Day#1: Laying the Foundation of your Marketplace Journey

Step# 1: Choose Your Marketplace Type

Type: E-commerce

Purpose: Online platform for buying and selling furniture

Step #2: Define Your Business Goals

Problem:

Our marketplace will provide to customers a convinient solution for buying high-quality furniture online.

Target Audience:

Homeowners, interior designers, and business looking for stylish and affordable furniture.

Products/Services:

Sofas, beds, dining tables, office furniture, plastic/Wooden chairs.

Unique Selling Point:

Wide range of furniture styles, customization options, and affordable prices on high quality.

Step#3: Create a Data Schema

[Product]
- ID
- Name
- Price
- Stock
- Description
- Category
- Dimensions
  |
  |

[order] ------> [customer]

[order]	[customer]
- order ID	- Customer ID
- Product ID	- Name
- Quantity	- Email
- Order Date	- Phone
- Delivery Date	- Address
- Status	

  |
  |

[order] ------> [Delivery Zone]
- Zone ID
- Zone Name
- Coverage Area
- Assigned Drivers

Relationships between Entities:

- Products are linked to orders via product ID
- orders are linked to customers via customer ID
- orders are linked to Delivery Zones based on the delivery address
- Delivery Zones are linked to specific Drivers for shipping

Marketplace Technical Foundation – Marketplace Builder Hackathon 2025 (Day-2)

1. Frontend Requirements:

- User Interface (UI):

  A user-friendly interface has been developed for browsing furniture products on the Avion website.

  The design is clean, modern, and aesthetically pleasing to enhance user experience.

- Responsive Design:

  The Avion website is fully responsive, ensuring it works seamlessly on both mobile and desktop devices.

Essential Pages:

- Home Page:

The main page of Avion, showcasing featured products, promotions, and including a search bar for easy navigation.

- Product Listing Page:

  A page displaying a list of furniture products with filters for categories, price, and popularity, making it easy for users to find what they are looking for.

- Product Details Page:

A detailed view of a single product, including images, description, price, and customer reviews (to be added later).

- **Shopping Cart:**

  The functionality for adding items to the cart has been implemented. This page displays selected products, their quantities, prices, and provides the option to remove items.

- **Checkout Page (to be added):**

  This page will be developed in the coming days, allowing users to enter their details, choose shipping options, and make payments.

- **Order Confirmation Page (to beadded):**

  This page will be developed in the coming days to confirm the user's order with an order summary and estimated delivery date.

Additional Features:

- **Search Bar:**

A search functionality has been integrated into the Avion website, allowing users to quickly find products.

- **Navigation Menu:**

  Easy navigation has been provided to different categories and sections of the Avion website, ensuring a seamless browsing experience.

- **User Account (to be added):**

  Login and sign-up functionality along with account management features will be provided later.

- **Wishlist (to be added):**

  There are plans to provide users the option to save products to a wishlist for future reference.

- Customer Reviews (to be added):

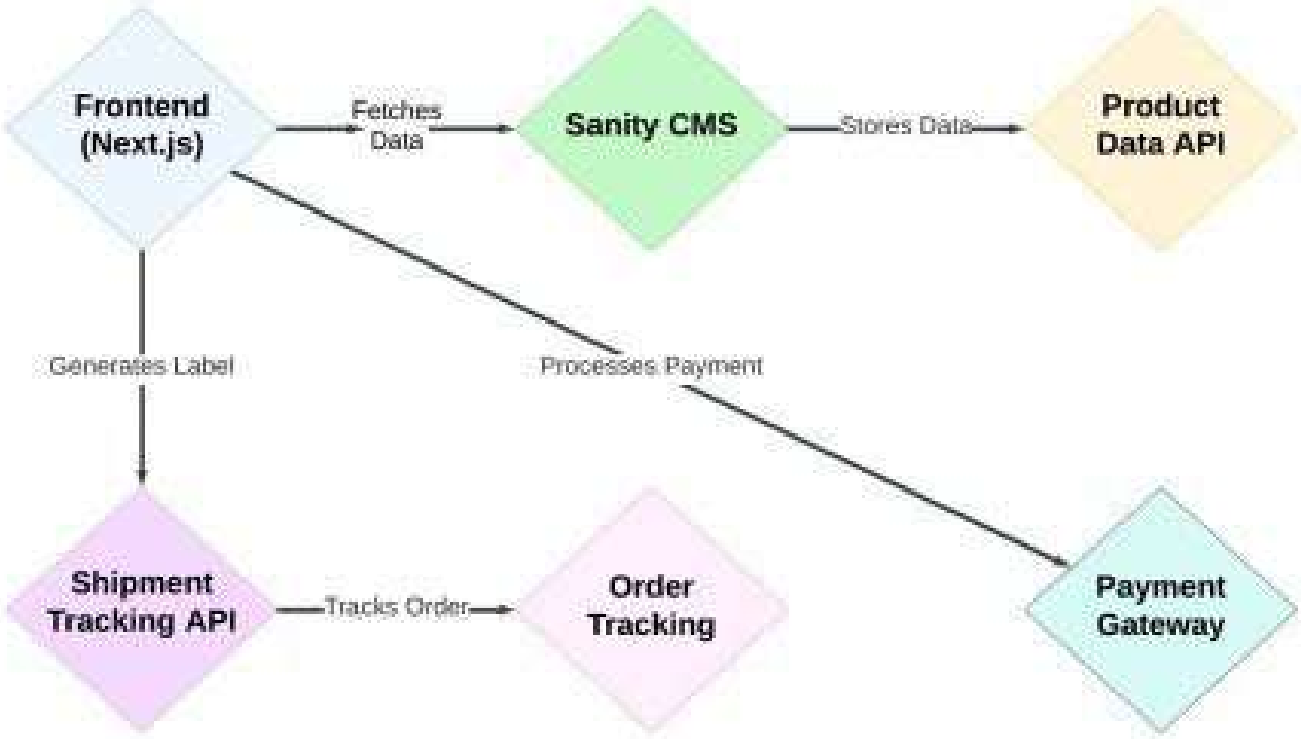
  A section will be provided where users can read and write reviews on products.

2. Design System Architecture:

- Introduction:

  The system architecture diagram below outlines the interactions and data flow between different components of the Avion e-commerce website. This architecture ensures a scalable, efficient, and user-friendly platform for managing furniture products, handling orders, and processing payments.

- System Architecture Diagram:



Component Descriptions:

- ### Frontend (Next.js):

  Reason for Use: Next.js is used for its powerful server-side rendering capabilities, which improve the performance and SEO of the website. It provides a seamless user experience with fast page loads and dynamic content rendering.

  Function: It serves as the user interface layer where users interact with the website, browse products, manage their cart, and proceed to checkout.

- ### Sanity CMS:

  Reason for Use: Sanity CMS is chosen for its flexibility and real-time content management capabilities. It allows for easy management of product data and orders through a user-friendly interface.

