

## Day 3 - API Integration Report – Online Grocery store

### a. Setting Up Sanity.io

#### 1. Initialize Sanity Studio:

- Install the Sanity CLI: `npm install -g @sanity/cli`
- Run `sanity init` to initialize a new Sanity project.

#### 2. Configure CORS:

- Add the Next.js development URL and production URL to the CORS settings in the Sanity dashboard.

### b. Adding API in Next.js

#### 1. Install Dependencies:

2. `npm install @sanity/client`

#### 3. Initialize Sanity Client:

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

5.

6. `const client = sanityClient({`

7.  `projectId: 'your-project-id',`

8.  `dataset: 'your-dataset',`

9.  `useCdn: true, // `false` if you want to ensure fresh data`

10.  `apiVersion: '2023-01-01', // Use the latest ISO date`

11. `});`

12.

13. `export default client;`

#### 14. Fetch Data from API:

- Use the groq query language to fetch data.
- Example query:
- `import client from './sanityClient';`
- 
- `export async function getData() {`
- `const query = `*[_type == 'post']`;`
- `const posts = await client.fetch(query);`
- `return posts;`
- `}`

#### 15. Display Data in Next.js:

- Use `useEffect` or `useState` for data rendering

---

## 2. Adjustments Made to Schemas

### 1. Define Schemas:

```
src > sanity > TS products > product > fields
1 import { defineType } from "sanity";
2
3 export const product = defineType({
4   name: "product",
5   title: "Product",
6   type: "document",
7   fields: [
8     {
9       name: "title",
10      title: "Title",
11      // validation: (rule) => rule.required(),
12      type: "string",
13    },
14    {
15      name: "description",
16      type: "text",
17      // validation: (rule) => rule.required(),
18      title: "Description",
19    },
20    {
21      name: "productImage",
22      type: "image",
23      // validation: (rule) => rule.required(),
24      title: "Product Image",
25    },
26    {
27      name: "price",
28      type: "number",
29      // validation: (rule) => rule.required(),
30      title: "Price",
31    },
32    {
33      name: "tags",
```

```
34      name: "tags",
35      type: "array",
36      title: "tags",
37      of: [{ type: "string" }],
38    },
39    {
40      name: "discountPercentage",
41      type: "number",
42      title: "Discount Percentage",
43    },
44    {
45      name: "isNew",
46      type: "boolean",
47      title: "New Badge",
48    },
49  ],
50 });
```

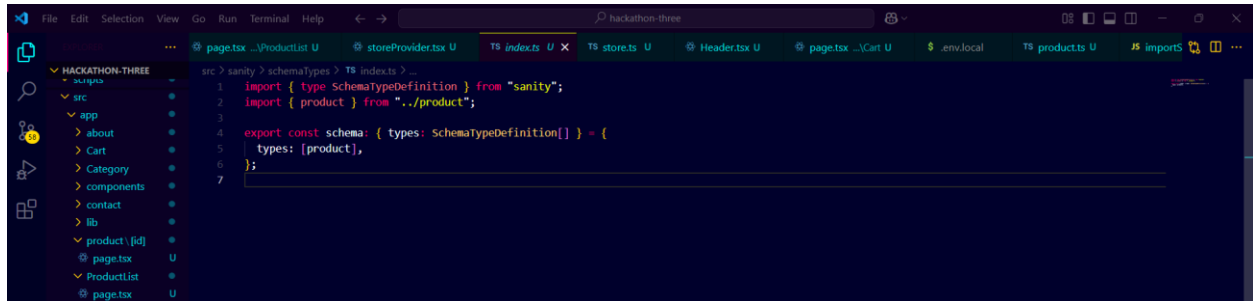
## 2. Testing Adjustments:

- Test new fields using the Sanity Studio interface.
- Verify correct data fetching in Next.js.

