Day3 API INTEGRATION

API Integration Process

• API EndPoint Setup

1- Products Endpoint: https://giaic-hackathon-template-08.vercel.app/api/products

- Provides product data with fields such as productName, category, price, inventory, colors, status, description, and image.

2- Categories Endpoint: https://giaic-hackathon-template-08.vercel.app/api/categories

- Provides category data with fields such as title, images, products

Library Used: Fetch API was used to fetch data from the API.

• Fetch Data From API

The API call retrieves an array of product objects from the endpoint, ensuring proper data handling and structure.

Code Snippet:

```
export default function Home() {
  const { products, setProducts } = useProducts();
  const { setCategories } = useCategories();
  useEffect(() => {
    (async function () {
      try {
        let query = await client.fetch(
          `*[_type == "products"]{_id, name, description, category, price, discountPercent, colors , 'image':image.
          asset->url, sizes, isNew}`
        );

        const productsArr: Product[] = query.map((product: any) => {
          product.slug = stringToSlug(product.name);
          return product;
        });
        console.log(productsArr);

        setProducts(productsArr);
        query = await client.fetch(
          `*[_type == 'category']{_id, name, 'image':image.asset->url, productsCount}`
        );
        console.log(query);
        setCategories(query);
      } catch (error) {
        throw new Error("Error in fetch");
      }
    })();
  }, []);
```

1. Product Schema

```jsx
export default defineType({
  name: "product",
  title: "Products",
  type: "document",
  fields: [
    {
      name: "name",
      title: "Name",
      type: "string",
    },
    {
      name: "price",
      title: "Price",
      type: "number",
    },
    {
      name: "description",
      title: "Description",
      type: "text",
    },
    {
      name: "image",
      title: "Image",
      type: "image",
    },
    {
      name: "category",
      title: "Category",
      type: "string",
      options: {
        list: [
```

```jsx
export default defineType({
  fields: [
      options: {
        list: [
          { title: "Jeans", value: "jeans" },
          { title: "Hoddie", value: "hoodie" },
          { title: "Shirt", value: "shirt" },
        ],
      },
    },
    {
      name: "discountPercent",
      title: "Discount Percent",
      type: "number",
    },
    {
      name: "new",
      type: "boolean",
      title: "New",
    },
    {
      name: "colors",
      title: "Colors",
      type: "array",
      of: [{ type: "string" }],
    },
    {
      name: "sizes",
      title: "Sizes",
      type: "array",
      of: [{ type: "string" }],
```

Migration Setup and Tool Used

1. Environment Setup:

- Installed required dependencies: @sanity/client, dotenv.
- Created a .env.local file to securely store environment variables

2. Data Fetching:

- Retrieved product data from the API endpoint using the Fetch API.
- Parsed and logged the data to confirm its structure and integrity.

3. Image Upload:

- Downloaded images from the API's image field using the Fetch API.
- Uploaded images to Sanity's Asset Manager using the Sanity client.

```
async function uploadImageToSanity(imageUrl: any) {
  try {
    console.log(`Uploading image: ${imageUrl}`);

    const response = await fetch(imageUrl);
    if (!response.ok) {
      throw new Error(`Failed to fetch image: ${imageUrl}`);
    }

    const buffer = await response.arrayBuffer();
    const bufferImage = Buffer.from(buffer);

    const asset = await client.assets.upload("image", bufferImage, {
      filename: imageUrl.split("/").pop(),
    });

    console.log(`Image uploaded successfully: ${asset._id}`);
    return asset._id;
  } catch (error) {
    console.error("Failed to upload image:", imageUrl, error);
    return null;
  }
}
```

Tool Used:

- Sanity Client: For interacting with the Sanity CMS.
- Fetch API: For making HTTP requests to fetch API data and images.
- Dotenv: To load environment variables from .env.local.

ScreenShot:

