🌐 API Integration Report

📌 Introduction

This report outlines the integration process of **Sanity CMS API** with a **Next.js** project. It covers schema structure, data fetching using **GROQ queries**, and dynamic rendering in the frontend for seamless product management.

📂 Product Schema

The following schema defines the structure of the **Product** document in **Sanity CMS:**

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

export const product = defineType({
  name: "product",
  title: "Product",
  type: "document",
  fields: [
    defineField({
      name: "image",
      title: "🖼️ Image",
      type: "image",
      options: { hotspot: true },
    }),
    defineField({
      name: "title",
      type: "string",
      title: "📝 Post Title",
      description: "Title of the post",
      validation: (Rule) => Rule.required(),
    }),
    defineField({
      name: "slug",
      type: "slug",
      title: "🔗 Slug",
      options: { source: "title", maxLength: 96 },
      validation: (Rule) => Rule.required(),
    }),
    defineField({
      name: "description",
      type: "string",
      title: "📄 Description",
      description: "Add a brief two-line description of the product here.",
      validation: (Rule) => Rule.required(),
    }),
    defineField({
      name: "price",
      type: "number",
      title: "💲 Price",
      validation: (Rule) => Rule.required(),
    }),
    defineField({
      name: "discountPrice",
      type: "number",
      title: "💸 Discount Price",
    }),
    defineField({
      name: "colors",
      type: "array",
      title: "🎨 Colors",
      of: [{ type: "string" }],
      description: "Available colors (e.g., blue, green, orange, etc.).",
      validation: (Rule) => Rule.unique().error("Each color must be unique."),
    }),
    defineField({
      name: "sizes",
      type: "array",
      title: "📏 Sizes",
      of: [{ type: "string" }],
      description: "Available sizes (e.g., small, medium, large, etc.).",
      validation: (Rule) => Rule.unique().error("Each size must be unique."),
    }),
  ],
});

export default product;
```

🔍 Fetching Data with GROQ Queries

GROQ (Graph-Relational Object Queries) is used to fetch data from **Sanity CMS**.

✅ 1. Fetch All Products

```
import { createClient } from "next-sanity";

const client = createClient({
  projectId: "<your_project_id>",
  dataset: "production",
  useCdn: true,
});

export async function getProducts() {
  const query = `*[_type == "product"]{
    title,
    slug,
    description,
    price,
    discountPrice,
    colors,
    sizes,
    "imageUrl": image.asset->url
  }`;

  return await client.fetch(query);
}
```

✅ 2. Fetch a Single Product by Slug

```
export async function getProductBySlug(slug) {
  const query = `*[_type == "product" && slug.current == $slug][0]{
    title,
    description,
    price,
    discountPrice,
    colors,
    sizes,
    "imageUrl": image.asset->url
  }`;

  return await client.fetch(query, { slug });
}
```

🎯 Conclusion

- ✅ **Sanity CMS** manages product data efficiently, offering flexibility for content updates.
- 🚀 **GROQ queries** enable seamless, structured data fetching, enhancing performance.

🔐 Security Best Practices

- **Secure API Keys:** Store sensitive data in environment variables (`.env.local`).
- **Role-Based Access:** Limit read/write permissions based on user roles in Sanity settings.

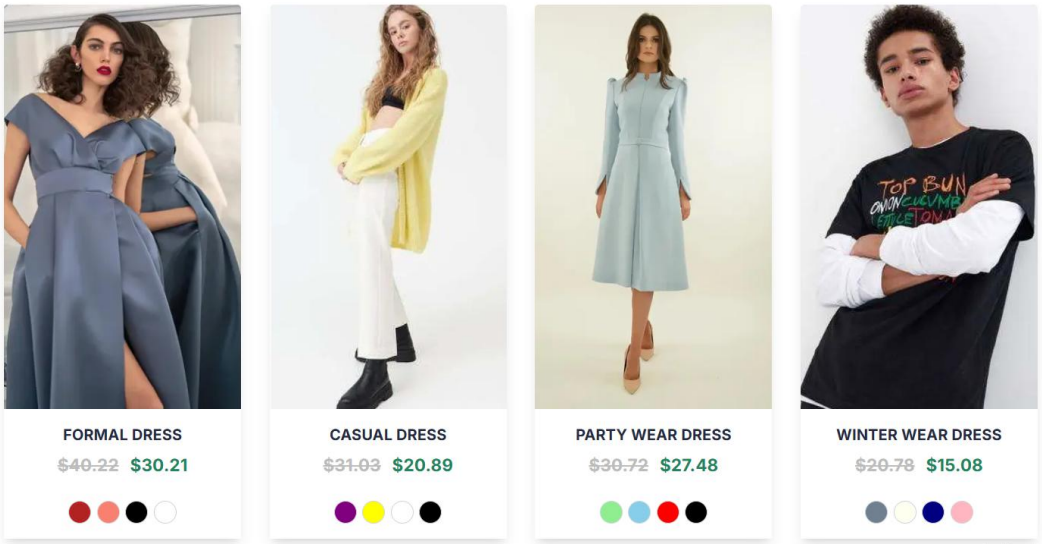
⚡ Caching Strategies

- **Static Site Generation (SSG):** Use `getStaticProps` to pre-render pages at build time.
- **Incremental Static Regeneration (ISR):** Automatically update static pages without full redeployment.

🛎️ Error Handling

- **Try-Catch Blocks:** Gracefully handle API errors with fallback UIs.
- **Validation Checks:** Ensure data completeness before rendering to prevent UI breaks

PRODUCT DISPLAY LISTNING ON BROWSER



PRODUCT DISPLAY LISTNING ON BROWSER