**Day 2 Tasks and Solutions**

**Marketplace Technical Foundation – FOODTUCK**

**1. Define Technical Requirements**

Based on my **business goals** and **data schema**, here are the technical requirements:

**Frontend Requirements**

* **User Interface**:
  + Home Page: Showcase featured meals, promotions, and a search bar.
  + Product Listing: Display meals, beverages, and desserts with filters (e.g., category, price).
  + Product Details: Show detailed information (e.g., name, price, description, customization options).
  + Cart: Allow users to view and modify their cart.
  + Checkout: Collect delivery details and process payments.
  + Order Confirmation: Display order summary and tracking information.
* **Responsive Design**:
  + Ensure the website works seamlessly on mobile, tablet, and desktop devices.
* **Dynamic Content**:
  + Fetch product data from **Sanity CMS** and display it dynamically.

**Sanity CMS as Backend**

**1. Product Schema**

The **Product schema** defines the structure for the meals, beverages, and desserts offered by my restaurant.

**Fields**

* **name**: Name of the product (e.g., "Margherita Pizza").
* **description**: Short description of the product.
* **price**: Price of the product.
* **category**: Category of the product (e.g., "Pizza", "Beverage", "Dessert").
* **image**: Image of the product.
* **customizationOptions**: List of customization options (e.g., "Extra Cheese", "No Onions").
* **availability**: Stock status (e.g., "In Stock", "Out of Stock").

**Sanity Schema**

javascript

export default {

name: 'product',

type: 'document',

title: 'Product',

fields: [

{

name: 'name',

type: 'string',

title: 'Product Name',

validation: Rule => Rule.required(),

},

{

name: 'description',

type: 'text',

title: 'Description',

validation: Rule => Rule.required(),

},

{

name: 'price',

type: 'number',

title: 'Price',

validation: Rule => Rule.required().min(0),

},

{

name: 'category',

type: 'string',

title: 'Category',

options: {

list: [

{ title: 'Pizza', value: 'pizza' },

{ title: 'Beverage', value: 'beverage' },

{ title: 'Dessert', value: 'dessert' },

],

},

validation: Rule => Rule.required(),

},

{

name: 'image',

type: 'image',

title: 'Product Image',

validation: Rule => Rule.required(),

},

{

name: 'customizationOptions',

type: 'array',

title: 'Customization Options',

of: [{ type: 'string' }],

},

{

name: 'availability',

type: 'string',

title: 'Availability',

options: {

list: [

{ title: 'In Stock', value: 'in-stock' },

{ title: 'Out of Stock', value: 'out-of-stock' },

],

},

validation: Rule => Rule.required(),

},

],

};

**2. Order Schema**

The **Order schema** defines the structure for customer orders.

**Fields**

* **orderId**: Unique identifier for the order.
* **customerId**: ID of the customer placing the order.
* **products**: List of products in the order (with quantities).
* **totalPrice**: Total cost of the order.
* **orderStatus**: Current status of the order (e.g., "Pending", "Preparing", "Out for Delivery", "Delivered").
* **orderTimestamp**: Date and time of order placement.
* **deliveryAddress**: Address where the order will be delivered.
* **riderId**: ID of the rider assigned to deliver the order.

**Sanity Schema**

javascript

export default {

name: 'order',

type: 'document',

title: 'Order',

fields: [

{

name: 'orderId',

type: 'string',

title: 'Order ID',

validation: Rule => Rule.required(),

},

{

name: 'customerId',

type: 'reference',

to: [{ type: 'customer' }],

title: 'Customer',

validation: Rule => Rule.required(),

},

{

name: 'products',

type: 'array',

title: 'Products',

of: [

{

type: 'object',

fields: [

{ name: 'productId', type: 'reference', to: [{ type: 'product' }], title: 'Product' },

{ name: 'quantity', type: 'number', title: 'Quantity', validation: Rule => Rule.required().min(1) },

],

},

],

validation: Rule => Rule.required(),

},

{

name: 'totalPrice',

type: 'number',

title: 'Total Price',

validation: Rule => Rule.required().min(0),

},

{

name: 'orderStatus',

type: 'string',

title: 'Order Status',

options: {

list: [

{ title: 'Pending', value: 'pending' },

{ title: 'Preparing', value: 'preparing' },

{ title: 'Out for Delivery', value: 'out-for-delivery' },

{ title: 'Delivered', value: 'delivered' },

],

},

validation: Rule => Rule.required(),

},

{

name: 'orderTimestamp',

type: 'datetime',

title: 'Order Timestamp',

validation: Rule => Rule.required(),

},

{

name: 'deliveryAddress',

type: 'string',

title: 'Delivery Address',

validation: Rule => Rule.required(),

},

{

name: 'riderId',

type: 'reference',

to: [{ type: 'rider' }],

title: 'Rider',

validation: Rule => Rule.required(),

},

],

};

**3. Customer Schema**

The **Customer schema** defines the structure for customer information.

