Day 3 - API Integration and Data Migration Report

Marketplace Name: SXN by NASH

API Integration

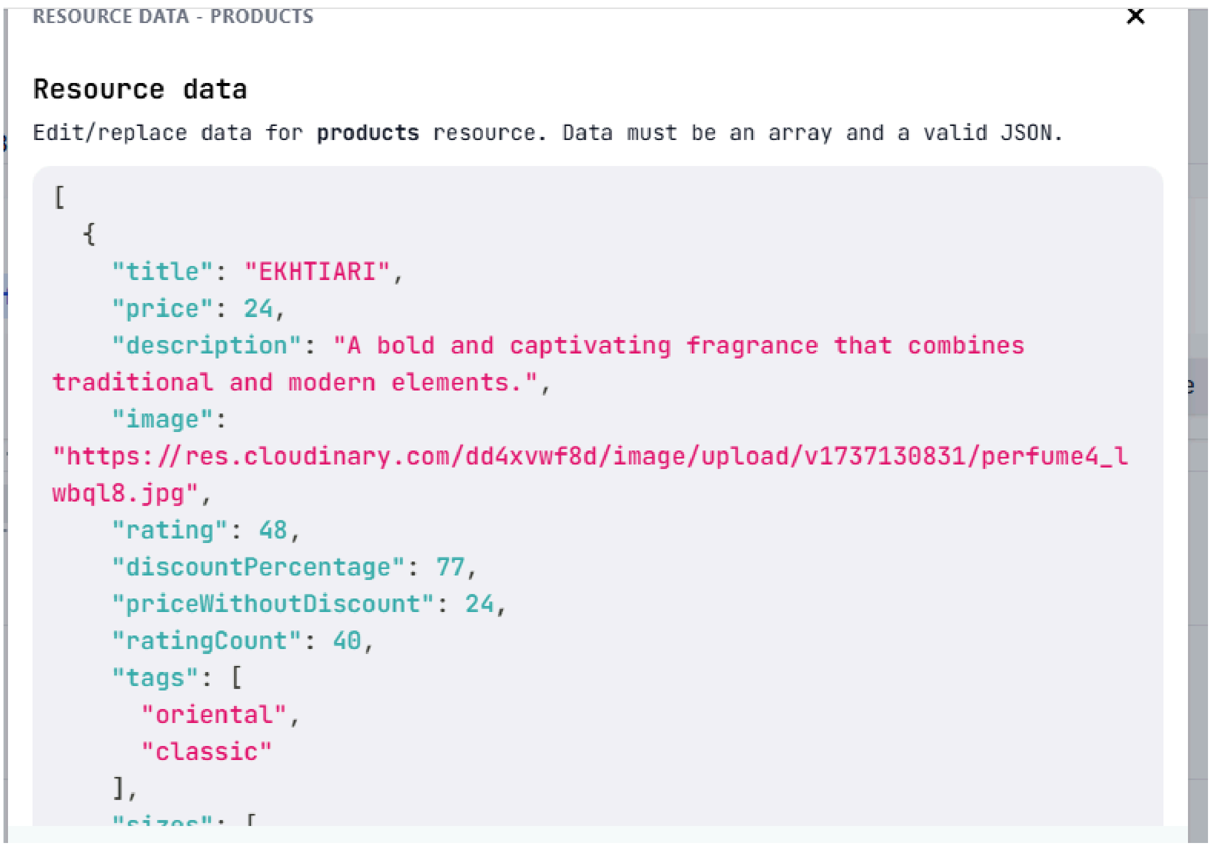
Overview

API integration is a crucial step in connecting external data sources with our frontend and CMS. For this project, we created a mock API with custom data to simulate real-world scenarios. This API provides endpoints for product listings, categories, and other relevant data, enabling a seamless data flow into our Sanity CMS and Next.js frontend.