### 3. Migration Steps and Tools

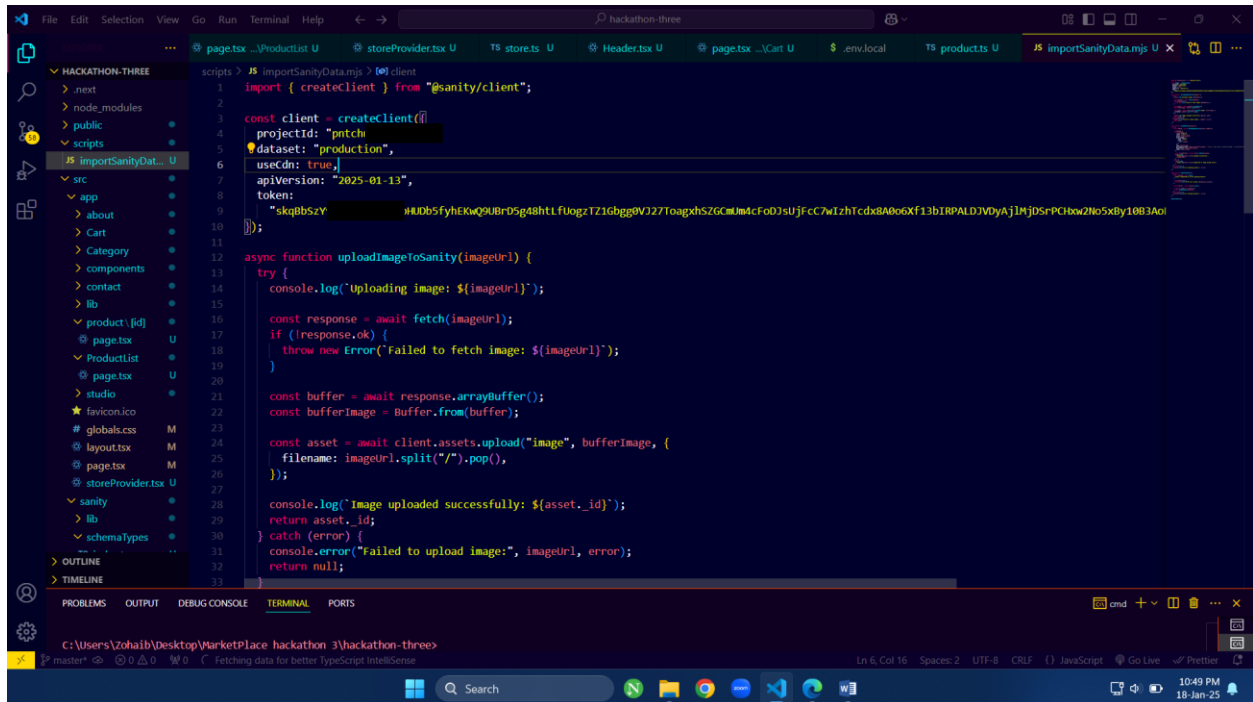
#### 1. Schema Updates:

- Use `sanity schema update` to deploy schema changes.
- Validate the schema locally before deploying.



```
src > sanity > schemaTypes > TS index.ts > ...
1 import { type SchemaTypeDefinition } from "sanity";
2 import { product } from "../product";
3
4 export const schema: { types: SchemaTypeDefinition[] } = {
5   types: [product],
6 };
7
```

#### 2. Data Migration:



```
scripts > JS importSanityData.mjs > @client
1 import { createClient } from "@sanity/client";
2
3 const client = createClient({
4   projectId: "pntch",
5   dataset: "production",
6   useCdn: true,
7   apiVersion: "2025-01-13",
8   token:
9     "skqBbszYHUdb5fyhEKwQ9UBr05g48htLfUogzrTZ1Gbgg0V27ToagxhS7GcmImdcFoD3sUjfcC7w1zhTcdx8A0e6Xf13bIRPALD3VDyAjjMJDSrPCHow2No5xBy10B3Apl";
10 });
11
12 async function uploadImageToSanity(imageUrl) {
13   try {
14     console.log("Uploading image: ${imageUrl}");
15
16     const response = await fetch(imageUrl);
17     if (!response.ok) {
18       throw new Error("Failed to fetch image: ${imageUrl}");
19     }
20
21     const buffer = await response.arrayBuffer();
22     const bufferImage = Buffer.from(buffer);
23
24     const asset = await client.assets.upload("image", bufferImage, {
25       filename: imageUrl.split("/").pop(),
26     });
27
28     console.log("Image uploaded successfully: ${asset._id}");
29     return asset._id;
30   } catch (error) {
31     console.error("Failed to upload image:", imageUrl, error);
32     return null;
33   }
34 }
```

