Day 3 - API Integration Report - SHOP.CO

Table of Contents:

1. API Integration Process

Overview

- Objective: Integrate the provided API into the **Next.js** frontend to dynamically fetch and display product data

.

- API Used: custom API.
- Tools Used:
  - Postman for testing API endpoints.
  - Next.js for frontend development.
  - Sanity CMS for managing product data.

Steps Taken

1. API Understanding:
   - Reviewed the API documentation to identify key endpoints (e.g., `/products`, `/categories`).
   - Tested API endpoints using **Postman** to understand the response structure.

2. Frontend Integration:
   - Created utility functions in **Next.js** to fetch data from the API.
   - Integrated the API into the front end to display product listings, categories, and prices.
   - Implemented error handling to manage API failures (e.g., network errors, invalid data).

3. Testing:
   - Tested the API integration using **Postman** and browser developer tools.
   - Simulated error scenarios (e.g., empty responses, slow network) to ensure robust error handling.

Challenges Faced

- **Schema Mismatch:** Initially, the API fields did not match the **Sanity CMS** schema. Adjusted the schema to align with the API data structure.
- **Error Handling:** Implemented fallback data and user-friendly error messages to improve the user experience.

2. Schema Adjustments

Original Schema

- The provided schema was basic, with fields like `name`, `price`, and `category`.

Improved Schema

- Added additional fields for better representation of product data:
- Applied validation rules:

- `description:` Detailed product description.
- `images:` Array of product images (limit of 5 images).
- `tags:` Array of tags for better searchability.
- `colors:` Array of available colors.
- `reviews:` Array of references to customer reviews.
- `name:` and `price` are required fields.
- `price:` must be a positive number.
- `images:` are limited to 5 per product.

Schema Code

```
import { defineType, defineField } from 'sanity';

export default defineType({
  name: 'product',
  type: 'document',
  title: 'Product',
  fields: [
    defineField({
      name: 'name',
      type: 'string',
      title: 'Name',
      validation: (Rule) => Rule.required(),
    }),
    defineField({
      name: 'price',
      type: 'number',
      title: 'Price',
      validation: (Rule) => Rule.required().min(0),
    }),
    defineField({
      name: 'discount',
      type: 'number',
      title: 'Discount (%)',
    }),
    defineField({
      name: 'category',
      type: 'reference',
      title: 'Category',
      to: [{ type: 'category' }],
      validation: (Rule) => Rule.required(),
    }),
    defineField({
      name: 'description',
      type: 'text',
      title: 'Description',
    }),
    defineField({
      name: 'images',
      type: 'array',
      title: 'Images',
      of: [{ type: 'image' }],
      validation: (Rule) => Rule.max(5).warning('You can upload up to 5
images only.'),
    }),
    defineField({
      name: 'tags',
      type: 'array',
      title: 'Tags',
      of: [{ type: 'string' }],
    }),
    defineField({
      name: 'colors',
      type: 'array',
      title: 'Colors',
      of: [{ type: 'string' }],
    }),
    defineField({
      name: 'reviews',
      type: 'array',
      title: 'Reviews',
      of: [{ type: 'reference', to: [{ type: 'review' }] }],
    }),
  ],
  preview: {
    select: {
      title: 'name',
      subtitle: 'price',
      media: 'images.0',
    },
    prepare({ title, subtitle, media }) {
      return {
        title,
        subtitle: `Price: $${subtitle}`,
        media,
      };
    },
  },
});
```

3. Data Migration

Migration Method

- Used **migration scripts** to fetch data from the custom API and populate **Sanity CMS**
- Additional data was manually imported using **Sanity's** built-in import tools.

Steps Taken

1. Fetched data from the API using a script.
2. Transformed the data to match the **Sanity CMS** schema.
3. Imported the data into **Sanity CMS**
4. Verified that all fields were correctly populated.

Challenges Faced

- **Data Transformation:** Some API fields required transformation to match the schema (e.g., converting `product_title` to `name`).
- **Validation Errors:** Fixed validation errors during data import (e.g., missing required fields).

