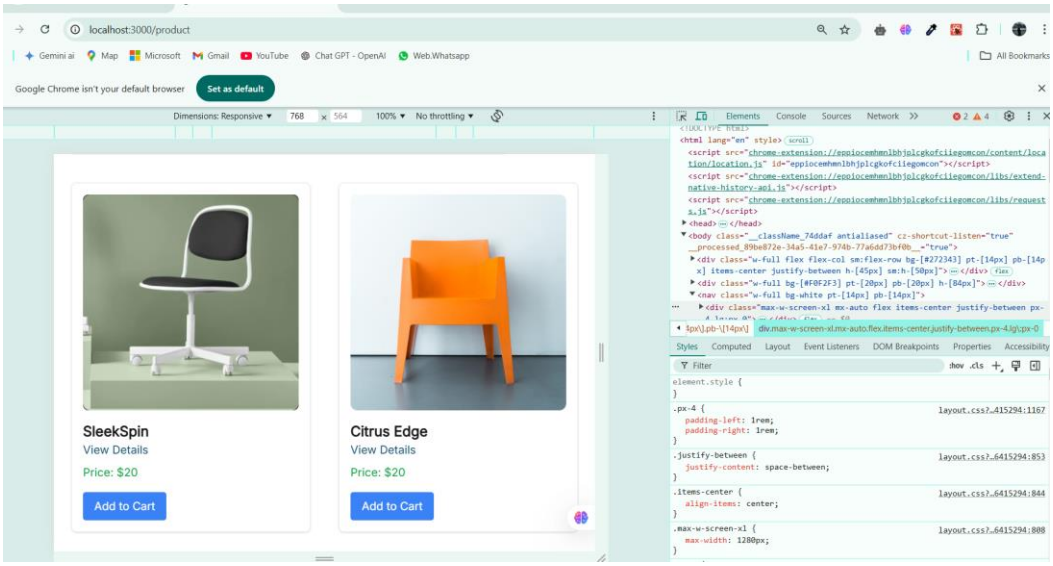
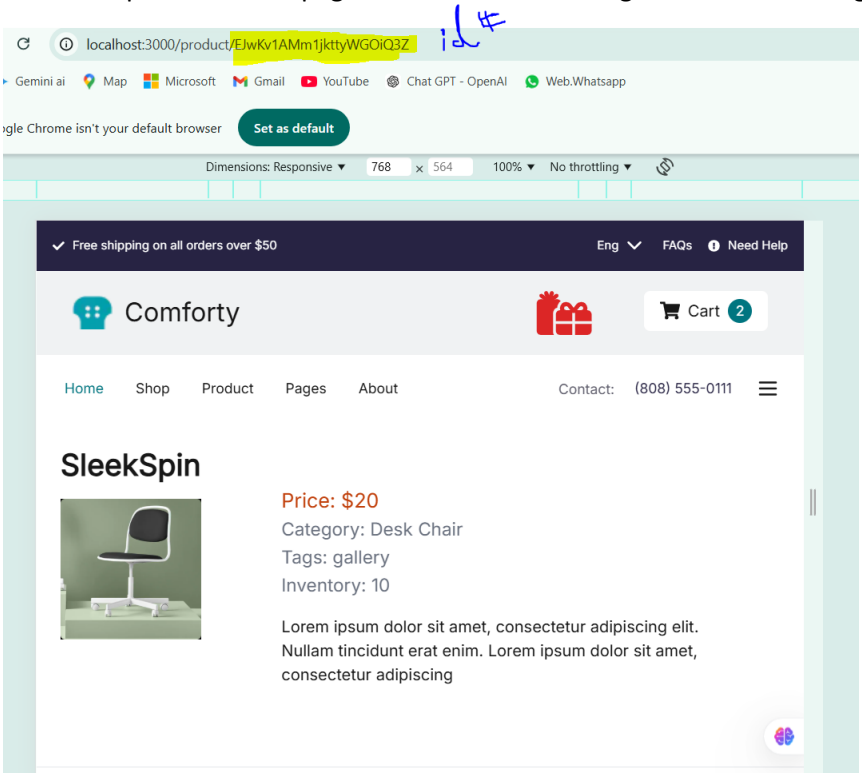
Day4