The screenshot shows the VS Code editor with the file `importSanityData.mjs` open. The code defines an `uploadProduct` function that takes a `product` object and attempts to upload it to Sanity. It uses `await uploadImageToSanity` to get an `imageId`, then constructs a document object with fields like `_type`, `title`, `price`, `productImage`, `tags`, `discountPercentage`, `description`, and `isNew`. The document is then created using `client.create`. A `catch` block handles errors.

```
35 import { client } from './client';
36
37 async function uploadProduct(product) {
38   try {
39     const imageId = await uploadImageToSanity(product.imageUrl);
40
41     if (imageId) {
42       const document = {
43         _type: "product",
44         title: product.title,
45         price: product.price,
46         productImage: {
47           _type: "image",
48           asset: {
49             _ref: imageId,
50           },
51         },
52         tags: product.tags,
53         discountPercentage: product.discountPercentage, // Typo in field name: discountPercentage -> discountPercentage
54         description: product.description,
55         isNew: product.isNew,
56       };
57
58       const createdProduct = await client.create(document);
59       console.log(
60         `Product ${product.title} uploaded successfully:`,
61         createdProduct
62       );
63     } else {
64       console.log(
65         `Product ${product.title} skipped due to image upload failure.`
66       );
67     }
68   } catch (error) {
69     console.error("Error uploading product:", error);
70   }
71 }
```

The screenshot shows the VS Code editor with the file `importSanityData.mjs` open. The code defines an `importProducts` function that fetches products from a remote API and uploads them. It uses `await fetch` to get the products, then iterates over them and calls `uploadProduct` for each. A `catch` block handles errors.

```
72
73 async function importProducts() {
74   try {
75     const response = await fetch(
76       "https://template6-six.vercel.app/api/products"
77     );
78
79     if (!response.ok) {
80       throw new Error(`HTTP error! Status: ${response.status}`);
81     }
82
83     const products = await response.json();
84
85     for (const product of products) {
86       await uploadProduct(product);
87     }
88   } catch (error) {
89     console.error("Error fetching products:", error);
90   }
91 }
92
93 importProducts();
```


### 3. Verify Migration:

- Query the updated dataset to ensure data integrity.


### 4. Tools Used:

- **Postman:** To test API endpoints.


