PREPARED BY MISBAH

Day 3: API Integration and Data Migration

Objective

The goal of Day 3 is to integrate APIs and migrate data into **Sanity CMS**, creating a functional backend for the clothing marketplace. By replicating real-world scenarios, this exercise equips students to meet diverse client needs, such as integrating headless APIs or transferring data from popular e-Commerce platforms.

Overview This report discusses the integration of product and category information from an external API into the backend system of Shop.co, a clothing e-commerce site. The integration uses Sanity CMS for managing content and Next.js for frontend rendering.

The primary goals of this integration include:

1. Retrieving data from an external API.
2. Storing and organizing this data in Sanity CMS.
3. Dynamically displaying the data on the frontend of the platform.

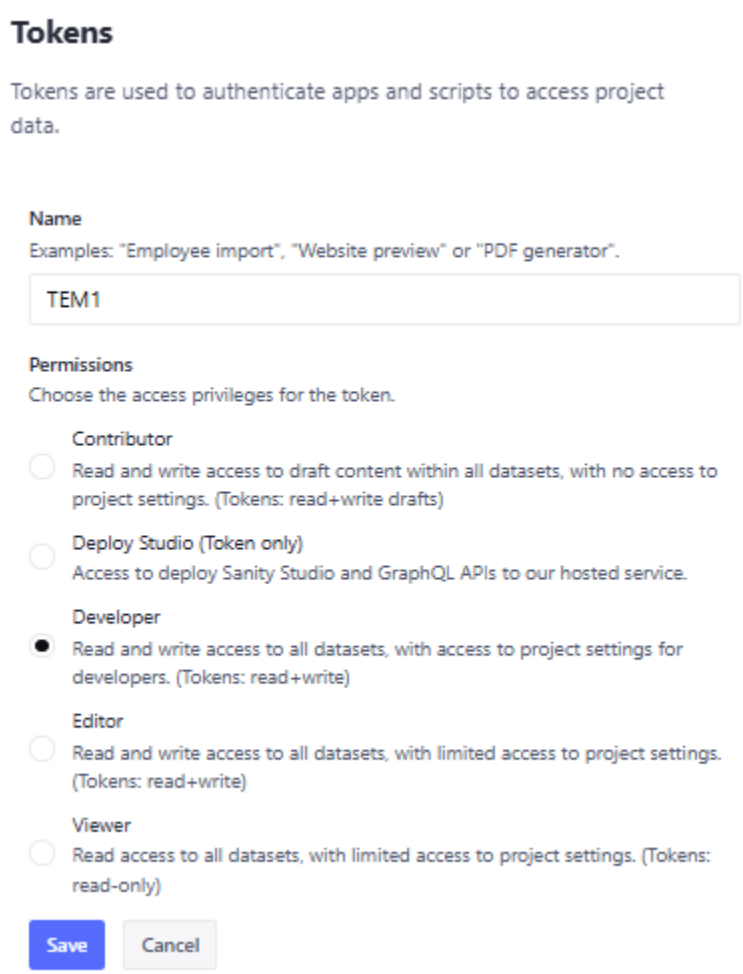
Step 1: Fetching Data from an External API

The API provided the following endpoints:

● Products Endpoint: Includes details such as titles, prices, descriptions, categories, inventory, and images.

Choose the External API

For this exercise, let's assume you're integrating with an external eCommerce API (e.g., Shopify, WooCommerce, or any custom API for product data). You will likely need to fetch product data like product names, descriptions, images, prices, and stock levels.



Step 2:Storing and Managing the Data in Sanity CMS

1. Log in to Sanity.io:

- Go to Sanity.io and log in to your account.

2. Go to the Project Settings:

- From the Sanity dashboard, click on the "Manage project" link of the project you want to generate the token for.
- This will take you to the Project Settings page.

3. Navigate to the API Section:

- In the Project Settings page, on the left sidebar, click on the API section.

4. Create a New API Token:

- In the API section, you will see an option for Tokens. Click on "Create token".
- You will be asked to specify the name of the token (e.g., "Frontend API Token") and choose the permissions for this token.