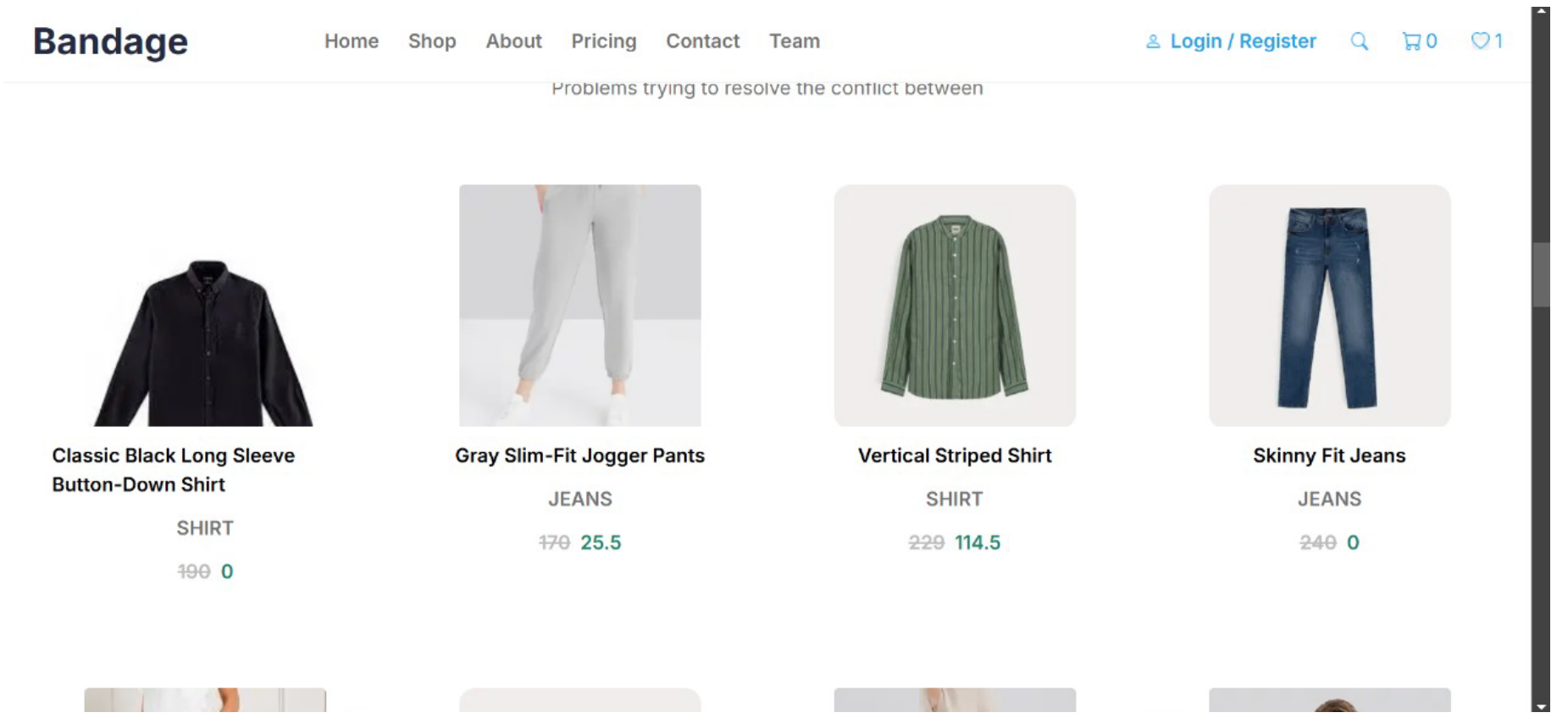
1. API Call Output

- Output showing the retrieved product data.


```
  _id: 'tiTonAuk6WGSOb8ZgZziE9',
  _rev: 'tiTonAuk6WGSOb8ZgZziDO',
  _type: 'product',
  _updatedAt: '2025-01-18T05:16:15Z',
  category: 'tshirt',
  description: 'Classic Polo Shirt\n' +
    '\n' +
    'Upgrade your wardrobe with this timeless classic polo shirt, perfect for any occasion. Crafted from premium-quality fabric, it offers a
 soft, breathable, and comfortable fit that lasts all day. Featuring a stylish collar, button placket, and short sleeves, this versatile pol
o blends elegance with a casual touch.\n' +
    '\n' +
    'Key features:\n' +
    '\n' +
    'Made with durable and lightweight material\n' +
    'Available in a variety of colors and sizes\n' +
    'Easy to pair with jeans, chinos, or shorts for a polished look\n' +
    'Ideal for work, casual outings, or weekend wear\n' +
    'Stay effortlessly stylish with this must-have polo shirt!',
  discountPercent: 0,
  image: {
    _type: 'image',
    asset: {
      _ref: 'image-fb62308b5fa5f6c0d5af2b40e5be28065c42454d-295x298-png'
    }
  },
  isNew: true,
  name: 'Classic Polo Shirt ',
  price: 180
```

