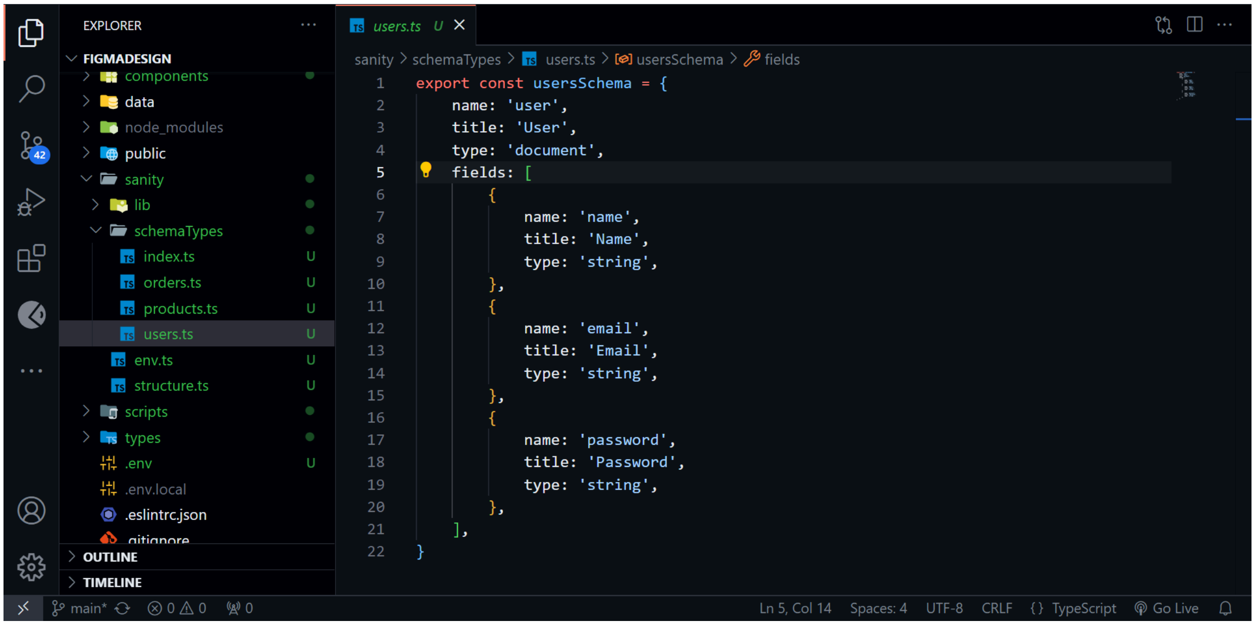
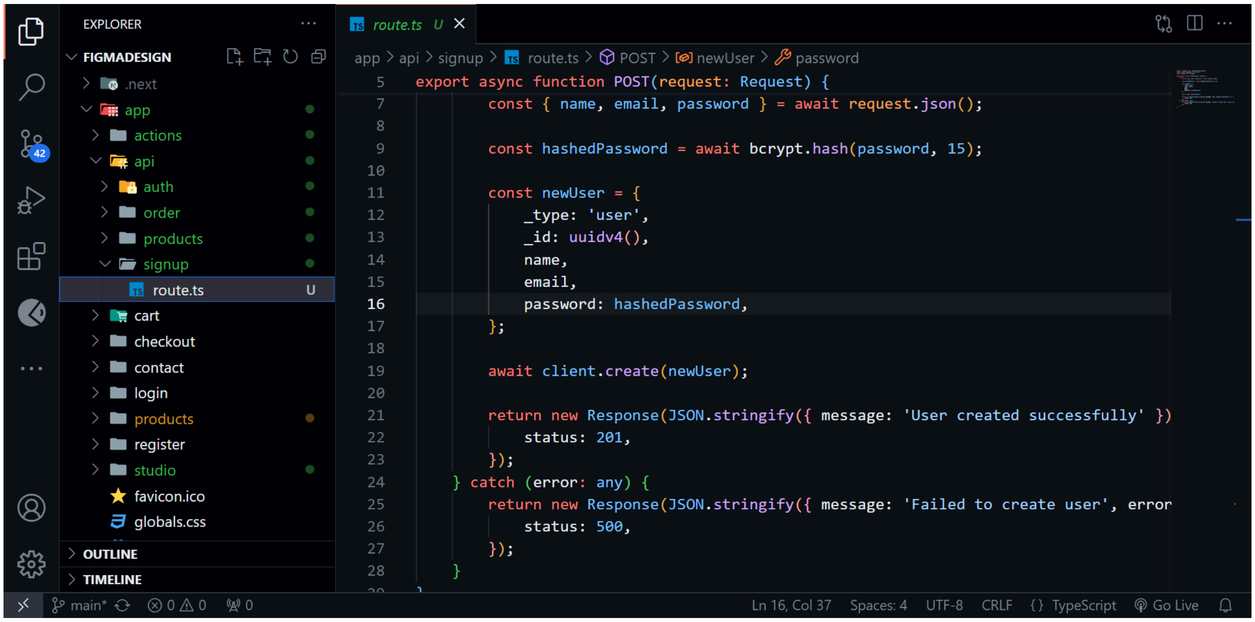
Building Dynamic Frontend Components

User Registration:

- I created an API endpoint called /signup to create a user to Sanity CMS document called User created through a given schema.



