Day 3 - API Integration Report and Data Migration

API integration process

Introduction:

JavaScript code demonstrates the integration of Sanity's CMS for uploading and managing product data. The code utilizes Sanity's client library to interact with the platform and processes data from an external API. The report details the workflow, functionalities, and best practices incorporated in this implementation.

Sanity Client Setup:

The create Client function initializes a connection to the Sanity CMS:

This setup ensures secure and efficient interaction with the Sanity API.

Key Functionalities

1. Image Upload:

- The uploadImageToSanity function handles image uploads to Sanity:
- Downloads the image using the fetch API.
- Converts the image to a buffer format compatible with Sanity's asset manager.
- Uploads the image to Sanity and retrieves its unique ID for further use.
- Error Handling: Ensures proper error logging and null returns for failed uploads.

2. Product Upload:

- The uploadProduct function creates and uploads product documents to Sanity:
- Image Integration: Incorporates the uploaded image asset.
- Product Properties: Includes name, description, price, category, discounts, and other metadata.
- Sanity Document Creation: Utilizes the client.create method to store products in the CMS.
- Error Handling: Logs errors and skips product uploads in case of image upload failure.

3. Data Import:

- The importProducts function fetches product data from an external API:
- Retrieves data from the URL https://template1-neon-nu.vercel.app/api/products.
- Iterates through the product list and calls uploadProduct for each item.
- Error Handling: Manages HTTP errors and logs failed requests.

Highlights and Strengths

1. Modular Design

The implementation uses distinct functions for each task, ensuring:

- Reusability
- Improved readability
- Simplified debugging

2. Comprehensive Error Handling

The code employs try-catch blocks to handle potential failures in:

- Image downloads
- API requests
- Product uploads
- This approach minimizes disruptions and ensures detailed logging.

3. Secure Token Usage

Authentication is managed through an API token, enabling secure access to Sanity CMS.

4. Dynamic Product Integration

The script dynamically handles product data, making it suitable for varied datasets.

Recommendations for Enhancement

- Environment Variables:
- Store sensitive information like the API token and project ID in environment variables to enhance security.

Rate Limiting:

Incorporate rate limiting for API requests to avoid potential throttling or denial of service.

Detailed Logging:

Expand logging to include timestamps and categorized error levels for better debugging.

Validation:

Implement schema validation for incoming product data to ensure consistency and prevent upload errors.

Retry Mechanism:

Add retry logic for transient failures, especially for image uploads and API requests.

Conclusion:

The code effectively integrates Sanity CMS for product data management and demonstrates best practices in modular design and error handling. With minor enhancements, this implementation can be further optimized for scalability and security. The current structure is robust and adaptable for real-world applications.

Adjustments made to schemas.

The schema was tailored to include essential product attributes, enhance data integrity, and streamline the content management process. The schema serves as the backbone for a comprehensive product management system, enabling seamless integration with frontend and backend systems.

Schema Adjustments Overview

Key Additions and Enhancements

Core Product Attributes

- Name (name): A string field added to store the product's name, serving as the primary identifier.
- Price (price): A number field to represent the product's price.
- Description (description): A text field for detailed product information.
- Image (image): An image field to upload and manage product visuals.

Categorization

Category (category):

A string field with predefined options using the list property to ensure uniform categorization.

Categories include: T-Shirt, Short, Jeans, Hoodie, and Shirt.

This adjustment improves consistency and simplifies filtering and sorting products by category.

Discount Management

Discount Percent (discountPercent): A number field added to store discount information as a percentage. This allows for dynamic pricing adjustments and promotional strategies.

Product Status

New Product Indicator (new):

A boolean field to identify whether a product is new.

Enables dynamic tagging and highlighting of new arrivals in the storefront.

Variants and Attributes

Colors (colors):

An array of strings to store available color variants for each product.

Improves the user experience by supporting multiple options per product.

Sizes (sizes):

An array of strings to define size variants (e.g., S, M, L, XL).

Ensures product variants are well-represented.

Advanced Adjustments

Validation

Validation rules were incorporated to maintain data integrity:

Price must be a positive number.

Discount Percent must range between 0 and 100.

Scalable Categorization

The category field was optimized for scalability by using predefined options with clear titles and values.

Future expansion to a reference-based system is feasible for dynamic category management.

Slug Field for SEO

A slug field can be added for creating unique URLs for each product.

Strengths of the Adjusted Schema

- Data Consistency

Predefined categories and validation rules ensure consistent data entry.

- Enhanced User Experience

Support for product variants like colors and sizes provides flexibility for end-users.

- Scalability

The modular design of the schema supports future enhancements, such as integrating additional attributes or external references.

- Optimization for E-commerce

Fields like discountPercent and new make the schema well-suited for dynamic storefronts and marketing strategies.

Conclusion

The adjusted schema provides a robust framework for product data management, balancing flexibility with structure. These enhancements enable seamless integration with e-commerce platforms and ensure data accuracy and scalability. With the outlined recommendations, the schema is well-positioned to support complex product catalogs and dynamic storefronts.

Migration steps and tools used.

Migration Steps

1. Schema Definition and Adjustment

- Task:

Define the updated schema in the Sanity Studio.

- Steps:

1. Update the schema file with the adjusted fields, validation rules, and configurations.
2. Ensure all field names and types align with the requirements of the project.
3. Add necessary validation for fields like price, discount percentage, and product metadata.
4. Test the schema locally in Sanity Studio to validate its structure and functionality.

2. Data Preparation

- Task:

Prepare the existing product data for migration.

- Steps:

1. Export product data from the existing source (e.g., database, API).
2. Format the data to match the new schema structure (e.g., JSON).
3. Ensure all necessary fields (name, price, category, etc.) are present and follow the updated schema's validation rules.
4. Clean the data to remove duplicates or incomplete entries.

3. Image Assets Preparation

- Task:

Manage product images for the migration process.

- Steps:

1. Collect all associated product images.
2. Organize the images with proper naming conventions for easy mapping with product data.
3. Optimize image sizes for web performance (e.g., compression).
4. Ensure metadata like alt text is added for accessibility and SEO.

4. Data Upload

- Task:

Migrate the prepared product data and images into Sanity.