4. Error Handling

Implementation

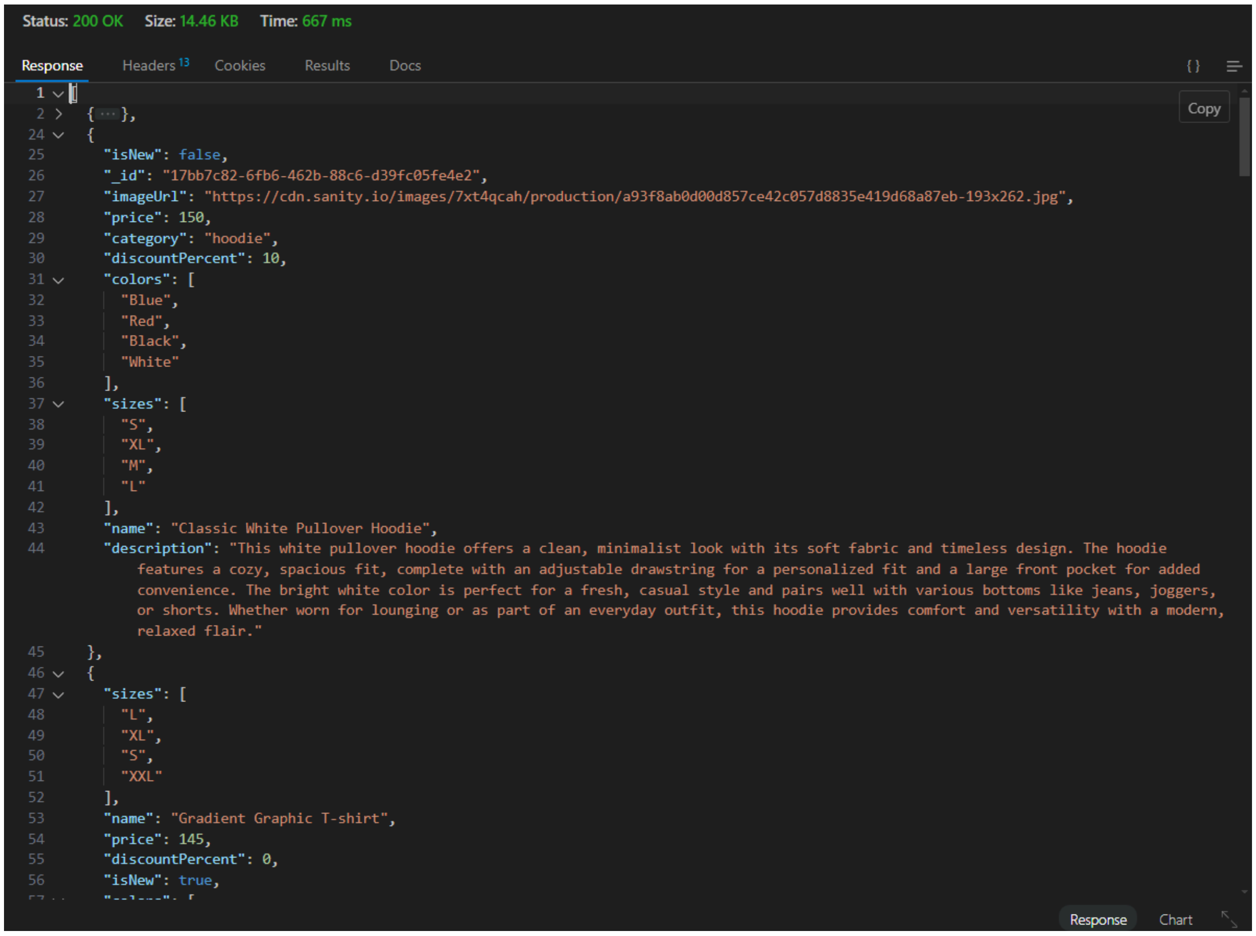
- Added error handling for API calls:
  - Displayed user-friendly error messages (e.g., "Failed to load products. Please try again later.").
  - Used fallback data or skeleton loaders to improve the user experience during loading or errors.

Error Scenarios Tested

- **API Downtime**: Displayed an error message when the API was unavailable.
- **Invalid Data**: Handled cases where the API returned incomplete or invalid data.

