

“Day 3 Integration Report”

Api Integration and Data migration

Table of Contents

1. Setting Up Environment Variables
2. Obtaining Sanity Project ID and API Token
3. Creating the Sanity Schema
4. Setting Up the Data Import Script
5. Running the Import Script

1. Setting Up Environment Variables

First, let's set up our environment variables. Create a ``.env.local`` file in your project root if it doesn't already exist. Add the following variables:

```
NEXT_PUBLIC_SANITY_PROJECT_ID=your_project_id
NEXT_PUBLIC_SANITY_DATASET=production
SANITY_API_TOKEN=your_sanity_token
```

Remember, variables prefixed with `NEXT_PUBLIC_` will be exposed to the browser, so be cautious about what you prefix.

2. Obtaining Sanity Project ID and API Token

Project ID

To find your Sanity project ID:

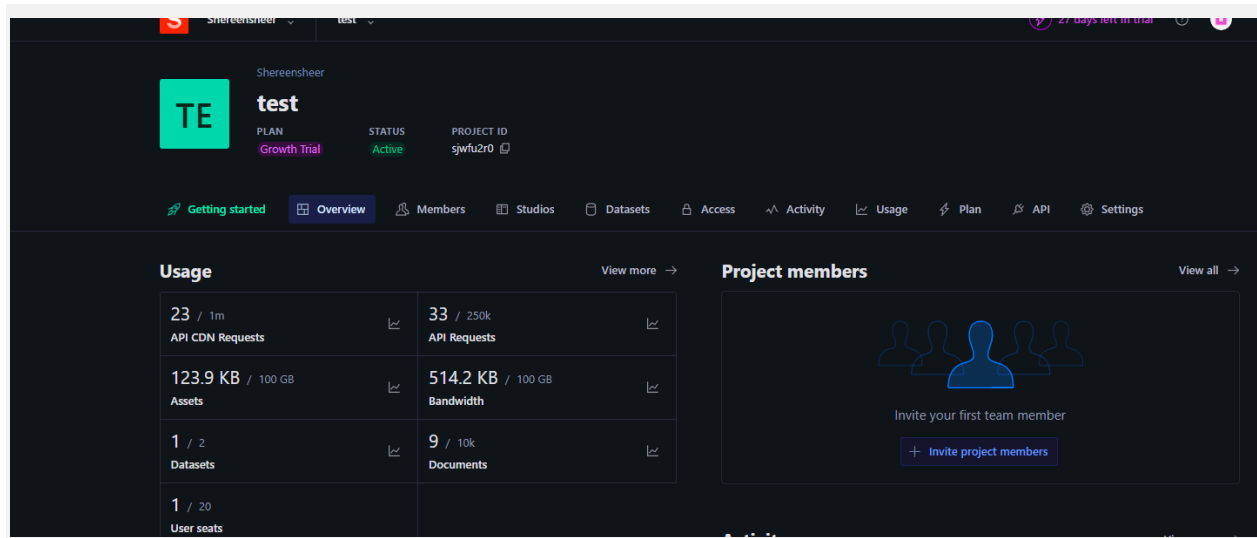
1. Log in to your Sanity account at

<https://www.sanity.io/manage>

2. Select your project

3. In the project dashboard, you'll see the project ID listed

Use this ID for the `NEXT_PUBLIC_SANITY_PROJECT_ID`



Use the `NEXT_PUBLIC_SANITY_PROJECT_ID` in your `.env.local` file.

API Token

To generate a Sanity API token:

1. Go to <https://www.sanity.io/manage> and select your project
2. Navigate to the "API" tab
3. Under "Tokens," click "Add API token"
4. Give your token a name and select the appropriate permissions (usually "Editor" for full read/write access)

5. Copy the generated token

The screenshot displays the Sanity Studio interface. The top navigation bar includes a sidebar with 'Getting started', 'Overview', 'Members', 'Studios', 'Datasets', 'Access', 'Activity', 'Usage', 'Plan', 'API', and 'Settings'. The 'API' tab is selected, highlighted by a blue arrow. The left sidebar shows 'Webhooks', 'CORS origins', and 'Tokens', with 'Tokens' highlighted by a blue arrow. The main content area is divided into two sections: 'CORS origins' and 'Tokens'.

CORS origins

Hosts that can connect to the project API.

ORIGIN	CREDENTIALS	CREATED
http://localhost:3000	Allowed	3 days
http://localhost:3333	Allowed	3 days

Tokens

Tokens are used to authenticate apps and scripts to access project data.

