REPORT:DAY-3

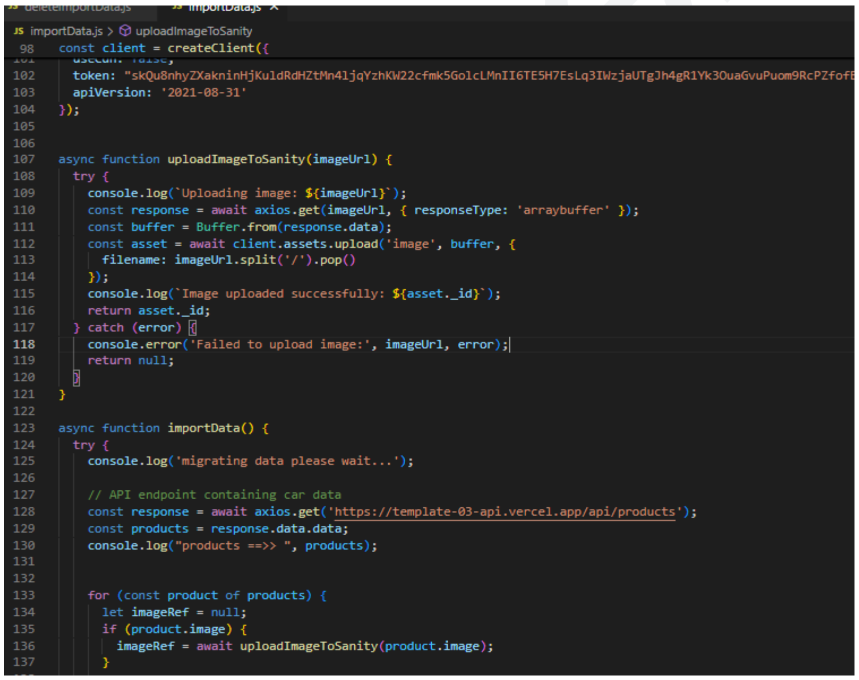
<u>API Integration Report - General E-commerce:</u>

1. Introduction:

In this report, I will explain how I used Sanity CMS to dynamically fetch and display product cards on the frontend of the marketplace website. These cards were populated with product details stored in Sanity, and I used a combination of GROQ queries and schema to display them dynamically on the browser.

1. API Details

- **API Endpoint**: `https://template-03-api.vercel.app/api/products`
- **Purpose**: Fetch product data (including images) for migration into Sanity CMS.



```
importData.js > uploadImageToSanity
const client = createClient({
  token: "skQu8nhyZXakninHjKuldRdHZtMn4ljqYzhKW22cfmk5GolcLMnII6TE5H7EsLq3IWzjaUTgJh4gR1Yk3OuaGvuPuom9RcPZfofE
  apiVersion: '2021-08-31'
});


async function uploadImageToSanity(imageUrl) {
  try {
    console.log(`Uploading image: ${imageUrl}`);
    const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
    const buffer = Buffer.from(response.data);
    const asset = await client.assets.upload('image', buffer, {
      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 importData() {
  try {
    console.log('migrating data please wait...');

    // API endpoint containing car data
    const response = await axios.get('https://template-03-api.vercel.app/api/products');
    const products = response.data.data;
    console.log("products ==>> ", products);


    for (const product of products) {
      let imageRef = null;
      if (product.image) {
        imageRef = await uploadImageToSanity(product.image);
      }
```

2. Functions and Their Purpose

- **`uploadImageToSanity(imageUrl)`**:
  - Uploads images from the API to Sanity CMS.

    - Converts image URLs into assets compatible with Sanity.
    - Logs success or failure for debugging.
- `importData()`:
    - Fetches product data using `axios.get()`.
    - Processes each product, uploading images and storing other details in Sanity.
    - Handles errors gracefully during the migration process.

3. Key Features

- **Error Handling**: Logs errors during image uploads or data processing.
- **Data Validation**: Ensures schema compatibility and logs successful uploads.

