

Name:Alina Fatima

Roll No: 347720

Integration of Sanity with Frontend

Overview

Integrating Sanity with the frontend in our e-commerce marketplace ensures dynamic content management, real-time updates, and a structured data flow between the backend and frontend components. This integration is powered by GROQ queries, helper functions, and schema definitions, enabling smooth data fetching and display.

1. Sanity Schema Definitions

To structure the data correctly, we defined schemas for different entities such as Products, Orders, Categories, and Users. Below is an example of the product schema:

```
import { defineField, defineType } from 'sanity';
import { TrolleyIcon } from '@sanity/icons';

export const productType = defineType({
  name: 'product',
  title: 'Products',
  type: 'document',
  icon: TrolleyIcon,
  fields: [
    defineField({ name: 'name', title: 'Product Name', type: 'string', validation: Rule => Rule.required() }),
    defineField({ name: 'slug', title: 'Slug', type: 'slug', options: { source: 'name', maxLength: 96 }, validation: Rule => Rule.required() }),
    defineField({ name: 'image', title: 'Product Image', type: 'image', options: { hotspot: true } }),
    defineField({ name: 'description', title: 'Description', type: 'text' }),
    defineField({ name: 'price', title: 'Price', type: 'number', validation: Rule => Rule.required().min(0) }),
    defineField({ name: 'categories', title: 'Categories', type: 'array', of: [{ type: 'reference', to: { type: 'category' } }] }),
    defineField({ name: 'stock', title: 'Stock', type: 'number', validation: Rule => Rule.required().min(0) })
  ]
});
```

```

1 export const productType = defineType({
2   name: 'product',
3   title: 'Products',
4   type: 'document',
5   icon: TrolleyIcon,
6   fields: [
7     defineField({
8       name: 'name',
9       title: 'Product Name',
10      type: 'string',
11      validation: Rule => Rule.required(),
12    }),
13
14    defineField({
15      name: 'slug',
16      title: 'Slug',
17      type: 'slug',
18      options: {
19        source: 'name',
20        maxLength: 96,
21      },
22      validation: Rule => Rule.required(),
23    }),
24
25    defineField({
26      name: 'image',
27      title: 'Product Image',
28      type: 'image',
29      options: {
30        hotspot: true,
31      },
32    }),
33
34    defineField({
35      name: 'description',
36      title: 'Description',
37      type: 'blockContent', // Replace blockContent with text
38      description: 'Provide a short description of the product',
39    }),
40
41    defineField({
42      name: 'price',
43      title: 'Price',
44      type: 'number',
45      validation: Rule => Rule.required().min(0),
46    }),
47
48    defineField({
49      name: 'categories',
50      title: 'Categories',
51      type: 'array',
52      of: [{ type: 'reference', to: { type: 'category' } }],
53      validation: Rule => Rule.required(),
54    }),
55
56    defineField({
57      name: 'stock',
58      title: 'Stock',
59      type: 'number',
60      validation: Rule => Rule.required().min(0),
61    }),
62  ],
63
64  preview: {
65    select: {
66      title: 'name',
67      media: 'image',
68      price: 'price',
69    },
70    prepare(select) {
71      return {
72        title: select.title,
73        subtitle: `${select.price}`,
74        media: select.media,
75      }
76    },
77  },
78 });
79
80

```

Schemas ensure data consistency and proper structuring, making frontend integration seamless.

2. Fetching Data with GROQ Queries

Sanity provides GROQ (Graph-Relational Object Queries) to fetch structured data efficiently. Below is an example GROQ query to fetch all products:

```
const ALL_PRODUCTS_QUERY = `*_type == "product" {title, description, price, image}`;
```

We use helper functions to abstract data fetching, ensuring reusability and maintainability.

3. Helper Functions for Data Fetching

To optimize fetching data from Sanity, we created helper functions using `sanityFetch`.

```
import { defineQuery } from "next-sanity";
import { sanityFetch } from "../live";

export const getAllCategories = async () => {
  const ALL_CATEGORIES_QUERY = defineQuery(`*_type=="category" | order(name asc)`);
  try {
    const categories = await sanityFetch({ query: ALL_CATEGORIES_QUERY });
    return categories.data || [];
  } catch (error) {
    console.error("Error fetching categories", error);
    return [];
  }
};
```

Similar helper functions exist for fetching products, orders, and user data, ensuring efficient API communication between the frontend and Sanity CMS.

4. Rendering Data in Next.js 15 Frontend

We dynamically fetch and display products using React components and Next.js features like Server Components or `getServerSideProps`.

Example:



```

1 import { defineQuery } from "next-sanity";
2 import { sanityFetch } from "../live";
3
4
5 export const getAllProducts = async () => {
6   const ALL_PRODUCTS_QUERY = defineQuery(
7     `*[_type=="product"]
8       | order(name asc)`
9   )
10  try{
11    // SANITY DATA FETCHING using sanityfetch
12    const products = await sanityFetch({
13      query: ALL_PRODUCTS_QUERY,
14    });
15
16    // Return the products
17    return products.data || [];
18  } catch (error){
19    console.error("Error fetching all products", error);
20    return [];
21  }
22
23 };
24

```

This approach ensures that the frontend fetches data efficiently while keeping the UI responsive and performant.

5. Real-Time Updates with Webhooks and Stripe CLI

We use Stripe CLI to handle real-time payment updates and sync orders in Sanity.

Webhook example:

```

1 import stripe from "@lib/stripe";
2
3 import { headers } from "next/headers";
4 import { NextRequest, NextResponse } from "next/server";
5 import Stripe from "stripe";
6 import { Metadata } from "../../../../../actions/createCheckoutSession";
7 import { backendClient } from "@sanity/lib/backendClient";
8
9
10
11
12 export async function POST(req :NextRequest){
13   const body = await req.text();
14   const headersList = await headers();
15   const sig = headersList.get("stripe-signature");
16
17   if (!sig) {
18     return NextResponse.json({error: "No signature"},{status:400})
19   }
20
21   const webhookSecret = process.env.STRIPE_WEBHOOK_SECRET;
22
23   if (!webhookSecret) {
24     console.log("Stripe webhook secret is not set.")
25     return NextResponse.json(
26       {error:"Stripe webhook Secret is not set"},
27       {status:400}
28     );
29   }
30
31   let event :Stripe.Event;
32   try{
33     event = stripe.webhooks.constructEvent(body, sig,webhookSecret);
34   }catch(err){
35     console.error("webhook signature varification failed: " , err);
36     return NextResponse.json({error: `Webhook Error: ${err}`}, {status:400}
37   );}
38
39
40
41
42

```

This ensures that successful Stripe payments trigger automatic order creation in Sanity.

Conclusion

Sanity and Next.js 15 integration, combined with structured schemas, GROQ queries, and helper functions, provides a seamless backend-to-frontend workflow. By leveraging Sanity's real-time content management and Stripe's secure payments, we ensure a highly efficient e-commerce marketplace. Future improvements include adding streaming capabilities, AI-driven recommendations, and performance optimizations.