NAME	PERMISSIONS	CREATED
Editor Token	Editor	3 days

The bottom status bar shows 'SANITY' and 'ALL SYSTEMS OPERATIONAL'.

Tokens

Tokens are used to authenticate apps and scripts to access project data.

Name

Examples: "Employee import", "Website preview" or "PDF generator".

revalidate

Permissions

Choose the access privileges for the token.

☐ Contributor

Read and write access to draft content within all datasets, with no access to project settings. (Tokens: read+write drafts)

☐ Deploy Studio (Token only)

Access to deploy Sanity Studio and GraphQL APIs to our hosted service.

☐ Developer

Read and write access to all datasets, with access to project settings for developers. (Tokens: read+write)

☒ Editor

Read and write access to all datasets, with limited access to project settings. (Tokens: read+write)

☐ Viewer

Read access to all datasets, with limited access to project settings. (Tokens: read-only)

Save

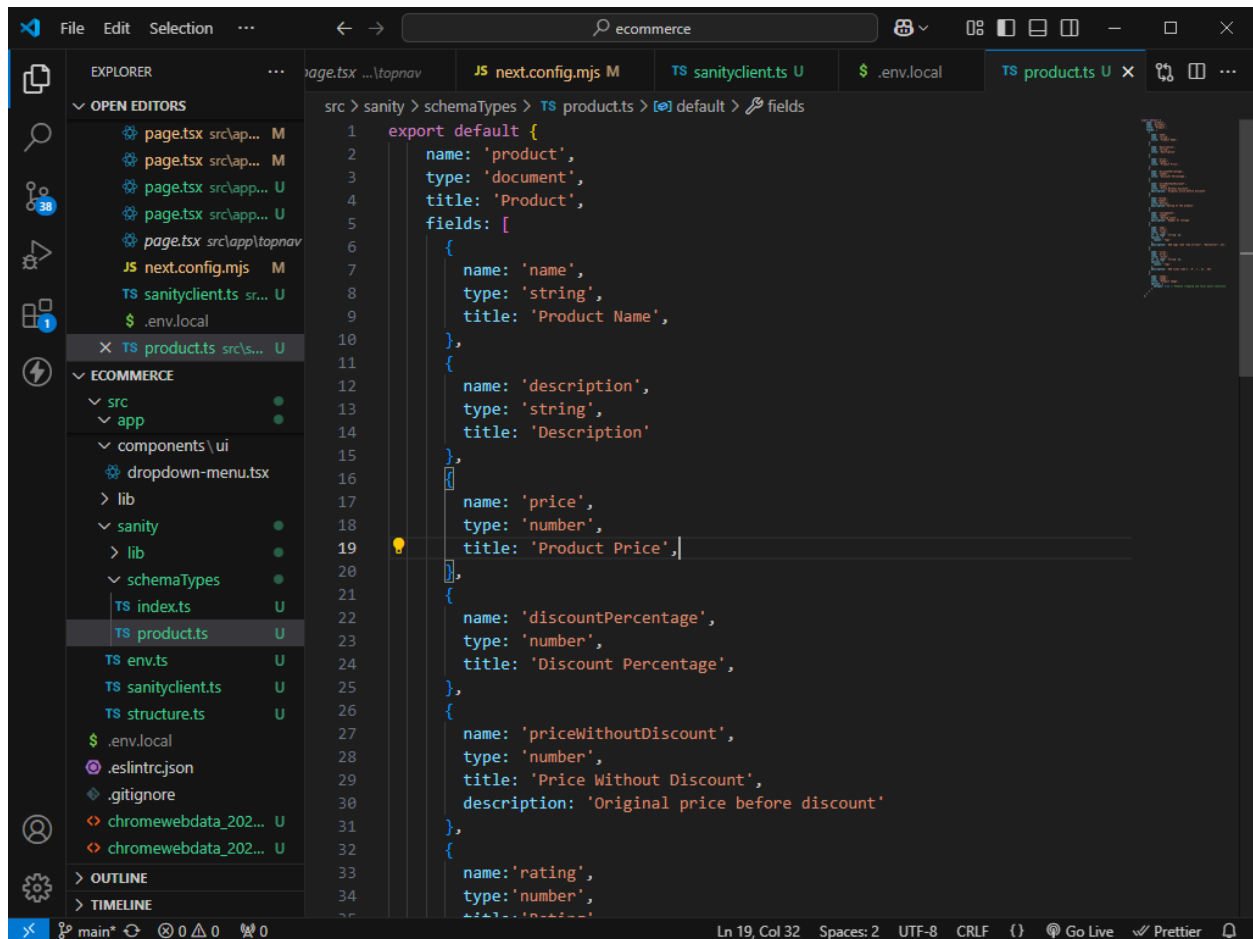
Cancel

The screenshot shows the GitHub 'Tokens' page. At the top, there's a 'Webhooks' tab and a '+ Add API token' button. Below, a section titled 'Tokens' explains their purpose. A table lists a token named 'revalidate' with 'Editor' permissions, created 'just now'. A blue arrow points to the 'Copy' icon next to the token value, which is partially obscured by a blue scribble. The token value is a long alphanumeric string.