**Fields**

* **customerId**: Unique identifier for the customer.
* **name**: Full name of the customer.
* **contactInfo**: Phone number and email address.
* **address**: Default delivery address.
* **orderHistory**: List of past orders placed by the customer.
* **preferences**: Customer preferences (e.g., favorite cuisine, dietary restrictions).

**Sanity Schema**

javascript

export default {

name: 'customer',

type: 'document',

title: 'Customer',

fields: [

{

name: 'customerId',

type: 'string',

title: 'Customer ID',

validation: Rule => Rule.required(),

},

{

name: 'name',

type: 'string',

title: 'Name',

validation: Rule => Rule.required(),

},

{

name: 'contactInfo',

type: 'object',

title: 'Contact Info',

fields: [

{ name: 'phone', type: 'string', title: 'Phone' },

{ name: 'email', type: 'string', title: 'Email' },

],

validation: Rule => Rule.required(),

},

{

name: 'address',

type: 'string',

title: 'Address',

validation: Rule => Rule.required(),

},

{

name: 'orderHistory',

type: 'array',

title: 'Order History',

of: [{ type: 'reference', to: [{ type: 'order' }] }],

},

{

name: 'preferences',

type: 'object',

title: 'Preferences',

fields: [

{ name: 'favoriteCuisine', type: 'string', title: 'Favorite Cuisine' },

{ name: 'dietaryRestrictions', type: 'array', of: [{ type: 'string' }], title: 'Dietary Restrictions' },

],

},

],

};

**4. Rider Schema**

The **Rider schema** defines the structure for delivery personnel.

**Fields**

* **riderId**: Unique identifier for the rider.
* **name**: Full name of the rider.
* **contactInfo**: Phone number and email address.
* **vehicleType**: Type of vehicle used for delivery (e.g., "Bike", "Scooter").
* **availabilityStatus**: Current status of the rider (e.g., "Available", "Busy", "Offline").
* **zoneId**: ID of the delivery zone the rider operates in.

**Sanity Schema**

javascript

export default {

name: 'rider',

type: 'document',

title: 'Rider',

fields: [

{

name: 'riderId',

type: 'string',

title: 'Rider ID',

validation: Rule => Rule.required(),

},

{

name: 'name',

type: 'string',

title: 'Name',

validation: Rule => Rule.required(),

},

{

name: 'contactInfo',

type: 'object',

title: 'Contact Info',

fields: [

{ name: 'phone', type: 'string', title: 'Phone' },

{ name: 'email', type: 'string', title: 'Email' },

],

validation: Rule => Rule.required(),

},

{

name: 'vehicleType',

type: 'string',

title: 'Vehicle Type',

options: {

list: [

{ title: 'Bike', value: 'bike' },

{ title: 'Scooter', value: 'scooter' },

],

},

validation: Rule => Rule.required(),

},

{

name: 'availabilityStatus',

type: 'string',

title: 'Availability Status',

options: {

list: [

{ title: 'Available', value: 'available' },

{ title: 'Busy', value: 'busy' },

{ title: 'Offline', value: 'offline' },

],

},

validation: Rule => Rule.required(),

},

{

name: 'zoneId',

type: 'reference',

to: [{ type: 'deliveryZone' }],

title: 'Delivery Zone',

validation: Rule => Rule.required(),

},

],

};

**Summary**

* **Product Schema**: Defines meals, beverages, and desserts.
* **Order Schema**: Tracks customer orders.
* **Customer Schema**: Stores customer information and preferences.
* **Rider Schema**: Manages delivery personnel.

**Third-Party APIs**

* **Payment Gateway**:
  + Integrate a payment gateway (e.g., Stripe, PayPal) for secure transactions.
* **Shipment Tracking**:
  + Use a third-party API for real-time order tracking (e.g., Google Maps API for delivery tracking).

**2. System Architecture Overview**

The system architecture consists of the following components:

1. **Frontend (Next.js)**: Handles user interactions and displays dynamic content.
2. **Sanity CMS**: Manages product data, customer details, and order records.
3. **Third-Party APIs**: Integrates payment processing and shipment tracking.
4. **Database (Sanity CMS)**: Stores all data (products, orders, customers, riders).

**High-Level Diagram**

plaintext

[Frontend (Next.js)]

| | |

[Sanity CMS] [Third-Party APIs]

**Detailed Workflow**

Here’s how the components interact in a real-world scenario:

**1. User Registration**

* **Frontend**: User signs up by entering their details (name, contact info, address).
* **Sanity CMS**: Stores the customer data in the **Customer schema**.
* **Response**: Confirmation message is sent to the user.

