Day- 3

API Integration Report – Heckto

1. Introduction

This report details the steps taken to integrate Sanity CMS with the Heckto chair marketplace. The primary objective of this integration was to simplify and enhance product data management through Sanity's powerful CMS system. The process involved creating schemas, configuring environment variables, importing data using the npm run import-data command, and rendering the data on the frontend using dynamic routes from Sanity.

---

2. Importing Data into Sanity

To import data into Sanity through the API, the process followed can be found in the documentation: https://github.com/anasseth/next-ecommerce-template-4/blob/master/README.md

The import was conducted seamlessly using the API, ensuring data integrity and aligning the product information with the requirements of the Heckto chair marketplace.

I have made changes to my project according to my requirements while following this documentation.

3. Displaying Sanity Data on the Frontend

3.1 Sanity Schema

To fetch products based on specific tags, I updated the Sanity schema by adding a tags property. The schema allows products to be categorized using predefined tags for dynamic filtering. Below is the updated schema:

```
name: "tags",
title: "Tags",
type: "array",
of: [{ type: "string" }],
options: {
  list: [
    { title: "Featured", value: "featured" },
    { title: "Latest", value: "latest" },
    { title: "Unique", value: "unique" },
    { title: "Trending", value: "trending" },
```

```
      { title: "Discount", value: "discount" },
      { title: "Top", value: "top" },
      { title: "All Products", value: "allProducts" },
      { title: "Executive Seat", value: "executiveSeat" },
      { title: "Wooden", value: "wooden" },
    ],
  },
}
```

This enhancement allows the flexibility to filter and retrieve products based on specific tags, improving data organization and accessibility.

3.2 Fetching Data into the Frontend

To fetch data from Sanity and render it on the frontend, the following approach was implemented:

1. **State Management:**
   A state was created to store the product data fetched from Sanity:

```
const [product, setProduct] = useState<Product[]>([]);
```

2. **Fetching Data:**
   An async function was used to retrieve product data using the GetProductData API:

```
useEffect(() => {
  async function fetchCategoryData() {
    const categoryData: Product[] = await GetProductData();
    setProduct(categoryData);
  }
  fetchCategoryData();
}, []);
```

3. **Rendering Data:**
   The fetched products were displayed on the UI using the map() method:

```
{product.map((item: Product, index) => (
  // Render the product UI here
))}
```

This process enabled seamless integration of Sanity data into the frontend, ensuring a smooth user experience.

3.3 Displaying Data with Dynamic Routes

Dynamic routes were used to render product details based on a specific params.id. Below is the implementation:

1. **Fetching Product Details:**
   A state was created to store a specific product, and the data was fetched using the params.id:

```
const [usproduct, setProduct] = useState<Product | undefined>(undefined);
useEffect(() => {
  async function fetchCategoryData() {
    try {
      const categoryData: Product[] = await GetProductData2();
      const productData = categoryData.find(
        (data: Product) => String(data._id) === params.id
      );
      setProduct(productData);
    } catch (error) {
      console.error("Failed to fetch products:", error);
    }
  }
  fetchCategoryData();
}, [params.id]);
```

2. **Rendering the Product:**
   If the product is not found, a loading message is displayed. Otherwise, the product details are rendered:

```
if (!usproduct) {
  return <div>Loading...</div>;
}
return (
  // Render product details here
);
```

Using dynamic routes allowed specific product details to be fetched and displayed based on the user's selection, further enhancing the usability of the system.

---

4. Conclusion

The integration of Sanity CMS with the Heckto chair marketplace significantly optimized product data management and improved the overall frontend experience. By customizing schemas, importing data seamlessly, and leveraging dynamic routes, the marketplace now supports efficient product filtering and detailed displays. This integration has laid a strong foundation for scalable and maintainable data handling, paving the way for future enhancements to the platform.