Name :Ramesha javed

Sunday 2-5

Day 3 - API Integration Report

[General E-com website]

Objective

The focus of Day 3 was on API integration, data migration into Sanity CMS, and building functional features for the dress shopping marketplace backend. Additionally, blog content and comment functionality were integrated using APIs to enhance the user experience, achieving Hackathon Day 3 milestones.

API Integration Process

To integrate APIs into the Next.js project and Sanity CMS, the following steps were performed:

1. API Selection:

- **API Used:** Template 0
- **API URL:** https://template6-six.vercel.app/api/products
- Tested API endpoints in **Postman** to understand responses and structure.

2. Integration Steps:

- Installed `axios` for HTTP requests.
- Developed a utility function to fetch data from the API with error handling.
- Used `getServerSideProps` in Next.js for server-side rendering to improve performance and SEO.

Code Snippet: API Call

```
import axios from 'axios';


export async function fetchProducts() {
```

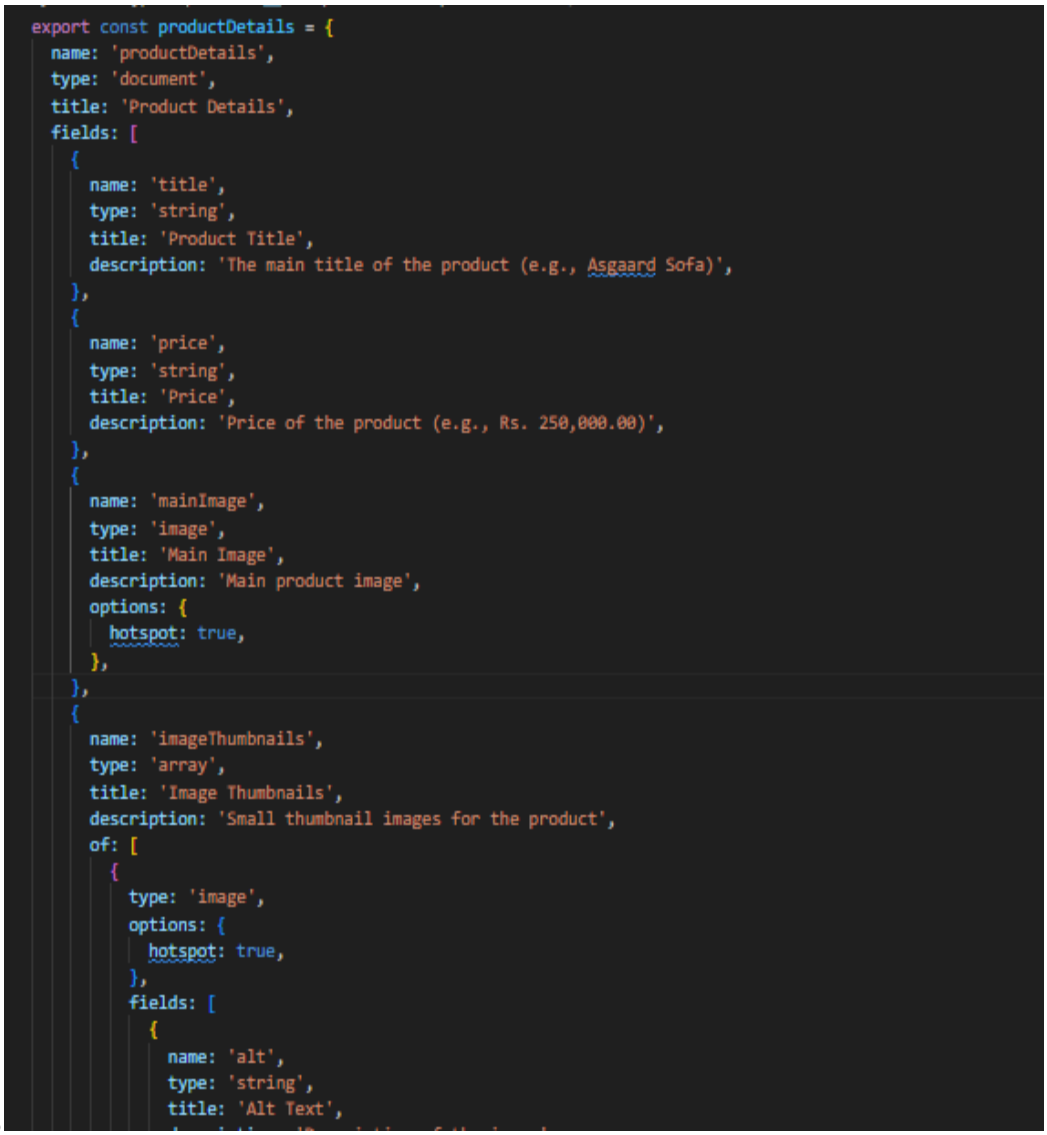
```
  try {
    const response = await axios.get(process.env.API_URL);
    return response.data;
  } catch (error) {
    console.error('Error fetching products:', error);
    return [];
  }
}
```

