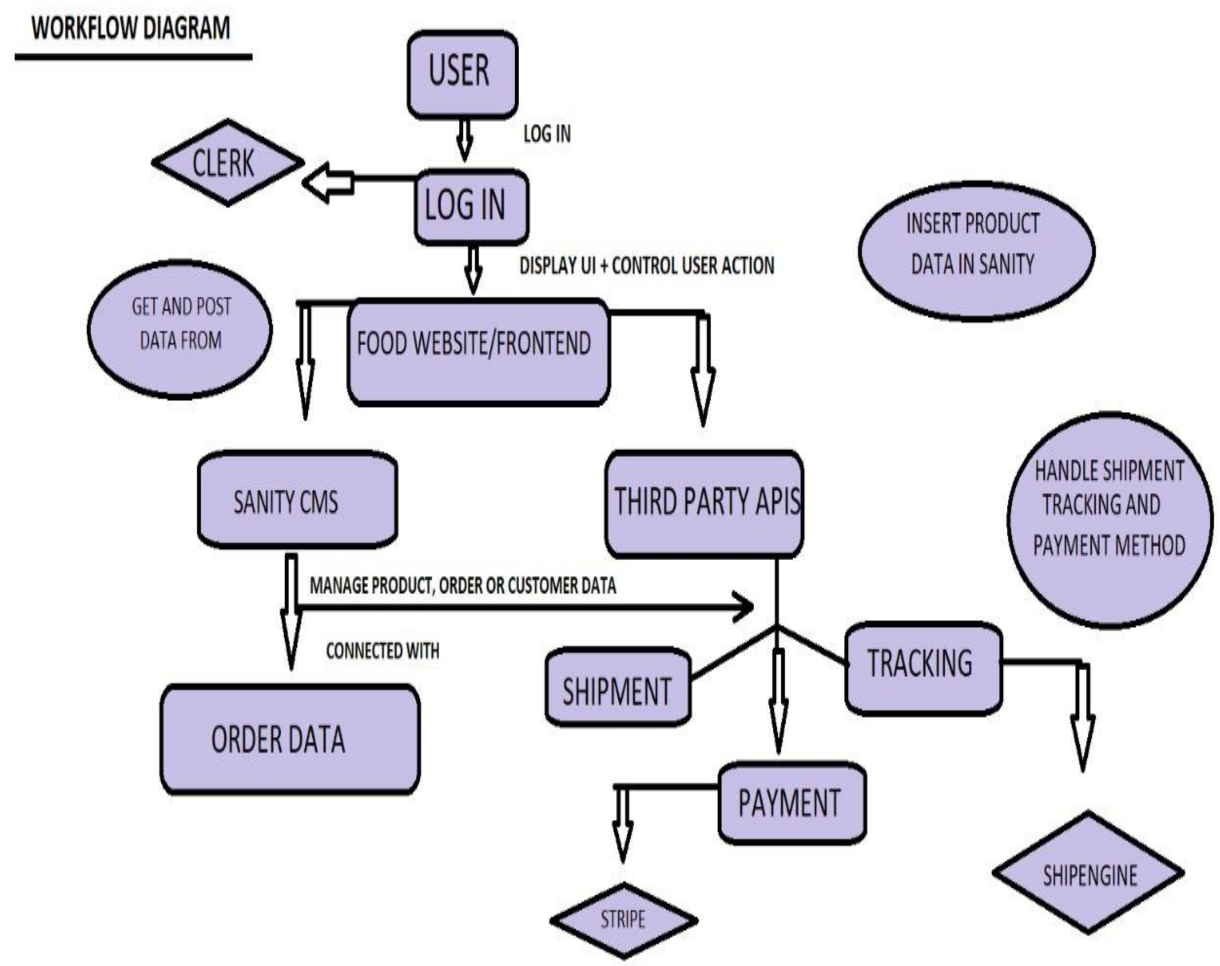
1. **Data Management in Sanity:** All the e-commerce data (e.g., products, orders, customers) is stored and managed in Sanity's content studio.
2. **Fetching Data:** Next.js fetches data from Sanity using GROQ queries via Sanity's API endpoints. These queries retrieve structured content, ensuring high 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.