- Steps:

1. Image Upload:

   - Use Sanity's client.assets.upload method to upload images.

   - Retrieve and store the image IDs for linking with corresponding product data.

2. Document Creation:

   - Use the Sanity client library (client.create or client.createIfNotExists) to upload product documents.

   - Include all fields (e.g., name, price, category, colors, sizes, etc.) and link uploaded image assets.

   - Handle errors during upload with appropriate logging and retry mechanisms.

5. Testing and Verification

- Task:

Ensure the migrated data aligns with the schema and frontend requirements.

- Steps:

1. Verify that all documents are successfully uploaded to Sanity.
2. Test the Sanity Studio interface for content managers to ensure ease of use.
3. Check the frontend integration to confirm that products display correctly.

6. Deployment

- Task:

Deploy the updated schema and data.

- Steps:

1. Deploy the updated schema to production using the sanity deploy command.
2. Test the production environment for any issues with content display or functionality.
3. Set up regular backups of the Sanity dataset to prevent data loss.

Tools Used

1. Sanity CMS

Purpose:

- Content management and schema handling.

Features Used:

- Schema definition and validation.
- Asset management for product images.
- Content organization and querying.

2. Sanity Client Library

Purpose:

- Interact with Sanity programmatically.

Features Used:

- createClient: Initialize connection to the Sanity project.
- client.assets.upload: Upload and manage images.
- client.create: Create and update product documents.

3. Data Preparation Tools

- Tools:

- Spreadsheet Software (e.g., Excel or Google Sheets): Format and clean data.

- Scripts (e.g., JavaScript OR typeScript): Automate data transformation.

4. API and Fetch Libraries

- Purpose: Fetch and process external data for migration.
- Example: Fetch API or axios library for handling API requests and image downloads.

5. Image Optimization Tools

- Tools:

- ImageOptim: Compress images for web usage.

- Custom Scripts: Automate resizing and optimization.

6. Testing Tools

- Tools:

- Sanity Studio: Test schema changes and data uploads.

- Frontend Framework (Next.js): Verify integration and data rendering.

7. Version Control and Deployment

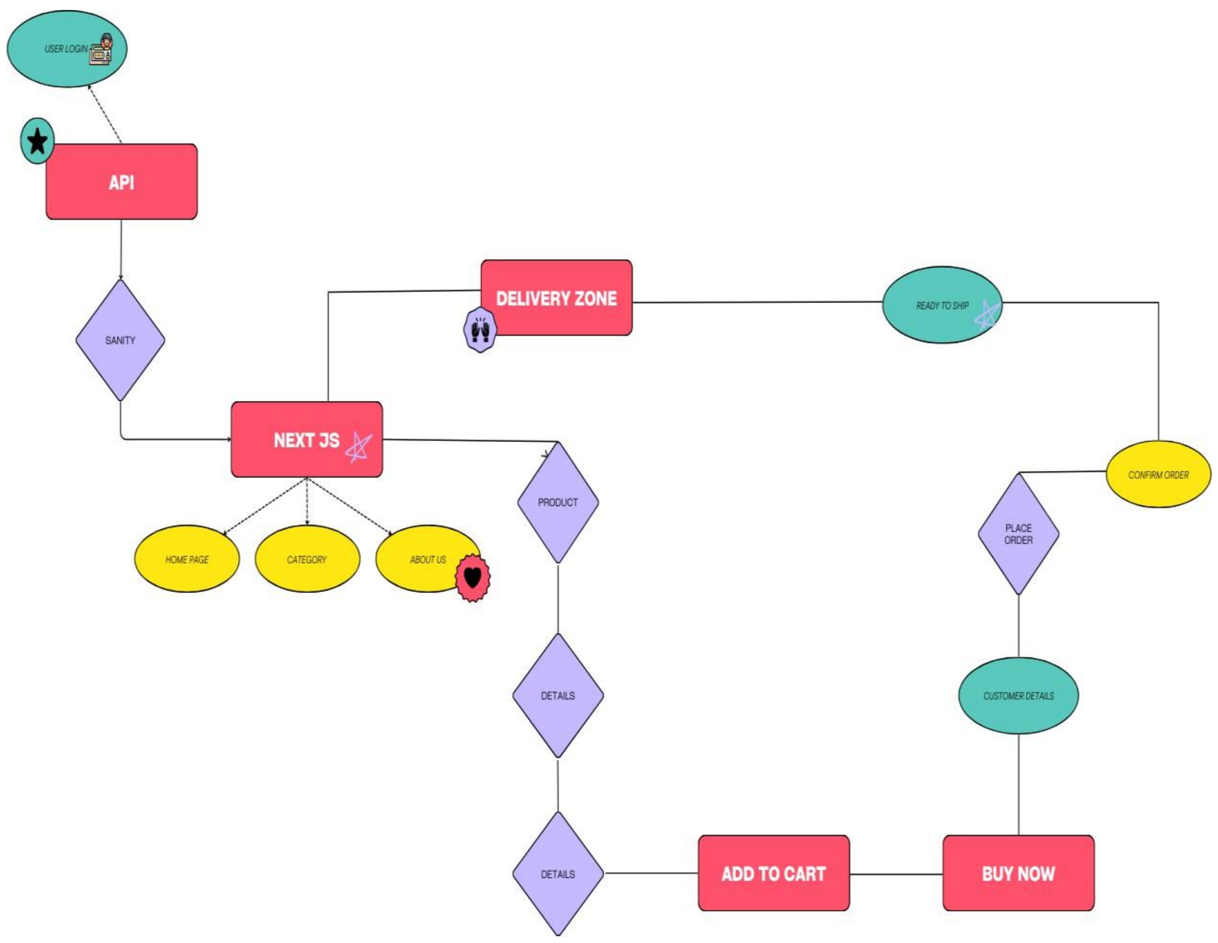
- Tools:

- Sanity CLI: Deploy schema updates and manage datasets (sanity deploy, sanity dataset import).