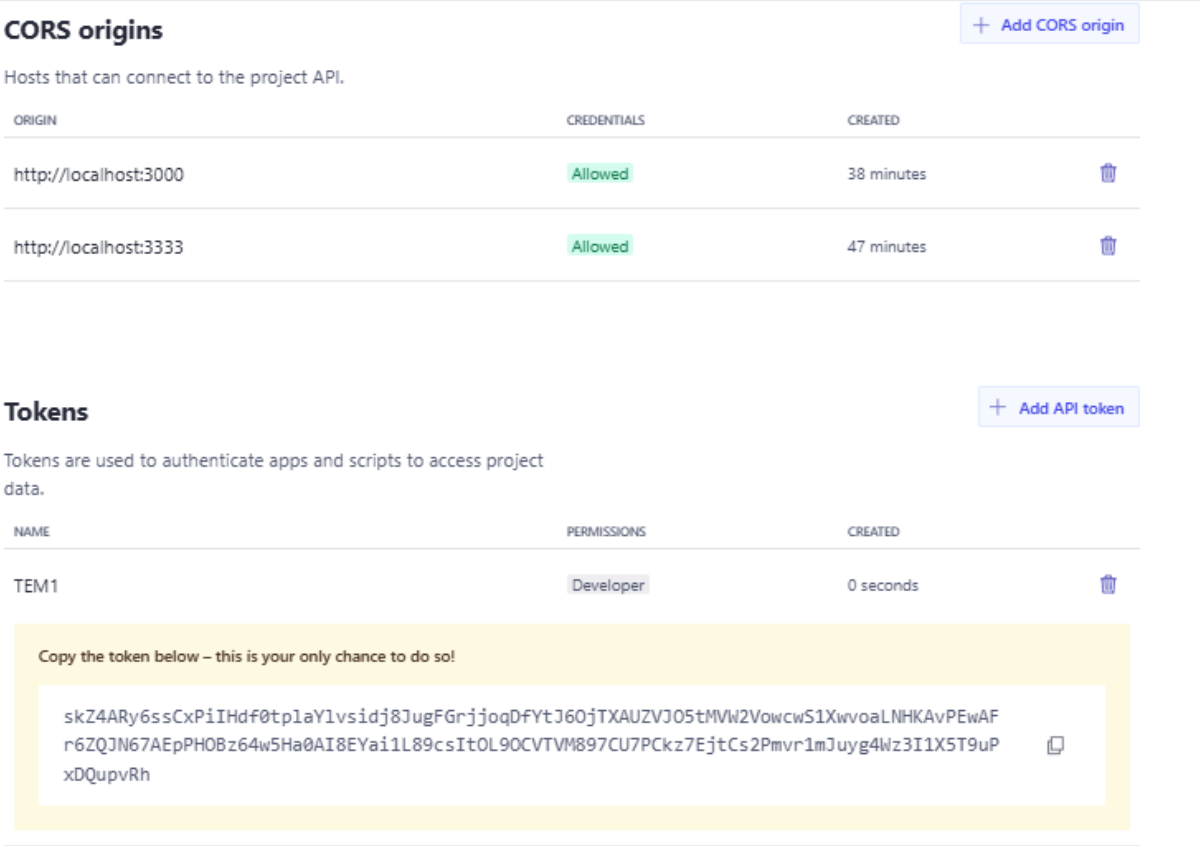
Choose the appropriate permissions based on your use case:

- Read: If your application only needs to read data (e.g., displaying products or categories).
- Write: If you want to push or update data into the Sanity CMS.
- Delete: If you want to delete data (be cautious with this option).

Best Practice: For a frontend application that only needs to fetch data, select "Read" permission.

1. Save the Token:

o After specifying the permissions, click "Save". You will be shown your new API token. Copy and store the token securely, as you won't be able to see it again once you leave the page.



Modifications in .env.local

For this project, I added the following key environment variables to the .env.local file:

1.SANITY_API_TOKEN: This is the token used to authenticate API requests to Sanity CMS. This token was generated in the Sanity dashboard and grants the necessary permissions (read/write) to interact with the Sanity CMS.

2.SANITY_PROJECT_ID: This is the unique identifier for my Sanity project, which is required for connecting to the correct project in Sanity CMS.

