Day 3: API Integration & Data Migration

Process of API Integration

To begin the API integration, I followed the steps below:

1. **Creating the Script Folder**

2. First, I created a `script` folder at the root of my project directory.

3. **Creating the importData.mjs File**

Inside the `script` folder, I created a file named `importData.mjs`.

4. **Adding the Script to package.json**

In the `package.json` file, I added a new script under the `"scripts"` section:

`"import-Data": "node script/importData.mjs"`.

5. **Running the Script**

After setting up the script, I ran it in the terminal using the command:

`npm run import-Data`.

6. **Fetching Data from the API**

Once the script was executed, it successfully fetched the data from the API.

7. **Exporting Data to Sanity**The script then exported the fetched data to Sanity one by one, completing the integration process.

Adjustments Made to Schema

During the integration process, I made the following adjustments to the schema:

1. **Field Name Correction**

I corrected the name of a field from `new` to `isNew` to ensure successful export of all API data to Sanity.

2. **Adding Extra Fields**

I added two extra fields—`stars` and `rating`—to match the requirements of the UI.

API Calls Data:

$ npm run import-Data

> ecommerce-website@0.1.0 import-Data

> node script/importData.mjs

Uploading image: https://cdn.sanity.io/images/7xt4qcah/production/4e2ed6a9eaa6e1413843e53f3113ccfd2104c301-278x296.pngImage uploaded successfully: image-4e2ed6a9eaa6e1413843e53f3113ccfd2104c301-278x296-png

Product Casual Green Bomber Jacket uploaded successfully: {

_createdAt: '2025-01-17T21:09:56Z',

_id: 'EJwKv1AMm1jkttyWGNCaHJ',

_rev: 'EJwKv1AMm1jkttyWGNCaEH',

_type: 'products',

_updatedAt: '2025-01-17T21:09:56Z',

category: 'hoodie',

colors: ['Blue', 'Red', 'Black', 'Yellow'],

description: "This stylish green bomber jacket offers a sleek and modern twist on a classic design. Made from soft and comfortable fabric, it features snap buttons and ribbed cuffs, giving it a sporty yet refined look. The minimalist

style makes it perfect for layering over casual t-shirts or hoodies. Whether you're out with friends or just lounging, this jacket provides a laid-back yet fashionable vibe. Its muted green color adds a subtle, earthy tone that pairs well with a variety of outfits, making it a versatile addition to your casual wardrobe.",

discountPercent: 20,

image: {

_type: 'image',

asset: {

_ref: 'image-4e2ed6a9eaa6e1413843e53f3113ccfd2104c301-278x296-png'