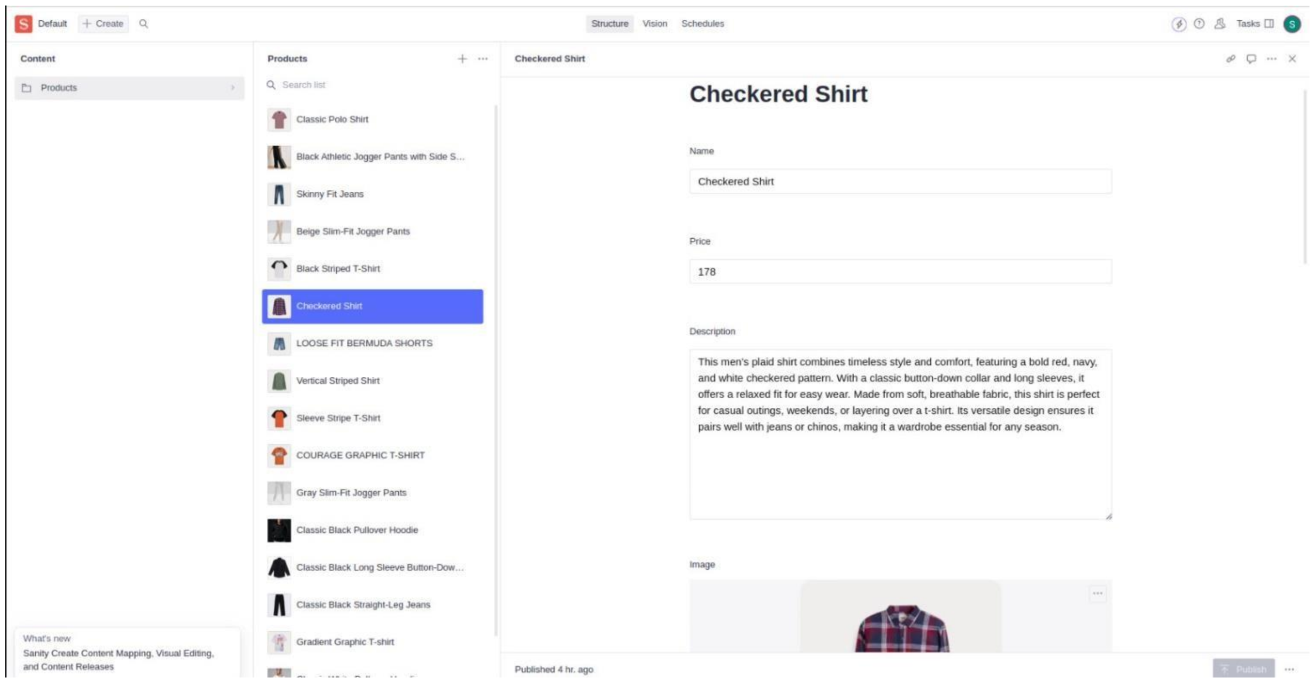
Conclusion:

The migration process involves defining an updated schema, preparing data, uploading it to Sanity, and verifying the integration. By utilizing tools like Sanity CMS, client libraries, and image optimization utilities, the process ensures a seamless transition with minimal disruption. The described steps and tools collectively enhance data accuracy, scalability, and maintainability in the CMS.

Workflow



Populated Sanity CMS fields



Classic Polo Shirt
Black Athletic Jogger Pants with Side S...
Skinny Fit Jeans
Beige Slim-Fit Jogger Pants
Black Striped T-Shirt
Checkered Shirt
LOOSE FIT BERMUDA SHORTS
Vertical Striped Shirt
Sleeve Stripe T-Shirt
COURAGE GRAPHIC T-SHIRT
Gray Slim-Fit Jogger Pants
Classic Black Pullover Hoodie
Classic Black Long Sleeve Button-Dow...
Classic Black Straight-Leg Jeans
Gradient Graphic T-shirt