3. SANITY_DATASET: This specifies the dataset I'm working with (e.g., "production"). The dataset is where all the content (such as products, categories, and other documents) will be stored.

```
.env.local
NEXT_PUBLIC_SANITY_PROJECT_ID="mz8zifxl"
NEXT_PUBLIC_SANITY_DATASET="production"
SANITY_API_TOKIN="skZ4ARy6ssCxPiIHdf0tplaYlvsidj8JugFGrjjoqDfYtJ6OjTXAUZVJO5tMVW2VowcwS1XwvoaLNHKAvPEwAEr6ZOJN67AEnPHOBz64
w5Ha0AI8EYai1L89csItOL9OCVTVM897CU7PCkz7EjtCs2Pmvr1mJuyg4Wz3I1X5T9uPxDQupvRh"
```

Chat (CTRL + I) / Edit (CTRL + L)

Prepare Data for Sanity Schema

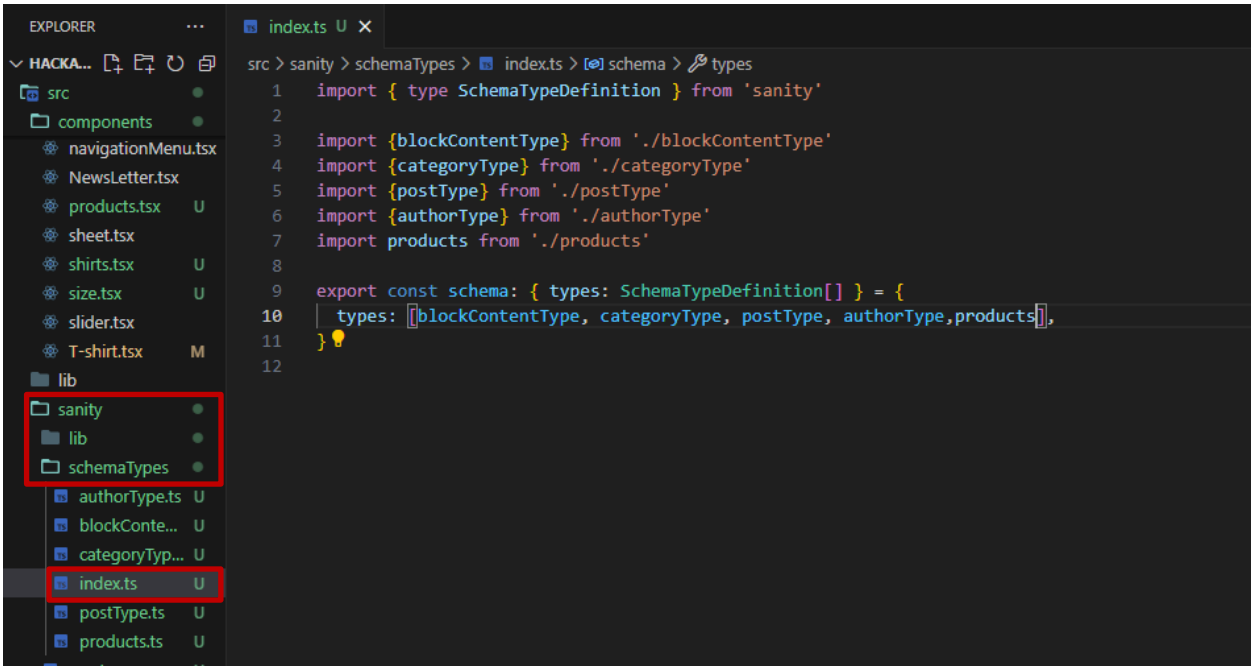
Just like with API integration, map the CSV data to the Sanity CMS schema.

```
src
components
navigationMenu.tsx
NewsLetter.tsx
products.tsx U
sheet.tsx
shirts.tsx U
size.tsx U
slider.tsx
T-shirt.tsx M
lib
sanity
lib
schemaTypes
authorType.ts U
blockConte... U
categoryTyp... U
index.ts U
postType.ts U
products.ts U
env.ts U
structure.ts U
styles
fonts.tsx
.env.local
.eslintrc.json
.gitignore
components.json
next-env.d.ts
PROBLEMS
OUTLINE
TIMELINE
NPM SCRIPTS
APPLICATION BUILDER
```

```
import { defineType } from "sanity"

export default defineType({
    name: 'products',
    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:[
                    {title: 'T-Shirt', value: 'tshirt'},
                    {title: 'Short', value: 'short'},
                    {title: 'Jeans', value: 'jeans'} ,
                    {title: 'Hoddie', value: 'hoodie'} ,
                    {title: 'Shirt', value: 'shirt'} ,
                ]
            }
```

main* ⊗ 0 ⚠ 0 0 AWS BLACKBOX Chat Add Logs CyberCoder Improve Code Cloud Code - No Project Share Code Link Generate Commit Message CRLF



Step 3: Migrating Data to Sanity CMS

The fetched data was structured and stored in Sanity CMS using its JavaScript client library.