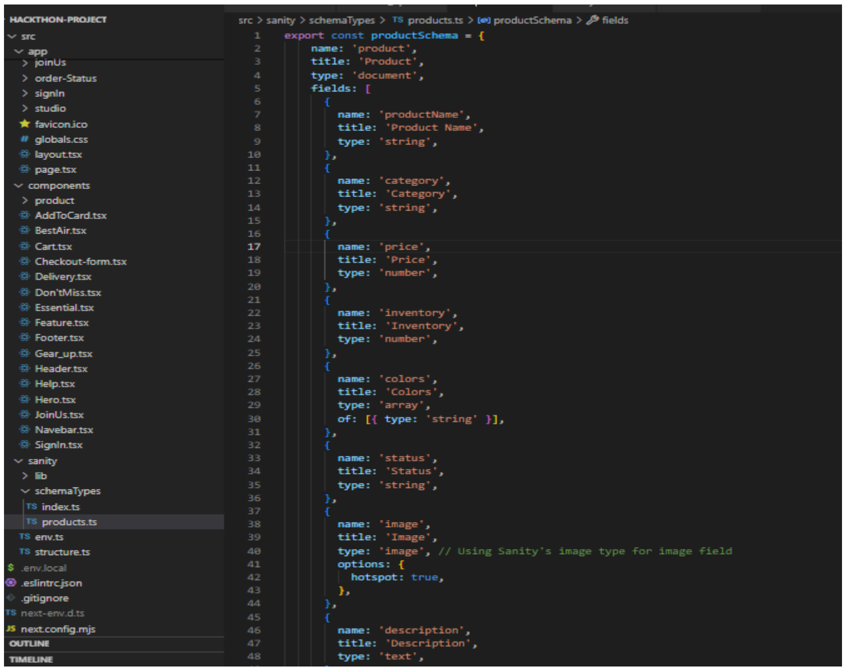
4. Tools Used

- **Sanity Client**: For image uploads and data storage.
- **Axios**: For API calls and data fetching.

2. Sanity CMS Schema for Product Cards:

To begin, I created a schema for the product cards to store data such as the product title, description, price, and an image. Each product is stored as a document in Sanity CMS, which can then be fetched dynamically using the GROQ query.

Schema for Product Card:



```
export const productSchema = {
    name: 'product',
    title: 'Product',
    type: 'document',
    fields: [
      {
        name: 'productName',
        title: 'Product Name',
        type: 'string',
      },
      {
        name: 'category',
        title: 'Category',
        type: 'string',
      },
      {
        name: 'price',
        title: 'Price',
        type: 'number',
      },
      {
        name: 'inventory',
        title: 'Inventory',
        type: 'number',
      },
      {
        name: 'colors',
        title: 'Colors',
        type: 'array',
        of: [{ type: 'string' }],
      },
      {
        name: 'status',
        title: 'Status',
        type: 'string',
      },
      {
        name: 'image',
        title: 'Image',
        type: 'image', // Using Sanity's image type for image field
        options: {
          hotspot: true,
        },
      },
      {
        name: 'description',
        title: 'Description',
        type: 'text',
```

