**CLOTHCOVE**

**THE TECHNICAL FOUNDATION FOR**

**Transitioning to Marketplace**

My 2nd day task of hackathon 3:

**Definition of system architecture:**

System architecture is the foundational structure of any system, detailing its components, data flow, and interactions. It ensures the system is scalable, secure, reliable, and efficient. A well-defined system architecture leads to better performance, ease of maintenance, and a more seamless user experience.

**WORKFLOW OF MY SYSTEM ARCHITECTURE:**

[Frontend (Next.js)]

|

|— [Pages]

|— [Components]

|— [Static Assets] (e.g., images, fonts)

|— [API Routes] (Optional)

|

v

[Sanity CMS]

|

|— [Content Models] (Products, Blogs, etc.)

|— [Content Delivery API] (Fetching dynamic content)

|

v

[Backend / API Server] (Optional)

|

|— [Database] (e.g., MongoDB, PostgreSQL)

|— [Custom Business Logic]

|

v

[3rd Party APIs]

|

|— [Payment Gateway] (e.g., Stripe, PayPal)

|— [Authentication Services] (e.g., Auth0, Firebase)

|— [Shipping APIs] (e.g., UPS, FedEx)

|— [Other Services] (e.g., Email services like SendGrid)

|

v

[Hosting / Deployment]

|

|— [Vercel / Netlify / AWS]

|

v

[Monitoring & Analytics]

|

|— [Google Analytics]

|— [Error Tracking] (e.g., Sentry)

|— [Performance Monitoring] (e.g., New Relic)

**Explanation:**

1. **Frontend (Next.js)**:
   * **Pages**: Your dynamic and static pages that represent different routes in your app (e.g., Home, Product Details, Contact).
   * **Components**: Reusable UI components like buttons, headers, footers, product cards, etc.
   * **Static Assets**: This includes images, fonts, icons, etc., that are loaded on the frontend.
   * **API Routes**: Optional in Next.js for handling backend logic directly inside the Next.js app.
2. **Sanity CMS**:
   * **Content Models**: Structures used for managing different types of content, such as products, blogs, user testimonials, etc.
   * **Content Delivery API**: This is how you fetch dynamic content from the Sanity CMS and render it on your frontend.
3. **Backend / API Server** (Optional):
   * **Database**: If you have custom server-side logic or need to store user data, transactions, or any custom data, you'll have a database like MongoDB or PostgreSQL.
   * **Custom Business Logic**: Backend logic to handle things like product recommendations, user authentication, etc.
4. **3rd Party APIs**:
   * **Payment Gateway**: Services like Stripe or PayPal for handling payments in your store.
   * **Authentication Services**: If you need to authenticate users (e.g., login, registration), services like Auth0 or Firebase Authentication can be used.
   * **Shipping APIs**: APIs to manage shipping costs, tracking, and logistics.
   * **Other Services**: Other services like email notifications (SendGrid, Mailgun) or marketing tools.
5. **Hosting / Deployment**:
   * **Vercel / Netlify / AWS**: Hosting platforms where your Next.js application is deployed, taking care of scaling, caching, and server-side rendering.
6. **Monitoring & Analytics**:
   * **Google Analytics**: Track user behavior on your site, conversions, and other relevant metrics.
   * **Error Tracking**: Tools like Sentry help in tracking and monitoring errors in production.
   * **Performance Monitoring**: Tools like New Relic help in measuring the performance of your site, checking for bottlenecks, or issues in real-time.

**Flow of Data in the Architecture:**

1. **Frontend (Next.js)**:
   * The Next.js frontend communicates with **Sanity CMS** via API calls to fetch dynamic content like products, blog posts, etc.
   * It can also interact with **3rd Party APIs** for handling payments, shipping, or other external services.
2. **Backend (Optional)**:
   * If you need custom server-side processing, the backend API server can interact with a database or other business logic before returning data to the frontend.
3. **Sanity CMS**:
   * Manages and serves content to the frontend using its **Content Delivery API**. This helps in rendering dynamic content like product descriptions or blog posts.
4. **3rd Party APIs**:
   * Handle specific tasks (e.g., payment processing, user authentication) and pass relevant information back to the frontend for display or processing.
5. **Deployment & Hosting**:
   * The app is deployed to platforms like **Vercel**, which are designed for server-side rendering, optimization, and easy deployment of Next.js apps.
6. **Monitoring & Analytics**:
   * Use analytics tools to monitor the performance, track user behavior, and handle errors that might arise in production