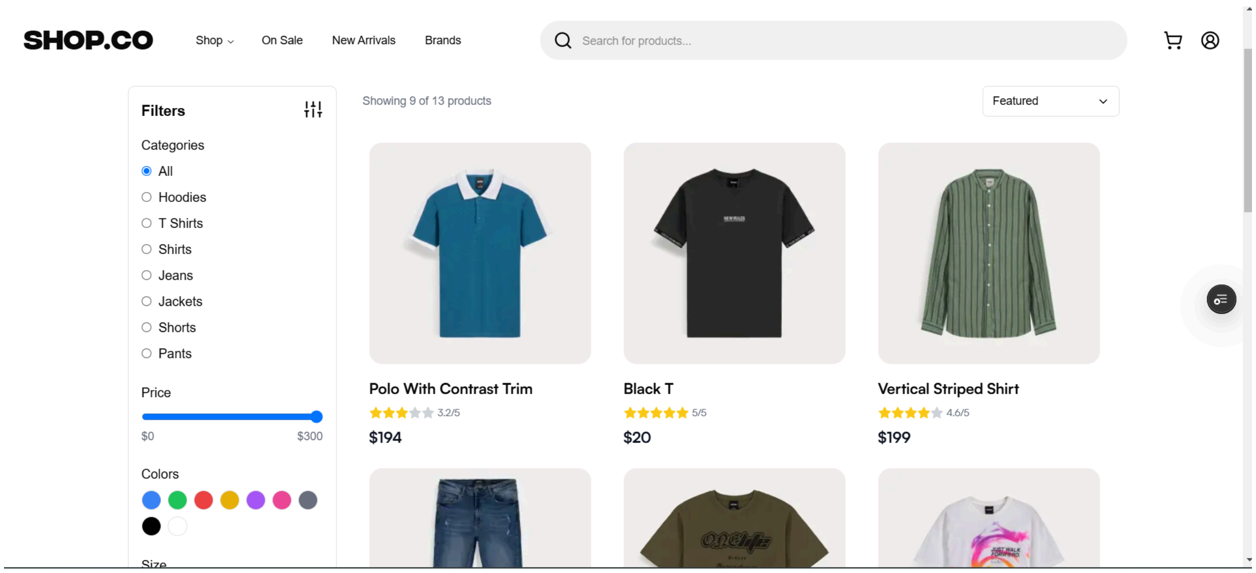
<u>5. Screenshots</u>

API Calls



```
[
  {…},
  {
    "isNew": false,
    "_id": "17bb7c82-6fb6-462b-88c6-d39fc05fe4e2",
    "imageUrl": "https://cdn.sanity.io/images/7xt4qcah/production/a93f8ab0d00d857ce42c057d8835e419d68a87eb-193x262.jpg",
    "price": 150,
    "category": "hoodie",
    "discountPercent": 10,
    "colors": [
      "Blue",
      "Red",
      "Black",
      "White"
    ],
    "sizes": [
      "S",
      "XL",
      "M",
      "L"
    ],
    "name": "Classic White Pullover Hoodie",
    "description": "This white pullover hoodie offers a clean, minimalist look with its soft fabric and timeless design. The hoodie
        features a cozy, spacious fit, complete with an adjustable drawstring for a personalized fit and a large front pocket for added
        convenience. The bright white color is perfect for a fresh, casual style and pairs well with various bottoms like jeans, joggers,
        or shorts. Whether worn for lounging or as part of an everyday outfit, this hoodie provides comfort and versatility with a modern,
        relaxed flair."
  },
  {
    "sizes": [
      "L",
      "XL",
      "S",
      "XXL"
    ],
    "name": "Gradient Graphic T-shirt",
    "price": 145,
    "discountPercent": 0,
    "isNew": true,
```