**2. Product Browsing**

* **Frontend**: User views product categories (e.g., Pizza, Beverage, Dessert).
* **Sanity CMS**: Fetches product data from the **Product schema**.
* **Frontend**: Displays the products dynamically on the website.

**3. Order Placement**

* **Frontend**: User adds items to the cart and proceeds to checkout.
* **Sanity CMS**: Saves the order details in the **Order schema**.
* **Third-Party API (Payment Gateway)**: Processes the payment.
* **Response**: Order confirmation is displayed to the user.

**4. Shipment Tracking**

* **Frontend**: User checks the status of their order.
* **Third-Party API (Shipment Tracking)**: Fetches real-time delivery updates.
* **Frontend**: Displays the shipment status (e.g., "In Transit", "Delivered").

**Component Details**

**1. Frontend (Next.js)**

* **Pages**:
  + **Home**: Displays featured products and promotions.
  + **Product Listing**: Shows all products with filters (e.g., category, price).
  + **Product Details**: Displays detailed information about a product.
  + **Cart**: Allows users to view and modify their cart.
  + **Checkout**: Collects delivery details and processes payments.
  + **Order Confirmation**: Displays order summary and tracking information.
* **Features**:
  + Responsive design for mobile and desktop.
  + Dynamic content fetched from Sanity CMS.
  + Integration with third-party APIs for payment and shipment tracking.

**2. Sanity CMS**

* **Schemas**:
  + **Product**: Manages product data (name, description, price, category, image, customization options, availability).
  + **Order**: Tracks customer orders (order ID, customer ID, products, total price, order status, delivery address, rider ID).
  + **Customer**: Stores customer information (name, contact info, address, order history, preferences).
  + **Rider**: Manages delivery personnel (name, contact info, vehicle type, availability status, delivery zone).
* **API**:
  + Provides endpoints to fetch and update data (e.g., /products, /orders, /customers, /riders).

**3. Third-Party APIs**

* **Payment Gateway**:
  + Processes payments securely (e.g., Stripe, PayPal).
  + Endpoint: /process-payment.
  + Payload: Order details (e.g., total price, customer ID).
  + Response: Payment status (e.g., "Success", "Failed").
* **Shipment Tracking**:
  + Provides real-time delivery updates (e.g., Google Maps API).
  + Endpoint: /track-shipment.
  + Payload: Order ID.
  + Response: Shipment status (e.g., "In Transit", "Delivered").

**Key Workflows**

**1. User Registration**

plaintext

[Frontend] → [Sanity CMS] → [Database]

1. User enters details (name, contact info, address).

2. Frontend sends data to Sanity CMS.

3. Sanity CMS stores data in the Customer schema.

4. Confirmation message is sent to the user.

**2. Product Browsing**

plaintext

[Frontend] → [Sanity CMS] → [Database]

1. User views product categories.

2. Frontend fetches product data from Sanity CMS.

3. Sanity CMS retrieves data from the Product schema.

4. Frontend displays products dynamically.

**3. Order Placement**

plaintext

[Frontend] → [Sanity CMS] → [Third-Party API] → [Database]

1. User adds items to the cart and proceeds to checkout.

2. Frontend sends order details to Sanity CMS.

3. Sanity CMS saves data in the Order schema.

4. Frontend sends payment details to the Payment Gateway API.

5. Payment Gateway processes the payment and sends a response.

6. Order confirmation is displayed to the user.

**4. Shipment Tracking**

plaintext

[Frontend] → [Third-Party API] → [Database]

1. User checks the status of their order.

2. Frontend sends a request to the Shipment Tracking API.

3. Shipment Tracking API fetches real-time updates.

4. Frontend displays the shipment status to the user.

**System Architecture Diagram**

Here’s a **visual representation** of the system architecture:

plaintext

+-----------------+ +-----------------+ +-----------------+

| Frontend | | Sanity CMS | | Third-Party APIs|

| (Next.js) | | | | |

+-----------------+ +-----------------+ +-----------------+

| | |

| | |

v v v

+-----------------+ +-----------------+ +-----------------+

| Product Data | | Order Data | | Payment Gateway |

| (Sanity CMS) | | (Sanity CMS) | | |

+-----------------+ +-----------------+ +-----------------+

| | |

| | |

v v v

+-----------------+ +-----------------+ +-----------------+

| Customer Data | | Rider Data | | Shipment Tracking|

| (Sanity CMS) | | (Sanity CMS) | | |

+-----------------+ +-----------------+ +-----------------+

**Summary**

* **Frontend (Next.js)**: Handles user interactions and displays dynamic content.
* **Sanity CMS**: Manages product data, customer details, and order records.
* **Third-Party APIs**: Integrates payment processing and shipment tracking.
* **Workflows**: User registration, product browsing, order placement, and shipment tracking.