Step 1: Creating a Mock API

1. **Purpose**: To simulate a real API for testing data integration and migration.
2. **Implementation**:
   - Defined endpoints such as `/products` and `/categories`.
   - Used tools like `json-server` or a custom Express.js server to serve mock data.

Data Format Example:



Resource data

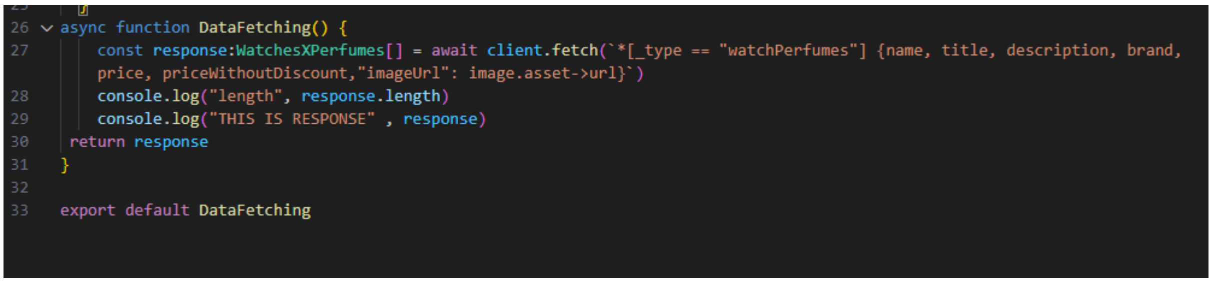
Edit/replace data for **products** resource. Data must be an array and a valid JSON.

```
[
  {
    "title": "EKHTIARI",
    "price": 24,
    "description": "A bold and captivating fragrance that combines
traditional and modern elements.",
    "image":
"https://res.cloudinary.com/dd4xvwf8d/image/upload/v1737130831/perfume4_l
wbql8.jpg",
