Hackathon Day 2

Marketplace Technical Foundation

Furni Nest

Overview

This document outlines the technical and functional foundation of a General E-Commerce Marketplace, developed to provide a seamless shopping experience for users. The primary aim is to design a scalable and user-friendly platform for selling furniture items like sofas and chairs, while incorporating modern tools and APIs.

Purpose

To build an efficient and responsive system for customers and admins by integrating:

- **Sanity CMS** for real-time content management.
- **Stripe API** for secure payment processing.
- **Shipment Platform API** for real-time order tracking.

Technical Plan

This technical plan aligns with the specific requirements of a General E-Commerce Marketplace. This plan ensures the platform's goals and functionality are met effectively

1. **User-Centric Design:**

- Provide a seamless user experience for browsing products (e.g., sofas, chairs), managing carts, and placing orders.
- Ensure mobile and desktop responsiveness through Next.js.

2. **Real-Time Updates:**

- Integrate Sanity CMS to handle real-time inventory updates and product management.

3. **Secure Payments:**

- Use Stripe API for safe and reliable payment processing.

4. **Order Tracking:**

- Incorporate Shipment Platform APIs for real-time order tracking and delivery status updates.

5. **Admin Tools:**

- Develop an admin dashboard to manage products, orders, and sales reports effectively.

6. **Scalable and Flexible Architecture:**

- Design a modular system that can scale with increasing users and products.

Goals and Objectives

1. **User Experience:** Deliver a seamless and responsive shopping journey.
2. **Scalability:** Build a system to handle growing users and data efficiently.
3. **Easy Management:** Simplify product and order management for admins.
4. **Secure Payments:** Ensure safe transactions with trusted payment gateways.
5. **Real-Time Updates:** Enable live inventory management and order tracking.
6. **Business Growth:** Attract more users and vendors to the platform.

Architecture Overview

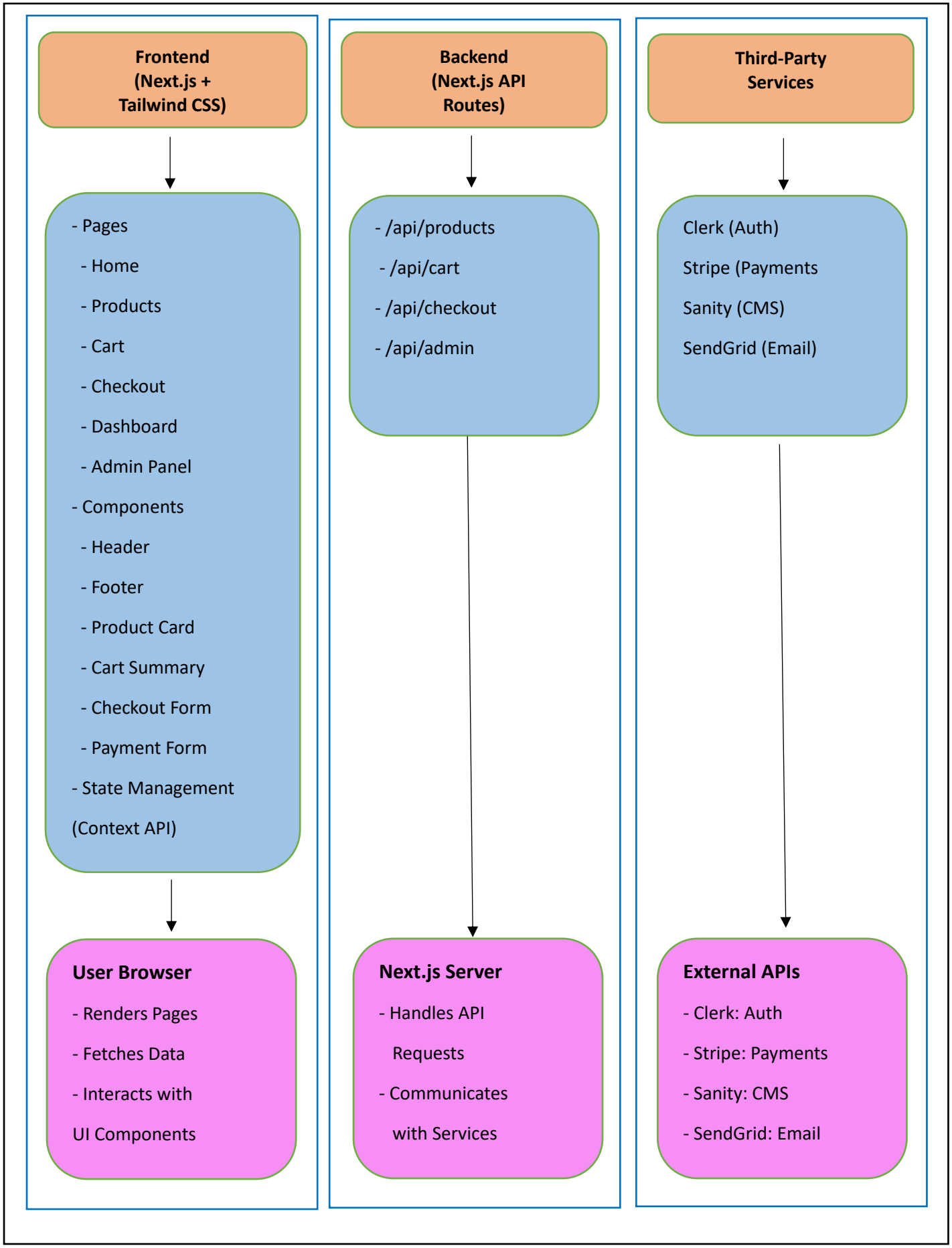
The architecture links Next.js for the frontend, Node.js for backend processing, Sanity CMS for data management, and APIs like Stripe and Shipment for payments and tracking.