  Function: It manages and stores product and order data. Sanity CMS connects to the Product Data API to retrieve product information and update the frontend with the latest data.

- ### Product Data API:

  Reason for Use: The Product Data API provides a structured way to access detailed product information. It ensures that the data is consistent and up-to-date across the system.

  Function: It provides detailed information about the products to Sanity CMS, which is then used to display product details on the website.

- ### Third-Party API (Shipment Tracking):

  Reason for Use: The Shipment Tracking API is integrated to provide users with real-time tracking information for their orders. It enhances the user experience by keeping them informed about their shipment status.

  Function: It generates shipping labels and tracks orders, providing tracking information to the frontend.

- **Payment Gateway:**

  Reason for Use: A secure payment gateway is essential for processing payments efficiently and safely. It handles payment transactions and ensures that sensitive payment information is processed securely.

  Function: It processes payments securely, interacting with the frontend to handle payment details and confirm successful transactions.

- **Workflow Description:**

  In this architecture, a typical data flow could look like this:

  A user visits the marketplace frontend to browse products.

  The frontend makes a request to the Product Data API (powered by Sanity CMS) to fetch product listings and details, which are displayed dynamically on the site.

  When the user places an order, the order details are sent to Sanity CMS via an API request, where the order is recorded.

  Shipment tracking information is fetched through a Third-Party API and displayed to the user in real-time.

  Payment details are securely processed through the Payment Gateway, and a confirmation is sent back to the user and recorded in Sanity CMS.

Key Workflows to Include:

- **User Registration:**

  User signs up -> Data is stored in Sanity -> Confirmation sent to the user.

- **Product Browsing:**

  User views product categories -> Sanity API fetches data -> Products displayed on frontend.

- **Order Placement:**

  User adds items to the cart -> Proceeds to checkout -> Order details saved in Sanity.

- **Shipment Tracking:**

  Order status updates fetched via 3rd-party API -> Displayed to the user.

Conclusion:

This detailed workflow ensures students understand how components interact in a real-world scenario and how data flows seamlessly between them. This demonstrates how frontend interactions are supported by Sanity CMS for content management, third-party APIs for logistics, and payment gateways for transaction processing. Including such details will help you visualize dependencies and plan integrations effectively.

3. Plan API Requirements:

- ### Introduction:

  This section defines the API endpoints needed based on the data schema for the Avion e-commerce website. Each endpoint is described with its method, purpose, payload, and response.

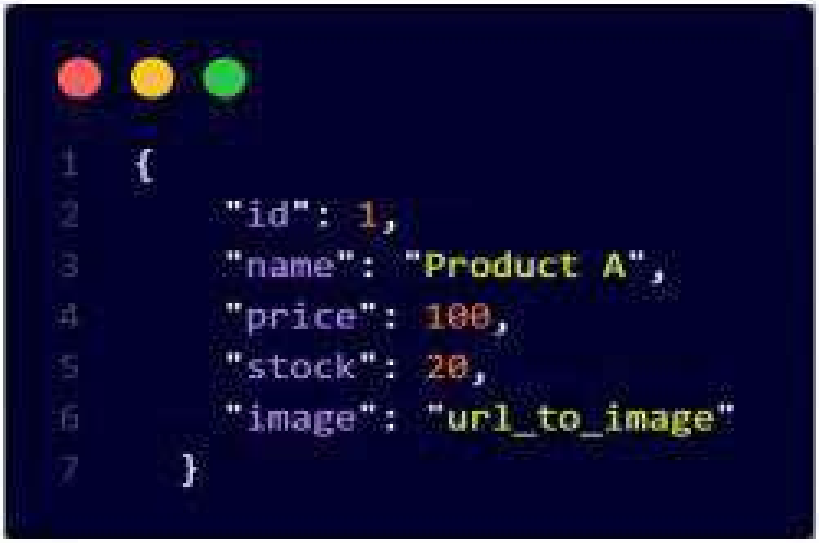
API Endpoints:

1. **Endpoint Name:**/products

   Method: GET

   Description: Fetch all available products from Sanity.

   Response Example:



2. Endpoint Name:/orders

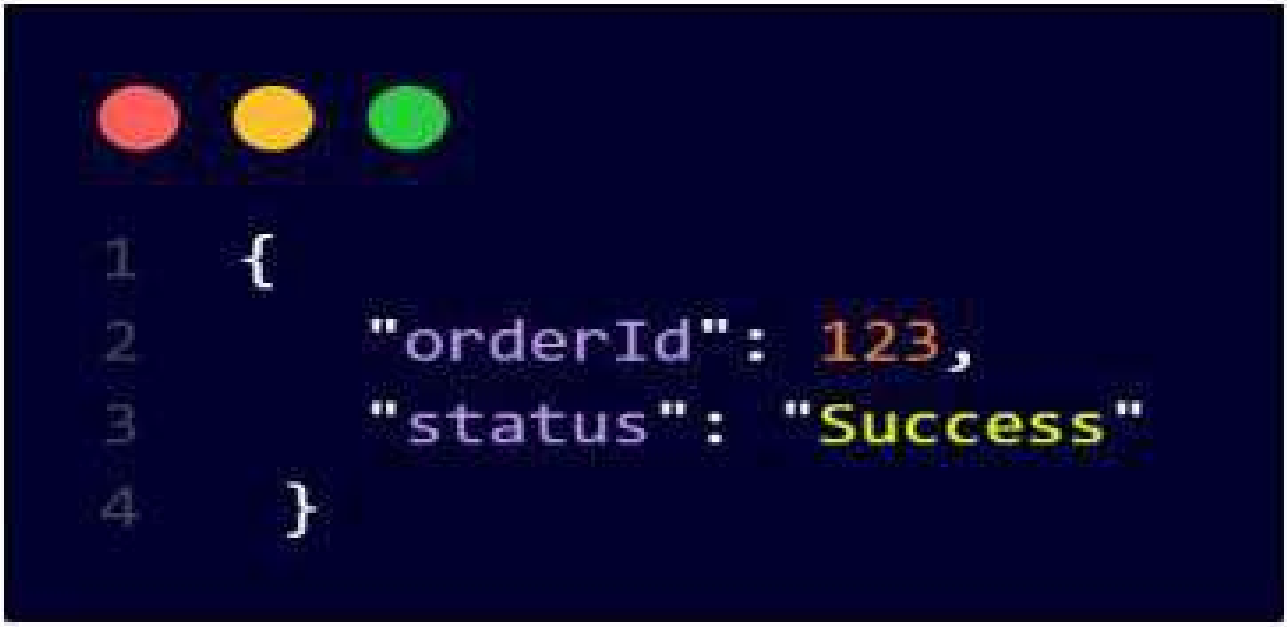
Method: POST

Description:Create a new order in Sanity.