Frontend Data Display



Showing 9 of 13 products

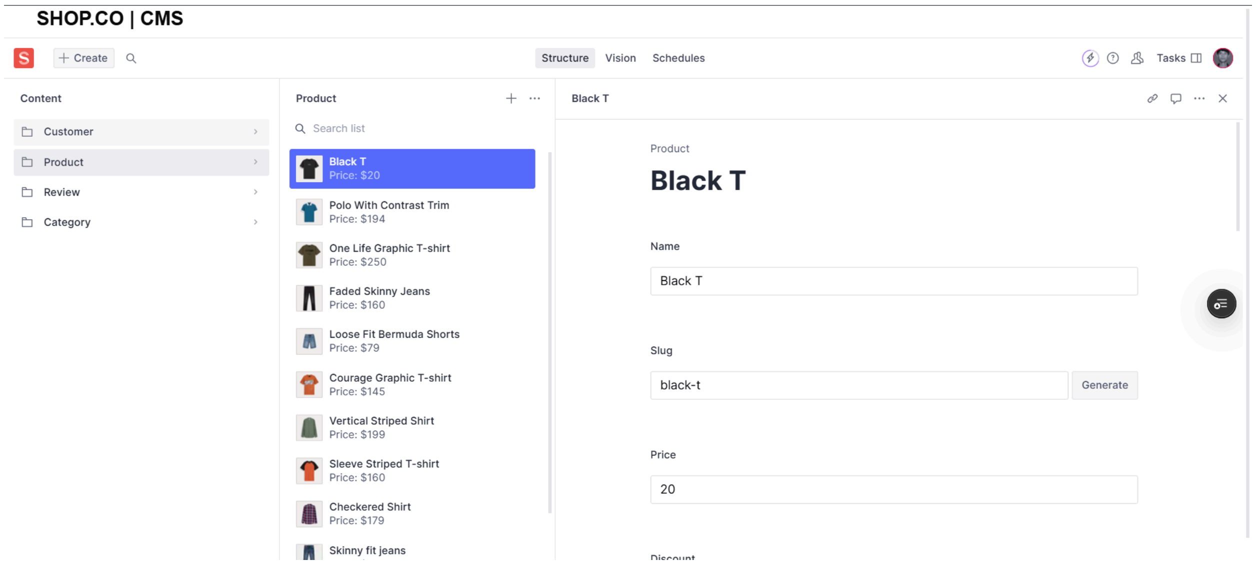
Categories
- All
- Hoodies
- T Shirts
- Shirts
- Jeans
- Jackets
- Shorts
- Pants

Polo With Contrast Trim
3.2/5
$194

Black T
5/5
$20

Vertical Striped Shirt
4.6/5
$199

Populated Sanity CMS Fields



Product

Black T
Price: $20

Polo With Contrast Trim
Price: $194

One Life Graphic T-shirt
Price: $250

Faded Skinny Jeans
Price: $160

Loose Fit Bermuda Shorts
Price: $79

Courage Graphic T-shirt
Price: $145

Vertical Striped Shirt
Price: $199

Sleeve Striped T-shirt
Price: $160

Checkered Shirt
Price: $179

Skinny fit jeans

Black T

Product

Black T

Name: Black T

Slug: black-t

Price: 20

Discount

<u>6. Code Snippets</u>

API Integration

```
// Utility function to fetch products
export async function fetchProducts() {
  try {
    const response = await fetch('/api/products');
    if (!response.ok) throw new Error('Failed to fetch products');
    return await response.json();
  } catch (error) {
    console.error(error);
    return [];
  }
}
```

Error Handling

```
// Display error message in UI
function ProductList() {
  const [products, setProducts] = useState([]);
  const [error, setError] = useState('');

  useEffect(() => {
    fetchProducts()
      .then(setProducts)
      .catch(() => setError('Failed to load products. Please try again
later.'));
  }, []);

  if (error) return <div className="error">{error}</div>;
  return <div>{products.map((product) => <ProductCard key={product.id}
{...product} />)}</div>;
}
```

7. Best Practices Followed

- Used `.env` files to store sensitive data like API keys.
- Followed clean coding practices (e.g., modular functions, descriptive variable names).
- Documented every step of the process for future reference.
- Used version control (Git) to track changes and tag milestones.

8. Conclusion

- Successfully integrated the API into the **Next.js** frontend.
- Migrated data into **Sanity CMS** and adjusted the schema for better compatibility.
- Implemented robust error handling to ensure a smooth user experience.
- Prepared for submission with detailed documentation, screenshots, and code snippets.