Sanity CMS Schema Adjustments

To support the API data structure, schema adjustments were made in the **Sanity CMS** project.

Adjustments Made:

1. **Added Categories Schema**:
   - **Path:** `sre > sanity > schemaTypes > TS categories.ts`
   - Included fields for `name` and `description`.
2. **Updated Product Schema:**
   - Added fields for `price`, `category`, `images`, `size`, and `tags`.
   - Ensured validation for required fields



```
export const productDetails = {
  name: 'productDetails',
  type: 'document',
  title: 'Product Details',
  fields: [
    {
      name: 'title',
      type: 'string',
      title: 'Product Title',
      description: 'The main title of the product (e.g., Asgaard Sofa)',
    },
    {
      name: 'price',
      type: 'string',
      title: 'Price',
      description: 'Price of the product (e.g., Rs. 250,000.00)',
    },
    {
      name: 'mainImage',
      type: 'image',
      title: 'Main Image',
      description: 'Main product image',
      options: {
        hotspot: true,
      },
    },
    {
      name: 'imageThumbnails',
      type: 'array',
      title: 'Image Thumbnails',
      description: 'Small thumbnail images for the product',
      of: [
        {
          type: 'image',
          options: {
            hotspot: true,
          },
          fields: [
            {
              name: 'alt',
              type: 'string',
              title: 'Alt Text',
```

Code Snippet: Updated Product Schema

Data Migration Steps and Tools Used

The following steps were taken to migrate data into Sanity CMS programmatically:

1. Preparation:

- Fetched data from the API to understand fields.
- Installed `dotenv` to manage environment variables securely.

2. Migration Process:

- Created a script named `migrate.mjs` in the **scripts** folder.
- Used the `@sanity/client` library to populate data into Sanity.

Added a migration command in `package.json` for automation:
"migrate": "node scripts/migrate.mjs"

-

Installed dependencies:
npm install dotenv @sanity/client

-

Ran the migration script:
npm run migrate

-

Code Snippet: Data Migration Script

```
import sanityClient from '@sanity/client';

import axios from 'axios';


const client = sanityClient({

  projectId: 'your_project_id',

  dataset: 'production',

  token: process.env.SANITY_TOKEN,

  useCdn: false,

});


async function migrateData() {

  try {

    const response = await axios.get(process.env.API_URL);

    const products = response.data.map(item => ({

      _type: 'product',
```

```
      name: item.name,
      price: item.price,
      category: { _type: 'reference', _ref: item.categoryId },
      size: item.size,
      images: item.images.map(img => ({
        _type: 'image',
        asset: { _ref: img.assetRef },
      })),
    }));

    for (const product of products) {
      await client.create(product);
    }
    console.log('Data migration successful!');
  } catch (error) {
    console.error('Error during migration:', error);
  }
}

migrateData();
```

Blog Content and Comment Functionality

Blog Content:

- **Schema Creation:** Added a new `blog` schema with fields for `title`, `content`, `author`, and `publishedDate`.

