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.

- **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) })
  ]
});
```

```
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: 'blockContent', // Replace blockContent with text
            description: 'Provide a short description of the product',

        }),

        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' } }],
            validation: Rule => Rule.required(),
        }),

        defineField({
            name: 'stock',
            title: 'Stock',
            type: 'number',
            validation: Rule => Rule.required().min(0),
        }),
    ],

    preview: {
        select: {
            title: 'name',
            media: 'image',
            price: 'price',
        },
        prepare(select) {
```

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

- **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.

- **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.

- **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:

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


export const getAllProducts = async () => {
    const ALL_PRODUCTS_QUERY = defineQuery(
        `*[_type=="product"]
          | order(name asc)`
    )
try{
    // SANITY DATA FECTHING using sanityfetch
    const products = await sanityFetch({
        query: ALL_PRODUCTS_QUERY,
    });

    // Return the products
    return products.data || [];
} catch (error){
    console.error("Error fetching all products", error);
    return [];
}

};

```

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

- **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:

```
import stripe from "@/lib/stripe";

import { headers } from "next/headers";
import { NextRequest, NextResponse } from "next/server";
import Stripe from "stripe";
import { Metadata } from "../../../../actions/createCheckoutSession";
import { backendClient } from "@/sanity/lib/backendClient";




export async  function POST(req :NextRequest){
    const body = await req.text();
    const headersList = await headers();
    const sig = headersList.get("stripe-signature");

    if (!sig) {

        return NextResponse.json({error: "No signature"},{status:400})

    }

    const webhookSecret = process.env.STRIPE_WEBHOOK_SECRET;

    if (!webhookSecret) {
        console.log("Stripe webhook secret is not set.")
        return NextResponse.json(
            {error:"Stripe webhook Secret is not set"},
            {status:400}
        );
    }

    let event :Stripe.Event;
    try{
        event = stripe.webhooks.constructEvent(body, sig,webhookSecret);

    }catch(err){
        console.error("webhook signature varification failed: " , err);
        return NextResponse.json({error: `Webhook Error: ${err}`}, {status:400}

        );}

```

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.