Quick Commerce-Specific Features:

- **Delivery Time Estimates:**
  - Integration with delivery APIs to display accurate ETAs for food orders.
- **Location-Based Menus:**
  - Serve menus based on the user's location using geofencing.
- **Real-Time Inventory Updates:**
  - Automatically update menu availability based on stock levels.
- **Push Notifications:**
  - Notify users about order updates, promotions, and delivery milestones.

Work Flow



Schemas

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

Products

Field	Type	Description
`id`	String	Unique ID for the food item.
`name`	String	Name of the food item (e.g., "Burger").
`price`	Number	Selling price of the food item.
`description`	String	Short description (e.g., ingredients).
`image`	String	URL of the food item's image.
`category`	String	Category of the food item (e.g., "Snacks").
`isAvailable`	Boolean	Is the food item available for order?
`rating`	Number	Average customer rating (optional).
`tags`	Array[String]	Keywords for searching (e.g., "spicy", "vegan").

Customer

Field	Type	Description
`id`	String	Unique ID for the customer.
`name`	String	Full name of the customer.
`email`	String	Email address of the customer.
`phone`	String	Phone number of the customer.
`address`	Object	Customer's delivery address.
`password`	String	Encrypted password for account login.
`createdAt`	Date	Date the customer account was created.

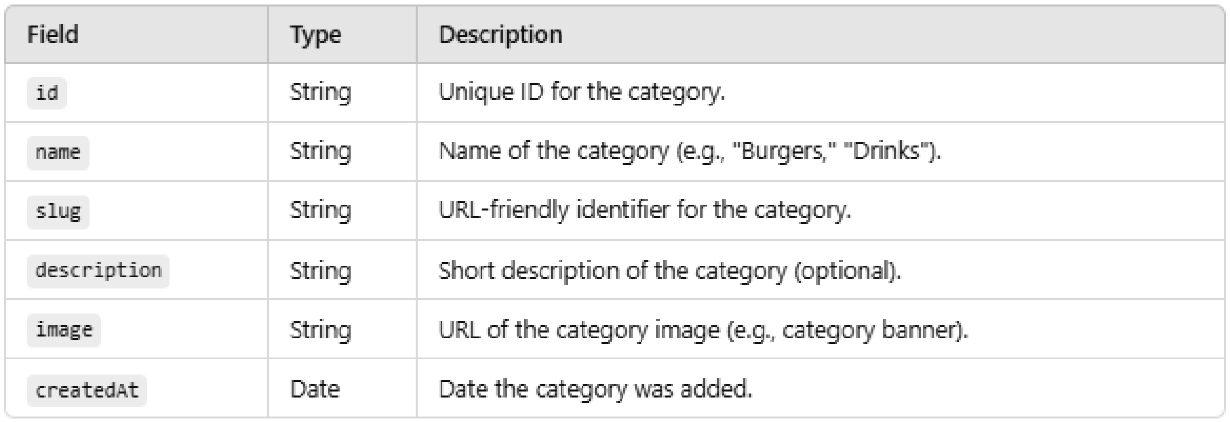
Order

Field	Type	Description
`id`	String	Unique ID for the order.
`customerId`	String	ID of the customer placing the order.
`items`	Array[Object]	List of food items in the order.
`totalPrice`	Number	Total price of the order.
`status`	String	Current status of the order (e.g., "Pending," "Delivered").
`createdAt`	Date	Date and time the order was placed.
`deliveryAddress`	Object	Delivery address for the order.
`paymentMethod`	String	Payment method (e.g., "Cash on Delivery," "Card").