What's new
Sanity Create Content Mapping, Visual Editing, and Content Releases

Checkered Shirt

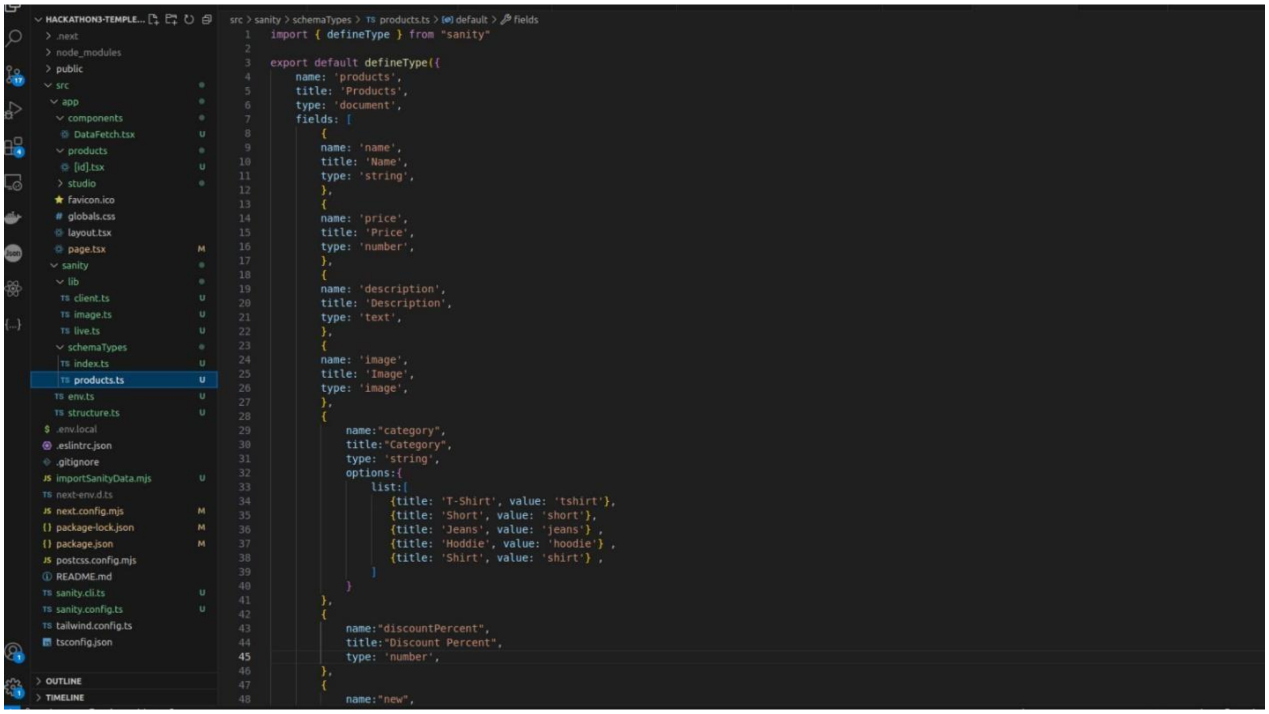
Name: Checkered Shirt

Price: 178

Description: This men's plaid shirt combines timeless style and comfort, featuring a bold red, navy, and white checkered pattern. With a classic button-down collar and long sleeves, it offers a relaxed fit for easy wear. Made from soft, breathable fabric, this shirt is perfect for casual outings, weekends, or layering over a t-shirt. Its versatile design ensures it pairs well with jeans or chinos, making it a wardrobe essential for any season.

Image

Published 4 hr. ago

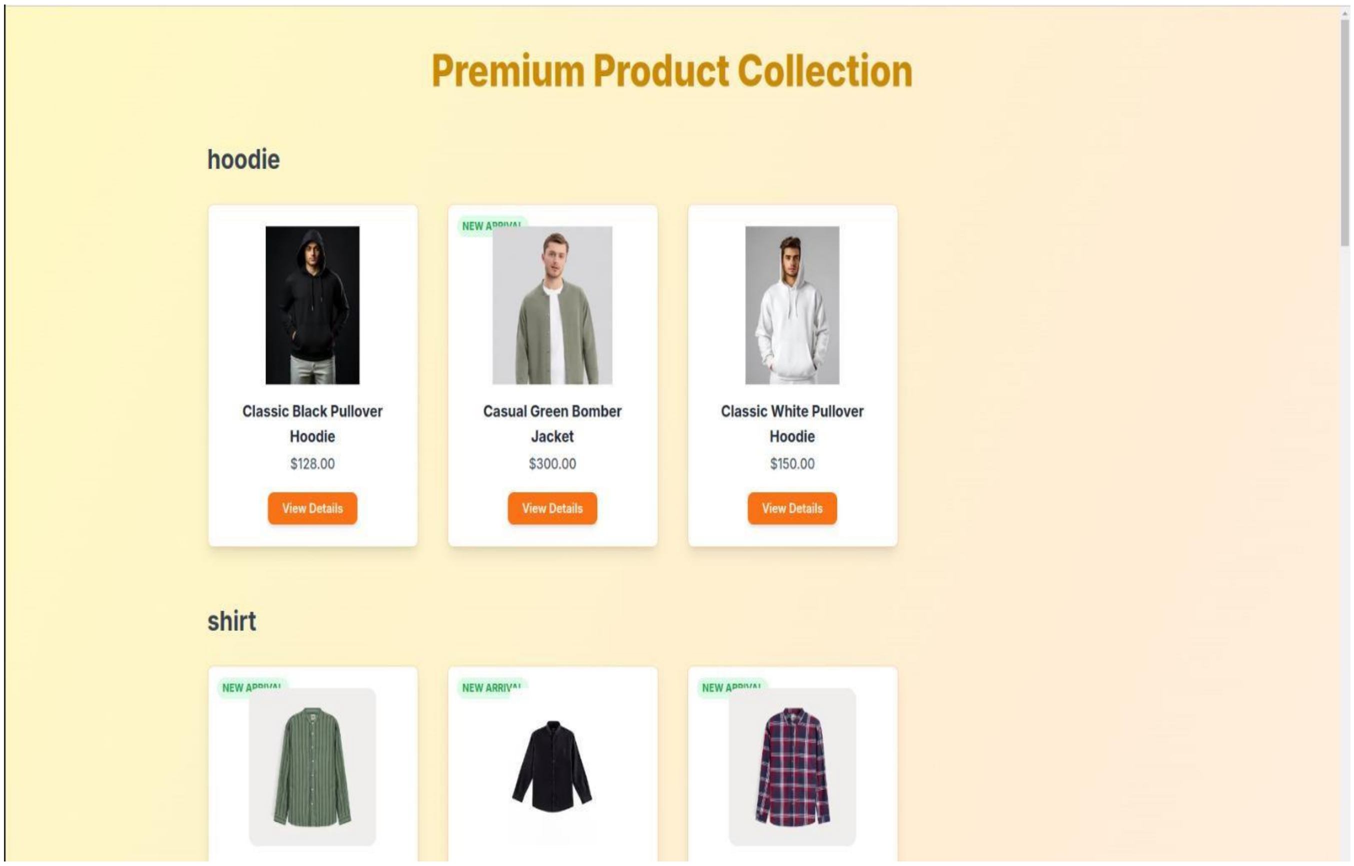


```
HACKATHON3-TEMPLE...
 > .next
 > node_modules
 > public
 v src                          •
   v app                        •
     v components               •
         DataFetch.tsx          U
     v products                 •
         [id].tsx               U
     > studio                   •
       favicon.ico
       globals.css
       layout.tsx
       page.tsx                 M
   v sanity                     •
     v lib                      •
         client.ts              U
         image.ts               U
         live.ts                U
     v schemaTypes              •
         index.ts               U
         products.ts            U
       env.ts                   U
       structure.ts             U
   .env.local
   .eslintrc.json
   .gitignore
   importSanityData.mjs         U
   next-env.d.ts
   next.config.mjs              M
   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
> OUTLINE
> TIMELINE
```

src > sanity > schemaTypes > products.ts > default > fields

```
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'} ,
                ]
            }
        },
        {
            name:"discountPercent",
            title:"Discount Percent",
            type: 'number',
        },
        {
            name:"new",
```