}

},

isNew: true,

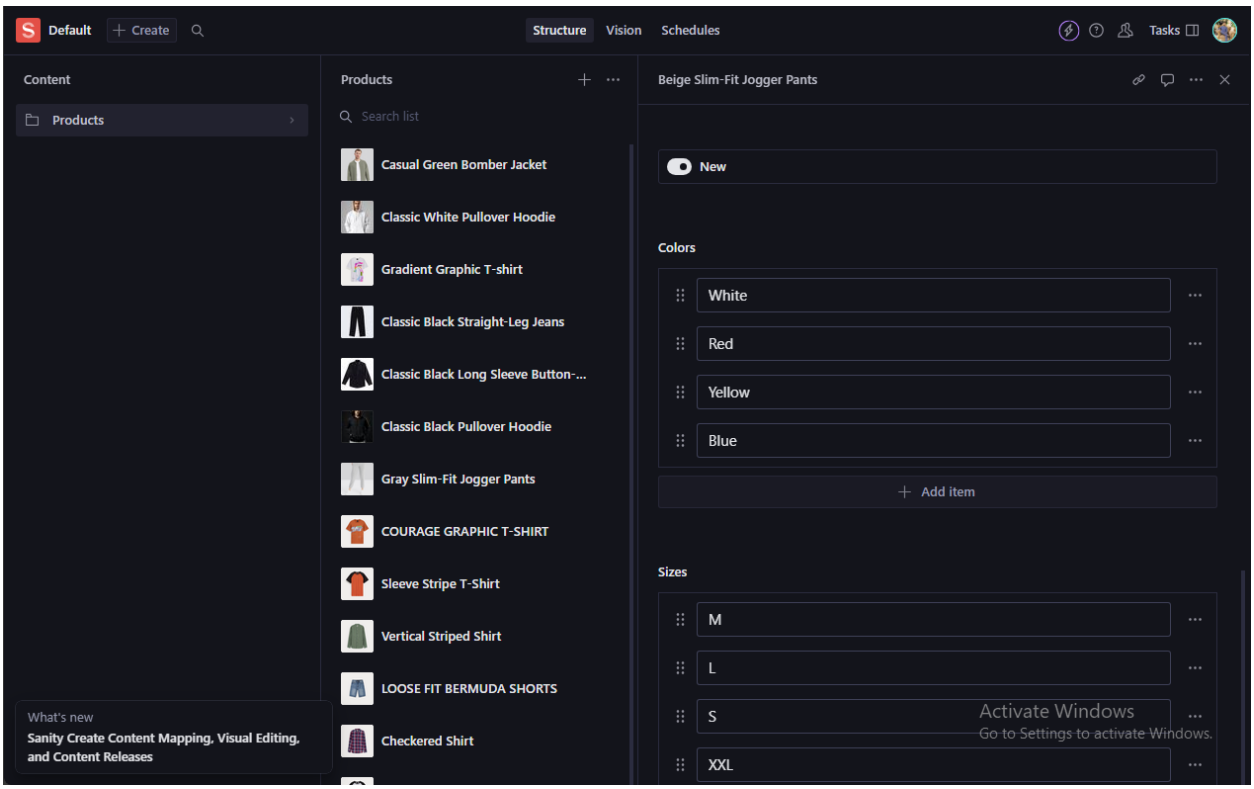
name: 'Casual Green Bomber Jacket',

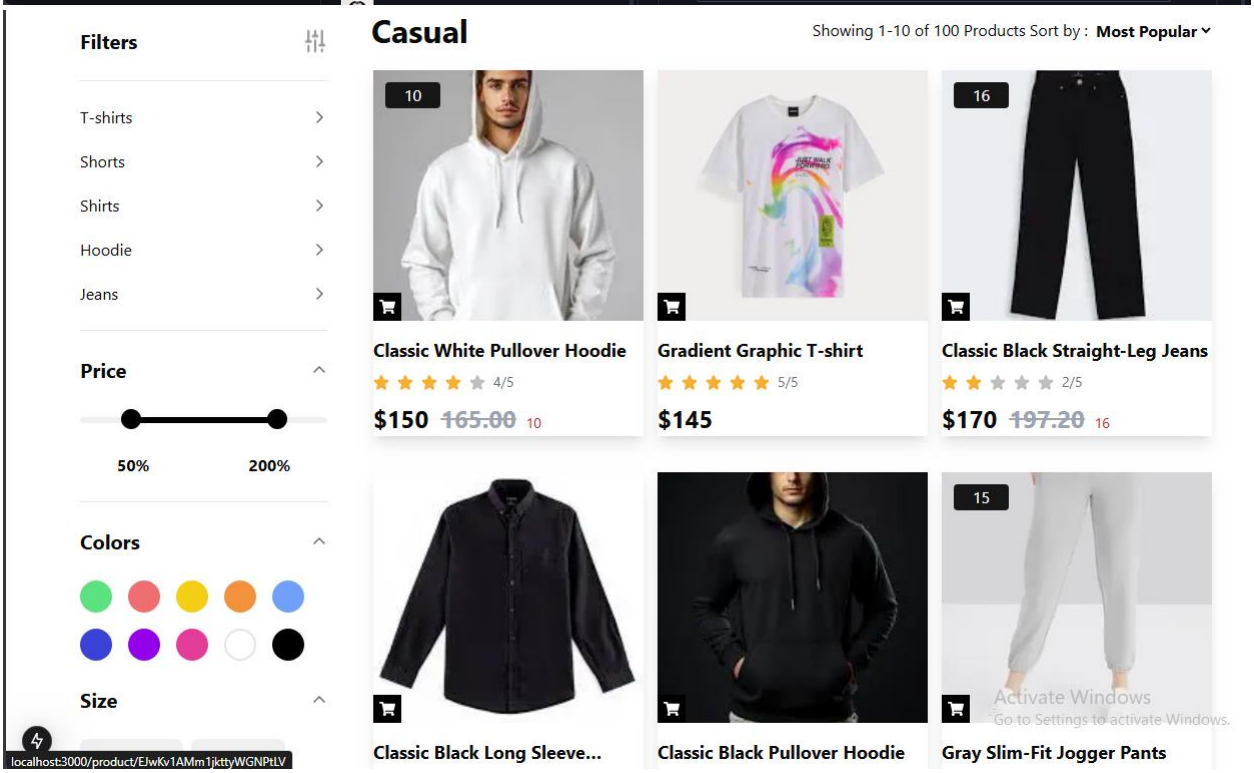
price: 300,

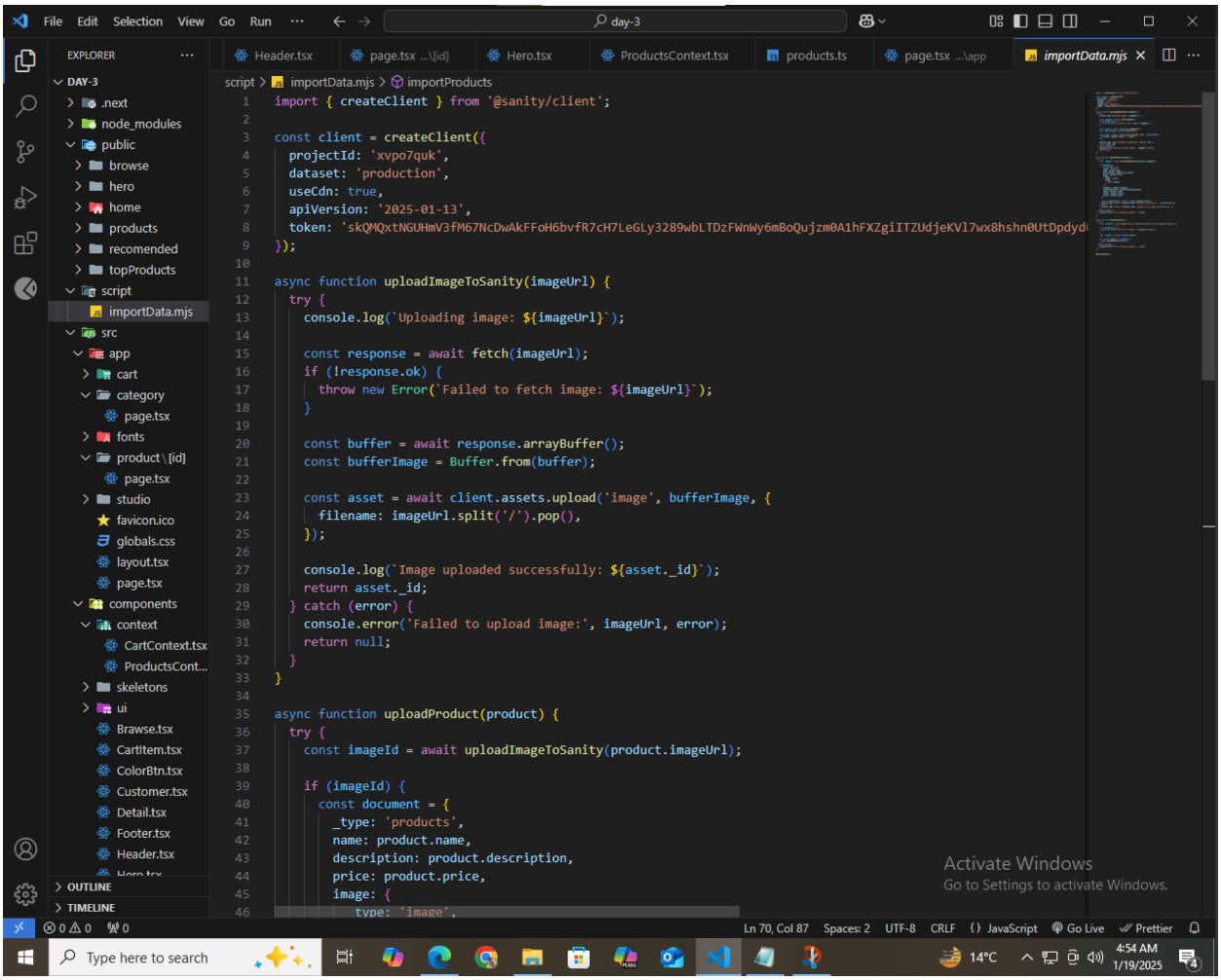
sizes: ['S', 'XXL', 'XL', 'L']

}

Snippets:


Default
Create
Structure
Vision
Schedules
Tasks
Content
Products
Products
Search list
Casual Green Bomber Jacket
Classic White Pullover Hoodie
Gradient Graphic T-shirt
Classic Black Straight-Leg Jeans
Classic Black Long Sleeve Button-...
Classic Black Pullover Hoodie
Gray Slim-Fit Jogger Pants
COURAGE GRAPHIC T-SHIRT
Sleeve Stripe T-Shirt
Vertical Striped Shirt
LOOSE FIT BERMUDA SHORTS
Checkered Shirt
What's new
Sanity Create Content Mapping, Visual Editing, and Content Releases
Beige Slim-Fit Jogger Pants
New
Colors
White
Red
Yellow
Blue
Add item
Sizes
M
L
S
XXL
Activate Windows
Go to Settings to activate Windows.


Filters
T-shirts
Shorts
Shirts
Hoodie
Jeans
Price
50%
200%
Colors
Size
localhost:3000/product/EJwKv1AMm1jkttyWGNPtLV
Casual
Showing 1-10 of 100 Products Sort by : Most Popular
10
Classic White Pullover Hoodie
4/5
$150 165.00 10
Gradient Graphic T-shirt
5/5
$145
16
Classic Black Straight-Leg Jeans
2/5
$170 197.20 16
Classic Black Long Sleeve...
Classic Black Pullover Hoodie
15
Gray Slim-Fit Jogger Pants
Activate Windows
Go to Settings to activate Windows.