    "rating": 48,
    "discountPercentage": 77,
    "priceWithoutDiscount": 24,
    "ratingCount": 40,
    "tags": [
      "oriental",
      "classic"
    ],
```

Step 2: Integrating API with Next.js

1. **Utility Functions**:

Created a utility functions for fetching data from Sanity:

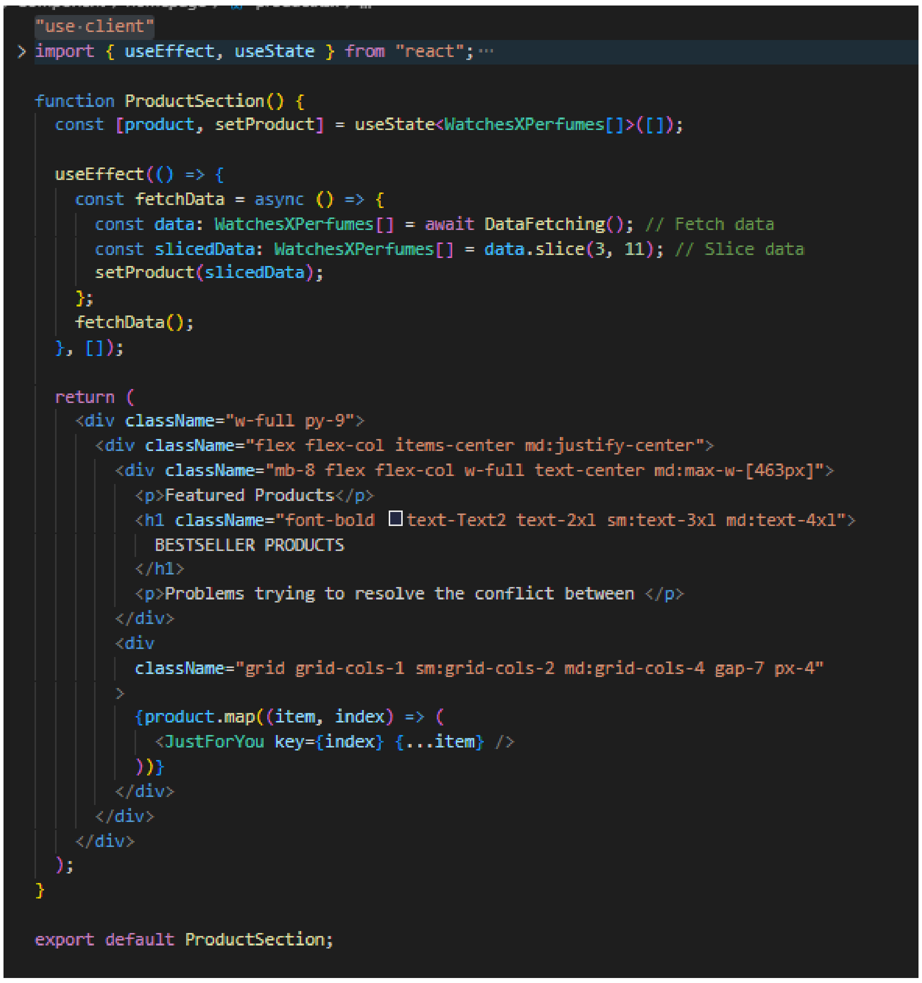


```
async function DataFetching() {
    const response:WatchesXPerfumes[] = await client.fetch(`*[_type == "watchPerfumes"] {name, title, description, brand,
    price, priceWithoutDiscount,"imageUrl": image.asset->url}`)
    console.log("length", response.length)
    console.log("THIS IS RESPONSE" , response)
  return response
}

export default DataFetching
```