Payload Example:

```
{
    "customerInfo": {
      "name": "John Doe",
      "email": "john@example.com",
      "address": "123 Main St"
    },
    "productDetails": [
      {
        "productId": 1,
        "quantity": 2
      }
    ],
    "paymentStatus": "Paid"
}
```

Response Example:



```
{
    "orderId": 123,
    "status": "Success"
}
```

3. **Endpoint Name:**/shipment

   Method: GET

   Description: Track order status via third-party API.

   Response Example:

```
{
    "shipmentId": "SHIP123",
    "orderId": 123,
    "status": "In Transit",
    "expectedDeliveryDate": "2025-01-20"
}
```

Conclusion:

This detailed API documentation ensures a clear understanding of the endpoints needed for the Avion e-commerce website. It outlines the methods, descriptions, payloads, and responses for each endpoint, providing clarity for implementation and ensuring seamless integration with the marketplace workflows.

4. Write Technical Documentation:

- Objective:
  - To build a scalable, user-friendly e-commerce platform with the following features:
  - Product browsingand management via Sanity CMS.
  - Authentication using Clerk.
  - Order tracking with ShipEngine API.
  - Secure payments via Stripe.
  - Modern tools like useContext for cart functionality.

1. System Architecture Overview:

- System Architecture Diagram:
  - User[User] -->|Signin| Clerk[Clerk Authentication]
  - User -->|Browses| Frontend[Next.js]
  - Frontend -->|Fetches Data| Sanity[Sanity CMS]
  - Frontend -->|Manages Cart| Context[useContext API]
  - Frontend -->|Places Order| Stripe[Stripe Checkout]
  - Frontend -->|Generates Label| ShipEngine[ShipEngine API]
  - Sanity -->|Stores| ProductData[Product Data]
  - ShipEngine -->|Tracks Order| OrderTracking[Order Tracking]

2. Key Workflows:

- User Authentication (Clerk):

  Use Clerk's pre-built authentication components.

  Manage user sessions without storing data in Sanity CMS.

- Product Browsing:

  Fetch and display products from Sanity CMS using GROQ queries.

- Cart Management:

  Use useContext to manage cart state globally.

  Add/remove items and calculate totals dynamically.

- Checkout Process:

  Collect user details and payment via Stripe-hosted checkout.

  Display order confirmation after successful payment.

- Order Tracking:

  Generate a shipping label ID using ShipEngine.

  Provide label ID to users for tracking.

3. API Endpoints:

Endpoint	Method	Purpose	Response Example
/api/products	Get	Fetch product data from Sanity CMS	{ "id": 1, "name": "Product A", "price": 100, "stock": 20, "image": "url_to_image" }
/api/shipping-label	Post	Generate a shipping label using ShipEngine	{ "labelId": "LABEL123", "status": "Generated" }
/api/track-order	Get	Retrieve order status using ShipEngine label ID	{ "shipmentId": "SHIP123", "orderId": 123, "status": "In Transit", "expectedDeliveryDate": "2025-01-20" }
/api/checkout	Post	Integrate Stripe for payment processing	{ "orderId": 123, "status": "Success" }

4. Sanity Schema Example

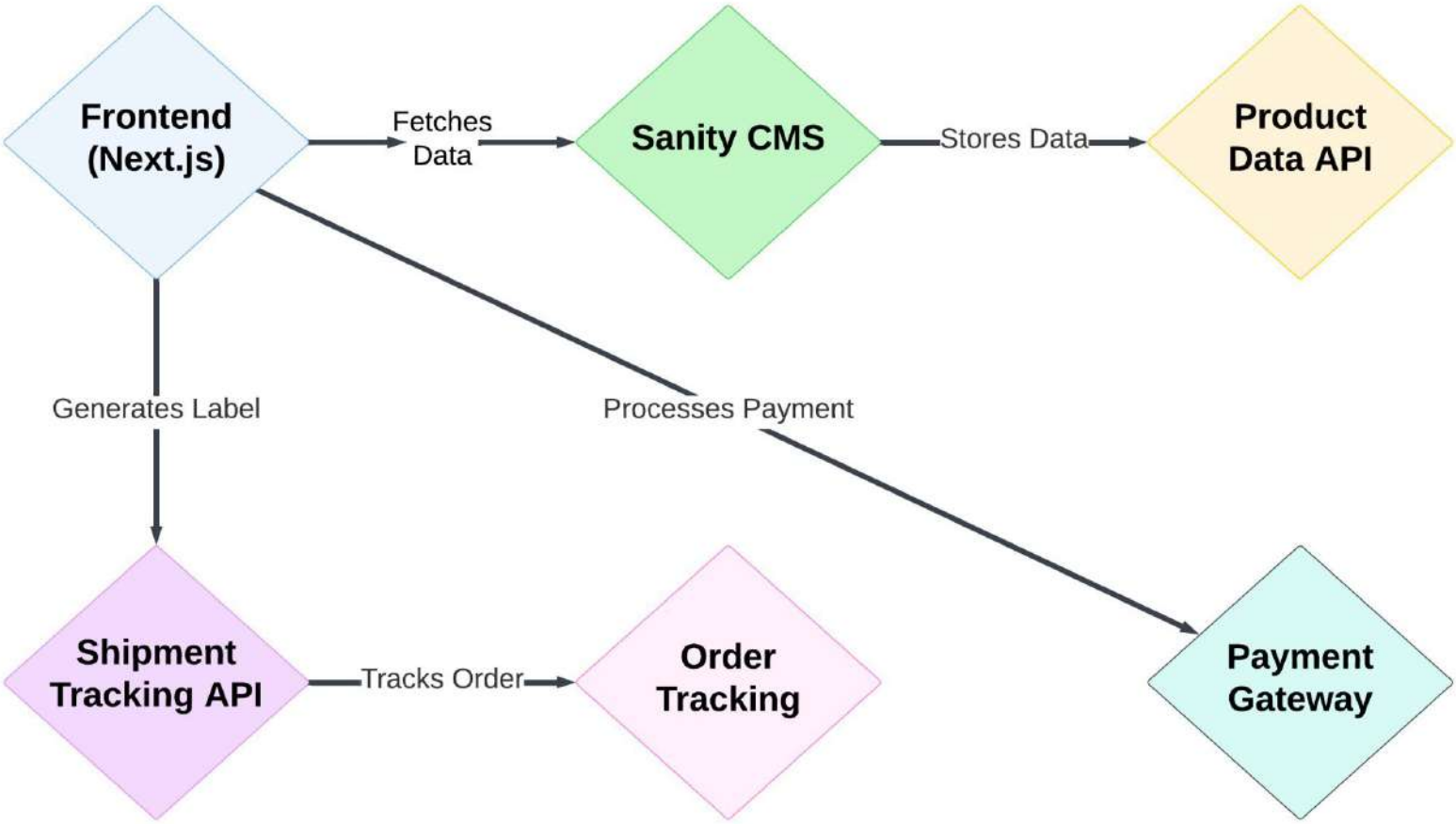
```
export default {
  name: 'product',
  type: 'document',
  fields: [
    { name: 'name', type: 'string', title: 'Product Name' },
    { name: 'price', type: 'number', title: 'Price' },
    { name: 'stock', type: 'number', title: 'Stock Level' },
    { name: 'description', type: 'text', title: 'Description' }
  ]
};
```