1)Functional Deliverables:

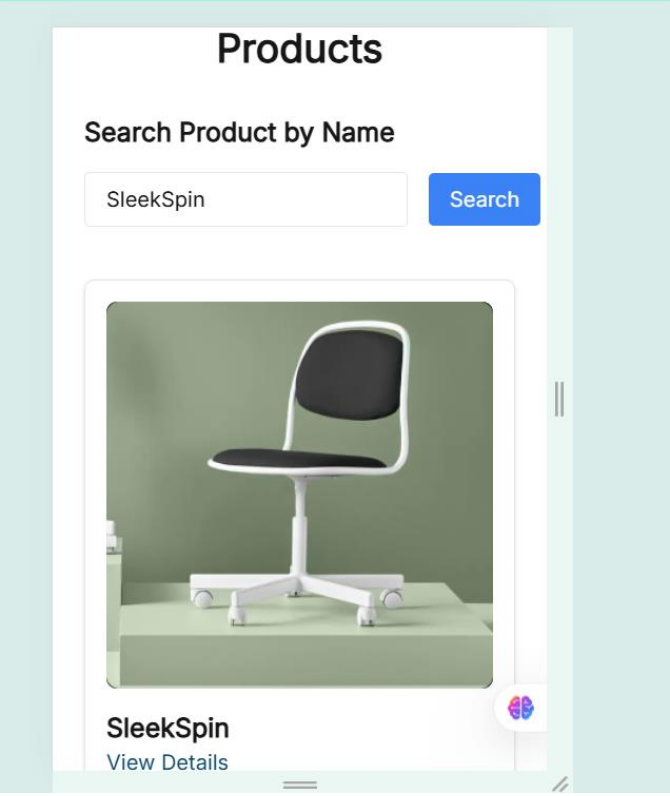
- The product listing page with dynamic data.