**WORKFLOW:**

**1. User Registration:**

1. **User signs up**:
   * The user navigates to the **Sign-Up page** and enters personal details (such as name, email, and password) in the registration form.
   * Once the form is filled, the user submits the registration details.
2. **Data is stored in Sanity**:
   * On form submission, the **Next.js backend API** receives the user's data and stores it in **Sanity CMS**.
   * Sanity's API interacts with the **User Content Model**, saving the user details securely.
3. **Confirmation sent to the user**:
   * After successfully saving the user's data in Sanity, a **confirmation email** is triggered.
   * The email can be sent through a **third-party email API** (e.g., **SendGrid**, **Mailgun**, or **Amazon SES**), confirming the user's registration and offering instructions to log in.

**2. Product Browsing:**

1. **User views product categories**:
   * The user navigates to the **Product Category page** (e.g., Men’s Apparel, Women’s Apparel, etc.).
   * The page loads, showing a list of available categories.
2. **Sanity API fetches data**:
   * The frontend sends a request to the **Sanity API** to fetch product data, such as names, prices, descriptions, and images.
   * The **Sanity Content API** returns product data based on the category selected.
3. **Products displayed on frontend**:
   * Once the data is fetched, the frontend dynamically renders the products using React components (e.g., **ProductGrid**, **ProductCard**).
   * The user can now browse products, view detailed descriptions, and interact with product features like sorting or filtering options.

**3. Order Placement:**

1. **User adds items to the cart**:
   * The user browses the products and selects items to add to the **shopping cart**.
   * The cart is managed either in the browser's **localStorage** or in a global state (e.g., **React Context API** or **Redux**).
2. **User proceeds to checkout**:
   * After reviewing the cart, the user clicks on the **"Checkout"** button.
   * The user provides necessary details like **shipping address** and **payment information** (via a payment gateway like **Stripe** or **PayPal**).
3. **Order details saved in Sanity**:
   * Upon successful payment, the backend (Next.js API) stores the order details in **Sanity CMS** or an alternative database (such as **MongoDB** or **PostgreSQL**).
   * The details stored include user information, product names, quantities, total cost, shipping address, and order status (pending, processed, shipped).

**4. Shipment Tracking:**

1. **Order status updates fetched via 3rd-party API**:
   * After an order is placed and shipped, the user can track the status of their order.
   * The frontend sends a request to a **3rd-party shipping API** (e.g., **UPS**, **FedEx**, **DHL**) to fetch the current order status using the tracking number received during checkout.
2. **Status displayed to the user**:
   * The frontend dynamically displays the shipment status (e.g., "In Transit", "Out for Delivery", "Delivered") using real-time data fetched from the shipping API.
   * The status can be shown in a **tracking page**, using **progress bars** or **status updates**.
   * Optionally, users can receive **email notifications** or **SMS updates** when the status changes (using third-party services like **Twilio**).

**Full Workflow Summary:**

1. **User Registration**:
   * User signs up → Data is stored in Sanity → Confirmation email sent.
2. **Product Browsing**:
   * User views product categories → Sanity API fetches product data → Products displayed on frontend.
3. **Order Placement**:
   * User adds items to the cart → Proceeds to checkout → Order details saved in Sanity.
4. **Shipment Tracking**:
   * Order status updates fetched via 3rd-party API → Displayed to the user.

**PLAN APIs REQUIREMENTS WITH ENDPOINT:**

**THIRD PARTY APIs:**

Database schema (Sanity CMS)

Shipment (ShipEngine)

Payment (Stripe)

**Endpoint Name: /products**

* **Method**: GET
* **Description**: Fetch all available products from Sanity CMS. This endpoint will retrieve product details, including the product's ID, name, price, stock, and image from the Sanity CMS API.
* **Response Example**:

json

[

{

"id": "12345",

"name": "Vita Classic Shirt",

"price": 16.48,

"stock": 100,

"image": "https://example.com/images/vita-shirt.jpg"

},

{

"id": "12346",

"name": "Vita Classic Jeans",

"price": 29.99,

"stock": 50,

"image": "https://example.com/images/vita-jeans.jpg"

}

]

**Endpoint Name: /products/{id}**

* **Method**: GET
* **Description**: Fetch a single product's details by its unique id from Sanity CMS.
* **Response Example**:

json

{

"id": "12345",

"name": "Vita Classic Shirt",

"price": 16.48,

"description": "Comfortable and stylish classic shirt for summer.",

"stock": 100,

"image": "https://example.com/images/vita-shirt.jpg"