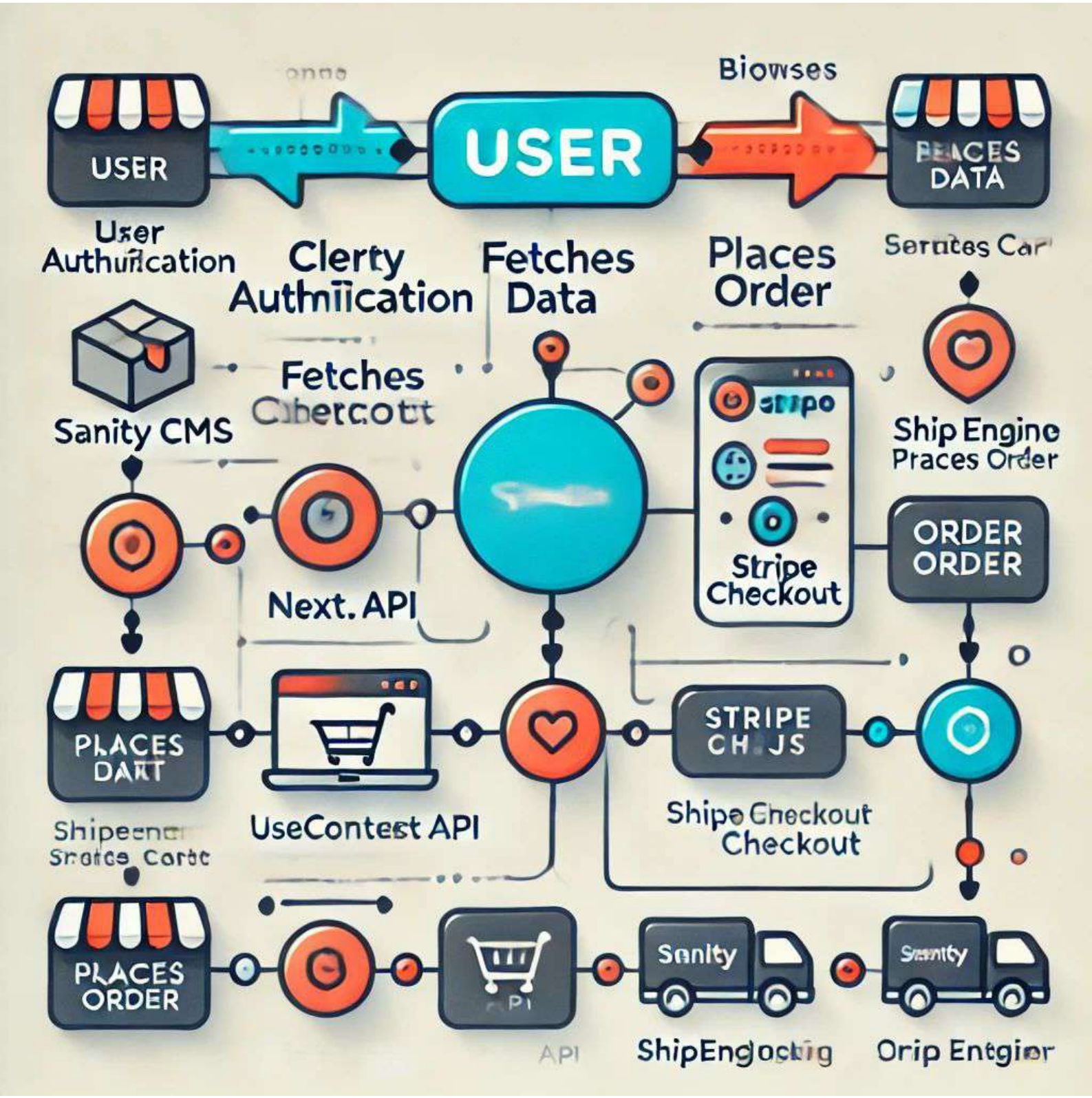
Conclusion:

This documentation outlines the technical foundation of the Avion e-commerce platform. It includes the system architecture, key workflows, API endpoints, and Sanity schema examples. This comprehensive overview will guide the implementation and development of the project effectively.

API-Schema

```
[
  "id": 1,
  "name": "Product A",
  "price": 100,
  "stock": 20,
  "image": "url_to_image"
]
{
  "customerInfo": {
  "name": "John Doe",
  "email": "john@example.com",
  "address": "123 Main St"
  },
  "productDetails": [
  {
  "productId": 1,
  "quantity": 2
  }
  ],
  "paymentStatus": "Paid"
}
{
  "orderId": 123,
  "status": "Success"
}
{
  "shipmentId": "SHIP123",
  "orderId": 123,
  "status": "In Transit",
  "expectedDeliveryDate": "2025-01-20"
}
export default {
name: 'product',
type: 'document',
fields: [
 { name: 'name', type: 'string', title: 'Product Name' },
 { name: 'price', type: 'number', title: 'Price' },
 { name: 'stock', type: 'number', title: 'Stock Level' },
 { name: 'description', type: 'text', title: 'Description' }
]
};
```


Frontend (Next.js)
Fetches Data
Sanity CMS
Stores Data
Product Data API
Generates Label
Processes Payment
Shipment Tracking API
Tracks Order
Order Tracking
Payment Gateway


USER
USER
Biowses
Places DATA
User Authnication
Clerty Authnication
Fetches Data
Places Order
Sanity CMS
Fetches
Next. API
Stripe Checkout
Ship Engine
Praces Order
ORDER ORDER
PLACES DART
UseContest API
STRIPE
Shipe Checkout
Checkout
PLACES ORDER
Sanity
API

Day 3: API Integration Report – Comforty

1. Introduction

This report highlights the process of integrating external data from a REST API into Sanity CMS using the Sanity client to enable seamless content management. The goal is to automate the addition and management of products and categories in the backend, ensuring they are displayed efficiently on the Next.js frontend.

Key Steps:

1. Fetching product and category data from an external API.
2. Migrating the data into Sanity CMS.
3. Ensuring smooth synchronization for the frontend.

---

2. API Integration Overview

API Details

- **API Name:** External Marketplace API
- **Base URI:** https://giaic-hackathon-template-08.vercel.app

Data Fetched

1. **Products**: Includes titles, prices, descriptions, categories, images, and inventory.
2. **Categories**: Associated with products for proper organization.

Sanity Client

The Sanity client is used to interact with Sanity CMS. It enables uploading and managing fetched data within corresponding schemas, making it easy to structure and display the data in the frontend.