Key Components

1. **Frontend (Next.js):**
    - Manages user interactions and provides a responsive, mobile-friendly interface.
    - Dynamically renders product data from the backend using APIs and Sanity CMS.
2. **Backend (Node.js):**
    - Serves as the intermediary between the frontend and the data layer.
    - Processes business logic, including payments, order management, and API integration (e.g., Stripe and Shipment APIs).

3. **Data Layer (Sanity CMS):**
    - Manages product details, categories, and user profiles.
    - Provides APIs for real-time synchronization of data with the frontend and backend.
4. **Payment Gateway (Stripe):**
    - Secures online payment processing and transaction tracking.
5. **Shipment API:**
    - Tracks real-time delivery updates and sends notifications to users about shipment status.

System Architecture Diagram



The architecture diagram illustrates the interaction between the frontend, backend, APIs, and Sanity CMS. Each component plays a critical role in ensuring smooth operation.

3. User and Admin Workflows

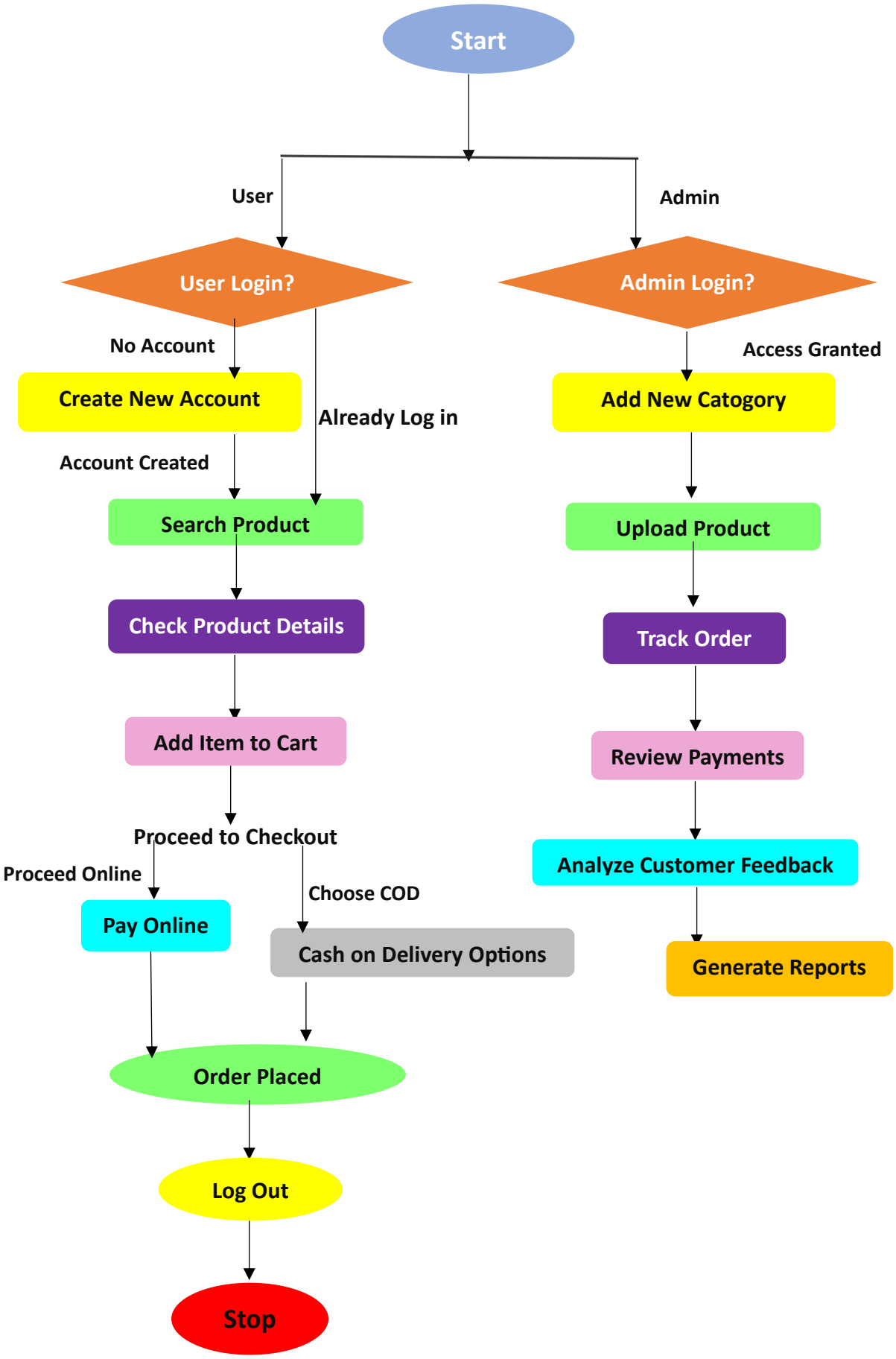
The workflows provide a step-by-step view of user and admin interactions with the system.