<u>**Data successfully displayed in the frontend.**</u>



Premium Product Collection

hoodie

Classic Black Pullover Hoodie
$128.00
View Details

NEW ARRIVAL
Casual Green Bomber Jacket
$300.00
View Details

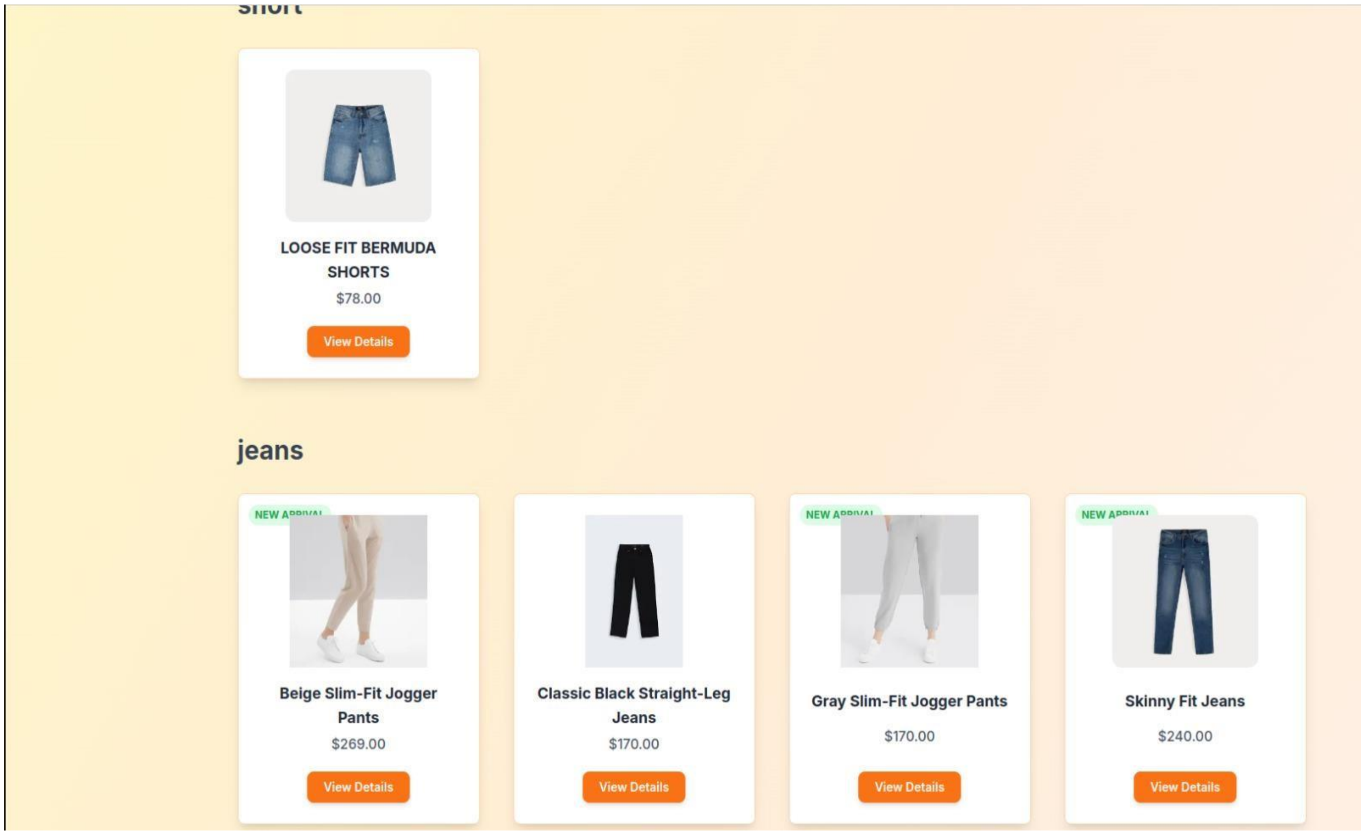
Classic White Pullover Hoodie
$150.00
View Details