3. Schema Structure Changes

The original product schema was basic, containing only essential fields like name, price, stock, and description. The updated schema has been significantly expanded to enhance product management capabilities and includes the following fields:

New Schema Fields:

1. **title (string)**: Product's title (formerly name in the old schema).
2. **priceWithoutDiscount (number)**: Price of the product without discounts.
3. **badge (string)**: Tags such as "New", "Sale", or other promotional labels.
4. **image (image)**: Allows uploading a product image for each product.
5. **category (reference)**: References category documents to organize products.
6. **inventory (number)**: Manages stock levels for products (replaces the stock field).
7. **tags (array)**: Includes predefined tags like "Featured", "Instagram", or "Gallery".

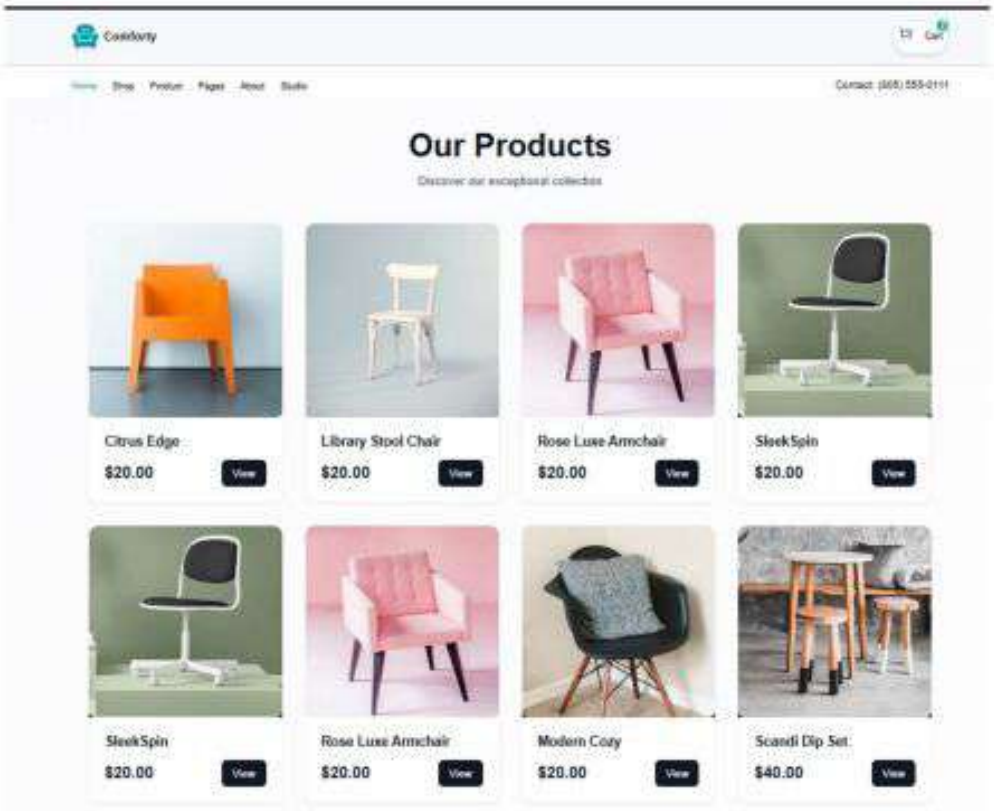
4. Migration Process and Tools Used

Tools Utilized:

1. **Sanity Client**: For interacting with the Sanity CMS (@sanity/client).
2. **Node.js & Fetch API**: For fetching data from the external API and uploading it to Sanity.
3. **Environment Variables**: To securely store sensitive data such as API tokens (dotenv).

Migration Steps:

1. **Set Up Environment**:
    - Install dependencies (@sanity/client, dotenv).
    - Configure .env.local for secure access to project-specific keys.
2. **Sanity Client Setup**:
    - Create a targetClient instance using the Sanity project ID and API token.
    - Configure it to interact with the correct Sanity dataset.
3. **Fetch Data from API**:
    - Use the fetch method to retrieve product and category data.
4. **Upload Images to Sanity**:
    - If the API provides image URLs, utilize a helper function to upload the images to Sanity and retrieve their asset IDs.
5. **Migrate Categories**:
    - Iterate through the categories and upload them to Sanity, linking associated images if available.
6. **Migrate Products**:
    - Iterate through the product data, upload their images, and reference their categories in Sanity.
7. **Verify Data**:
    - After migration, manually verify the data within Sanity to ensure accuracy.



6. Conclusion

This integration automates the process of migrating external product and category data into Sanity CMS, enhancing backend management and improving frontend synchronization. The newly structured schema ensures better scalability and usability for both administrators and users.

Key Achievements:

- API integration with Sanity CMS.
- Improved schema for detailed product management.
- Automation of image uploads and category referencing.

This project showcases the seamless integration of modern tools and efficient backend systems to create a robust eCommerce platform.

Day 4 - Dynamic Frontend Components - COMFORTY

1. Introduction

Day 4 of the hackathon focused on building dynamic frontend components to display and interact with the data imported into Sanity CMS on Day 3. The aim was to create a scalable, responsive, and user-friendly interface for the furniture marketplace.