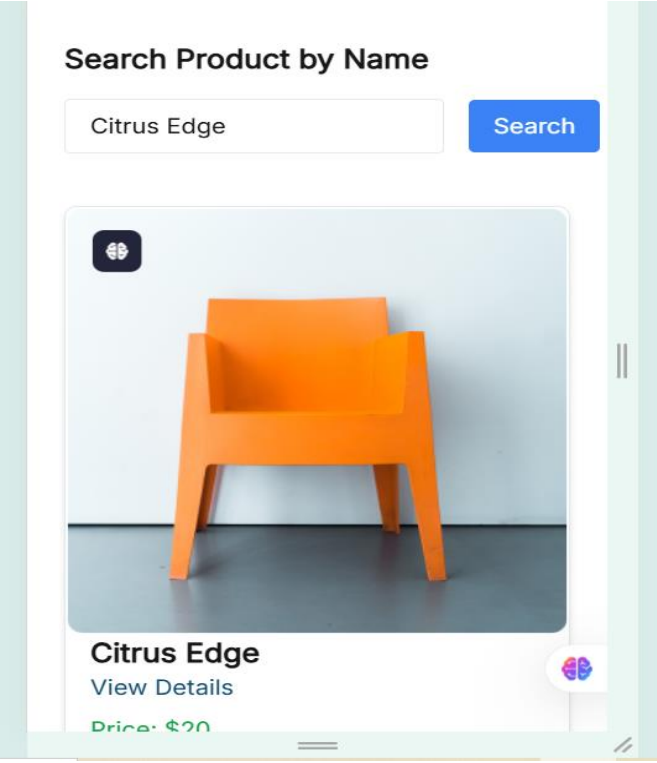


- Individual product detail pages with accurate routing and data rendering



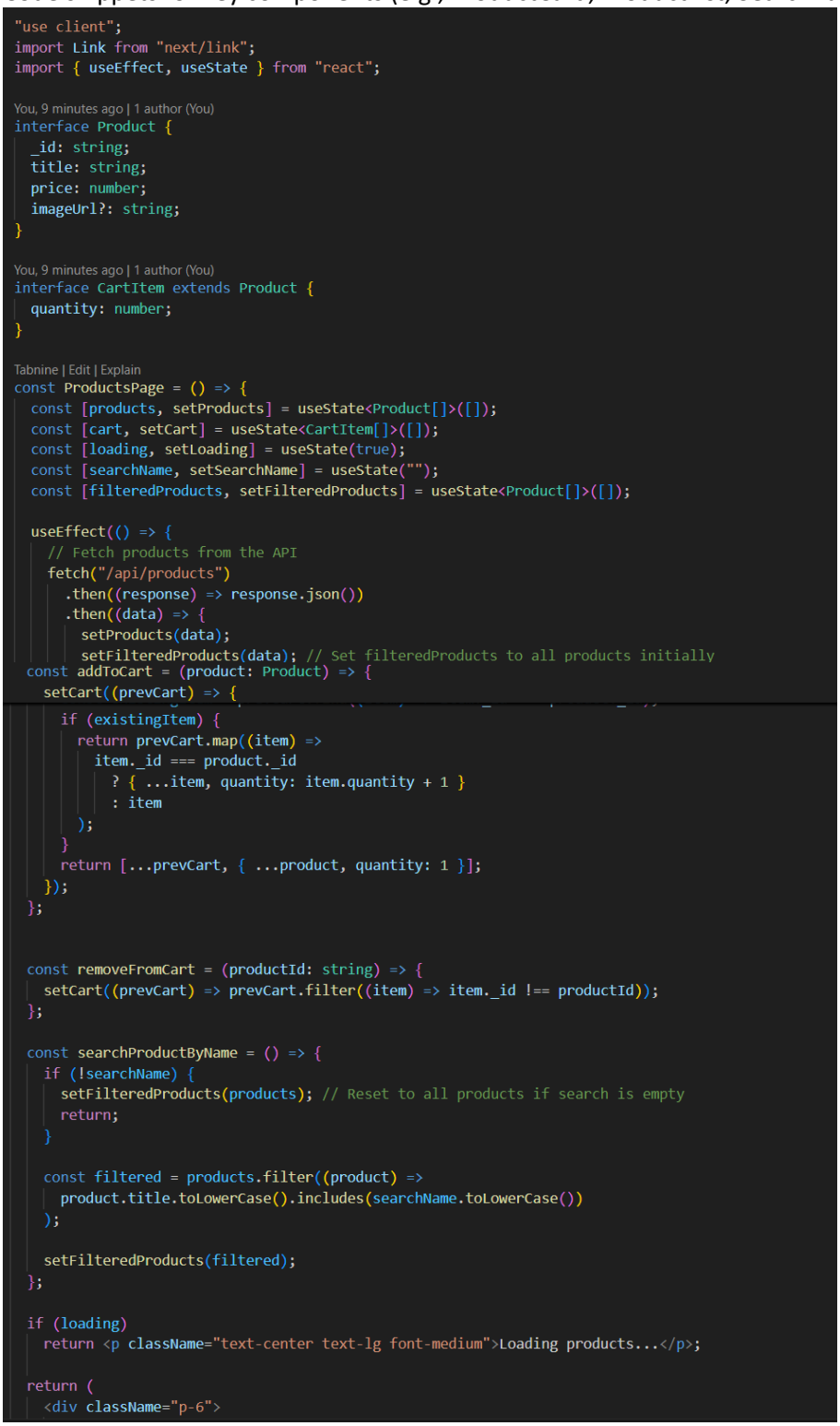
- Working category filters, search bar, and pagination.