The product schema was updated to include additional fields for more detailed data management.

Key Schema Fields:

- title: Name of the product.
- priceWithoutDiscount: Original price before any discounts.
- badge: Promotional labels such as "Sale" or "New Arrival".
- category: References the category associated with the product.
- image: Image URL for the product.
- inventory: Stock level of the product.

- tags: Tags like "Featured" or "Trending" for categorization

EXPLORER

HACKA...

.next
node_modules
public
icons
image
script
importData.js U
app
casual
page.tsx U
fonts
products
[id]
page.tsx M
page.tsx
studio \ [[...to...
page.tsx U
favicon.ico
globals.css
layout.tsx
page.tsx
components
ui
accordion.tsx U
AllReviews.tsx
Breadcrumb.t... M
carousel.tsx
checkbox.tsx

PROBLEMS
OUTLINE
TIMELINE
NPM SCRIPTS

index.ts U | tsconfig.json | importData.js U

script > importData.js > uploadImageToSanity

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

const client = createClient({
  projectId: 'mz8zifxl',
  dataset: 'production',
  useCdn: true,
  apiVersion: '2025-01-13',
  token:
  'skZ4ARy6ssCxPiIHdf0tplaYlvsidj8JugFGrjjoqDfYtJ6OjTXAUZVJO5tMVW2VowcwS1XwvoaLNHKAvPEwAFr6ZQJN67AEpPHOBz64w5Ha0AI8EYai1L89csItOL9OCVTVM
  vr1mJuyg4Wz3I1X5T9uPxDQupvRh',
});

Tabnine | Edit | Test | Explain | Document
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;
  }
}

Tabnine | Edit | Test | Explain | Document
async function uploadProduct(product) {
```

EXPLORER

HACKA...

src
sanity
lib
schemaTypes
authorType.ts U
blockConte... U
categoryTyp... U
index.ts U
postType.ts U
product.ts U
env.ts U
structure.ts 4, U
styles
fonts.tsx
.env.local
.eslintrc.json
.gitignore
components.json
next-env.d.ts
next.config.mjs
package-lock.json M
package.json M
postcss.config.mjs
README.md
sanity.cli.ts U
sanity.config.ts U
tailwind.config.ts
tsconfig.json

index.ts U | structure.ts 4, U | importSanityData.js U | package.json M

package.json > {} scripts

```
{
  "name": "hackathon",
  "version": "0.1.0",
  "private": true,
  "type": "module",
  Debug
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "import-data": "node scripts/importSanityData.js"
  },
  "dependencies": {
    "@radix-ui/react-accordion": "^1.2.2",
    "@radix-ui/react-checkbox": "^1.1.3",
    "@radix-ui/react-dialog": "^1.1.3",
    "@radix-ui/react-dropdown-menu": "^2.1.4",
    "@radix-ui/react-label": "^2.1.1",
    "@radix-ui/react-navigation-menu": "^1.2.3",
    "@radix-ui/react-separator": "^1.1.1",
    "@radix-ui/react-slider": "^1.2.2",
    "@radix-ui/react-slot": "^1.1.1",
    "@sanity/client": "^6.25.0",
    "@sanity/icons": "^3.5.7",
    "@sanity/image-url": "^1.1.0",
    "@sanity/vision": "^3.70.0",
    "@shadcn/ui": "^0.0.4",
    "axios": "^1.7.9",
    "class-variance-authority": "^0.7.1",
    "classnames": "^2.5.1",
```

OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS | COMMENTS