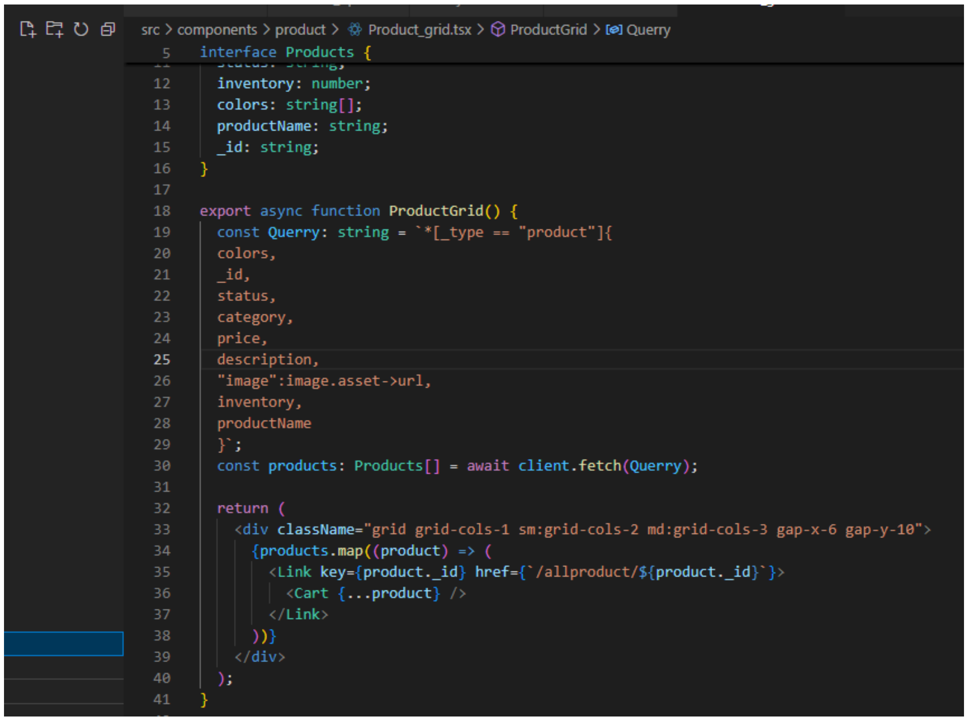
Explanation:

- `title`: The name of the product.
- `description`: A brief description of the product.
- `price`: The price of the product.
- `image`: An image field to upload the product image.

3. GROQ Query to Fetch Product Cards:

Next, I used a GROQ query to fetch the product card data from Sanity CMS. The query is designed to retrieve the title, description, price, and image of all the products stored in Sanity.

GROQ Query:



src > components > product > Product_grid.tsx > ProductGrid > Querry

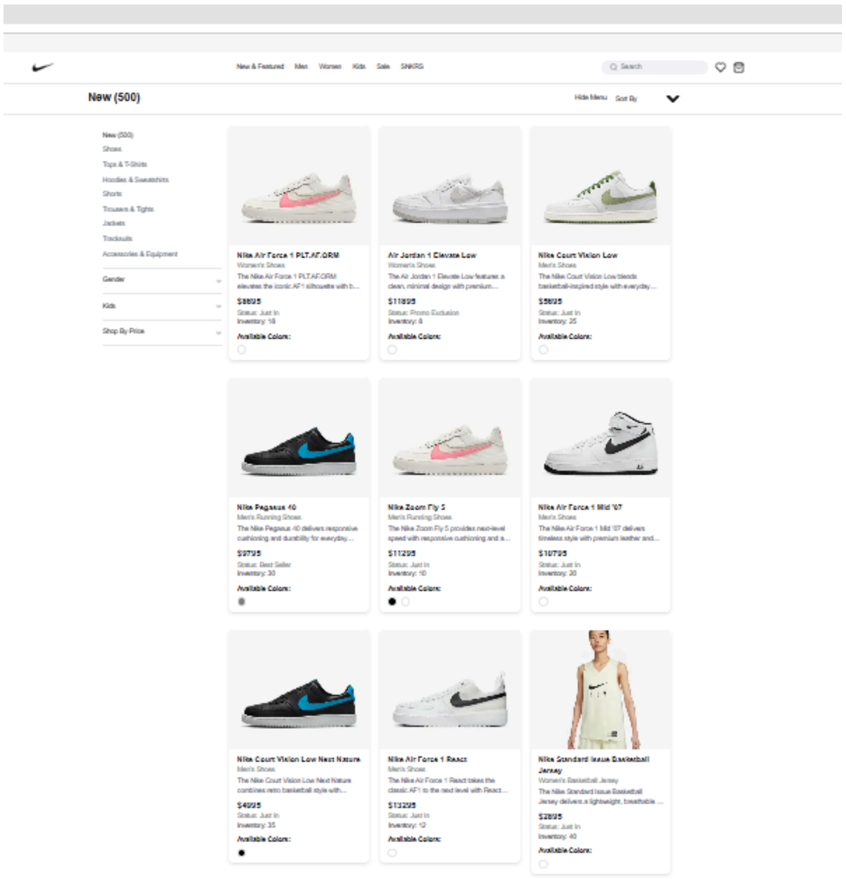
```tsx
interface Products {
  inventory: number;
  colors: string[];
  productName: string;
  _id: string;
}

export async function ProductGrid() {
  const Querry: string = `*[_type == "product"]{
  colors,
  _id,
  status,
  category,
  price,
  description,
  "image":image.asset->url,
  inventory,
  productName
  }`;
  const products: Products[] = await client.fetch(Querry);

  return (
    <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 gap-x-6 gap-y-10">
      {products.map((product) => (
        <Link key={product._id} href={`/allproduct/${product._id}`}>
          <Cart {...product} />
        </Link>
      ))}
    </div>
  );
}
```

