Day 3 - API Integration Report

[Hekto: Building the Future of Furniture Shopping]

Overview:

This report outlines the API integration process, schema adjustments, data migration steps, and the implementation of frontend rendering for the marketplace project. The document includes step-by-step explanations, screenshots, and code snippets to ensure a comprehensive understanding of the tasks performed.

API Integration Process

Step 1: Fetching Data from API

- **API URL:** https://next-ecommerce-template-4.vercel.app/api/product
- **Objective:** Retrieve product data including name, image, price, description, category, discount percentage, stock level, and featured product status.
- **Method:** Axios GET request.
- **Code snippet:**

```
const response = await axios.get("https://next-ecommerce-template-4.vercel.app/api/product");
const products = response.data.products;
```

Step 2: Setting up Sanity Client

- **Objective:** Connect to the Sanity CMS for data storage.
- **Environment Variables:**
    - NEXT_PUBLIC_SANITY_PROJECT_ID
    - NEXT_PUBLIC_SANITY_DATASET
    - SANITY_API_TOKEN

- **Sanity Client Initialization:**

```
const client = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
  token: process.env.SANITY_API_TOKEN,
  apiVersion: '2025-01-15',
  useCdn: false,
});
```

Step 3: Schema Design in Sanity

- **Schema Name: productData**
- **Fields:**
    - **Name (string, required)**
    - **Image (image, hotspot enabled)**
    - **Price (string, required)**
    - **Description (text, max 150 characters)**
    - **Discount Percentage (number, 0-100 range)**
    - **Is Featured Product (boolean)**
    - **Stock Level (number, non-negative)**
    - **Category (string, predefined options)**
- **Code Snippet:**

```
export default {
  name: 'productData',
  type: 'document',
  title: 'Product Data',
  fields: [
    { name: 'name', type: 'string', title: 'Name', validation: (Rule) => Rule.required() },
    { name: 'image', type: 'image', title: 'Image', options: { hotspot: true } },
    { name: 'price', type: 'string', title: 'Price', validation: (Rule) => Rule.required() },
    { name: 'description', type: 'text', title: 'Description', validation: (Rule) =>
Rule.max(150) },
    { name: 'discountPercentage', type: 'number', title: 'Discount Percentage', validation:
(Rule) => Rule.min(0).max(100) },
    { name: 'isFeaturedProduct', type: 'boolean', title: 'Is Featured Product' },
   { name: 'stockLevel', type: 'number', title: 'Stock Level', validation: (Rule) =>
Rule.min(0) },
```

```
   { name: 'category', type: 'string', title: 'Category', options: { list: [{ title: 'Chair',
value: 'Chair' }, { title: 'Sofa', value: 'Sofa' }] }, validation: (Rule) => Rule.required() },
  ],
};
```

Data Migration Steps

Step 4: Image Upload to Sanity

- **Objective: Upload product images from the API to Sanity CMS.**
- **Method:**
    - **Use Axios to fetch image data.**
    - **Upload image data as a Sanity asset.**
- **Code Snippet:**

```
async function uploadImageToSanity(imageUrl) {
  const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
  const buffer = Buffer.from(response.data);
  const asset = await client.assets.upload('image', buffer, { filename:
imageUrl.split('/').pop() });
  return asset._id;
}
```

Step 5: Uploading Data to Sanity

- **Objective: Store the product data in the `productData` schema.**
- **Process:**
    - **Loop through each product from the API.**
    - **Map API data to Sanity schema fields.**
    - **Create new entries in Sanity.**
- **Code Snippet:**

```
async function importData() {
  for (const item of products) {
    // Upload image if available and get the image reference
```

```
    const imageRef = item.imagePath ? await uploadImageToSanity(item.imagePath) : null;

    // Structure the sanityItem object
    const sanityItem = {
      _type: 'productData',
      name: item.name,
      category: item.category || null,
      price: item.price,
      description: item.description || '',
      discountPercentage: item.discountPercentage || 0,
      stockLevel: item.stockLevel || 0,
      isFeaturedProduct: item.isFeaturedProduct,
      image: imageRef
        ? {
            _type: 'image',
            asset: {
              _type: 'reference',
              _ref: imageRef,
            },
          }
        : undefined,
    };

    // Log the product data being imported
    console.log("Products: ", sanityItem);

    // Create a new entry in Sanity
    const result = await client.create(sanityItem);
  }
}
```