```
$ npm run import-data

> hackathon@0.1.0 import-data
> node scripts/importSanityData.js

Uploading image: https://cdn.sanity.io/images/7xt4qcah/production/4e2ed6a9eaa6e1413843e53f3113ccfd2104c301-278x296.png
Image uploaded successfully: image-4e2ed6a9eaa6e1413843e53f3113ccfd2104c301-278x296-png
Product Casual Green Bomber Jacket uploaded successfully: {
  _createdAt: '2025-01-22T15:13:39Z',
  _id: '8ptia17TGb7SGzDvivjCqT',
  _rev: '8ptia17TGb7SGzDvivjCpi',
  _type: 'products',
  _updatedAt: '2025-01-22T15:13:39Z',
  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 cas
ual 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' ]
}
Uploading image: https://cdn.sanity.io/images/7xt4qcah/production/a93f8ab0d00d857ce42c057d8835e419d68a87eb-193x262.jpg
Image uploaded successfully: image-a93f8ab0d00d857ce42c057d8835e419d68a87eb-193x262-jpg
Product Classic White Pullover Hoodie uploaded successfully: {
  _createdAt: '2025-01-22T15:13:41Z',
  _id: 'gbTY3B2TxF1zfilLOd2NV6',
  _rev: 'gbTY3B2TxF1zfilLOd2NRZ',
  _type: 'products',
```

Testing and Validation

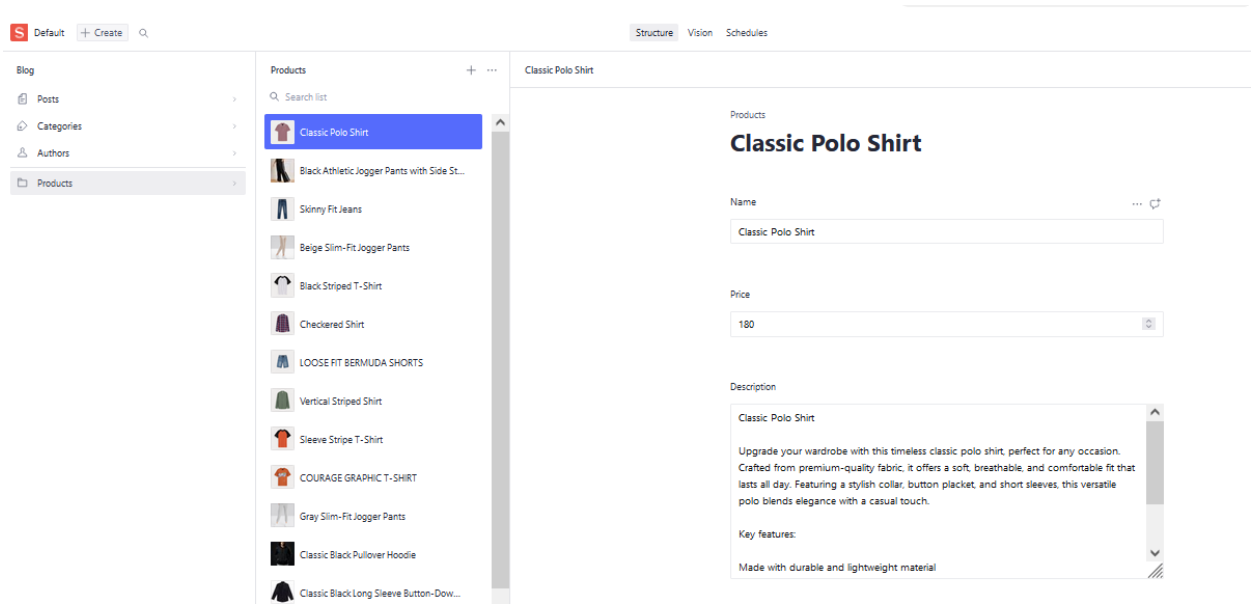
Validate Data

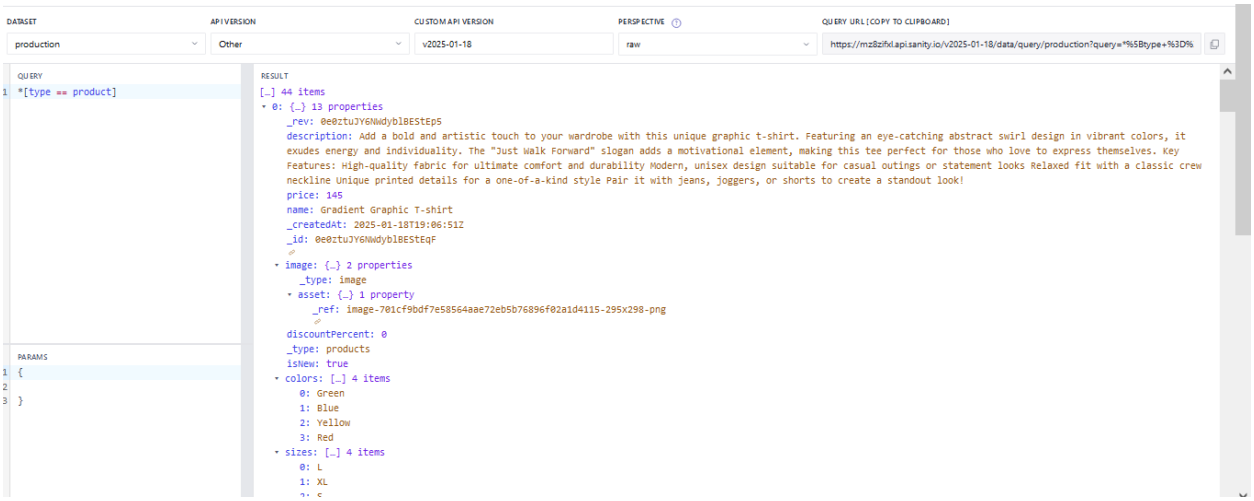
After pushing data, check your Sanity Studio to ensure that the data appears correctly. You should see your products listed with all the attributes (e.g., name, price, category).