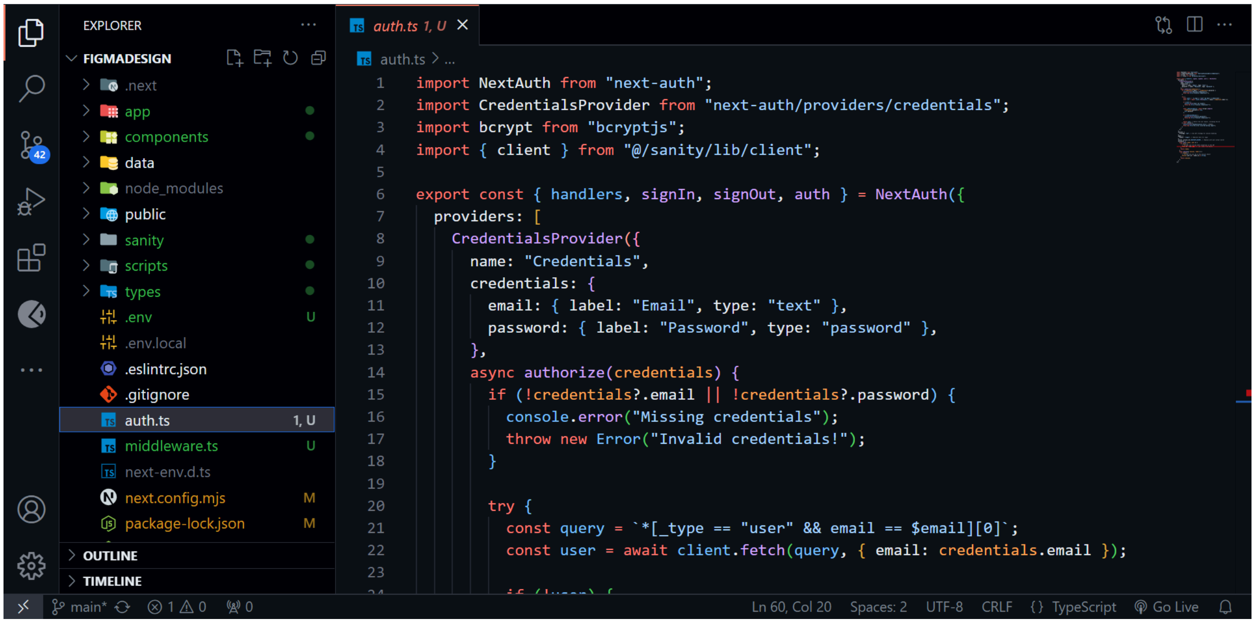
```
export const usersSchema = {
    name: 'user',
    title: 'User',
    type: 'document',
    fields: [
        {
            name: 'name',
            title: 'Name',
            type: 'string',
        },
        {
            name: 'email',
            title: 'Email',
            type: 'string',
        },
        {
            name: 'password',
            title: 'Password',
            type: 'string',
        },
    ],
}
```



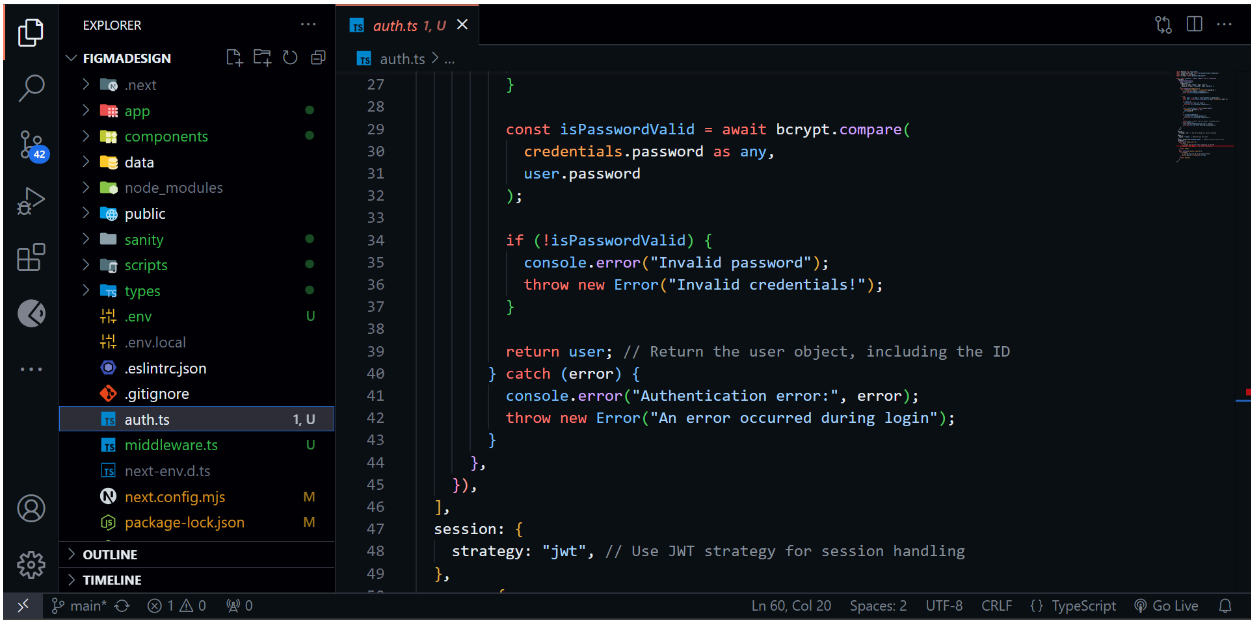
```
export async function POST(request: Request) {
        const { name, email, password } = await request.json();

        const hashedPassword = await bcrypt.hash(password, 15);

        const newUser = {
            _type: 'user',
            _id: uuidv4(),
            name,
            email,
            password: hashedPassword,
        };

        await client.create(newUser);

        return new Response(JSON.stringify({ message: 'User created successfully' })
            status: 201,
        });
    } catch (error: any) {
        return new Response(JSON.stringify({ message: 'Failed to create user', error
            status: 500,
        });
    }
```

User Authentication with Auth.js:

- I used Auth.js library for authentication, setup it from here https://authjs.dev/getting-started/installation?framework=next-js Below are the images showcasing how it been created and used.



```
import NextAuth from "next-auth";
import CredentialsProvider from "next-auth/providers/credentials";
import bcrypt from "bcryptjs";
import { client } from "@/sanity/lib/client";

export const { handlers, signIn, signOut, auth } = NextAuth({
  providers: [
    CredentialsProvider({
      name: "Credentials",
      credentials: {
        email: { label: "Email", type: "text" },
        password: { label: "Password", type: "password" },
      },
      async authorize(credentials) {
        if (!credentials?.email || !credentials?.password) {
          console.error("Missing credentials");
          throw new Error("Invalid credentials!");
        }

        try {
          const query = `*[_type == "user" && email == $email][0]`;
          const user = await client.fetch(query, { email: credentials.email });

```



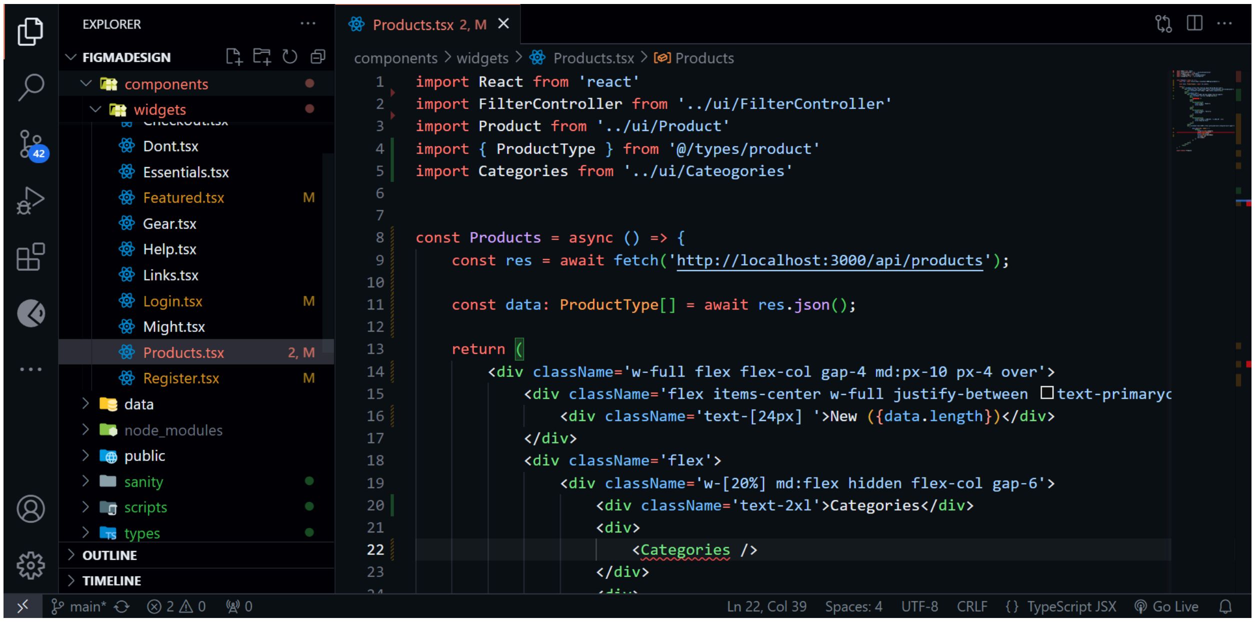
```
          }

          const isPasswordValid = await bcrypt.compare(
            credentials.password as any,
            user.password
          );

          if (!isPasswordValid) {
            console.error("Invalid password");
            throw new Error("Invalid credentials!");
          }

          return user; // Return the user object, including the ID
        } catch (error) {
          console.error("Authentication error:", error);
          throw new Error("An error occurred during login");
        }
      },
    }),
  ],
  session: {
    strategy: "jwt", // Use JWT strategy for session handling
  },
```

Dynamic Product Listings component:

- I fetched products saved in my Sanity CMS, through API endpoint /products where I fetched products using GROQ Query and then I fetched that API endpoint to the front-end to display products.

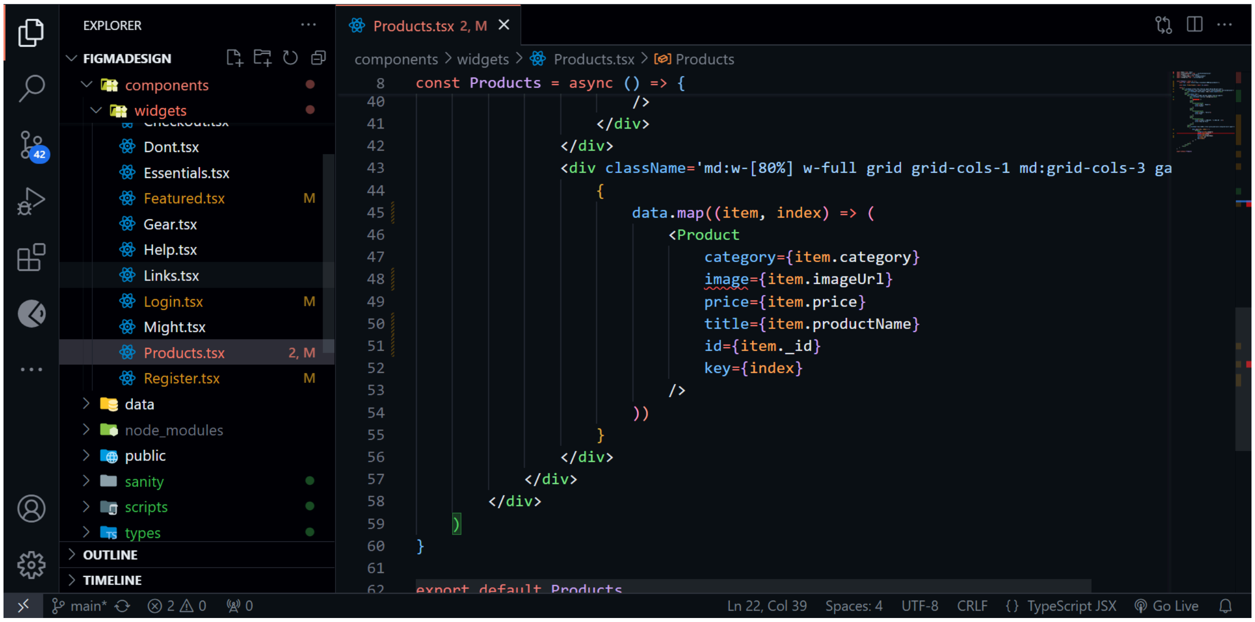
Screenshots of a API had mentioned in Day-3 Documentation:

Here they are fetched:

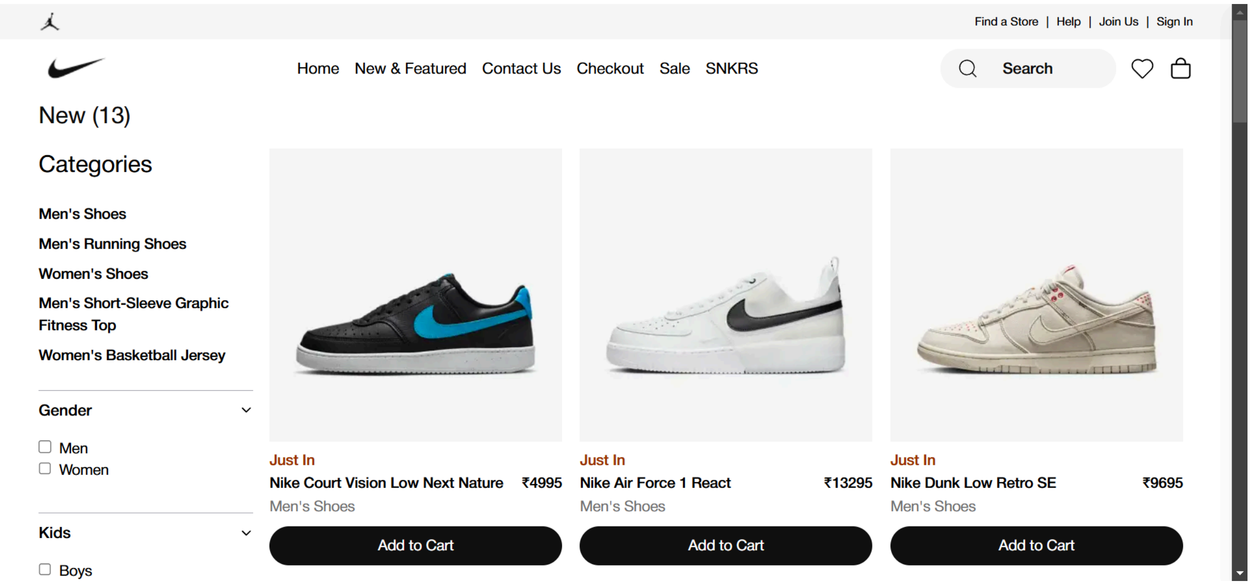


```
import React from 'react'
import FilterController from '../ui/FilterController'
import Product from '../ui/Product'
import { ProductType } from '@/types/product'
import Categories from '../ui/Cateogories'


const Products = async () => {
    const res = await fetch('http://localhost:3000/api/products');

    const data: ProductType[] = await res.json();

    return (
        <div className='w-full flex flex-col gap-4 md:px-10 px-4 over'>
            <div className='flex items-center w-full justify-between □text-primaryc
                <div className='text-[24px] '>New ({data.length})</div>
            </div>
            <div className='flex'>
                <div className='w-[20%] md:flex hidden flex-col gap-6'>
                    <div className='text-2xl'>Categories</div>
                    <div>
                        <Categories />
                    </div>
```

Here they are used:



```
const Products = async () => {
                        />
                    </div>
                </div>
                <div className='md:w-[80%] w-full grid grid-cols-1 md:grid-cols-3 ga
                    {
                        data.map((item, index) => (
                            <Product
                                category={item.category}
                                image={item.imageUrl}
                                price={item.price}
                                title={item.productName}
                                id={item._id}
                                key={index}
                            />
                        ))
                    }
                </div>
            </div>
        </div>
    )
}

export default Products
```



New (13)