2) Code Deliverables:

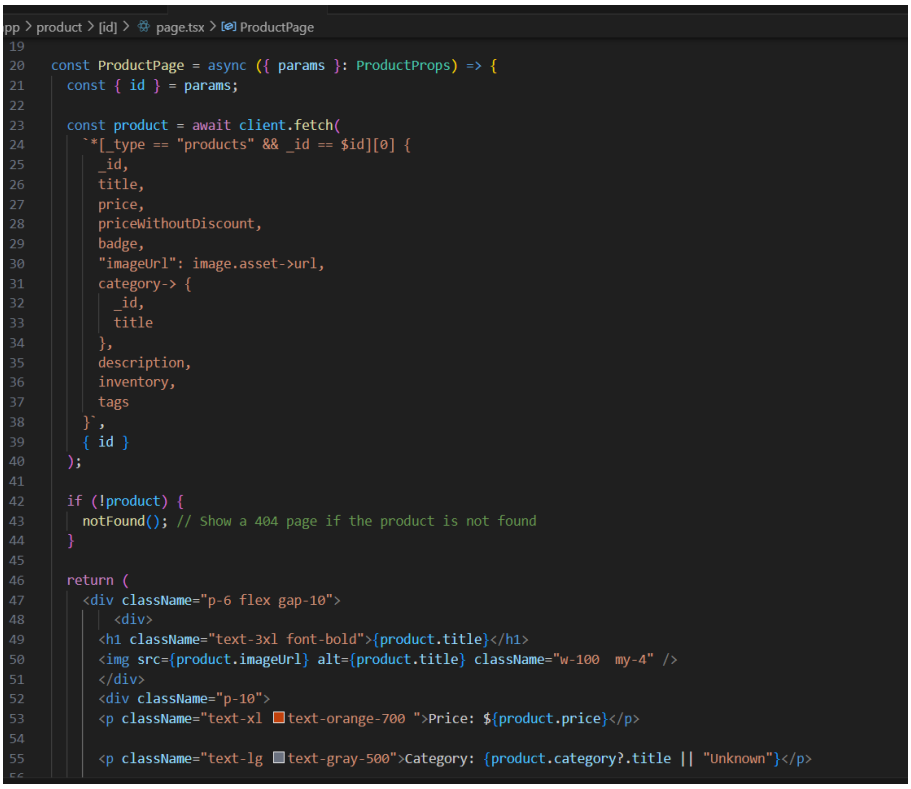
- Code snippets for key components (e.g., ProductCard, ProductList, SearchBar).