- **Fetching Blogs:** Used GROQ queries to retrieve blogs and display them on the frontend.

Comment Integration:

- Integrated a comment system using an API.
- Created `comment` schema in Sanity with fields for `name`, `email`, `commentText`, and `postId`.
- Comments are fetched dynamically and displayed below each blog post.

Code Snippet: Fetching Comments

Key Features Developed

1. **Product Management:**
    - CRUD operations on product listings using Sanity CMS.
    - Categorized products into `Men`, `Women`, and `Kids` sections.
2. **Shopping Cart:**
    - Users can add, update, and remove products.
    - Cart total dynamically updates.
3. **Blog System:**
    - Blogs displayed with real-time fetching from Sanity CMS.
    - Comments powered by API integration.

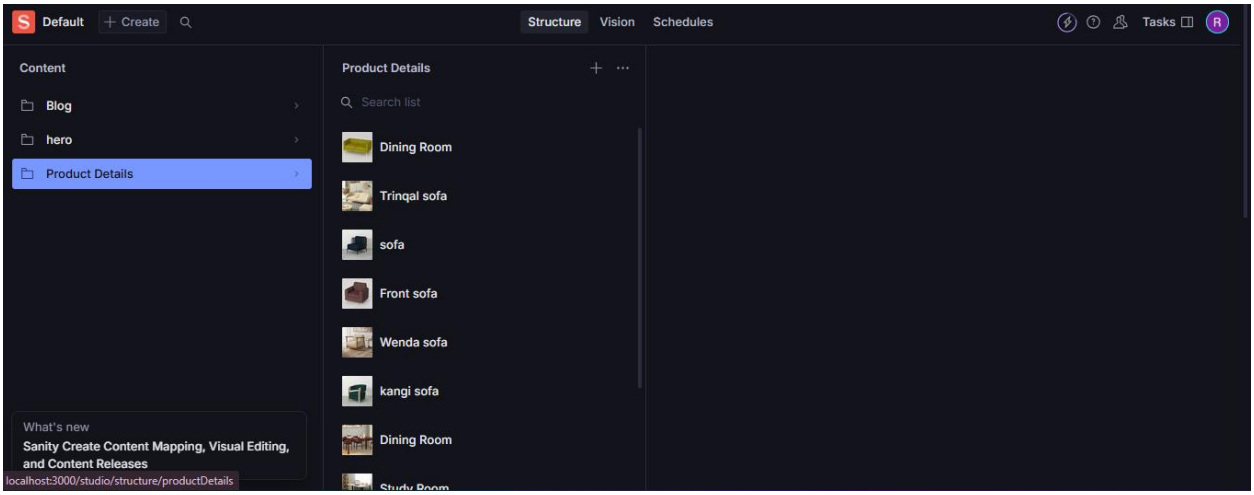
Error Handling

1. **Migration Errors:**
    - Resolved token mismatch by regenerating the Sanity API token.
    - Fixed schema discrepancies by updating field types.
2. **Dependency Issues:**
    - Downgraded to Next.js 14 as `dotenv` was incompatible with Next.js 15.

Best Practices Followed

- **Environment Variables:** Stored sensitive data securely in `.env`.
- **Code Modularity:** Encapsulated logic into reusable functions.
- **Validation:** Ensured data alignment with schemas.
- **Documentation:** Maintained detailed notes for each milestone.
- **Version Control:** Regular commits with meaningful messages.

1-



Conclusion

The API integration and data migration processes were successfully completed, achieving Hackathon Day 3 milestones. The inclusion of blog and comment functionality enriched the project further. The project now supports real-world use cases like product listing, shopping cart management, and user interaction through blogs and comments.