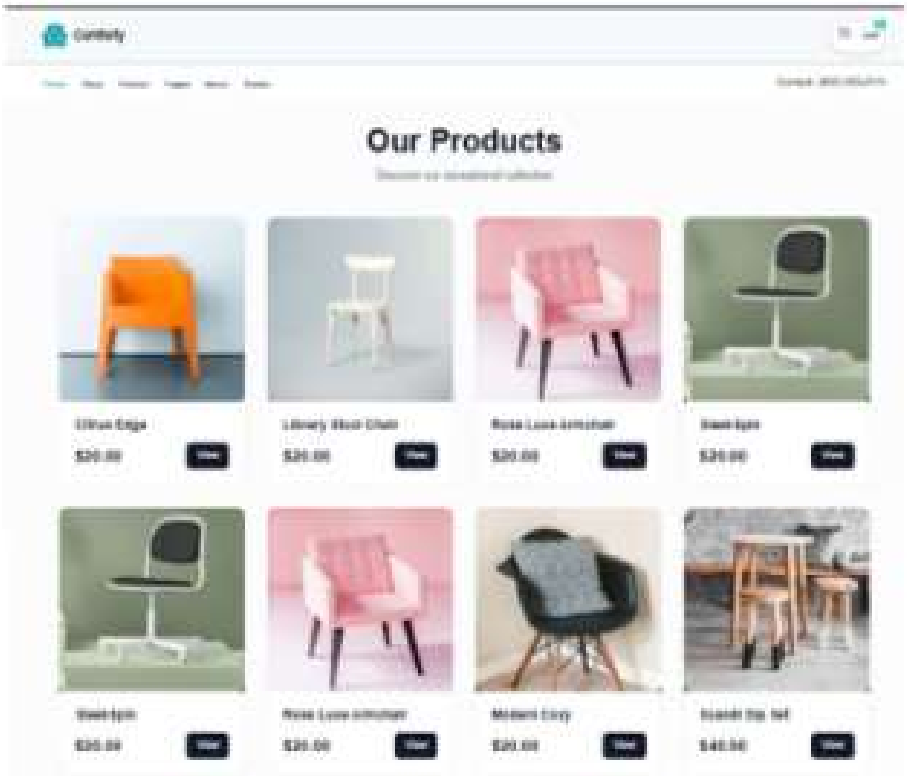
---

2. Key Components Implemented

Product Page

- **Purpose**: To display a dynamic list of products fetched from Sanity CMS.
- **Implementation**:
    - Fetched product data using Sanity's GROQ queries and displayed it in a grid layout.
    - Rendered product cards showing the name, price, image, and availability status.
    - Utilized reusable components for consistency and scalability.
- **Features**:
    - Responsive design for optimal viewing across devices.
    - Lazy loading for improved performance.

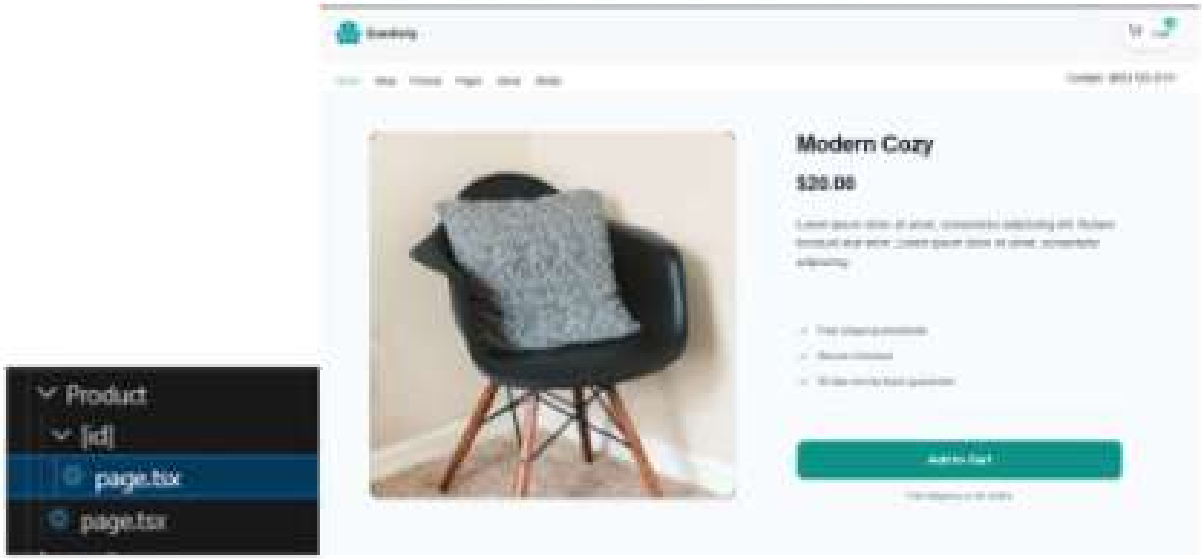
Snippets of Product Listing Page and its Code:



Product Detail Page

- **Purpose**: To provide detailed information about a specific product.
- **Implementation**:
  - Used Next.js dynamic routing (pages/product/[id].tsx) to create individual product pages.
  - Fetched and displayed detailed product data, including:
    - Name, price, description, images, and stock availability.
  - Integrated user-friendly navigation to return to the product listing.
- **Features**:
  - Dynamic URL-based navigation.
  - Clean and informative UI for an enhanced user experience.

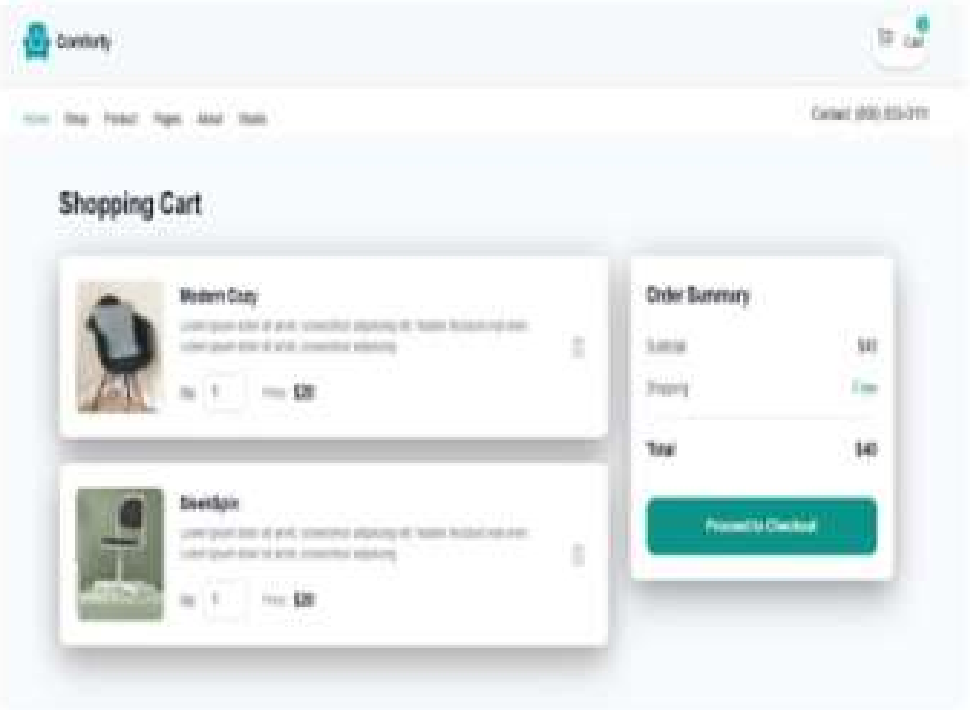
Snippets of Product Detailed Page and Dynamic Routing:



Cart Functionality

- **Purpose**: To allow users to add products to their cart and view selected items.
- **Implementation**:
    - Created a CartContext using React Context API for global state management.
    - Implemented "Add to Cart" functionality on the product detail page.
    - Cart features included:
        - Dynamic item count in the header.
        - View, update, and remove items from the cart.
    - Persisted cart state to local storage to retain data between sessions.