2. **Rendering Data in Components**:

Used the fetched data to render product listings and categories in Next.js components:



```
"use client"
> import { useEffect, useState } from "react";⋯

function ProductSection() {
  const [product, setProduct] = useState<WatchesXPerfumes[]>([]);

  useEffect(() => {
    const fetchData = async () => {
      const data: WatchesXPerfumes[] = await DataFetching(); // Fetch data
      const slicedData: WatchesXPerfumes[] = data.slice(3, 11); // Slice data
      setProduct(slicedData);
    };
    fetchData();
  }, []);

  return (
    <div className="w-full py-9">
      <div className="flex flex-col items-center md:justify-center">
        <div className="mb-8 flex flex-col w-full text-center md:max-w-[463px]">
          <p>Featured Products</p>
          <h1 className="font-bold text-Text2 text-2xl sm:text-3xl md:text-4xl">
            BESTSELLER PRODUCTS
          </h1>
          <p>Problems trying to resolve the conflict between </p>
        </div>
        <div
          className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-4 gap-7 px-4"
        >
          {product.map((item, index) => (
            <JustForYou key={index} {...item} />
          ))}
        </div>
      </div>
    </div>
  );
}

export default ProductSection;
```

○

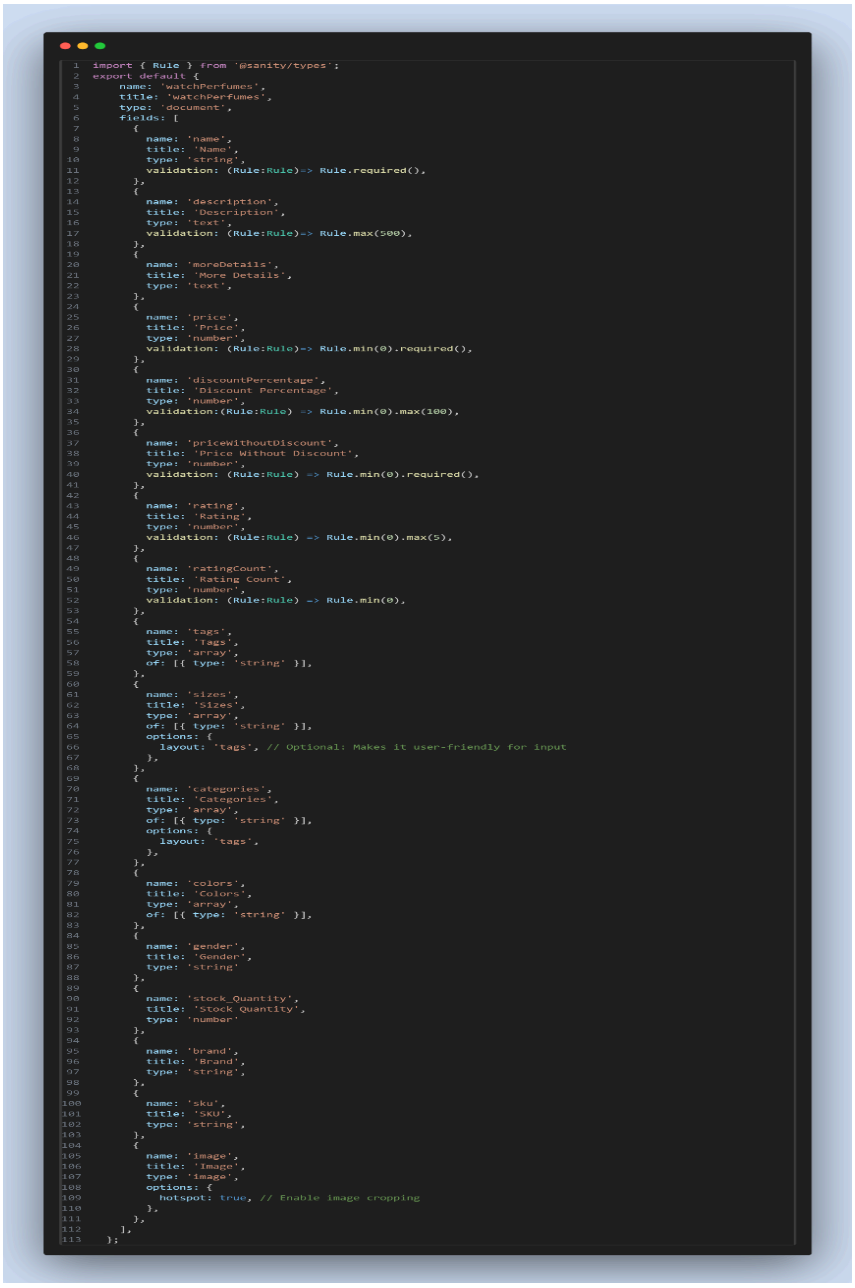
3. **Testing API Integration**:

        - Verified API responses using Postman and browser developer tools.
        - Logged responses to ensure data consistency and identify issues.