# Display Data on frontend




Cloud Haven Chair  
Rs. 230.00




Bright Space  
Rs. 180.00




Tropical Vibe  
Rs. 550.00




Cloud Haven Chair  
Rs. 230.00




Bright Space  
Rs. 180.00




Serene Seat  
Rs. 350.00




Nordic Elegance  
Rs. 280.00




Modern Serenity  
Rs. 480.00



Serene Seat  
Rs. 350.00



Nordic Elegance  
Rs. 280.00



Modern Serenity  
Rs. 480.00

## Fronend code:

```
"use client";
import { useState, useEffect } from "react";
import Link from "next/link";
import { client } from "@sanity/lib/client";
import imageUrlBuilder from "@sanity/image-url";

export default function Home() {
  const [products, setProducts] = useState<any[]>([]);

  const builder = imageUrlBuilder(client);
  const urlFor = (source: any) => builder.image(source);

  useEffect(() => {
    const fetchData = async () => {
      try {
        // Fetching products from Sanity
        const sanityData = await client.fetch(`
          *[_type == "product"]{
            _id,
            title,
            description,
            price,
            discountPercentage,
            isNew,
            tags,
            productImage{
              asset->{
                _id,
                url
              },
              alt
            }
          }
        `);

        // Setting state for products
        setProducts(sanityData);
      } catch (error) {
        console.error("Error fetching data:", error);
      }
    };

    fetchData();
  });
}
```

```
}, []); // Empty dependency array means this effect runs only once after the initial render
```

```
return (  
  <main>  
    {/* Displaying the products in grid format */  
    <div className="grid grid-cols-1 lg:grid-cols-3 gap-5 my-14 px-5 md:px-0">  
      {products.map((product: any) => (  
        <Link href={` /product/${product._id}`} key={product._id}>  
          <div className="border p-4 h-full rounded-lg shadow-md flex justify-center items-center flex-col">  
            {product.productImage?.asset?.url && (  
              <img  
                src={urlFor(product.productImage.asset)  
                  .width(400) // Adjust this size if needed  
                  .url()}  
                alt={product.productImage.alt || "Product Image"}  
                className="w-full max-h-[400px] object-scale-down mb-4"  
              />  
            )}  
            <h2 className="font-semibold text-lg text-center">  
              {product.title}  
            </h2>  
            <p className="text-green-600 font-bold mb-2">  
              Rs. {product.price.toFixed(2)}  
            </p>  
            {product.discountPercentage && (  
              <p className="text-red-500 text-sm">  
                {product.discountPercentage}% OFF  
              </p>  
            )}  
          </div>  
        </Link>  
      )})  
    </div>  
  </main>  
>);  
}
```

## Migration Script code :

```
import { createClient } from "@sanity/client";

const client = createClient({
  projectId: "my id here",
  dataset: "production",
  useCdn: true,
  apiVersion: "2025-01-13",
  token:
    "my token here",
});

async function uploadImageToSanity(imageUrl) {
  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;
  }
}

async function uploadProduct(product) {
  try {
    const imageId = await uploadImageToSanity(product.imageUrl);

    if (imageId) {
      const document = {
        _type: "product",
        title: product.title,
```



```

    price: product.price,
    productImage: {
      _type: "image",
      asset: {
        _ref: imageId,
      },
    },
    tags: product.tags,
    dicountPercentage: product.dicountPercentage, // Typo in field name:
dicountPercentage -> discountPercentage
    description: product.description,
    isNew: product.isNew,
  };

```

```

    const createdProduct = await client.create(document);
    console.log(
      `Product ${product.title} uploaded successfully:`,
      createdProduct
    );
  } else {
    console.log(
      `Product ${product.title} skipped due to image upload failure.`
    );
  }
} catch (error) {
  console.error("Error uploading product:", error);
}
}

```

```

async function importProducts() {
  try {
    const response = await fetch(
      "https://template6-six.vercel.app/api/products"
    );

    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }

    const products = await response.json();

    for (const product of products) {
      await uploadProduct(product);
    }
  } catch (error) {

```

```
    console.error("Error fetching products:", error);
  }
}

importProducts();
```

### Sanity Schema code :

```
import { defineType } from "sanity";

export const product = defineType({
  name: "product",
  title: "Product",
  type: "document",
  fields: [
    {
      name: "title",
      title: "Title",
      // validation: (rule) => rule.required(),
      type: "string",
    },
    {
      name: "description",
      type: "text",
      // validation: (rule) => rule.required(),
      title: "Description",
    },
    {
      name: "productImage",
      type: "image",
      // validation: (rule) => rule.required(),
      title: "Product Image",
    },
    {
      name: "price",
      type: "number",
      // validation: (rule) => rule.required(),
      title: "Price",
    },
    {
      name: "tags",
```

```
    type: "array",
    title: "Tags",
    of: [{ type: "string" }],
  },
  {
    name: "dicountPercentage",
    type: "number",
    title: "Discount Percentage",
  },
  {
    name: "isNew",
    type: "boolean",
    title: "New Badge",
  },
],
});
```