shirt

NEW ARRIVAL

NEW ARRIVAL

NEW ARRIVAL



short

LOOSE FIT BERMUDA SHORTS
$78.00
View Details

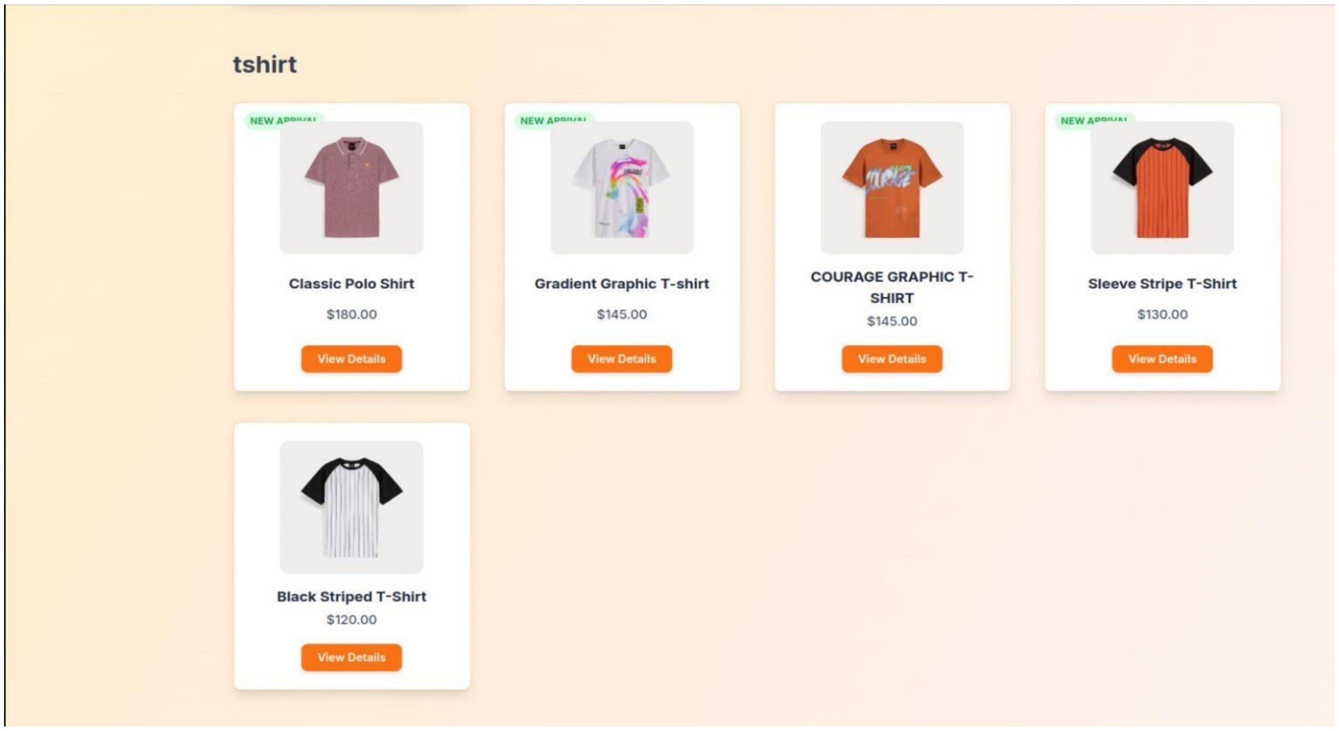
jeans

NEW ARRIVAL
Beige Slim-Fit Jogger Pants
$269.00
View Details

Classic Black Straight-Leg Jeans
$170.00
View Details

NEW ARRIVAL
Gray Slim-Fit Jogger Pants
$170.00
View Details

NEW ARRIVAL
Skinny Fit Jeans
$240.00
View Details



tshirt

NEW ARRIVAL
Classic Polo Shirt
$180.00
View Details

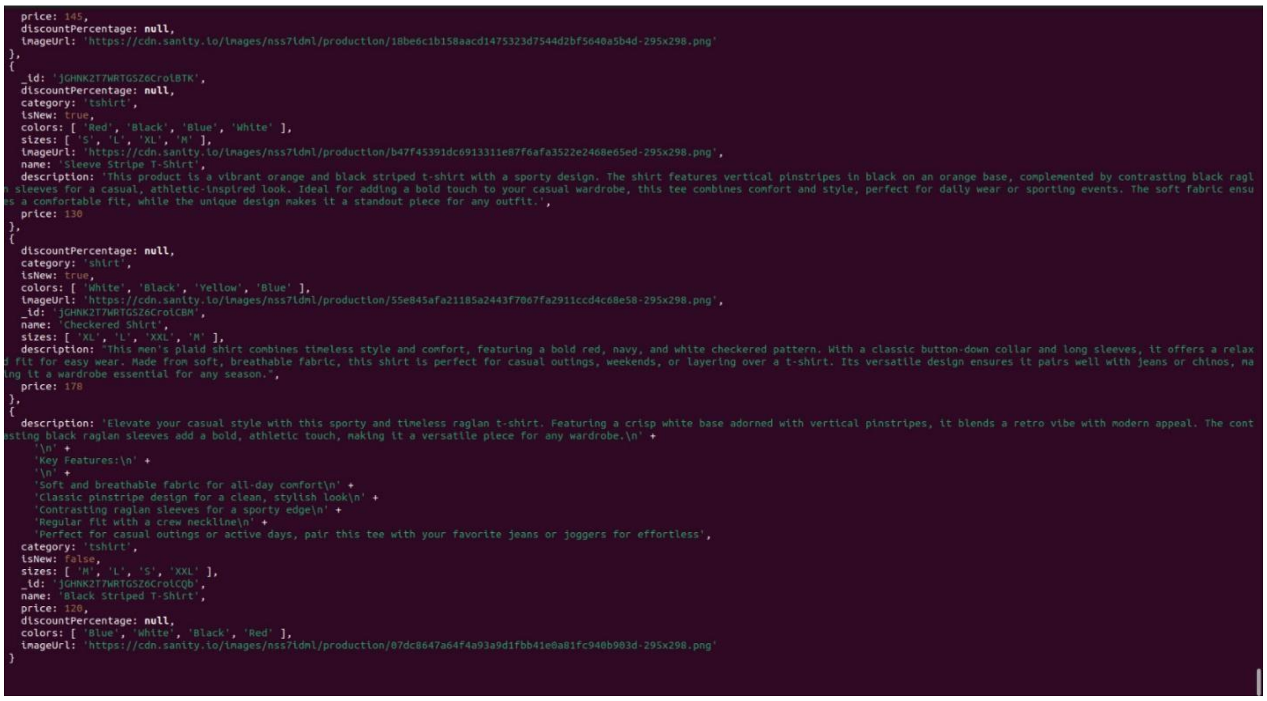
NEW ARRIVAL
Gradient Graphic T-shirt
$145.00
View Details

COURAGE GRAPHIC T-SHIRT
$145.00
View Details

NEW ARRIVAL
Sleeve Stripe T-Shirt
$130.00
View Details

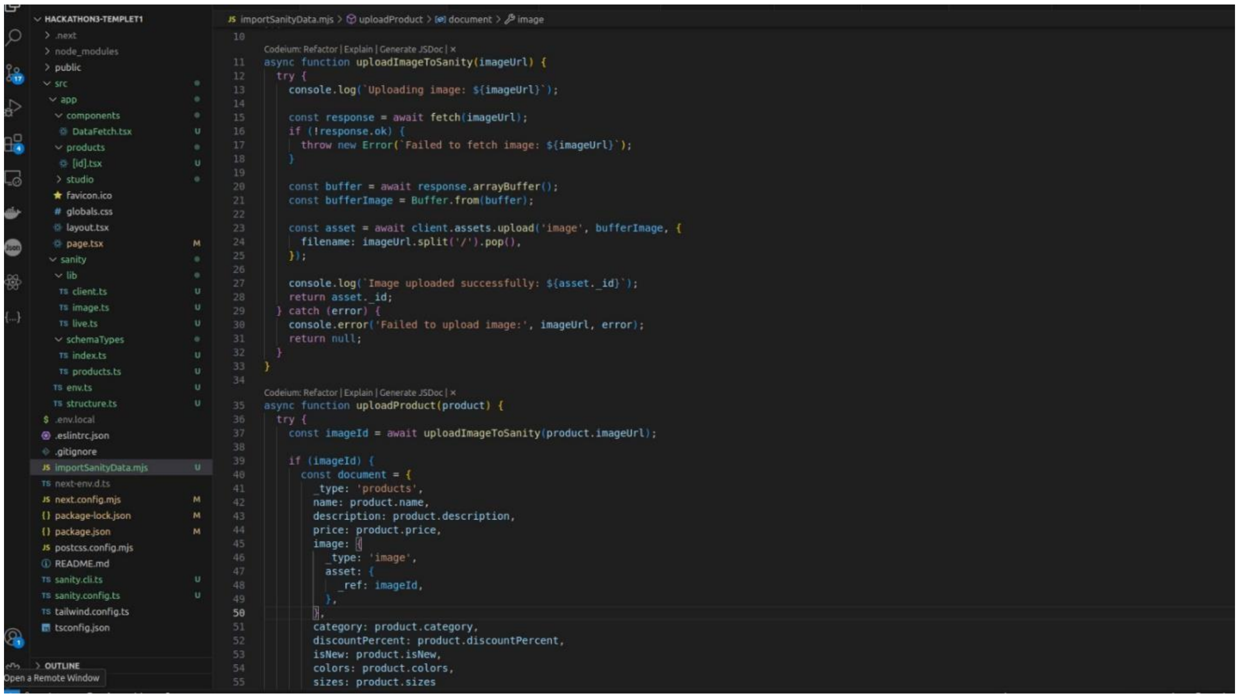
Black Striped T-Shirt
$120.00
View Details