Data Migration

Step 1: Validating and Adjusting Schema

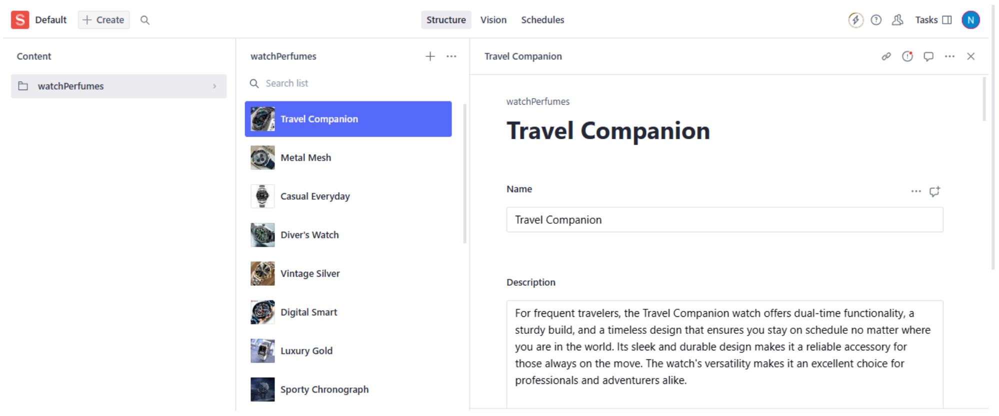
1. Compared the mock API fields with our Sanity CMS schema.
2. Adjusted field names, types, and relationships to ensure compatibility.
    - Example:
        - API Field: `product_title`
        - Schema Field: `name`



```
import { Rule } from '@sanity/types';
export default {
    name: 'watchPerfumes',
    title: 'watchPerfumes',
    type: 'document',
    fields: [
      {
        name: 'name',
        title: 'Name',
        type: 'string',
        validation: (Rule:Rule)=> Rule.required(),
      },
      {
        name: 'description',
        title: 'Description',
        type: 'text',
        validation: (Rule:Rule)=> Rule.max(500),
      },
      {
        name: 'moreDetails',
        title: 'More Details',
        type: 'text',
      },
      {
        name: 'price',
        title: 'Price',
        type: 'number',
        validation: (Rule:Rule)=> Rule.min(0).required(),
      },
      {
        name: 'discountPercentage',
        title: 'Discount Percentage',
        type: 'number',
        validation:(Rule:Rule) => Rule.min(0).max(100),
      },
      {
        name: 'priceWithoutDiscount',
        title: 'Price Without Discount',
        type: 'number',
        validation: (Rule:Rule) => Rule.min(0).required(),
      },
      {
        name: 'rating',
        title: 'Rating',
        type: 'number',
        validation: (Rule:Rule) => Rule.min(0).max(5),
      },
      {
        name: 'ratingCount',
        title: 'Rating Count',
        type: 'number',
        validation: (Rule:Rule) => Rule.min(0),
      },
      {
        name: 'tags',
        title: 'Tags',
        type: 'array',
        of: [{ type: 'string' }],
      },
      {
        name: 'sizes',
        title: 'Sizes',
        type: 'array',
        of: [{ type: 'string' }],
        options: {
          layout: 'tags', // Optional: Makes it user-friendly for input
        },
      },
      {
        name: 'categories',
        title: 'Categories',
        type: 'array',
        of: [{ type: 'string' }],
        options: {
          layout: 'tags',
        },
      },
      {
        name: 'colors',
        title: 'Colors',
        type: 'array',
        of: [{ type: 'string' }],
      },
      {
        name: 'gender',
        title: 'Gender',
        type: 'string'
      },
      {
        name: 'stock_Quantity',
        title: 'Stock Quantity',
        type: 'number'
      },
      {
        name: 'brand',
        title: 'Brand',
        type: 'string',
      },
      {
        name: 'sku',
        title: 'SKU',
        type: 'string',
      },
      {
        name: 'image',
        title: 'Image',
        type: 'image',
        options: {
          hotspot: true, // Enable image cropping
        },
      },
    ],
  };
```