}

**2. Order Endpoints**

**Endpoint Name: /orders**

* **Method**: POST
* **Description**: Create a new order in Sanity CMS. This endpoint will accept customer information (like name, address), product details, and payment status, and store the order in Sanity CMS.
* **Payload Example**:

json

{

"customer": {

"name": "John Doe",

"email": "johndoe@example.com",

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

},

"products": [

{

"productId": "12345",

"quantity": 2,

"price": 16.48

},

{

"productId": "12346",

"quantity": 1,

"price": 29.99

}

],

"paymentStatus": "paid",

"orderDate": "2025-01-17"

}

* **Response Example**:

json

CopyEdit

{

"orderId": "987654",

"customer": {

"name": "John Doe",

"email": "johndoe@example.com",

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

},

"products": [

{

"productId": "12345",

"quantity": 2,

"price": 16.48

},

{

"productId": "12346",

"quantity": 1,

"price": 29.99

}

],

"totalPrice": 62.95,

"paymentStatus": "paid",

"orderDate": "2025-01-17"

}

**Endpoint Name: /orders/{orderId}**

* **Method**: GET
* **Description**: Fetch the details of a specific order based on the orderId.
* **Response Example**:

json

{

"orderId": "987654",

"customer": {

"name": "John Doe",

"email": "johndoe@example.com",

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

},

"products": [

{

"productId": "12345",

"quantity": 2,

"price": 16.48

},

{

"productId": "12346",

"quantity": 1,

"price": 29.99

}

],

"totalPrice": 62.95,

"paymentStatus": "paid",

"orderDate": "2025-01-17",

"shipmentStatus": "In Transit"

}

**3. Shipment Tracking Endpoints**

**Endpoint Name: /shipment/{orderId}**

* **Method**: GET
* **Description**: Track the shipment status for a specific order via a third-party API (e.g., UPS, DHL). This endpoint will send a request to the third-party shipping provider's API and fetch the current shipment status.
* **Response Example**:

json

{

"shipmentId": "SH123456789",

"orderId": "987654",

"status": "In Transit",

"expectedDeliveryDate": "2025-01-20"

}

**Endpoint Name: /shipment/{orderId}/update**

* **Method**: POST
* **Description**: Update the shipment status for an order in the system (e.g., delivered, out for delivery, etc.). This endpoint is used when shipment status is updated via a third-party API.
* **Payload Example**:

json

{

"shipmentId": "SH123456789",

"orderId": "987654",

"status": "Delivered",

"expectedDeliveryDate": "2025-01-20"

}

* **Response Example**:

json

{

"message": "Shipment status updated successfully."

}

**4. Authentication and User Endpoints**

**Endpoint Name: /auth/register**

* **Method**: POST
* **Description**: Register a new user. The payload includes user details like name, email, and password, which are saved in Sanity CMS.
* **Payload Example**:

json

{

"name": "John Doe",

"email": "johndoe@example.com",

"password": "password123"

}

* **Response Example**:

json

{

"message": "User registered successfully."

}

**Endpoint Name: /auth/login**

* **Method**: POST
* **Description**: Log in an existing user by verifying their email and password. This will return a session token or JWT for future authentication.
* **Payload Example**:

json

{

"email": "johndoe@example.com",

"password": "password123"

}

* **Response Example**:

json

{

"message": "Login successful.",

"token": "JWT\_TOKEN\_HERE"

}

**5. Cart Management Endpoints**

**Endpoint Name: /cart**

* **Method**: GET
* **Description**: Fetch the user's current shopping cart, including product details, quantities, and prices.
* **Response Example**:

json

{

"cart": [

{

"productId": "12345",

"name": "Vita Classic Shirt",

"quantity": 2,

"price": 16.48

},

{

"productId": "12346",

"name": "Vita Classic Jeans",

"quantity": 1,

"price": 29.99

}

]

}

**Endpoint Name: /cart/add**

* **Method**: POST
* **Description**: Add a product to the user's shopping cart. The payload includes the product ID, quantity, and price.
* **Payload Example**:

json

{

"productId": "12345",

"quantity": 2,

"price": 16.48

}

* **Response Example**:

json

{

"message": "Product added to cart."

}

**Endpoint Name: /cart/remove**

* **Method**: POST
* **Description**: Remove a product from the user's shopping cart. The payload includes the product ID.
* **Payload Example**:

json

{

"productId": "12345"

}

* **Response Example**:

json

{

"message": "Product removed from cart."

}

**Flow Diagram:** 