**Marketplace Technical Foundation - TrendyCloth**

**1. System Architecture Document**

**1.1 Overview**

The marketplace website for TrendyCloth is designed to allow customers to browse and purchase trendy clothing. The system consists of several key components:

* **Frontend**: A React-based web application that provides the user interface for customers to interact with the marketplace.
* **Backend**: A mock API that simulates the processing of data, including fetching clothing items, handling user interactions, and performing CRUD operations.
* **CMS**: Sanity CMS to manage and store content such as clothing details, categories, and other relevant information.
* **Database**: The database is integrated with Sanity to store data related to clothing items, customer profiles, and order histories.

**1. Frontend**

* **Homepage**: A page that showcases banners, featured products, categories, and easy navigation.
* **Product Listing**: Displays a grid of clothing items with filters such as size, color, category, and price range.
* **Product Details**: A page showing detailed information for each clothing item (description, images, size options).
* **Cart**: A section to review and modify selected items before checkout.
* **Checkout**: A user-friendly interface for finalizing orders, entering payment information, and selecting delivery options.
* **Track Order** A dedicated page to track the delivery status of placed orders.

## 2. ****Sanity CMS****

* **Product Management**: Store and manage product details like name, price, category, sizes, images, and descriptions.
* **Order Management**: Track customer orders, payment status, and shipping details.

## ****Third-Party APIs****

1. **Payment Gateway**:
   * Integrate **Stripe** or **PayPal** for secure and reliable payment processing.
   * Features include:
     + Credit/debit card payment support.
     + Multi-currency support for international transactions.
     + Refund and dispute management.
2. **Shipment Tracking**:
   * Use **AfterShip** or similar courier APIs to provide real-time tracking information for customer orders.
   * Features include:
     + Real-time status updates (e.g., Shipped, Out for Delivery, Delivered).
     + Notifications for order movement.
3. **Mock API**:
   * Create a **Mock API** for development and testing purposes to simulate data and API responses without relying on live APIs.
   * Features include:
     + Simulating **product data**, **order statuses**, and **tracking updates**.

### ****API Structure****

#### **Products API**

* **Endpoint Name:** /api/products
  + **Method:** GET
  + **Purpose:** Fetch all products in the store.
  + **Request Parameters:** None
  + **Response:** List of products with details (e.g., name, price, category, etc.).
* **Endpoint Name:** /api/products/{id}
  + **Method:** GET
  + **Purpose:** Fetch a single product by ID.
  + **Request Parameters:** id (Product ID)
  + **Response:** Product details (e.g., name, price, description, images).
* **Endpoint Name:** /api/products
  + **Method:** POST
  + **Purpose:** Add a new product to the store.
  + **Request Parameters:** name, price, description, category, image
  + **Response:** Success message or product object with created data.
* **Endpoint Name:** /api/products/{id}
  + **Method:** PUT
  + **Purpose:** Update product details.
  + **Request Parameters:** id, name, price, description, category, image
  + **Response:** Updated product details.
* **Endpoint Name:** /api/products/{id}
  + **Method:** DELETE
  + **Purpose:** Delete a product by ID.
  + **Request Parameters:** id (Product ID)
  + **Response:** Success message or error message.

#### **Categories API**

* **Endpoint Name:** /api/categories
  + **Method:** GET
  + **Purpose:** Fetch all product categories.
  + **Request Parameters:** None
  + **Response:** List of categories (e.g., Korean style, Western, Old Money).
* **Endpoint Name:** /api/categories/{id}
  + **Method:** GET
  + **Purpose:** Fetch a single category by ID.
  + **Request Parameters:** id (Category ID)
  + **Response:** Category details (e.g., name, description).

#### **Orders API**

* **Endpoint Name:** /api/orders
  + **Method:** GET
  + **Purpose:** Fetch all orders for the user.
  + **Request Parameters:** user\_id (User ID)
  + **Response:** List of orders for the user, including status and product details.
* **Endpoint Name:** /api/orders/{id}
  + **Method:** GET
  + **Purpose:** Fetch details of a single order.
  + **Request Parameters:** id (Order ID)
  + **Response:** Order details (e.g., items, total price, shipping info).
* **Endpoint Name:** /api/orders
  + **Method:** POST
  + **Purpose:** Place a new order.
  + **Request Parameters:** user\_id, items (Product IDs with quantities), shipping\_address
  + **Response:** Order confirmation and details.
* **Endpoint Name:** /api/orders/{id}/ship
  + **Method:** POST
  + **Purpose:** Mark an order as shipped.
  + **Request Parameters:** id (Order ID), shipping\_provider, tracking\_number
  + **Response:** Shipment details with provider and tracking info.
* **Endpoint Name:** /api/orders/{id}/tracking
  + **Method:** GET
  + **Purpose:** Fetch tracking information for a shipped order.
  + **Request Parameters:** id (Order ID)
  + **Response:** Tracking details (e.g., shipping provider, status, delivery ETA).