Categories

Men's Shoes

Men's Running Shoes

Women's Shoes

Men's Short-Sleeve Graphic Fitness Top

Women's Basketball Jersey

Gender

□ Men

□ Women

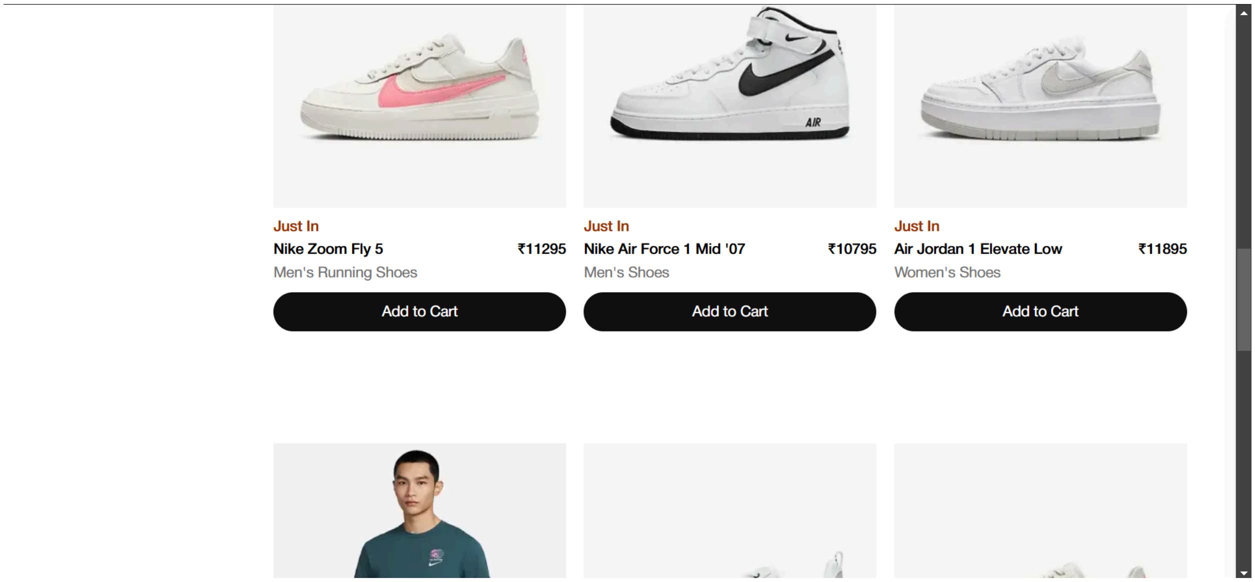
Kids

□ Boys

Just In
Nike Court Vision Low Next Nature ₹4995
Men's Shoes
Add to Cart

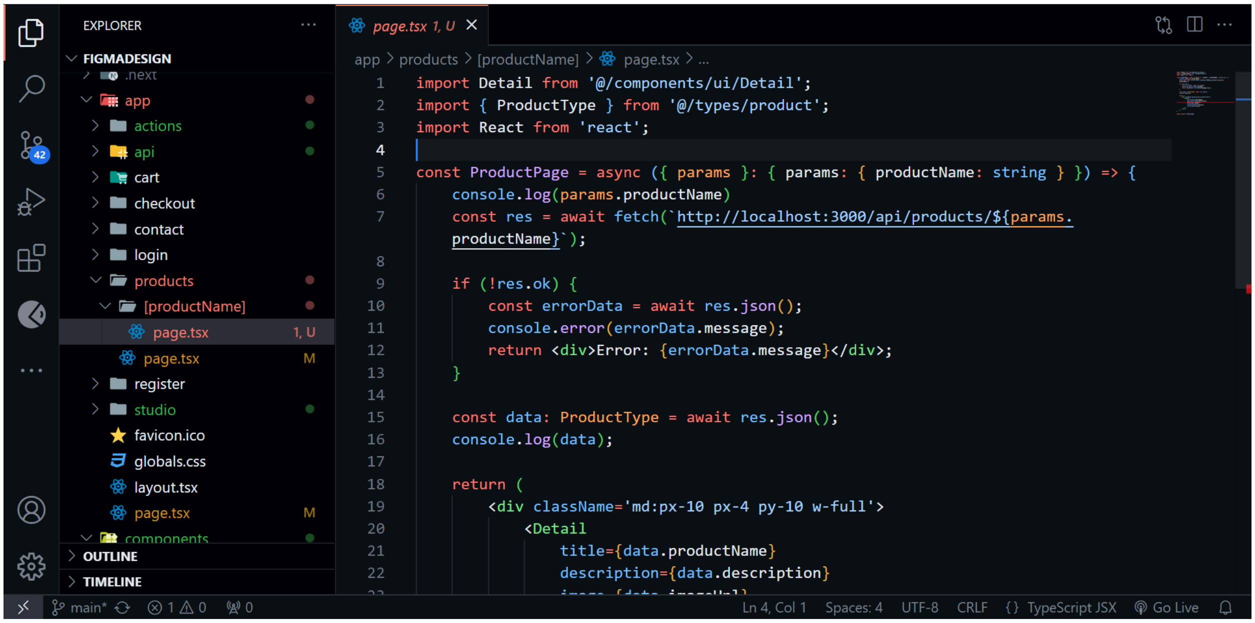
Just In
Nike Air Force 1 React ₹13295
Men's Shoes
Add to Cart

Just In
Nike Dunk Low Retro SE ₹9695
Men's Shoes
Add to Cart

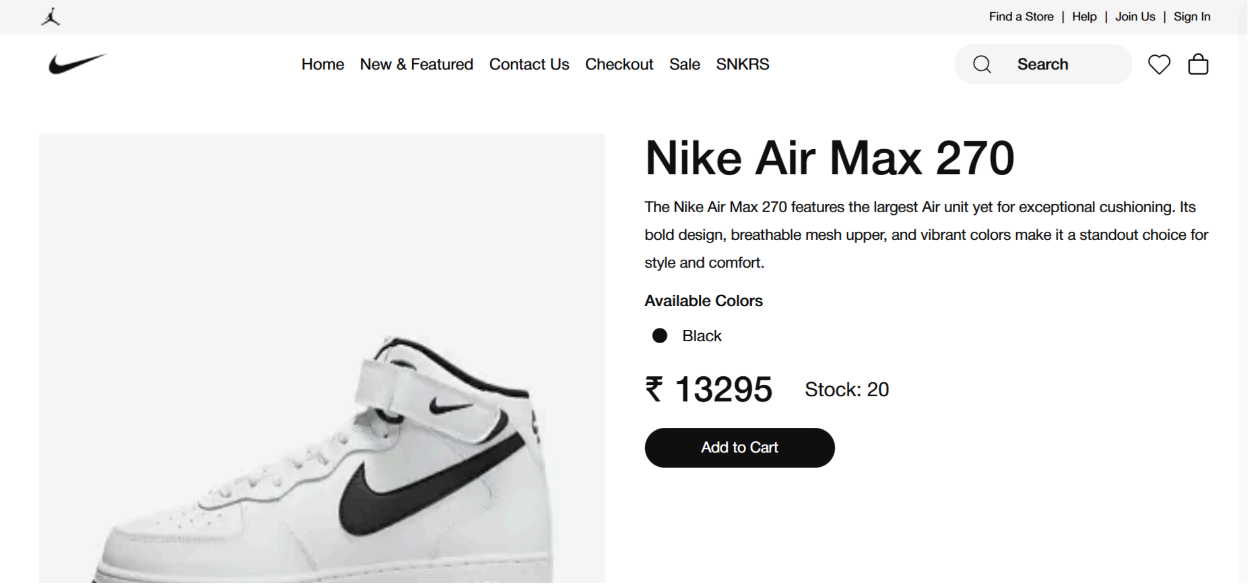


Dynamic Product Details Component:

- I also made product details page dynamic using dynamic API and GROQ Query and fetched them in the frontend component.



```
import Detail from '@/components/ui/Detail';
import { ProductType } from '@/types/product';
import React from 'react';

const ProductPage = async ({ params }: { params: { productName: string } }) => {
    console.log(params.productName)
    const res = await fetch(`http://localhost:3000/api/products/${params.productName}`);

    if (!res.ok) {
        const errorData = await res.json();
        console.error(errorData.message);
        return <div>Error: {errorData.message}</div>;
    }

    const data: ProductType = await res.json();
    console.log(data);

    return (
        <div className='md:px-10 px-4 py-10 w-full'>
            <Detail
                title={data.productName}
                description={data.description}
                image={data.imageUrl}
```



Nike Air Max 270

The Nike Air Max 270 features the largest Air unit yet for exceptional cushioning. Its bold design, breathable mesh upper, and vibrant colors make it a standout choice for style and comfort.

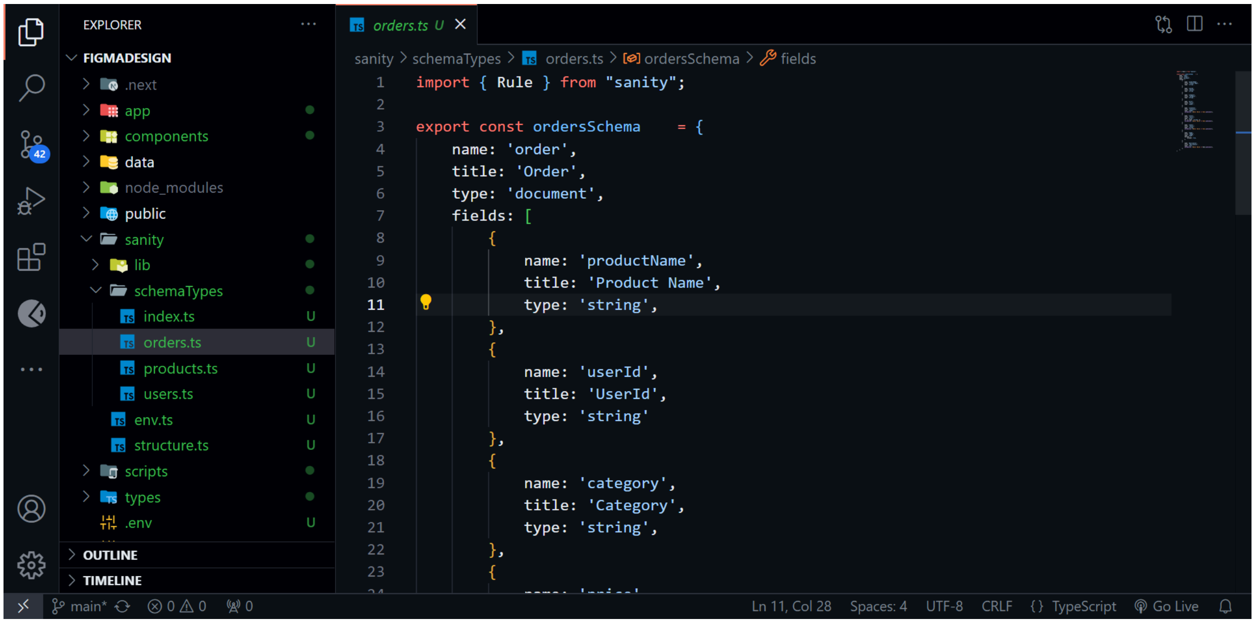
Available Colors

● Black

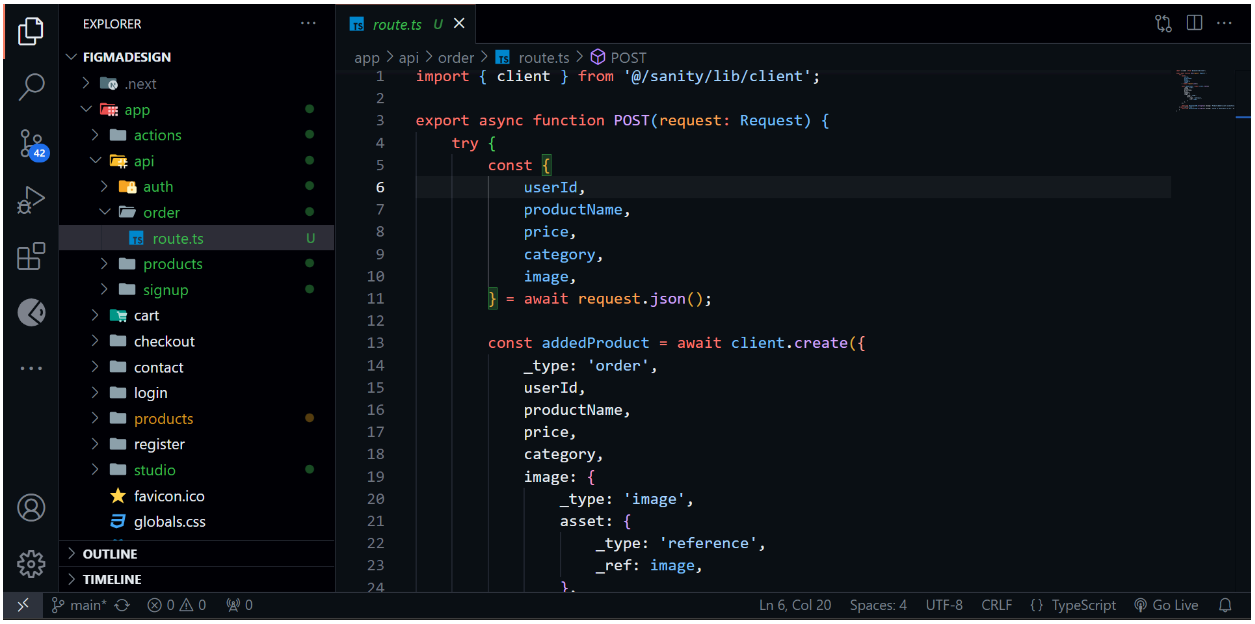
₹ 13295 Stock: 20

Dynamic Cart Items:

- I have made an API to put data into Sanity CMS to Order Document that I created with this provided Schema.
- Then I fetched the added cart items to cart on frontend.

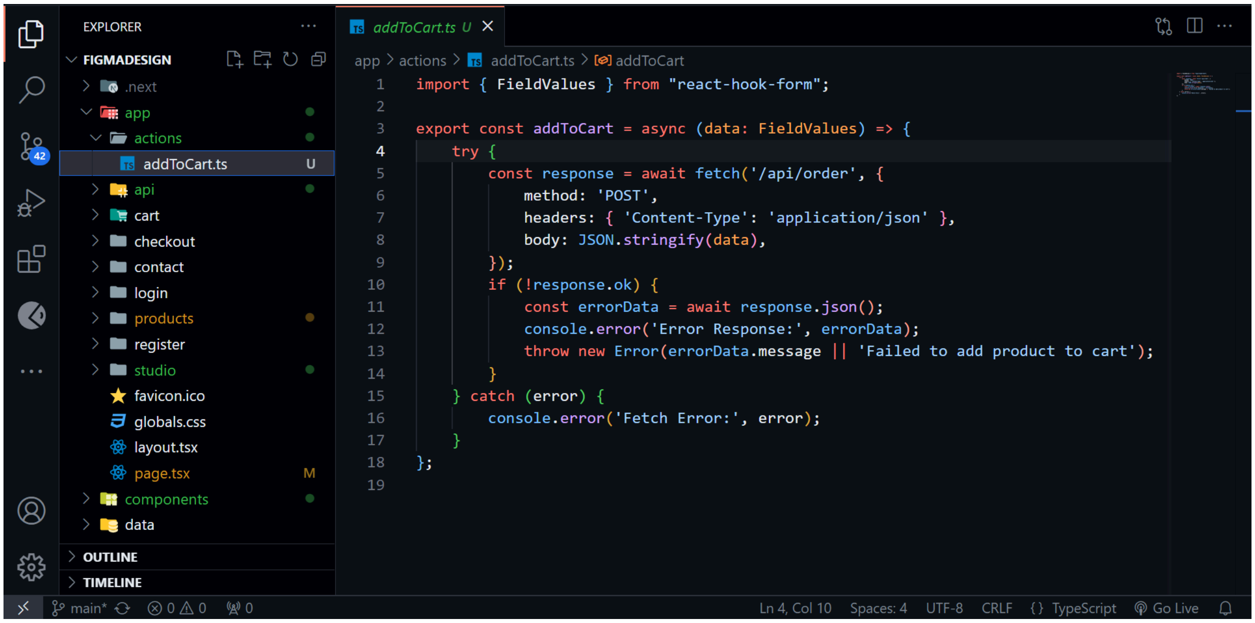


```
import { Rule } from "sanity";

export const ordersSchema    = {
    name: 'order',
    title: 'Order',
    type: 'document',
    fields: [
        {
            name: 'productName',
            title: 'Product Name',
            type: 'string',
        },
        {
            name: 'userId',
            title: 'UserId',
            type: 'string'
        },
        {
            name: 'category',
            title: 'Category',
            type: 'string',
        },
        {
            name: 'price',
```