Error Handling

Ensure that your scripts handle errors gracefully. For instance:

- Log errors during data fetching and pushing.
- Handle API rate limits.
- Ensure your mapping logic accounts for missing or incomplete data.

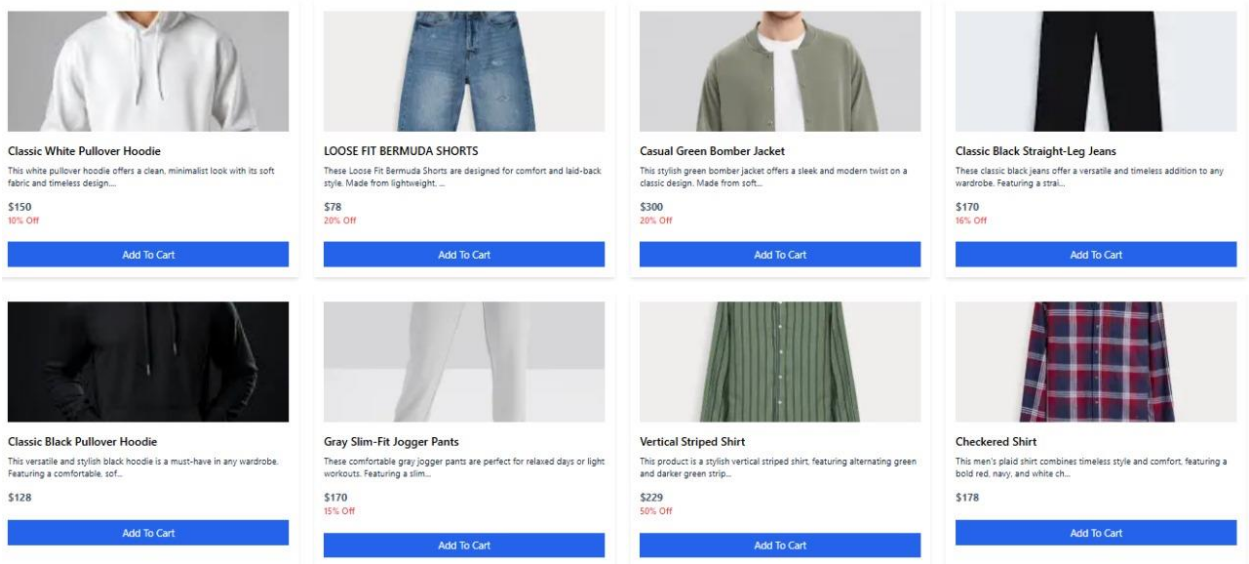




4. Frontend Integration

Step 4: Displaying Data on the Frontend

The next step was to retrieve the data stored in Sanity CMS and show it on the Shop.co frontend. By using Next.js, we dynamically pulled this data and displayed it on the product listing page. The dynamic rendering allowed for an up-to-date and smooth display of products, ensuring that any changes made in the CMS were instantly reflected on the website. This integration helped maintain real-time product information on the user interface.



Conclusion

The integration of the API for Shop.co has been successfully executed. This process involved retrieving data, storing it in Sanity CMS, and dynamically displaying it on the frontend. The attached screenshots confirm the successful implementation at each stage. With this integration, managing products and categories has become more efficient, offering a smooth experience for both administrators and users.

SATURDAY SLOT 2 TO 5

Task Given By : Sir Ameen Alam

Class Teacher : Sir Muhammad Bilal & Sir Ali Aftab Sheikh