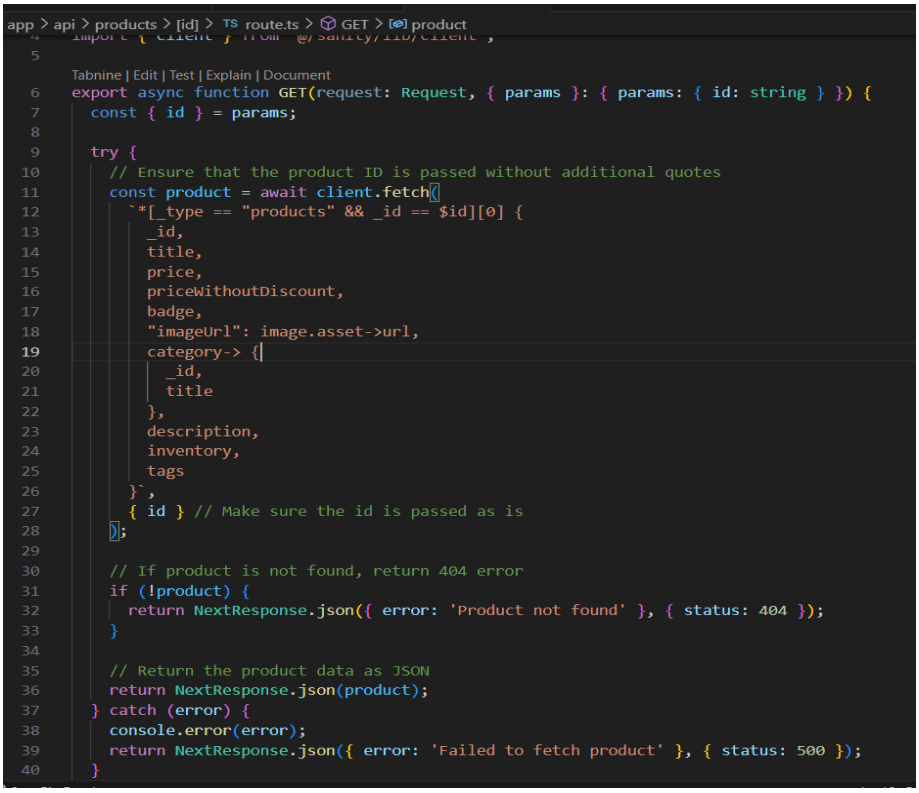
```tsx
"use client";
import Link from "next/link";
import { useEffect, useState } from "react";

You, 9 minutes ago | 1 author (You)
interface Product {
  _id: string;
  title: string;
  price: number;
  imageUrl?: string;
}

You, 9 minutes ago | 1 author (You)
interface CartItem extends Product {
  quantity: number;
}

Tabnine | Edit | Explain
const ProductsPage = () => {
  const [products, setProducts] = useState<Product[]>([]);
  const [cart, setCart] = useState<CartItem[]>([]);
  const [loading, setLoading] = useState(true);
  const [searchName, setSearchName] = useState("");
  const [filteredProducts, setFilteredProducts] = useState<Product[]>([]);

  useEffect(() => {
    // Fetch products from the API
    fetch("/api/products")
      .then((response) => response.json())
      .then((data) => {
        setProducts(data);
        setFilteredProducts(data); // Set filteredProducts to all products initially
const addToCart = (product: Product) => {
  setCart((prevCart) => {
    if (existingItem) {
      return prevCart.map((item) =>
        item._id === product._id
          ? { ...item, quantity: item.quantity + 1 }
          : item
      );
    }
    return [...prevCart, { ...product, quantity: 1 }];
  });
};


const removeFromCart = (productId: string) => {
  setCart((prevCart) => prevCart.filter((item) => item._id !== productId));
};

const searchProductByName = () => {
  if (!searchName) {
    setFilteredProducts(products); // Reset to all products if search is empty
    return;
  }

  const filtered = products.filter((product) =>
    product.title.toLowerCase().includes(searchName.toLowerCase())
  );

  setFilteredProducts(filtered);
};

if (loading)
  return <p className="text-center text-lg font-medium">Loading products...</p>;

return (
  <div className="p-6">
```

```jsx
{/* Search Product by Name */}
<div className="mb-10">
  <h2 className="text-xl font-semibold mb-4">Search Product by Name</h2>
  <div className="flex items-center gap-4">
    <input
      type="text"
      value={searchName}
      onChange={(e) => setSearchName(e.target.value)}
      placeholder="Enter Product Name"
      className="border rounded px-4 py-2 flex-1"
    />
    <button
      onClick={searchProductByName}
      className="bg-blue-500 text-white py-2 px-4 rounded hover:bg-blue-600"
    >
      Search
    </button>
  </div>
</div>

{/* Display Filtered Products */}
{/* <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-10">…
</div> */}
<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-10">
  {filteredProducts.map((product) => (
    <div
      key={product._id}
      className="border rounded-lg shadow p-4 hover:shadow-md transition-shadow"
    >
      {product.imageUrl && (
        <img
          src={product.imageUrl}
          alt={product.title}
          className="w-full object-cover rounded-lg transition-transform duration-300 hover:scale-110"
        />
      )}
      <div className="mt-4">
        <h3 className="text-xl font-semibold">{product.title}</h3>

        <Link href=  {`/product/${product._id}`} passHref className="text-sky-900 hover:text-amber-400
          View Details
          </Link>
        <p className="text-green-600 font-medium mt-2">
          Price: ${product.price}
        </p>

        <button
          onClick={() => addToCart(product)}
          className="mt-4 bg-blue-500 text-white py-2 px-4 rounded hover:bg-blue-600"
        >
          Add to Cart
        </button>
      </div>
    </div>
  ))}
</div>

{/* Cart Section */}
<div className="mt-10 border-2 border-red-600 pt-6">
  <h2 className="text-2xl font-bold mb-4">Cart</h2>
  {cart.length === 0 ? (
    <p className="text-gray-600">Your cart is empty.</p>
  ) : (
    <div className="space-y-4">
      {cart.map((item) => (
        <div
          key={item._id}
          className="flex items-center gap-4 border-b pb-4"
        >
```