User Workflow

1. Search Products: Users search for items like 'sofas' or 'chairs.'
2. View Product Details: Users select a product to view its description, price, and stock status.
3. Add to Cart: Items are added to the cart dynamically.
4. Checkout: Users provide payment and shipping details.
5. Track Order: Real-time updates via Shipment API.
6. Receive Product: Users confirm delivery upon receiving the item.

Admin Workflow

1. Login to Dashboard: Admins securely log in.
2. Manage Products: Add, update, or remove items via Sanity CMS.
3. Monitor Orders: Track orders and update their statuses.
4. Generate Reports: Analyze sales and generate insights.


Start
User
Admin
User Login?
Admin Login?
No Account
Access Granted
Create New Account
Already Log in
Add New Catogory
Account Created
Search Product
Upload Product
Check Product Details
Track Order
Add Item to Cart
Review Payments
Proceed to Checkout
Analyze Customer Feedback
Proceed Online
Choose COD
Pay Online
Cash on Delivery Options
Generate Reports
Order Placed
Log Out
Stop

4. API Endpoints

The following table summarizes the API endpoints required for the marketplace:

Endpoint	Method	Description	Parameters	Response Example
/products	GET	Fetch all products	None	{ "id": 1 ,"name": "Sofa", "price": 500 }
/products/:id	GET	Fetch a product by ID	id	{ "id": 1, "name": "Sofa", "price": 500 }
/orders	POST	Create a new order	products, userId	{ "orderId": 123, "status": "Confirmed" }
/orders/:id	GET	Retrieve order details by ID	id	{ "orderId": 123, "status": "Shipped" }
/auth/register	POST	User registration	email, password	password { "userId": 1, "message": "Registered" }
/auth/login	POST	User authentication	email, password	{ "userId": 1, "token": "abcd1234" }

Sanity Schemas

1. Product Schema

Manages product details for the marketplace.

- **Fields:**
  - name (string): Product name.
  - price (number): Product price.
  - description (string): Product details.
  - stock (number): Available quantity.
  - category (string): Product category (e.g., sofa, chair).

2. User Schema

Stores customer data and account details.

- **Fields:**
    - userID (string): Unique identifier for users.
    - name (string): Full name of the customer.
    - email (string): Contact email.
    - password (string): Encrypted password.

3. Order Schema

Tracks customer orders and related details.

- **Fields:**
    - orderID (string): Unique order identifier.
    - userID (string): Links to the user placing the order.
    - productIDs (array): List of products in the order.
    - totalAmount (number): Total price of the order.
    - status (string): Current order status (e.g., pending, shipped).

4. Payment Schema

Manages payment transactions for orders.

- **Fields:**
    - paymentID (string): Unique payment identifier.
    - orderID (string): Links to the order.
    - amount (number): Payment amount.
    - paymentDate (date): Payment transaction date.
    - status (string): Payment status (e.g., success, failed).

5. Category Schema

Organizes products into categories.

- **Fields:**
    - categoryID (string): Unique identifier for the category.
    - name (string): Name of the category.
    - description (string): Brief details about the category.