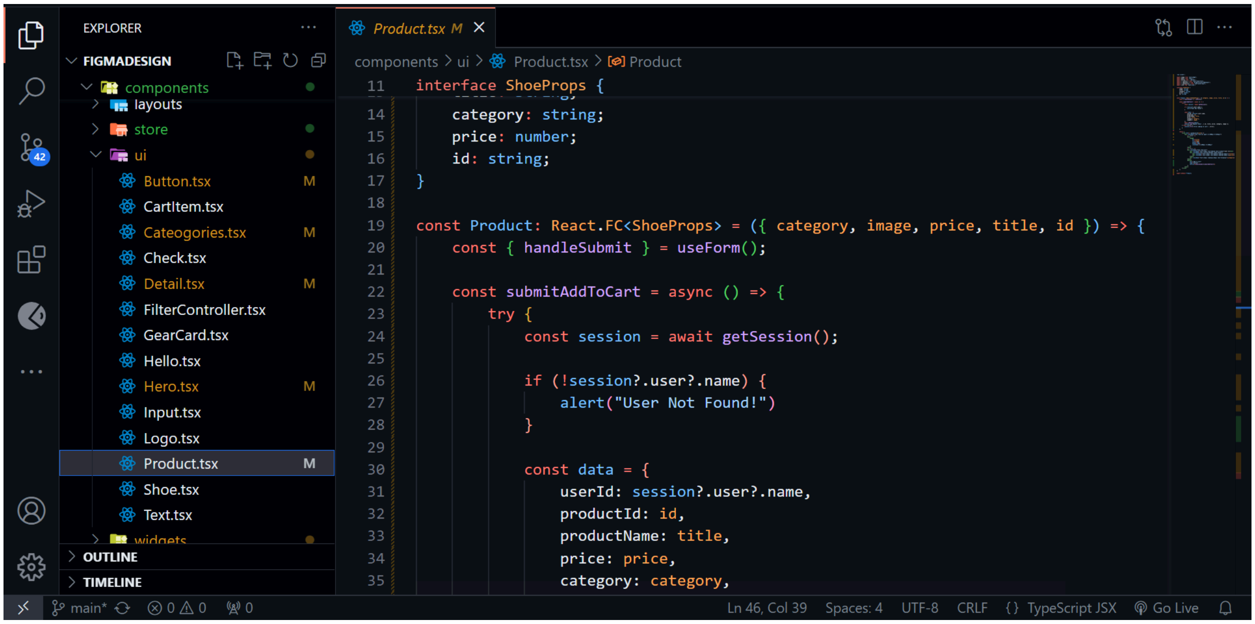
```
import { client } from '@/sanity/lib/client';

export async function POST(request: Request) {
    try {
        const {
            userId,
            productName,
            price,
            category,
            image,
        } = await request.json();

        const addedProduct = await client.create({
            _type: 'order',
            userId,
            productName,
            price,
            category,
            image: {
                _type: 'image',
                asset: {
                    _type: 'reference',
                    _ref: image,
                },
```

Server Action:

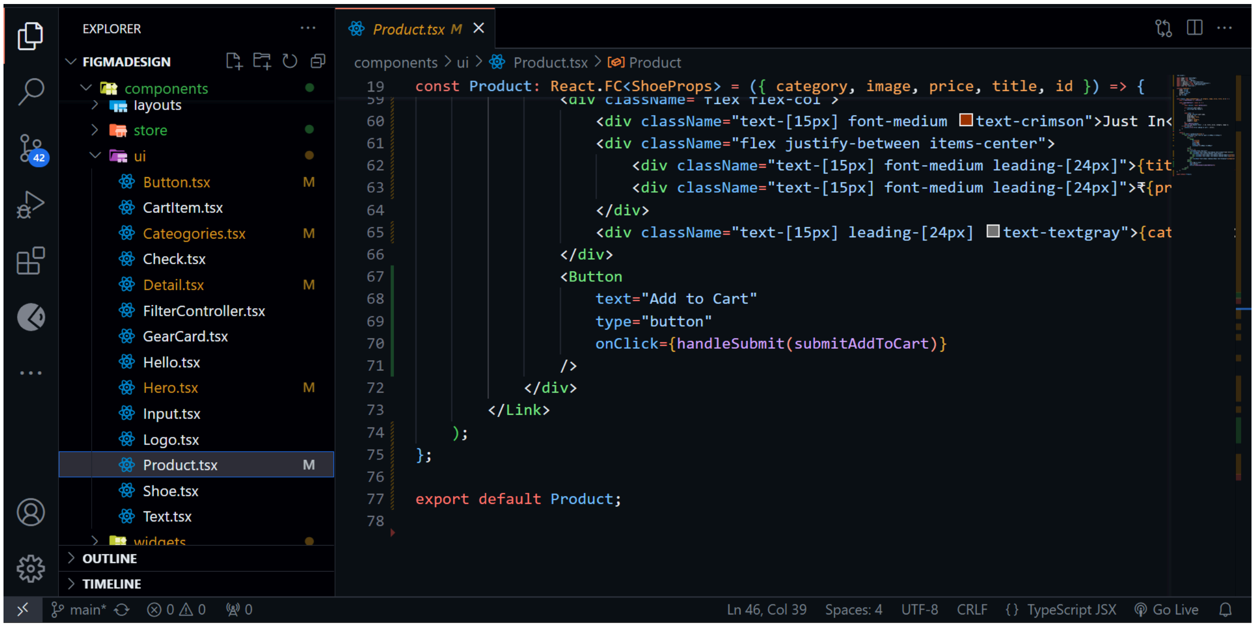


```
import { FieldValues } from "react-hook-form";

export const addToCart = async (data: FieldValues) => {
    try {
        const response = await fetch('/api/order', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(data),
        });
        if (!response.ok) {
            const errorData = await response.json();
            console.error('Error Response:', errorData);
            throw new Error(errorData.message || 'Failed to add product to cart');
        }
    } catch (error) {
        console.error('Fetch Error:', error);
    }
};
```

Fetching on front-end:



```
interface ShoeProps {
    category: string;
    price: number;
    id: string;
}

const Product: React.FC<ShoeProps> = ({ category, image, price, title, id }) => {
    const { handleSubmit } = useForm();

    const submitAddToCart = async () => {
        try {
            const session = await getSession();

            if (!session?.user?.name) {
                alert("User Not Found!")
            }

            const data = {
                userId: session?.user?.name,
                productId: id,
                productName: title,
                price: price,
                category: category,
```



```
const Product: React.FC<ShoeProps> = ({ category, image, price, title, id }) => {
                <div className="flex flex-col">
                    <div className="text-[15px] font-medium text-crimson">Just In<
                    <div className="flex justify-between items-center">
                        <div className="text-[15px] font-medium leading-[24px]">{tit
                        <div className="text-[15px] font-medium leading-[24px]">₹{pr
                    </div>
                    <div className="text-[15px] leading-[24px] text-textgray">{cat
                </div>
                <Button
                    text="Add to Cart"
                    type="button"
                    onClick={handleSubmit(submitAddToCart)}
                />
            </div>
        </Link>
    );
};

export default Product;
```