API call

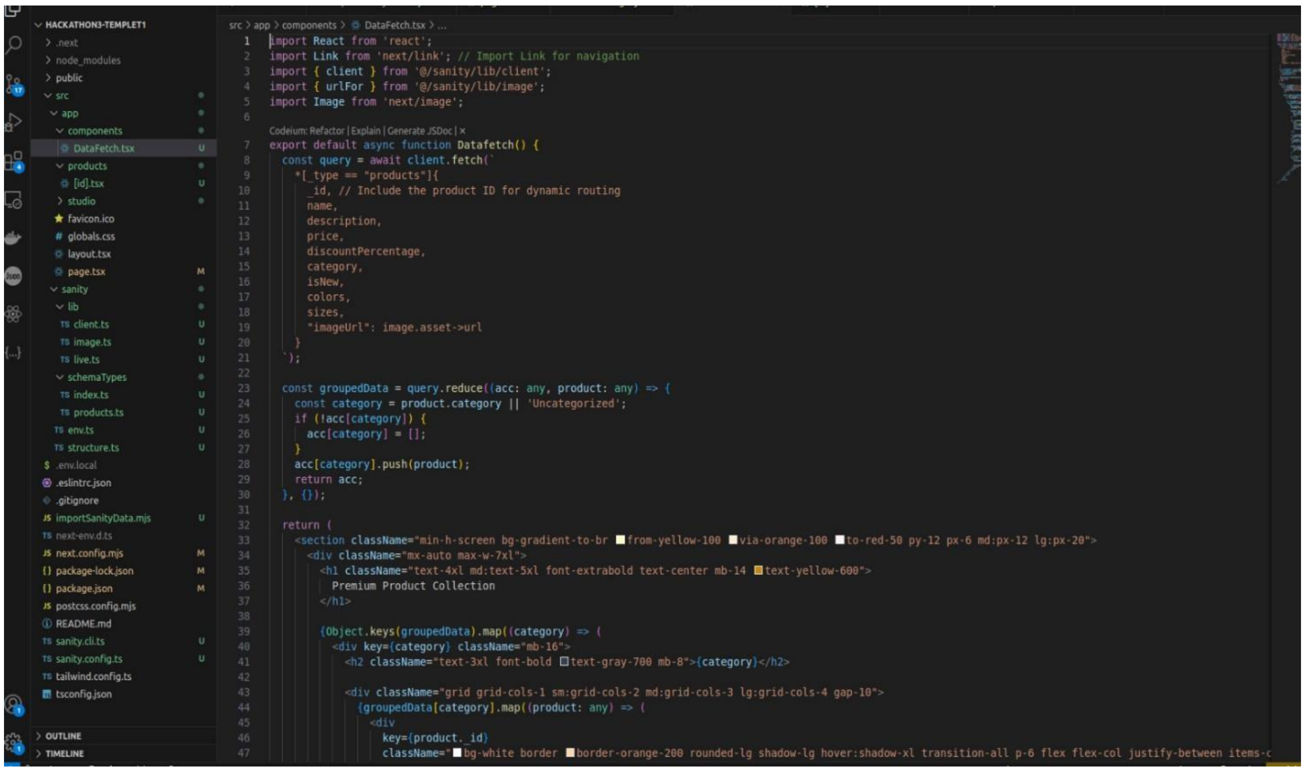


```
  price: 145,
  discountPercentage: null,
  imageUrl: 'https://cdn.sanity.io/images/nss7idml/production/18be6c1b158aacd1475323d7544d2bf5640a5b4d-295x298.png'
},
{
  _id: 'jGHNK2T7WRTGSZ6CroiBTK',
  discountPercentage: null,
  category: 'tshirt',
  isNew: true,
  colors: [ 'Red', 'Black', 'Blue', 'White' ],
  sizes: [ 'S', 'L', 'XL', 'M' ],
  imageUrl: 'https://cdn.sanity.io/images/nss7idml/production/b47f45391dc6913311e87f6afa3522e2468e65ed-295x298.png',
  name: 'Sleeve Stripe T-Shirt',
  description: 'This product is a vibrant orange and black striped t-shirt with a sporty design. The shirt features vertical pinstripes in black on an orange base, complemented by contrasting black ragl
n sleeves for a casual, athletic-inspired look. Ideal for adding a bold touch to your casual wardrobe, this tee combines comfort and style, perfect for daily wear or sporting events. The soft fabric ensu
es a comfortable fit, while the unique design makes it a standout piece for any outfit.',
  price: 130
},
{
  discountPercentage: null,
  category: 'shirt',
  isNew: true,
  colors: [ 'White', 'Black', 'Yellow', 'Blue' ],
  imageUrl: 'https://cdn.sanity.io/images/nss7idml/production/55e845afa21185a2443f7067fa2911ccd4c68e58-295x298.png',
  _id: 'jGHNK2T7WRTGSZ6CroiCBM',
  name: 'Checkered Shirt',
  sizes: [ 'XL', 'L', 'XXL', 'M' ],
  description: "This men's plaid shirt combines timeless style and comfort, featuring a bold red, navy, and white checkered pattern. With a classic button-down collar and long sleeves, it offers a relax
d fit for easy wear. Made from soft, breathable fabric, this shirt is perfect for casual outings, weekends, or layering over a t-shirt. Its versatile design ensures it pairs well with jeans or chinos, ma
ing it a wardrobe essential for any season.",
  price: 178
},
{
  description: 'Elevate your casual style with this sporty and timeless raglan t-shirt. Featuring a crisp white base adorned with vertical pinstripes, it blends a retro vibe with modern appeal. The cont
asting black raglan sleeves add a bold, athletic touch, making it a versatile piece for any wardrobe.\n' +
    '\n' +
    'Key Features:\n' +
    '\n' +
    'Soft and breathable fabric for all-day comfort\n' +
    'Classic pinstripe design for a clean, stylish look\n' +
    'Contrasting raglan sleeves for a sporty edge\n' +
    'Regular fit with a crew neckline\n' +
    'Perfect for casual outings or active days, pair this tee with your favorite jeans or joggers for effortless',
  category: 'tshirt',
  isNew: false,
  sizes: [ 'M', 'L', 'S', 'XXL' ],
  _id: 'jGHNK2T7WRTGSZ6CroiCQb',
  name: 'Black Striped T-Shirt',
  price: 120,
  discountPercentage: null,
  colors: [ 'Blue', 'White', 'Black', 'Red' ],
  imageUrl: 'https://cdn.sanity.io/images/nss7idml/production/07dc8647a64f4a93a9d1fbb41e0a81fc940b903d-295x298.png'
}
```