- Scripts or logic for API integration and dynamic routing.



```
pp > product > [id] > page.tsx > ProductPage

const ProductPage = async ({ params }: ProductProps) => {
  const { id } = params;

  const product = await client.fetch(
    `*[_type == "products" && _id == $id][0] {
      _id,
      title,
      price,
      priceWithoutDiscount,
      badge,
      "imageUrl": image.asset->url,
      category-> {
        _id,
        title
      },
      description,
      inventory,
      tags
    }`,
    { id }
  );

  if (!product) {
    notFound(); // Show a 404 page if the product is not found
  }

  return (
    <div className="p-6 flex gap-10">
        <div>
      <h1 className="text-3xl font-bold">{product.title}</h1>
      <img src={product.imageUrl} alt={product.title} className="w-100  my-4" />
      </div>
      <div className="p-10">
      <p className="text-xl text-orange-700 ">Price: ${product.price}</p>

      <p className="text-lg text-gray-500">Category: {product.category?.title || "Unknown"}</p>
```



```
app > api > products > [id] > route.ts > GET > product

Tabnine | Edit | Test | Explain | Document
export async function GET(request: Request, { params }: { params: { id: string } }) {
  const { id } = params;

  try {
    // Ensure that the product ID is passed without additional quotes
    const product = await client.fetch(
      `*[_type == "products" && _id == $id][0] {
        _id,
        title,
        price,
        priceWithoutDiscount,
        badge,
        "imageUrl": image.asset->url,
        category-> {
          _id,
          title
        },
        description,
        inventory,
        tags
      }`,
      { id } // Make sure the id is passed as is
    );

    // If product is not found, return 404 error
    if (!product) {
      return NextResponse.json({ error: 'Product not found' }, { status: 404 });
    }

    // Return the product data as JSON
    return NextResponse.json(product);
  } catch (error) {
    console.error(error);
    return NextResponse.json({ error: 'Failed to fetch product' }, { status: 500 });
  }
```

3) Documentation:

Technical Report Notes

1. Overview

- Project focused on [project objective, e.g., "creating a responsive portfolio website"].
- Built using [technologies/tools, e.g., "React, Tailwind CSS, and Sanity CMS"].
- Goal: [state the primary goal, e.g., "to create a professional, responsive, and dynamic user interface"].

---

2. Steps Taken to Build and Integrate Components

- **Planning Phase**:
  - Defined project requirements.
  - Created mockups in Figma.
- **Development Process**:
  - Set up the development environment using [e.g., "Node.js and Visual Studio Code"].
  - Built individual components (e.g., Navbar, Footer, Resume Section).
  - Integrated Sanity CMS for content management.
- **Admin Dashboard Component**:
  - Built an interface for managing products, orders, users, and categories.
  - Included tools for analytics and business insights.
- **Testing and Deployment**:
  - Tested responsiveness and functionality on multiple devices.
  - Deployed the project on [e.g., "Vercel"].

---

3. Challenges Faced and Solutions Implemented

- **Challenge 1**: Integration of CMS with front-end.
  Solution: Referred to Sanity documentation and implemented `client.fetch` queries for data retrieval.
- **Challenge 2**: Managing state across components.
  Solution: Used React Context API to simplify state management.
- **Challenge 3**: Browser compatibility issues.
  Solution: Tested with multiple browsers and resolved styling inconsistencies.

---

4. Best Practices Followed

- Clean code structure with consistent naming conventions.
- Used Git for version control and created feature branches for development.
- Followed the DRY (Don’t Repeat Yourself) principle to avoid redundant code.
- Used semantic HTML for better accessibility.

---

5. Conclusion and Future Recommendations

- Successfully delivered a responsive, dynamic, and user-friendly interface.
- Future Scope:
    - Add animations for enhanced user experience.
    - Optimize performance for larger datasets.

---

6. References

- Official Sanity CMS documentation.
- Tutorials from [source, e.g., "YouTube, MDN Web Docs"].
- GitHub repositories for similar projects.

Made by chandafatima$mylife