○

Step 2: Importing Data to Sanity CMS



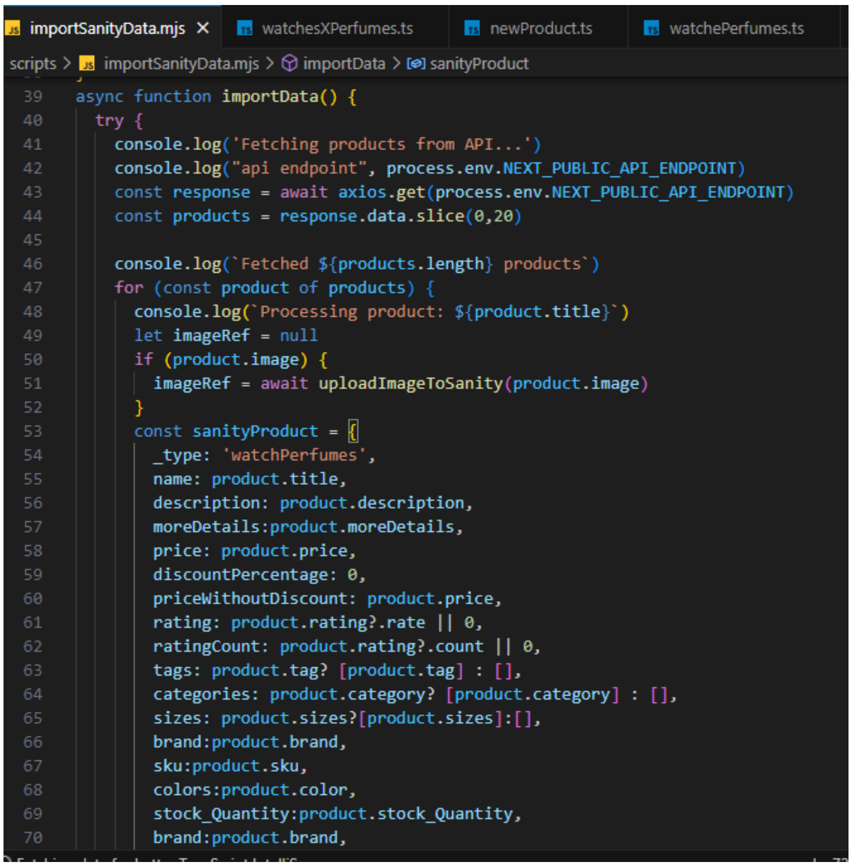
Travel Companion
Metal Mesh
Casual Everyday
Diver's Watch
Vintage Silver
Digital Smart
Luxury Gold
Sporty Chronograph

watchPerfumes

Travel Companion

Name: Travel Companion

Description: For frequent travelers, the Travel Companion watch offers dual-time functionality, a sturdy build, and a timeless design that ensures you stay on schedule no matter where you are in the world. Its sleek and durable design makes it a reliable accessory for those always on the move. The watch's versatility makes it an excellent choice for professionals and adventurers alike.



```js
async function importData() {
  try {
    console.log('Fetching products from API...')
    console.log("api endpoint", process.env.NEXT_PUBLIC_API_ENDPOINT)
    const response = await axios.get(process.env.NEXT_PUBLIC_API_ENDPOINT)
    const products = response.data.slice(0,20)

    console.log(`Fetched ${products.length} products`)
    for (const product of products) {
      console.log(`Processing product: ${product.title}`)
      let imageRef = null
      if (product.image) {
        imageRef = await uploadImageToSanity(product.image)
      }
      const sanityProduct = {
        _type: 'watchPerfumes',
        name: product.title,
        description: product.description,
        moreDetails:product.moreDetails,
        price: product.price,
        discountPercentage: 0,
        priceWithoutDiscount: product.price,
        rating: product.rating?.rate || 0,
        ratingCount: product.rating?.count || 0,
        tags: product.tag? [product.tag] : [],
        categories: product.category? [product.category] : [],
        sizes: product.sizes?[product.sizes]:[],
        brand:product.brand,
        sku:product.sku,
        colors:product.color,
        stock_Quantity:product.stock_Quantity,
        brand:product.brand,
```

1. **Validation**:
   - Verified data consistency post-import by checking the Sanity Studio interface.