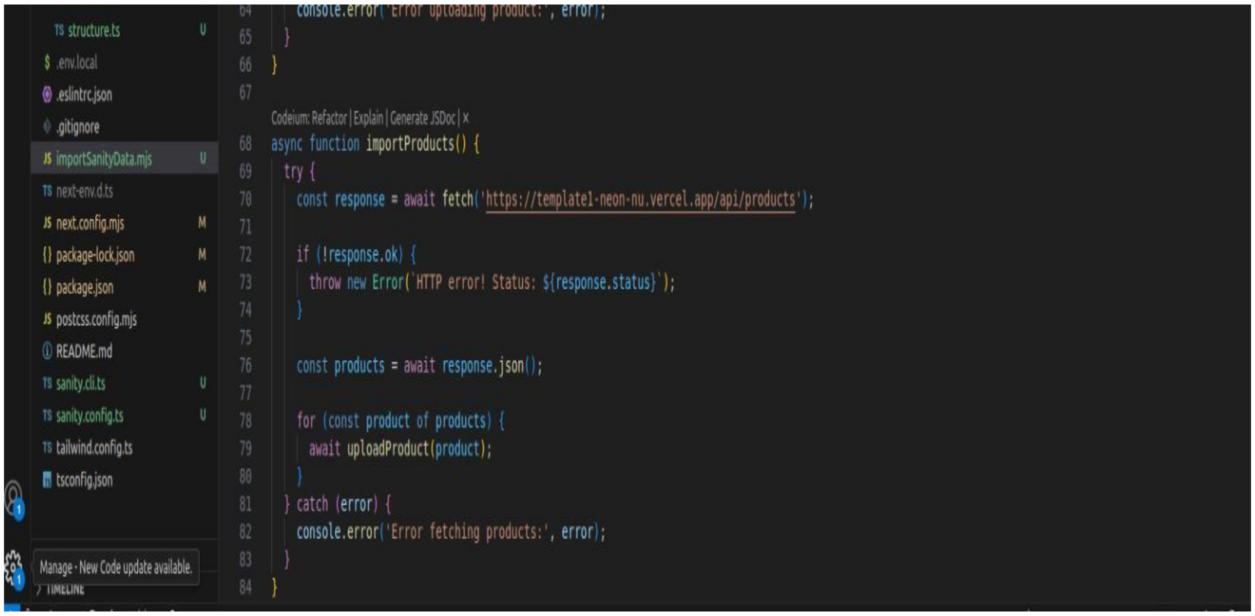
Code snippets for API integration and migration scripts



```
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: 'products',
        name: product.name,
        description: product.description,
        price: product.price,
        image: {
          _type: 'image',
          asset: {
            _ref: imageId,
          },
        },
        category: product.category,
        discountPercent: product.discountPercent,
        isNew: product.isNew,
        colors: product.colors,
        sizes: product.sizes
```



```tsx
import React from 'react';
import Link from 'next/link'; // Import Link for navigation
import { client } from '@/sanity/lib/client';
import { urlFor } from '@/sanity/lib/image';
import Image from 'next/image';

export default async function Datafetch() {
  const query = await client.fetch(`
    *[_type == "products"]{
      _id, // Include the product ID for dynamic routing
      name,
      description,
      price,
      discountPercentage,
      category,
      isNew,
      colors,
      sizes,
      "imageUrl": image.asset->url
    }
  `);

  const groupedData = query.reduce((acc: any, product: any) => {
    const category = product.category || 'Uncategorized';
    if (!acc[category]) {
      acc[category] = [];
    }
    acc[category].push(product);
    return acc;
  }, {});

  return (
    <section className="min-h-screen bg-gradient-to-br from-yellow-100 via-orange-100 to-red-50 py-12 px-6 md:px-12 lg:px-20">
      <div className="mx-auto max-w-7xl">
        <h1 className="text-4xl md:text-5xl font-extrabold text-center mb-14 text-yellow-600">
          Premium Product Collection
        </h1>

        {Object.keys(groupedData).map((category) => (
          <div key={category} className="mb-16">
            <h2 className="text-3xl font-bold text-gray-700 mb-8">{category}</h2>

            <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-10">
              {groupedData[category].map((product: any) => (
                <div
                  key={product._id}
                  className="bg-white border border-orange-200 rounded-lg shadow-lg hover:shadow-xl transition-all p-6 flex flex-col justify-between items-c
```



```js
    console.error('Error uploading product:', error);
  }
}

async function importProducts() {
  try {
    const response = await fetch('https://template1-neon-nu.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);
  }
}
```