#### **Users API**

* **Endpoint Name:** /api/users/{id}
  + **Method:** GET
  + **Purpose:** Fetch user profile.
  + **Request Parameters:** id (User ID)
  + **Response:** User profile details (e.g., name, email, shipping info).

#### **Authentication API**

* **Endpoint Name:** /api/auth/register
  + **Method:** POST
  + **Purpose:** User registration.
  + **Request Parameters:** name, email, password
  + **Response:** Success message or user object.
* **Endpoint Name:** /api/auth/login
  + **Method:** POST
  + **Purpose:** User login.
  + **Request Parameters:** email, password
  + **Response:** Success message and user authentication token.

#### **Cart API**

* **Endpoint Name:** /api/cart
  + **Method:** GET
  + **Purpose:** Fetch cart details for the user.
  + **Request Parameters:** user\_id
  + **Response:** Cart items with quantities and total price.
* **Endpoint Name:** /api/cart/{id}
  + **Method:** POST
  + **Purpose:** Add an item to the user's cart.
  + **Request Parameters:** user\_id, product\_id, quantity
  + **Response:** Updated cart details.
* **Endpoint Name:** /api/cart/{id}
  + **Method:** DELETE
  + **Purpose:** Remove an item from the user's cart.
  + **Request Parameters:** user\_id, product\_id
  + **Response:** Updated cart details.