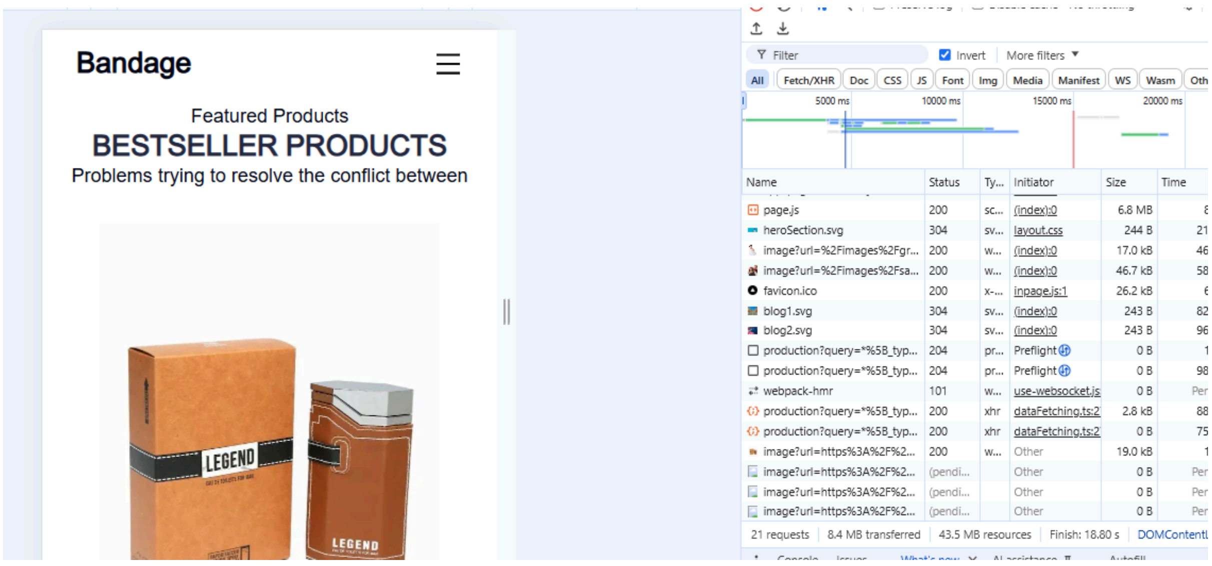
Step 3: Fetching Data from Sanity CMS

1. **Using `client.fetch()`**:

Queried the Sanity dataset using GROQ queries:

```ts
const response:WatchesXPerfumes[] = await client.fetch(`*[_type == "watchPerfumes"] {name, title, description, brand, price, priceWithoutDiscount,"imageUrl": image.asset->url}`)
```

2. **Rendering Sanity Data in Frontend**:
   - Replaced mock API data with Sanity CMS data in the Next.js components.
   - Confirmed seamless integration between frontend and Sanity.



Bandage

Featured Products

BESTSELLER PRODUCTS

Problems trying to resolve the conflict between

Name	Status	Ty...	Initiator	Size	Time
page.js	200	sc...	(index):0	6.8 MB	8
heroSection.svg	304	sv...	layout.css	244 B	21
image?url=%2Fimages%2Fgr...	200	w...	(index):0	17.0 kB	46
image?url=%2Fimages%2Fsa...	200	w...	(index):0	46.7 kB	58
favicon.ico	200	x-...	inpage.js:1	26.2 kB	6
blog1.svg	304	sv...	(index):0	243 B	82
blog2.svg	304	sv...	(index):0	243 B	96
production?query=*%5B_typ...	204	pr...	Preflight	0 B	1
production?query=*%5B_typ...	204	pr...	Preflight	0 B	98
webpack-hmr	101	w...	use-websocket.js	0 B	Per
production?query=*%5B_typ...	200	xhr	dataFetching.ts:2	2.8 kB	88
production?query=*%5B_typ...	200	xhr	dataFetching.ts:2	0 B	75
image?url=https%3A%2F%2...	200	w...	Other	19.0 kB	1
image?url=https%3A%2F%2...	(pendi...		Other	0 B	Per
image?url=https%3A%2F%2...	(pendi...		Other	0 B	Per
image?url=https%3A%2F%2...	(pendi...		Other	0 B	Per

21 requests | 8.4 MB transferred | 43.5 MB resources | Finish: 18.80 s | DOMContentL

Learnings and Insights

- Gained hands-on experience with API integration, mock data creation, and data migration.
- Improved understanding of schema design and validation in Sanity CMS.
- Enhanced skills in handling errors and creating a smooth user experience during API calls.
- Discovered best practices for data migration and maintaining data integrity.