**3. API Requirements Overview**

The APIs are divided into two categories:

1. **Internal APIs**: Used to interact with **Sanity CMS** for managing products, orders, customers, and riders.
2. **Third-Party APIs**: Used for **payment processing** and **shipment tracking**.

**1. Internal APIs (Sanity CMS)**

**Base URL**

* https://my-sanity-instance.api

**Authentication**

* Using **API tokens** for secure access to Sanity CMS.

**API Endpoints**

**1. Fetch All Products**

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

json

[

{

"id": "1",

"name": "Margherita Pizza",

"description": "Classic pizza with fresh mozzarella and basil.",

"price": 10.99,

"category": "Pizza",

"image": "https://example.com/pizza.jpg",

"customizationOptions": ["Extra Cheese", "No Onions"],

"availability": "In Stock"

}

]

**2. Fetch Product by ID**

* **Endpoint**: /products/{id}
* **Method**: GET
* **Description**: Fetch details of a specific product by its ID.
* **Response Example**:

json

{

"id": "1",

"name": "Margherita Pizza",

"description": "Classic pizza with fresh mozzarella and basil.",

"price": 10.99,

"category": "Pizza",

"image": "https://example.com/pizza.jpg",

"customizationOptions": ["Extra Cheese", "No Onions"],

"availability": "In Stock"

}

**3. Create Order**

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

json

{

"customerId": "123",

"products": [

{ "productId": "1", "quantity": 2 },

{ "productId": "2", "quantity": 1 }

],

"totalPrice": 25.98,

"deliveryAddress": "123 Main St, City, Country"

}

* **Response Example**:

json

{

"orderId": "456",

"status": "Pending",

"message": "Order created successfully."

}

**4. Fetch Order by ID**

* **Endpoint**: /orders/{id}
* **Method**: GET
* **Description**: Fetch details of a specific order by its ID.
* **Response Example**:

json

{

"orderId": "456",

"customerId": "123",

"products": [

{ "productId": "1", "quantity": 2 },

{ "productId": "2", "quantity": 1 }

],

"totalPrice": 25.98,

"orderStatus": "Pending",

"orderTimestamp": "2023-10-01T12:00:00Z",

"deliveryAddress": "123 Main St, City, Country",

"riderId": "789"

}

**5. Update Order Status**

* **Endpoint**: /orders/{id}
* **Method**: PATCH
* **Description**: Update the status of an order.
* **Payload**:

json

{

"orderStatus": "Out for Delivery"

}

* **Response Example**:

json

{

"orderId": "456",

"status": "Out for Delivery",

"message": "Order status updated successfully."

}

**6. Fetch Customer by ID**

* **Endpoint**: /customers/{id}
* **Method**: GET
* **Description**: Fetch details of a specific customer by their ID.
* **Response Example**:

json

{

"customerId": "123",

"name": "John Doe",

"contactInfo": {

"phone": "+1234567890",

"email": "john.doe@example.com"

},

"address": "123 Main St, City, Country",

"orderHistory": ["456", "789"],

"preferences": {

"favoriteCuisine": "Italian",

"dietaryRestrictions": ["Vegetarian"]

}

}

**7. Fetch Rider by ID**

* **Endpoint**: /riders/{id}
* **Method**: GET
* **Description**: Fetch details of a specific rider by their ID.
* **Response Example**:

json

{

"riderId": "789",

"name": "Jane Smith",

"contactInfo": {

"phone": "+0987654321",

"email": "jane.smith@example.com"

},

"vehicleType": "Bike",

"availabilityStatus": "Available",

"zoneId": "1"

}

**2. Third-Party APIs**

**1. Payment Gateway API**

* **Base URL**: https://api.paymentgateway.com
* **Endpoint**: /process-payment
* **Method**: POST
* **Description**: Process payment for an order.
* **Payload**:

json

{

"orderId": "456",

"amount": 25.98,

"paymentMethod": "Credit Card",

"cardDetails": {

"number": "4111111111111111",

"expiry": "12/25",

"cvv": "123"

}

}

* **Response Example**:

json

{

"paymentId": "789",

"status": "Success",

"message": "Payment processed successfully."

}

**2. Shipment Tracking API**

* **Base URL**: https://api.shipmenttracking.com
* **Endpoint**: /track-shipment
* **Method**: GET
* **Description**: Fetch real-time shipment tracking information.
* **Payload**:

json

{

"orderId": "456"

}

* **Response Example**:

json

{

"orderId": "456",

"status": "In Transit",

"eta": "15 mins",

"currentLocation": "123 Main St, City, Country"

}

**Summary**

* **Internal APIs**: Used to interact with Sanity CMS for managing products, orders, customers, and riders.
* **Third-Party APIs**: Used for payment processing and shipment tracking.
* **Endpoints**: Defined for fetching products, creating orders, updating order status, processing payments, and tracking shipments.