Frontend Rendering

Step 6: Fetching Data from Sanity

- **Objective: Retrieve data stored in Sanity CMS for rendering.**
- **Method: Use GROQ queries.**
- **Example Query:**

```
const query = `*[_type == "productData"]{ name, price, description, image, category }`;
const products = await client.fetch(query);
```

Step 7: Rendering Data on the Frontend

- **Objective: Display product details dynamically on the web page.**

- **Method: Map fetched data to frontend components.**
- **Code Example:**

```
products.map((product) => (
  <div key={product.name}>
    <h2>{product.name}</h2>
    <p>{product.price}</p>
    <img src={product.image.asset.url} alt={product.name} />
  </div>
));
```

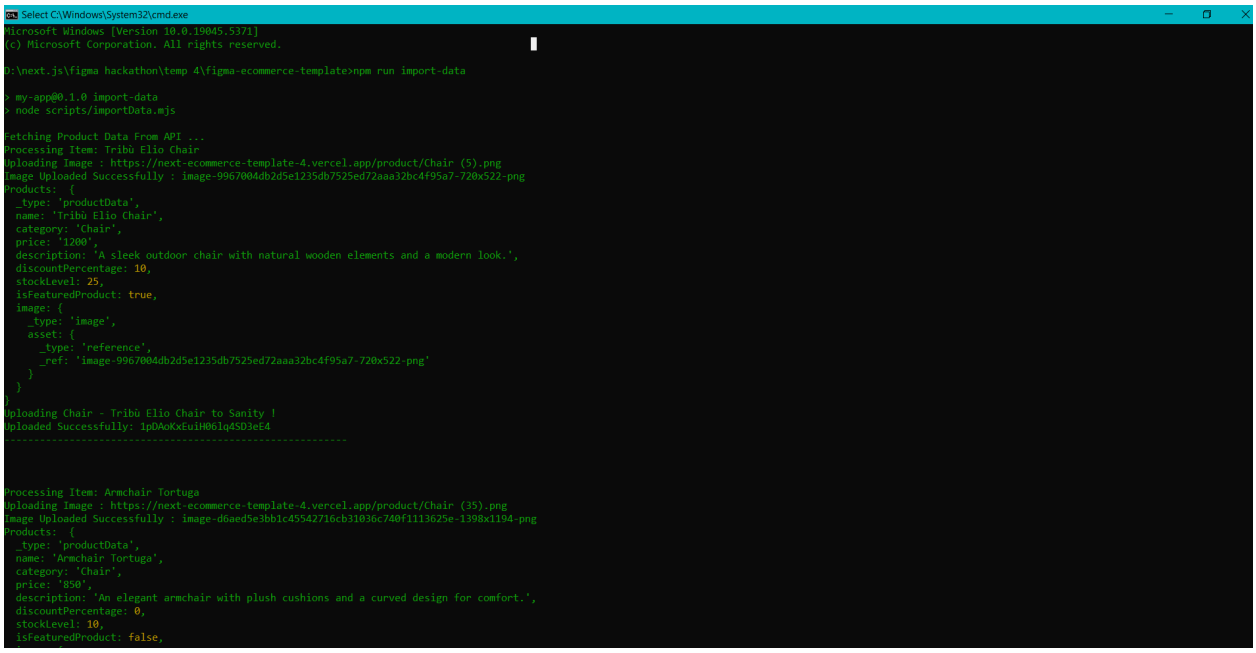
Tools Used

- **Sanity CMS**: Used for content management.
- **Axios**: Used to make HTTP requests and fetch product data from the external API.
- **dotenv**: Used to manage environment variables for Sanity project configuration.
- **Node.js**: Used to run the server-side code for API calls and data migration.