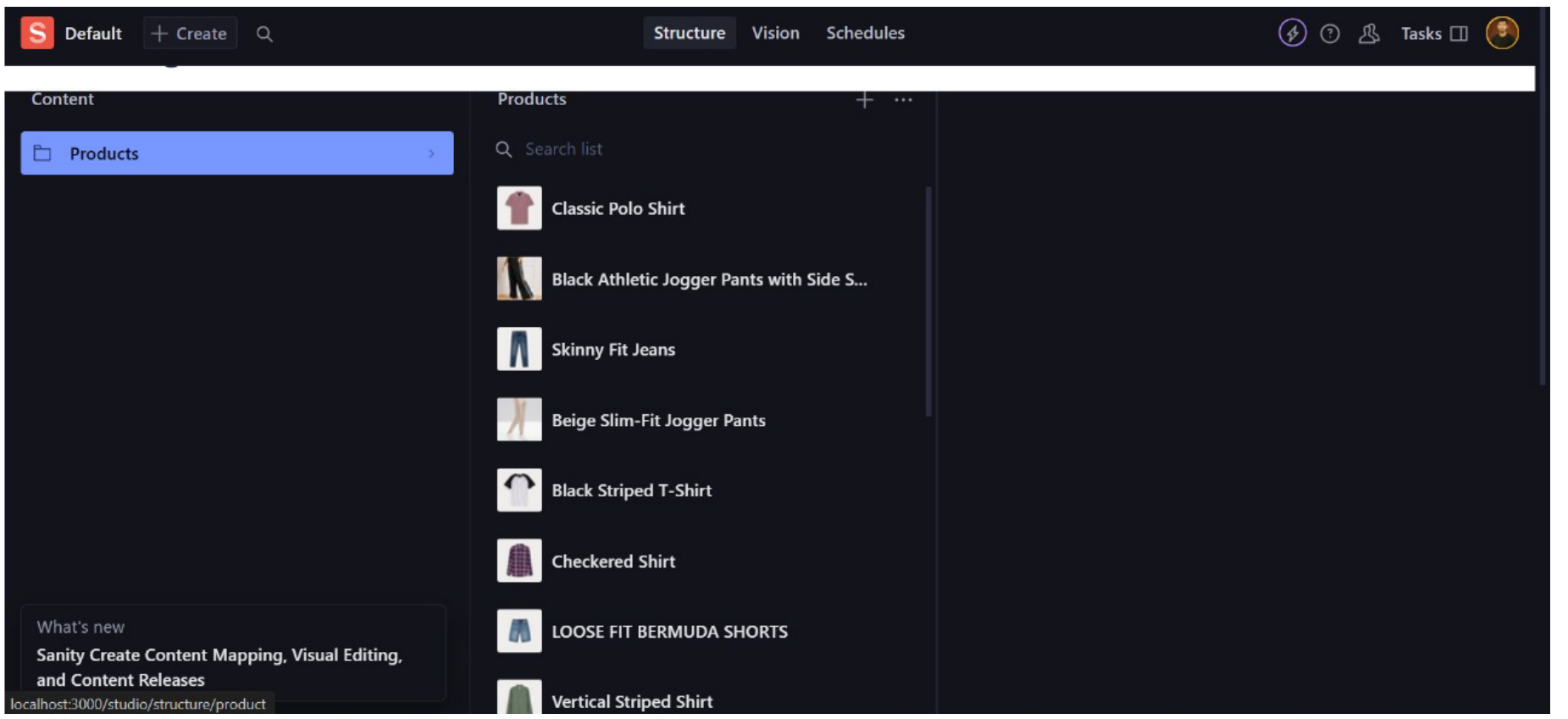
2. Frontend Display

- The data displayed on the front-end.



3. Populated Sanity CMS Fields

- Sanity's CMS populated with the product and image data.



MigrationScript:

importData.ts M ✕ | importData.js M

```
187  const client = createClient({
191    apiVersion: "2025-01-13",
192    token:
193      "skSLvGe6hhNjnYXFanlqu2cCSkCD10Rre401wM8I7mhdUjVCR5rUnIKHnW1zFNiJJ2mQ7Mhtt2C7LIwN7dDTyZ6fSV2xjIILanY0gXZ9hVuclYv
         Q68hdn7RG6WCAxdmobsDHzQlAyEQYrdTMj0ASBkvbFExN9ZZsvhPYSu7iXgvElLGkaocV",
194  });
195
196  async function uploadImageToSanity(imageUrl: any) {
197    try {
198      console.log(`Uploading image: ${imageUrl}`);
199
200      const response = await fetch(imageUrl);
201      if (!response.ok) {
202        throw new Error(`Failed to fetch image: ${imageUrl}`);
203      }
204
205      const buffer = await response.arrayBuffer();
206      const bufferImage = Buffer.from(buffer);
207
208      const asset = await client.assets.upload("image", bufferImage, {
209        filename: imageUrl.split("/").pop(),
210      });
211
212      console.log(`Image uploaded successfully: ${asset._id}`);
213      return asset._id;
214    } catch (error) {
215      console.error("Failed to upload image:", imageUrl, error);
216      return null;
217    }
218  }
219
```

CheckList:

Self-Validation Check-list:

API Understanding:

- ✔

Schema Validation:

- ✔

Data Migration:

- ✔

API Integration in Next.js:

- ✔

Submission Preparation:

- ✔