Use this token for the `SANITY_API_TOKEN` in your `.env.local` file.**3.**

Creating the Sanity Schema

Now, let's create a schema for our products. In your Sanity schema folder (usually `sanity/schemaTypes`), create a new file called `product.ts`:



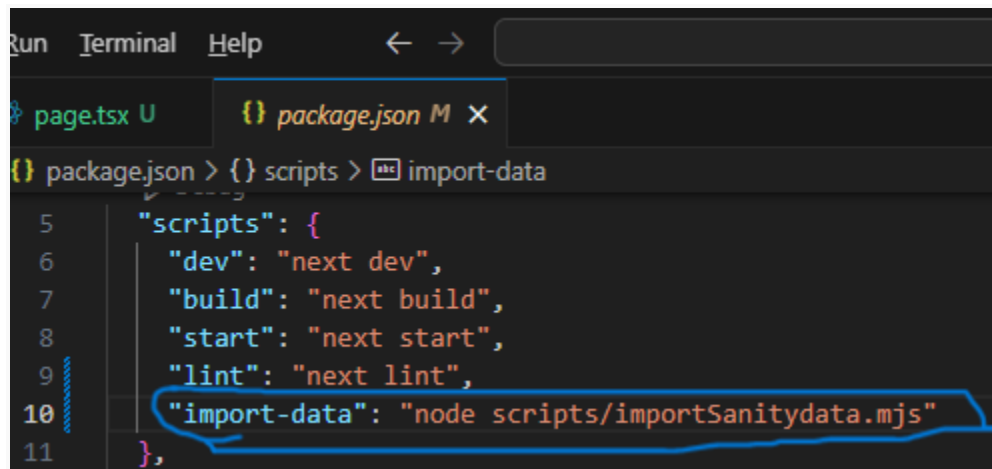
The screenshot shows a VS Code editor with a project named 'ecommerce'. The Explorer sidebar on the left shows the file structure, with the 'sanity/schemaTypes' folder expanded and 'product.ts' selected. The main editor window displays the content of 'product.ts', which defines a Sanity schema for a product. The schema includes fields for name, description, price, discount percentage, price without discount, and rating. The code is as follows:

```
1 export default {
2   name: 'product',
3   type: 'document',
4   title: 'Product',
5   fields: [
6     {
7       name: 'name',
8       type: 'string',
9       title: 'Product Name',
10    },
11    {
12      name: 'description',
13      type: 'string',
14      title: 'Description'
15    },
16    {
17      name: 'price',
18      type: 'number',
19      title: 'Product Price',
20    },
21    {
22      name: 'discountPercentage',
23      type: 'number',
24      title: 'Discount Percentage',
25    },
26    {
27      name: 'priceWithoutDiscount',
28      type: 'number',
29      title: 'Price Without Discount',
30      description: 'Original price before discount'
31    },
32    {
33      name: 'rating',
34      type: 'number',
35    }
36  ]
37 }
```

Then, update your `sanity/schemaTypes/index.ts` file to include the new product schema:

Now, let's install the necessary packages. Run the following command in your terminal:

```
npm install @sanity/client
```



The screenshot shows a code editor with a dark theme. The top bar has tabs for 'page.tsx U' and 'package.json M X'. The 'package.json' tab is active, showing the 'scripts' section of the file. The 'scripts' object contains several entries: 'dev', 'build', 'start', 'lint', and 'import-data'. The 'import-data' entry is highlighted with a blue circle and a blue arrow pointing to it from the left. The value for 'import-data' is 'node scripts/importSanitydata.mjs'. The line numbers 5 through 11 are visible on the left side of the editor.

```
5  "scripts": {  
6    "dev": "next dev",  
7    "build": "next build",  
8    "start": "next start",  
9    "lint": "next lint",  
10   "import-data": "node scripts/importSanitydata.mjs"  
11 },
```