Shipment

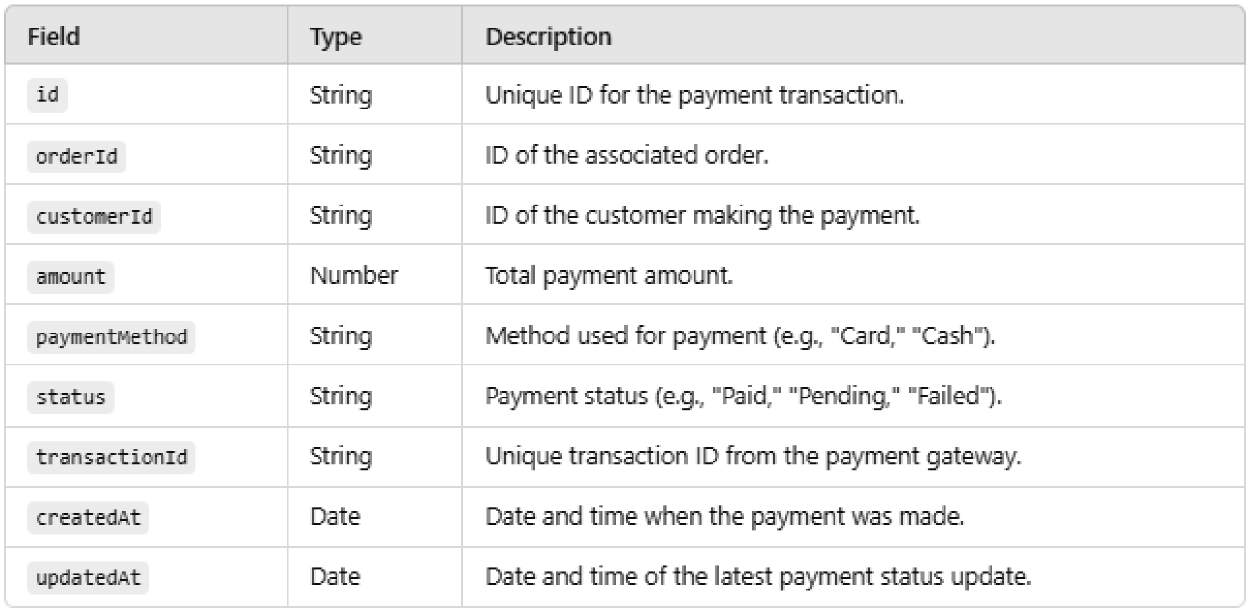
Field	Type	Description
`id`	String	Unique ID for the shipment.
`orderId`	String	ID of the associated order.
`customerId`	String	ID of the customer receiving the shipment.
`status`	String	Current status (e.g., "Pending," "In Transit," "Delivered").
`trackingNumber`	String	Tracking number provided by the delivery service.
`deliveryAddress`	Object	Delivery address for the shipment.
`createdAt`	Date	Date and time the shipment was created.
`updatedAt`	Date	Date and time the shipment status was last updated.

Category



Field	Type	Description
`id`	String	Unique ID for the category.
`name`	String	Name of the category (e.g., "Burgers," "Drinks").
`slug`	String	URL-friendly identifier for the category.
`description`	String	Short description of the category (optional).
`image`	String	URL of the category image (e.g., category banner).
`createdAt`	Date	Date the category was added.

Payment



Field	Type	Description
`id`	String	Unique ID for the payment transaction.
`orderId`	String	ID of the associated order.
`customerId`	String	ID of the customer making the payment.
`amount`	Number	Total payment amount.
`paymentMethod`	String	Method used for payment (e.g., "Card," "Cash").
`status`	String	Payment status (e.g., "Paid," "Pending," "Failed").
`transactionId`	String	Unique transaction ID from the payment gateway.
`createdAt`	Date	Date and time when the payment was made.
`updatedAt`	Date	Date and time of the latest payment status update.

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.