### 

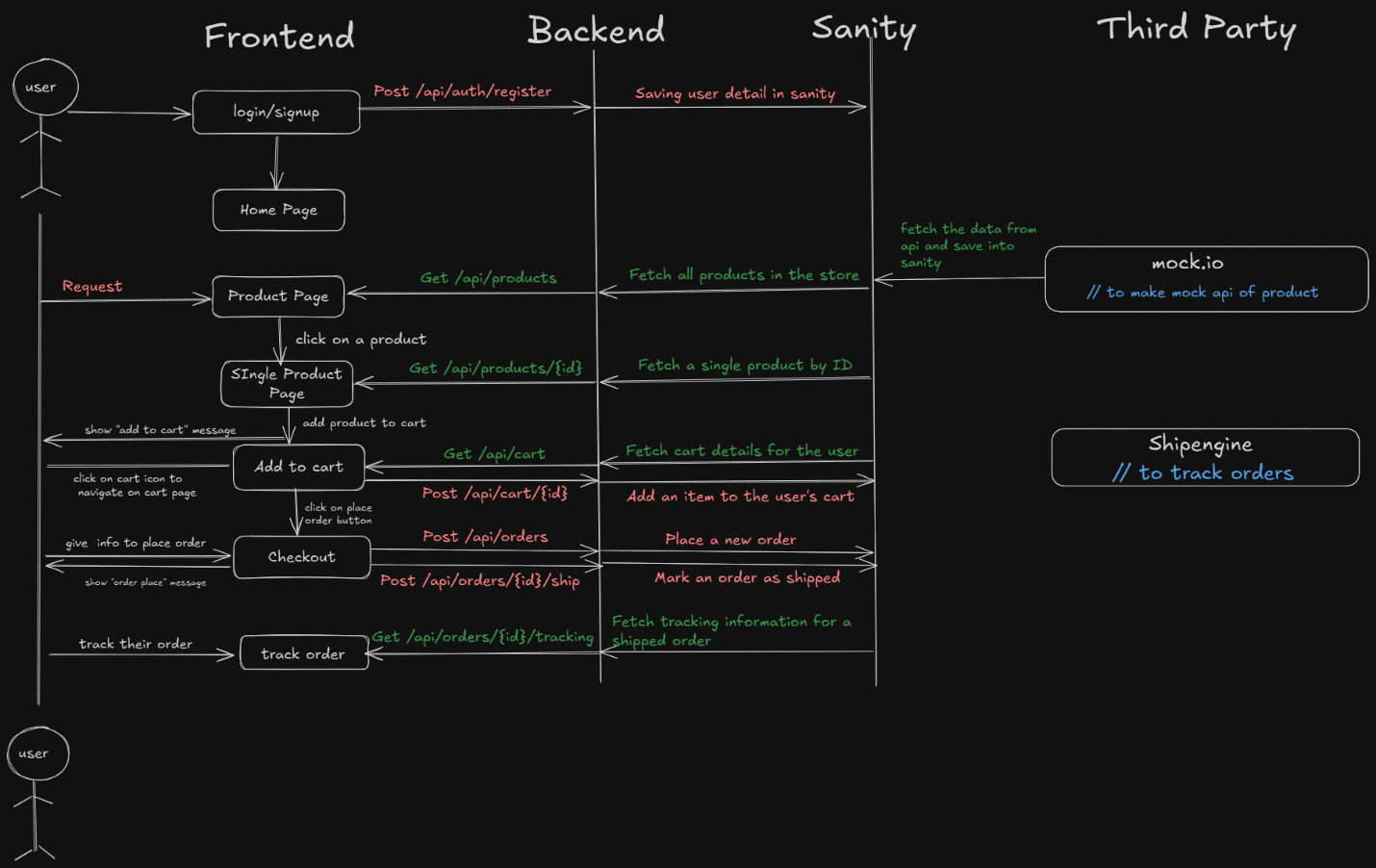
### ****System Workflow****

* **Login/Signup**
  + **User Action:** User initiates login or signup.
  + **API Call:** POST /api/auth/register (Frontend → Backend).
  + **Data Management:** User details are saved in Sanity.
* **Home Page**
  + **User Action:** User lands on the home page after login or signup.
* **Product Page**
  + **User Action:** User views a list of available products.
  + **API Call:** GET /api/products (Frontend → Backend).
  + **Data Management:** Backend fetches all product data from Sanity.
* **Single Product Page**
  + **User Action:** User clicks on a product to view details.
  + **API Call:** GET /api/products/{id} (Frontend → Backend).
  + **Data Management:** Backend fetches the product details by ID from Sanity.
* **Add to Cart**
  + **User Action:** User adds a product to their cart.
  + **API Call:**
    - GET /api/cart to fetch current cart details (Frontend → Backend).
    - POST /api/cart/{id} to add the product to the cart (Frontend → Backend).
  + **Data Management:** Backend updates the user's cart details in Sanity.
* **Cart Page**
  + **User Action:** User navigates to the cart page by clicking the cart icon.
* **Checkout**
  + **User Action:** User clicks the "Place Order" button and provides order details.
  + **API Calls:**
    - POST /api/orders to create a new order (Frontend → Backend).
    - POST /api/cart/{id} to confirm items in the cart (Frontend → Backend).
  + **Data Management:** Backend saves the order details and updates the cart in Sanity.
* **Track Order**
  + **User Action:** User tracks their order status.
  + **API Call:** GET /api/orders/{id}/tracking (Frontend → Backend).
  + **Data Management:** Backend fetches tracking details from ShipEngine and provides them to the user.

### ****Third-Party Services****

* **Mock.io**
  + Provides a mock API for product data.
  + Data fetched by the Backend and stored in Sanity for future use.
* **ShipEngine**
  + Used to track order shipping status.
  + Provides tracking information to the Backend, which is displayed to the user.

**Design System Architecture:**

****

### ****Sanity Schema**** Product Schema

export default {

name: 'product',

title: 'Product',

type: 'document',

fields: [

{

name: 'name',

title: 'Product Name',

type: 'string',

validation: Rule => Rule.required().min(3).max(50)

},

{

name: 'slug',

title: 'Slug',

type: 'slug',

options: {

source: 'name',

maxLength: 96,

},

validation: Rule => Rule.required()

},

{

name: 'description',

title: 'Description',

type: 'text',

validation: Rule => Rule.required().max(300)

},

{

name: 'price',

title: 'Price',

type: 'number',

validation: Rule => Rule.required().positive()

},

{

name: 'category',

title: 'Category',

type: 'string',

options: {

list: [

{ title: 'Korean Style', value: 'korean-style' },

{ title: 'Western Clothes', value: 'western-clothes' },

{ title: 'Old Money Fashion', value: 'old-money-fashion' },

],

},

validation: Rule => Rule.required()

},

{

name: 'images',

title: 'Images',

type: 'array',

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

options: {

hotspot: true,

},

},

{

name: 'stock',

title: 'Stock',

type: 'number',

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

},

{

name: 'sizes',

title: 'Sizes',

type: 'array',

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

options: {

list: [

{ title: 'Small', value: 'S' },

{ title: 'Medium', value: 'M' },

{ title: 'Large', value: 'L' },

{ title: 'Extra Large', value: 'XL' },

],

},

validation: Rule => Rule.required().min(1)

},

{

name: 'createdAt',

title: 'Created At',

type: 'datetime',

initialValue: () => new Date().toISOString(),

},

],

};

## ****Purpose of Documentation****

1. **Team Alignment**: Provides a shared understanding of the project architecture, workflows, and APIs to ensure all team members are aligned.
2. **Scalability**: Acts as a reference guide for adding new features or scaling the system without disrupting the existing architecture.
3. **Onboarding**: Simplifies the onboarding process for new developers by giving them clear insights into the system.
4. **Troubleshooting**: Helps identify and resolve issues by offering detailed workflows and data structures.
5. **Consistency**: Ensures uniformity in code standards and workflows across the team.
6. **Client Communication**: Serves as a professional document to explain the project’s architecture and workflows to stakeholders or clients.