Visual Representation of Process

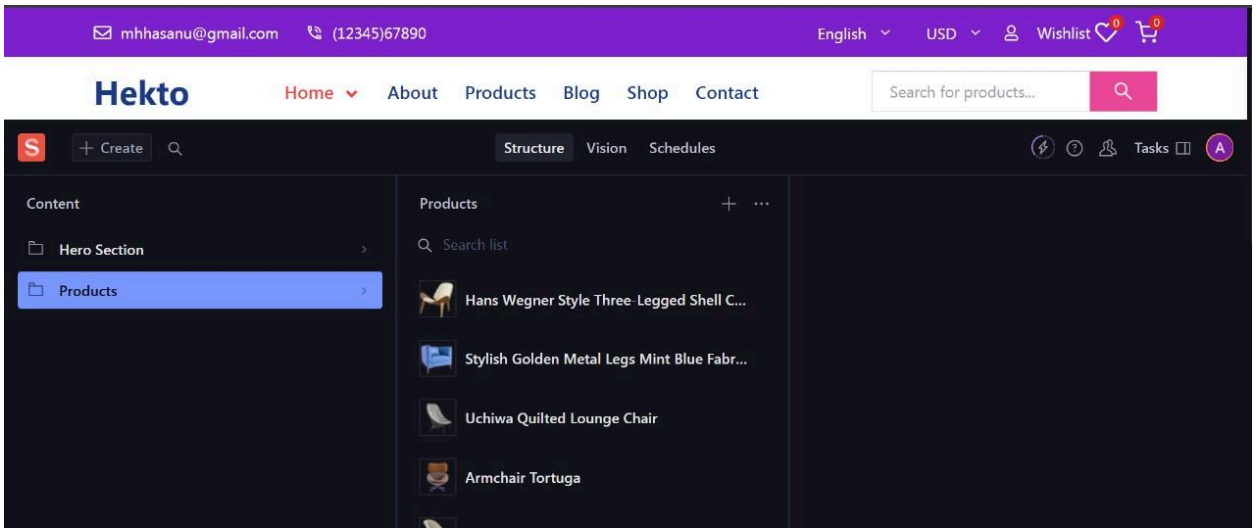
1. API Calls

A screenshot of the API call used to fetch product data from the external API:



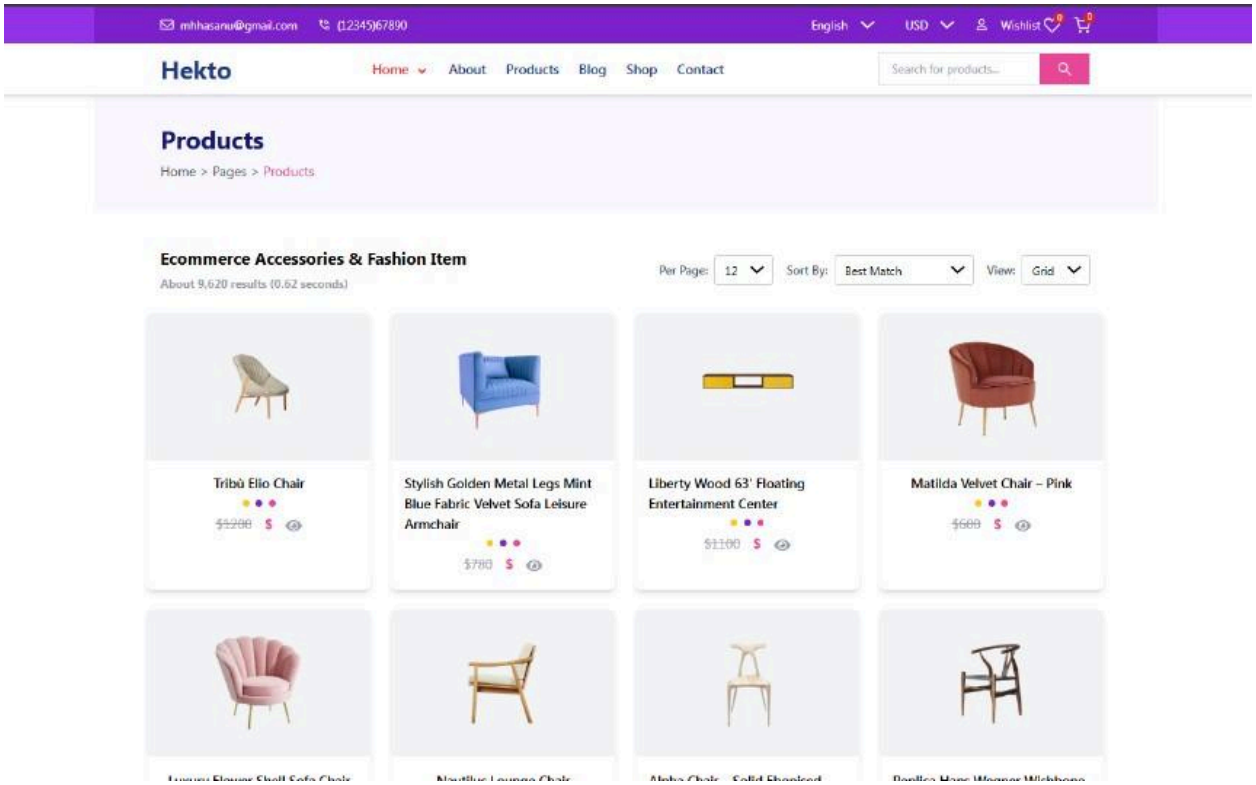
2. Populated Sanity CMS Fields

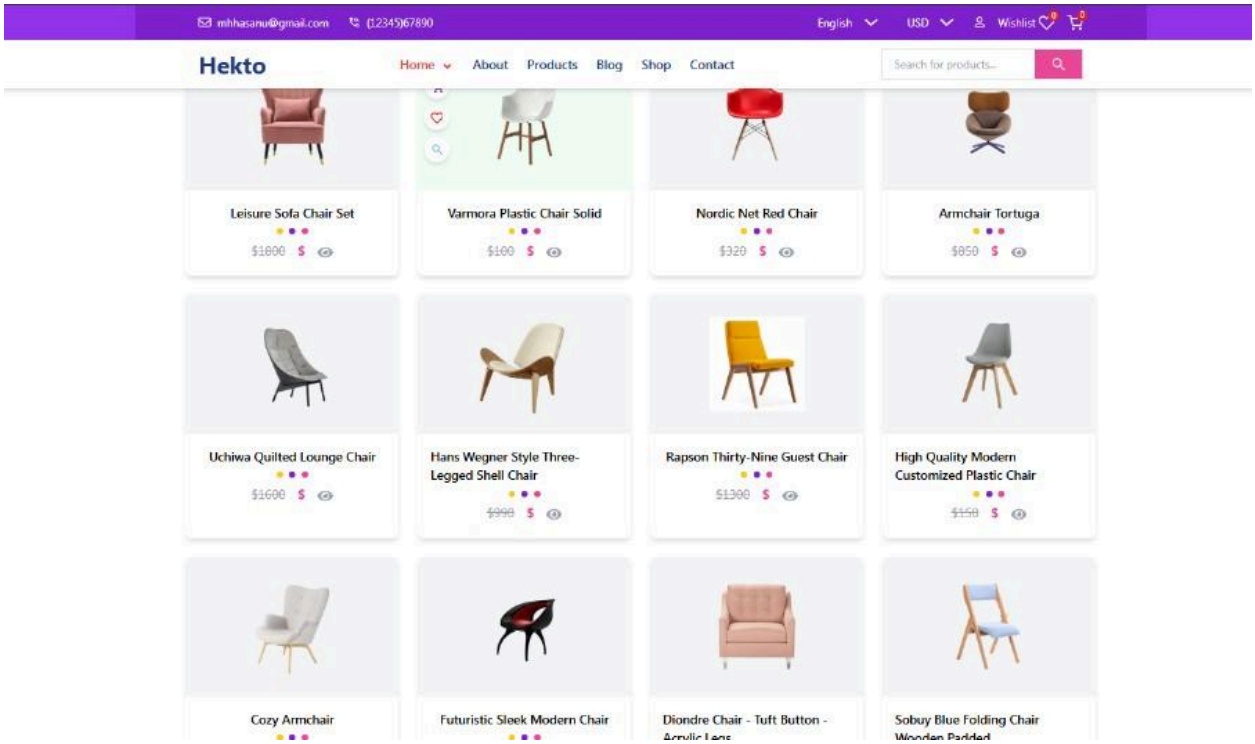
A screenshot of the Sanity CMS interface showing the populated product data:



3. Data Successfully Displayed in the Frontend

A screenshot showing the products being displayed on the frontend after fetching and integrating the data from Sanity:





Conclusion

This report provides a detailed walkthrough of the API integration, schema adjustments, data migration, and frontend rendering processes. The outlined steps ensure a robust and efficient implementation. Further improvements can include error handling, performance optimization, and additional features like search and filtering.

Day 3 Checklist

Task	Status
API Understanding	[✔]
Schema Validation	[✔]
Data Migration	[✔]
API Integration in Next.js	[✔]
Submission Preparation	[✔]