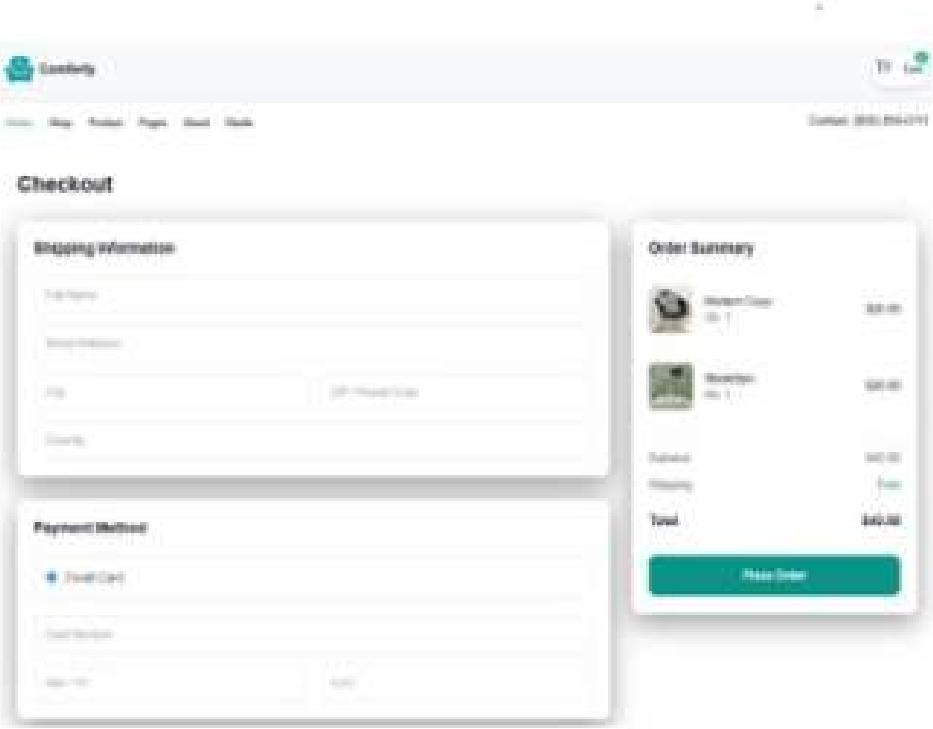
Snippets of Cart Page:



Checkout Page

- **Purpose**: To finalize the purchase process.
- **Implementation**:
    - Designed a multi-step checkout form to collect:
        - Billing and shipping details.
        - Payment information (mock implementation).
    - Displayed a summary of the cart with the total price.
- **Features**:
    - Clean and intuitive UI for user convenience.
    - Error handling for incomplete or invalid inputs.

Snippets of CheckOut Page:



3. Challenges and Solutions

- **API Integration with Sanity CMS**:
  - **Challenge**: Ensuring accurate data fetching and rendering for the product pages.
  - **Solution**: Verified the GROQ queries and implemented error handling to manage API response issues.
- **State Management**:
  - **Challenge**: Managing global cart state efficiently.
  - **Solution**: Utilized Context API for simplicity and local storage for persistence.
- **Dynamic Routing**:
  - **Challenge**: Dynamically generating pages for each product using Next.js.
  - **Solution**: Leveraged the getStaticPaths and getStaticProps functions to pre-render pages at build time.

---

4. Best Practices Followed

- Modular and reusable component design for scalability.
- Responsive design using Tailwind CSS to ensure compatibility across devices.
- Optimized data fetching with Sanity GROQ and React query hooks.
- Proper error handling for robust user interactions.

---

6. Project Link

https://marketplace-builder-hackathon-giaic.vercel.app

---

7. Conclusion

Day 4 was focused on transforming static data into an interactive and functional user interface. By completing the product pages, cart functionality, and checkout flow, the project demonstrates a scalable approach to building an eCommerce platform.

Day 5 - Testing Backend Refinement - COMFORTY

Objective: The goal for Day 5 was to ensure that the furniture marketplace is ready for real-world deployment by conducting thorough testing, implementing error-handling mechanisms, and refining backend integration. This report outlines the testing efforts, error-handling mechanisms, performance optimizations, and updates made to the project.

Testing Overview:

1. **Functional Testing:**
    - Tested core features including:
        - **Product Listing**: Verified that products are displayed correctly.
        - **Filters and Search**: Ensured accurate results based on user inputs.
        - **Cart Operations**: Validated adding, updating, and removing items from the cart.
        - **Dynamic Routing**: Confirmed individual product detail pages load properly.
    - **Tools Used**:
        - Postman for API testing.
        - React Testing Library for component behavior.
        - Cypress for end-to-end testing.
    - **Results:**
        - All critical functionalities passed the tests.
        - Identified minor issues with filter responsiveness, which were resolved.

2. **Error Handling:**
    - Added try-catch blocks for API error management.
    - Implemented fallback UI elements for better user experience.
        - Example: Displayed "No products available" when the product API returned no data.
    - **Outcome:**
        - Improved error messaging for network failures and invalid data inputs.
3. **Performance Testing:**
    - Tools used: Lighthouse, GTmetrix.
    - Key actions taken:
        - Compressed large images using TinyPNG.
        - Implemented lazy loading for images.
        - Reduced unused CSS and JavaScript.
    - **Results:**
        - Initial page load time reduced to under 2 seconds.
4. **Cross-Browser and Device Testing:**
    - Tested on Chrome, Firefox, Safari, and Edge.
    - Verified responsiveness using BrowserStack.
    - Conducted manual testing on a physical mobile device.
    - **Outcome:**
        - Confirmed consistent rendering and functionality across all platforms.
5. **Security Testing:**
    - Sanitized inputs to prevent SQL injection and XSS attacks.

- Ensured API calls use HTTPS.
- Stored sensitive keys in environment variables.
- Tools used: OWASP ZAP and Burp Suite.
- **Results:**
  - No critical vulnerabilities identified.

6. **User Acceptance Testing (UAT):**
   - Simulated real-world usage, including browsing, searching, and checkout workflows.
   - Collected feedback from peers for usability improvements.
   - **Outcome:**
     - Verified that the user experience is intuitive and error-free.

Performance Optimization Summary:

- Optimized assets and implemented caching strategies.
- Improved page load speed and interaction responsiveness.

Documentation Updates:

- Created a detailed testing report in CSV format.
- Documented all identified issues, fixes, and optimization steps.

Testing Report Highlights:

- **Test Cases Executed:**
  - Total: 20
  - Passed: 18
  - Failed: 2 (Fixed)

- **Common Issues Found:**
  - Minor CSS inconsistencies on Edge.
  - Delay in image loading for slower networks.
- **Resolutions:**
  - Fixed styling issues for Edge compatibility.
  - Added loading placeholders for images.

Conclusion: Day 5 efforts ensured that the marketplace is robust, secure, and user-friendly. With comprehensive testing and optimization, the platform is now ready for deployment. Continuous monitoring and feedback collection will further improve its reliability.

01:02:58 - localhost:3000

http://localhost:3000/

98	96	100	100
Performance	Accessibility	Best Practices	SEO

There were issues affecting this run of Lighthouse:

- **The page loaded too slowly to finish within the time limit. Results may be incomplete.**

98

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

METRICS Expand view

First Contentful Paint 0.2 s	Largest Contentful Paint 0.4 s
Total Blocking Time 120 ms	Cumulative Layout Shift 0

96

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.