5. Running the Import Script

To run the import script, we need to add a new script to our

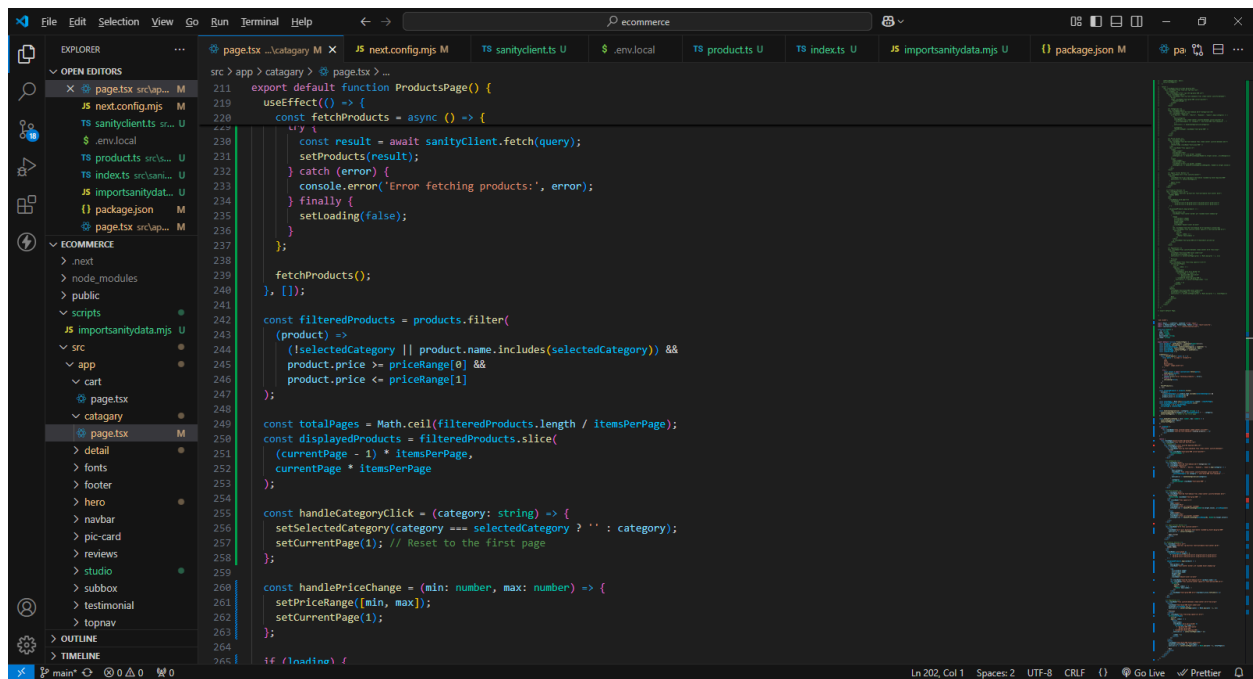
`package.json` file. Open your `package.json` and add the following to the `"scripts"` section:

```
"scripts": {  
  
  "dev": "next dev --turbopack",  
  
  "build": "next build",  
  
  "start": "next start",  
  
  "lint": "next lint",  
  
  "import-data": "node scripts/importSanityData.mjs"  
  
}
```

Now you can run the import script using:

```
npm run import-data
```


This script will fetch products from the FakeStoreAPI, upload any associated images to Sanity's asset store, and then create new product documents in your dataset sanity.



```
src > app > category > category.ts
211 export default function ProductsPage() {
212   useEffect(() => {
213     const fetchProducts = async () => {
214       try {
215         const result = await sanityClient.fetch(query);
216         setProducts(result);
217       } catch (error) {
218         console.error('Error fetching products:', error);
219       } finally {
220         setLoading(false);
221       }
222     };
223     fetchProducts();
224   }, []);
225
226   const filteredProducts = products.filter(
227     (product) =>
228       (!selectedCategory || product.name.includes(selectedCategory)) &&
229       product.price >= priceRange[0] &&
230       product.price <= priceRange[1]
231   );
232
233   const totalPages = Math.ceil(filteredProducts.length / itemsPerPage);
234   const displayedProducts = filteredProducts.slice(
235     (currentPage - 1) * itemsPerPage,
236     currentPage * itemsPerPage
237   );
238
239   const handleCategoryClick = (category: string) => {
240     setSelectedCategory(category === selectedCategory ? '' : category);
241     setCurrentPage(1); // Reset to the first page
242   };
243
244   const handlePriceChange = (min: number, max: number) => {
245     setPriceRange([min, max]);
246     setCurrentPage(1);
247   };
248
249   if (!loading) {
250     // ...
251   }
252 }
```