Explanation:

- `_type == "productCard"`: This filters the query to only return documents of type `productCard`.
- `image { asset -> { url } }`: This fetches the URL of the image asset, which is essential for rendering the product image dynamically.

4. Dynamic Card Rendering on Frontend:

I used Next.js to fetch this data and dynamically render the product cards on the browser.



Nike Air Force 1 PLT.AF.ORM
Women's Shoes
The Nike Air Force 1 PLT.AF.ORM elevates the iconic AF1 silhouette with b...
$8895
Status: Just In
Inventory: 18
Available Colors:
○

Air Jordan 1 Elevate Low
Women's Shoes
The Air Jordan 1 Elevate Low features a clean, minimal design with premium...
$11895
Status: Promo Exclusion
Inventory: 8
Available Colors:
○

Nike Court Vision Low
Men's Shoes
The Nike Court Vision Low blends basketball-inspired style with everyday...
$5895
Status: Just In
Inventory: 25
Available Colors:
○

Nike Pegasus 40
Men's Running Shoes
The Nike Pegasus 40 delivers responsive cushioning and durability for everyday...
$9795
Status: Best Seller
Inventory: 30
Available Colors:
●

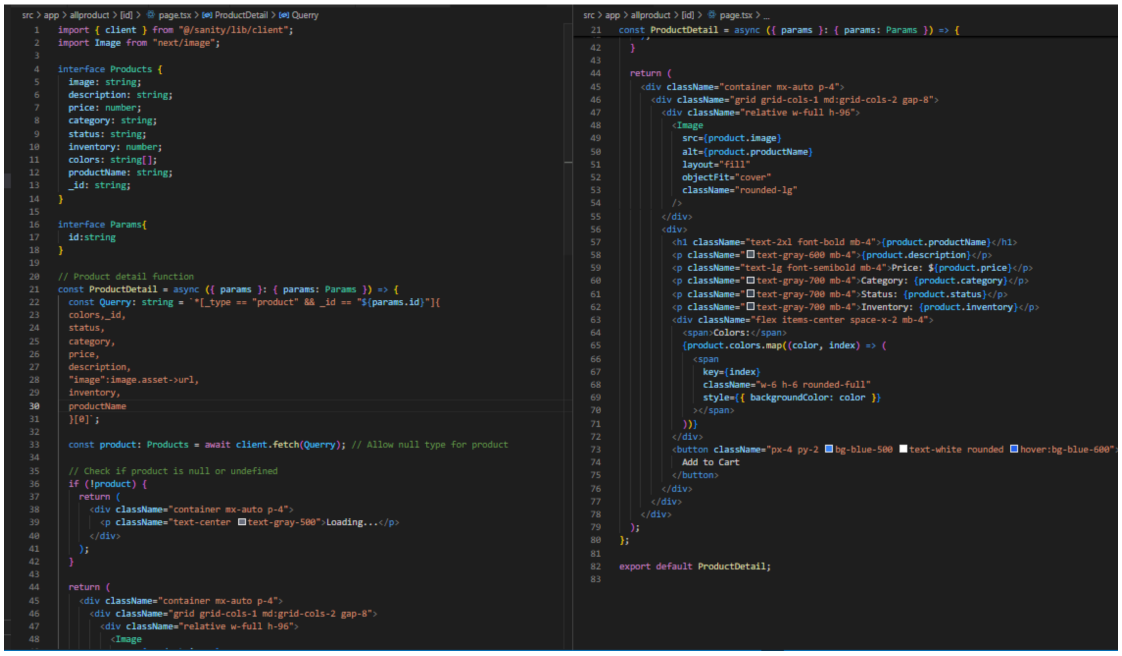
Nike Zoom Fly 5
Men's Running Shoes
The Nike Zoom Fly 5 provides next-level speed with responsive cushioning and a...
$11295
Status: Just In
Inventory: 10
Available Colors:
● ○