Fetch Data in UI

I successfully fetched the data in the UI using `useContext()`. Here's the process:

1. **Creating Product Context**

2. I created a `productContext` to hold the fetched data.

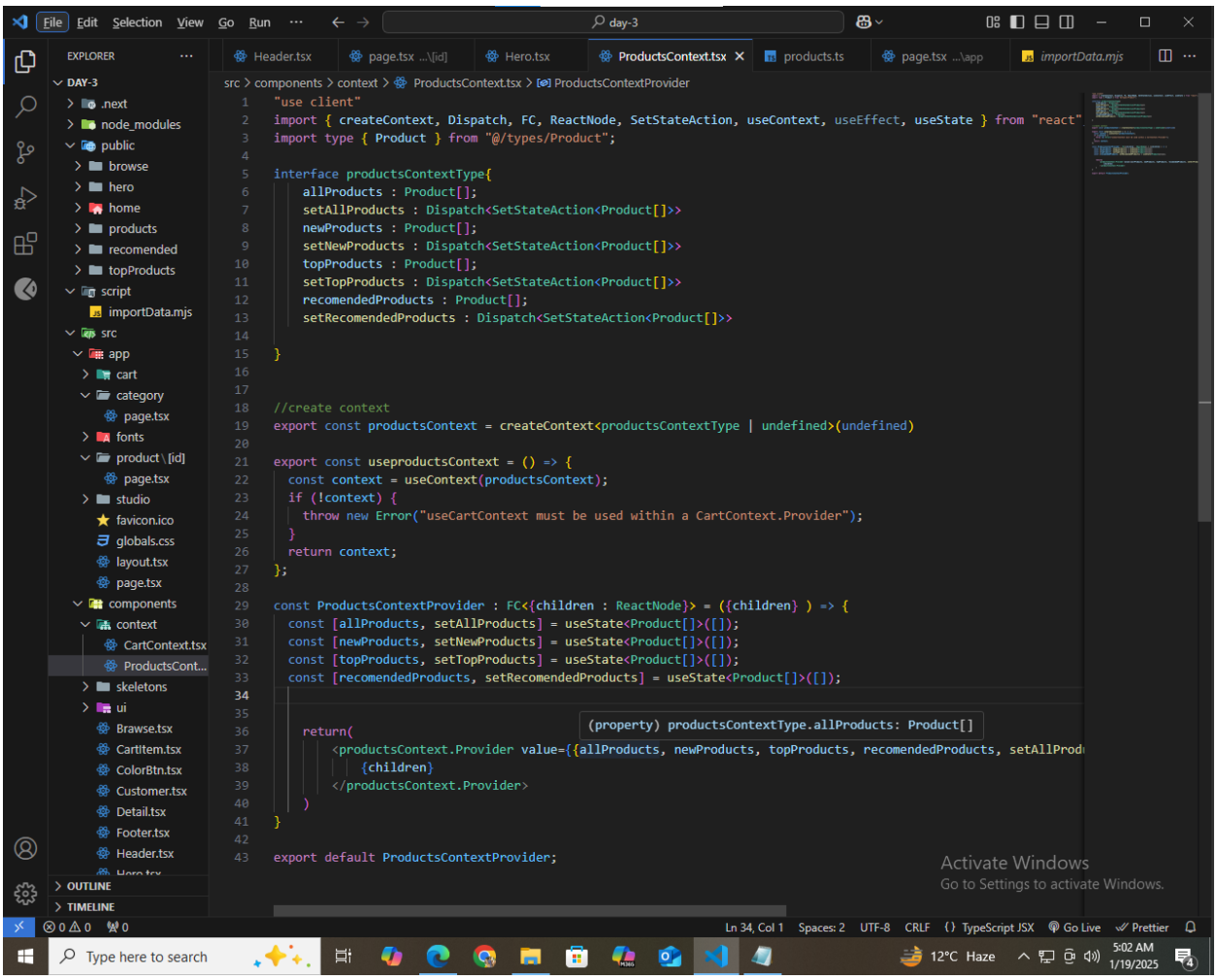
3. **Adding Data to Context**

The data fetched from the API was added to the `productContext`.

4. **Using Context in UI**

I utilized the context throughout the UI, ensuring the data was accessible across all components.

Product Context Component:



Conclusion

In conclusion, the API integration process was successfully completed by creating a script to fetch data and export it to Sanity. Adjustments were made to the schema to ensure the correct data structure. Additionally, the fetched data was integrated seamlessly into the UI using `useContext()`, making it available across the application. These steps have laid a strong foundation for further development and ensured that the data flow between the backend and frontend is smooth and efficient.