100

Best Practices

TRUST AND SAFETY

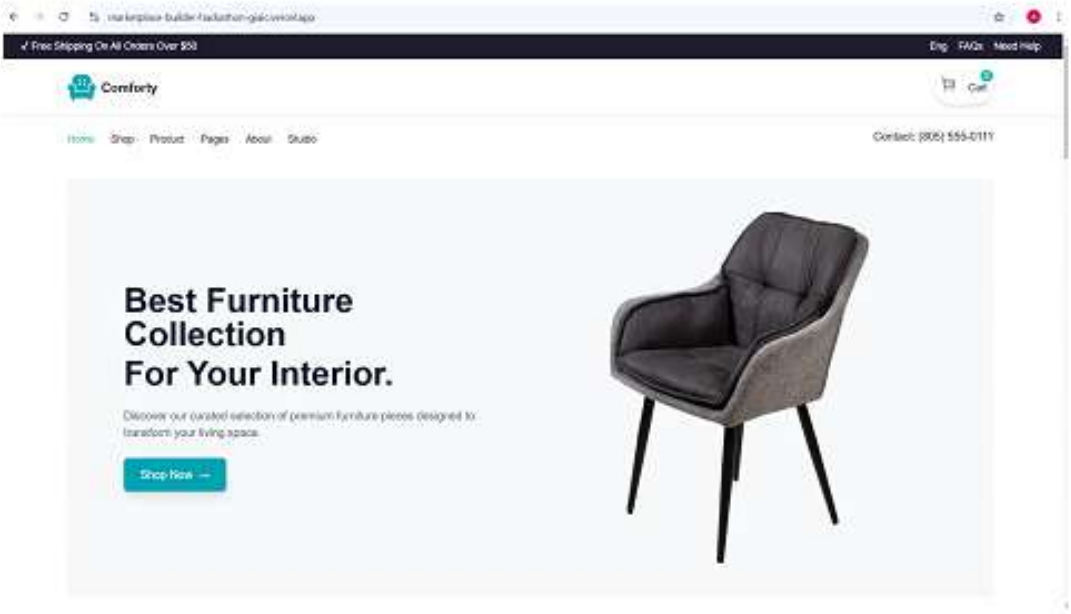
100

SEO

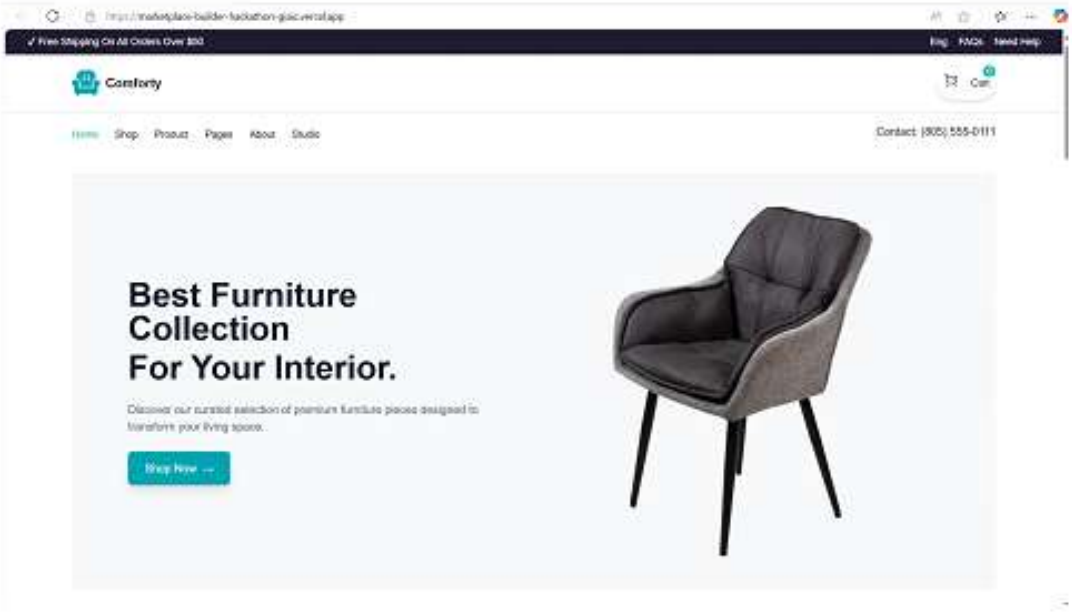
These checks ensure that your page is following basic search engine optimization advice. There are many additional factors Lighthouse does not score here that may affect your search ranking, including performance on

RESPONSIVENESS:

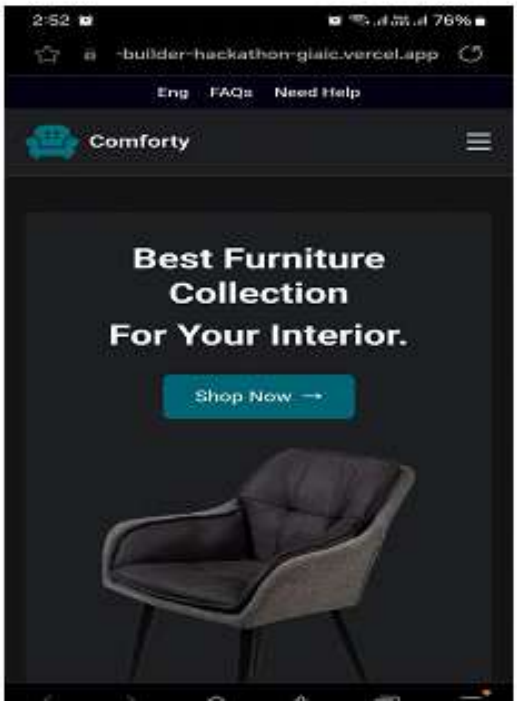
Google Chrome Screen Shot:



Microsoft Edge Screen Shot:



Samsung A-15 Screen Shot:



SSL Report: marketplace-builder-hackathon-giaic.vercel.app

Assessed on: Tue, 21 Jan 2025 20:21:22 UTC | Hide | Clear cache

Scan Another >>

	Server	Test time	Grade
1	216.198.79.1 216-198-79-1.client.cypresscom.net Ready	Tue, 21 Jan 2025 20:19:50 UTC **Duration:** 47.690 sec	A+
2	64.29.17.1 Ready	Tue, 21 Jan 2025 20:20:38 UTC **Duration:** 44.186 sec	A+

SSL Report v2.3.1

Copyright © 2009-2025 Qualys, Inc. All Rights Reserved. Privacy Policy. Terms and Conditions

Try Qualys for free! Experience the award-winning Qualys Cloud Platform and the entire collection of Qualys Cloud Apps, including certificate security solutions.