Nike Air Force 1 Mid '07
Men's Shoes
The Nike Air Force 1 Mid '07 delivers timeless style with premium leather and...
$10795
Status: Just In
Inventory: 20
Available Colors:
○

Nike Court Vision Low Next Nature
Men's Shoes
The Nike Court Vision Low Next Nature combines retro basketball style with...
$4995
Status: Just In
Inventory: 35
Available Colors:
●

Nike Air Force 1 React
Men's Shoes
The Nike Air Force 1 React takes the classic AF1 to the next level with React...
$13295
Status: Just In
Inventory: 12
Available Colors:
○

Nike Standard Issue Basketball Jersey
Women's Basketball Jersey
The Nike Standard Issue Basketball Jersey delivers a lightweight, breathable ...
$2895
Status: Just In
Inventory: 40
Available Colors:
○



src > app > allproduct > [id] > page.tsx > ProductDetail > Querry

```tsx
import { client } from "@/sanity/lib/client";
import Image from "next/image";

interface Products {
  image: string;
  description: string;
  price: number;
  category: string;
  status: string;
  inventory: number;
  colors: string[];
  productName: string;
  _id: string;
}

interface Params{
  id:string
}

// Product detail function
const ProductDetail = async ({ params }: { params: Params }) => {
  const Querry: string = `*[_type == "product" && _id == "${params.id}"]{
  colors,_id,
  status,
  category,
  price,
  description,
  "image":image.asset->url,
  inventory,
  productName
  }[0]`;

  const product: Products = await client.fetch(Querry); // Allow null type for product

  // Check if product is null or undefined
  if (!product) {
    return (
      <div className="container mx-auto p-4">
        <p className="text-center text-gray-500">Loading...</p>
      </div>
    );
  }

  return (
    <div className="container mx-auto p-4">
      <div className="grid grid-cols-1 md:grid-cols-2 gap-8">
        <div className="relative w-full h-96">
          <Image
```

```tsx
const ProductDetail = async ({ params }: { params: Params }) => {
  }

  return (
    <div className="container mx-auto p-4">
      <div className="grid grid-cols-1 md:grid-cols-2 gap-8">
        <div className="relative w-full h-96">
          <Image
            src={product.image}
            alt={product.productName}
            layout="fill"
            objectFit="cover"
            className="rounded-lg"
          />
        </div>
        <div>
          <h1 className="text-2xl font-bold mb-4">{product.productName}</h1>
          <p className="text-gray-600 mb-4">{product.description}</p>
          <p className="text-lg font-semibold mb-4">Price: ${product.price}</p>
          <p className="text-gray-700 mb-4">Category: {product.category}</p>
          <p className="text-gray-700 mb-4">Status: {product.status}</p>
          <p className="text-gray-700 mb-4">Inventory: {product.inventory}</p>
          <div className="flex items-center space-x-2 mb-4">
            <span>Colors:</span>
            {product.colors.map((color, index) => (
              <span
                key={index}
                className="w-6 h-6 rounded-full"
                style={{ backgroundColor: color }}
              ></span>
            ))}
          </div>
          <button className="px-4 py-2 bg-blue-500 text-white rounded hover:bg-blue-600"
            Add to Cart
          </button>
        </div>
      </div>
    </div>
  );
};

export default ProductDetail;
```

Conclusion:

This report documents the process of integrating and rendering product cards from Sanity CMS onto the frontend. Using GROQ queries, I was able to fetch the necessary data (including images), and render the product cards dynamically using Next.js. This approach allows for easy management and updates of product data in Sanity CMS, with changes automatically reflected on the frontend.