Name:Aqsa Arshad Rasheed

Day 4 - Building Dynamic Frontend Components for Your Marketplace

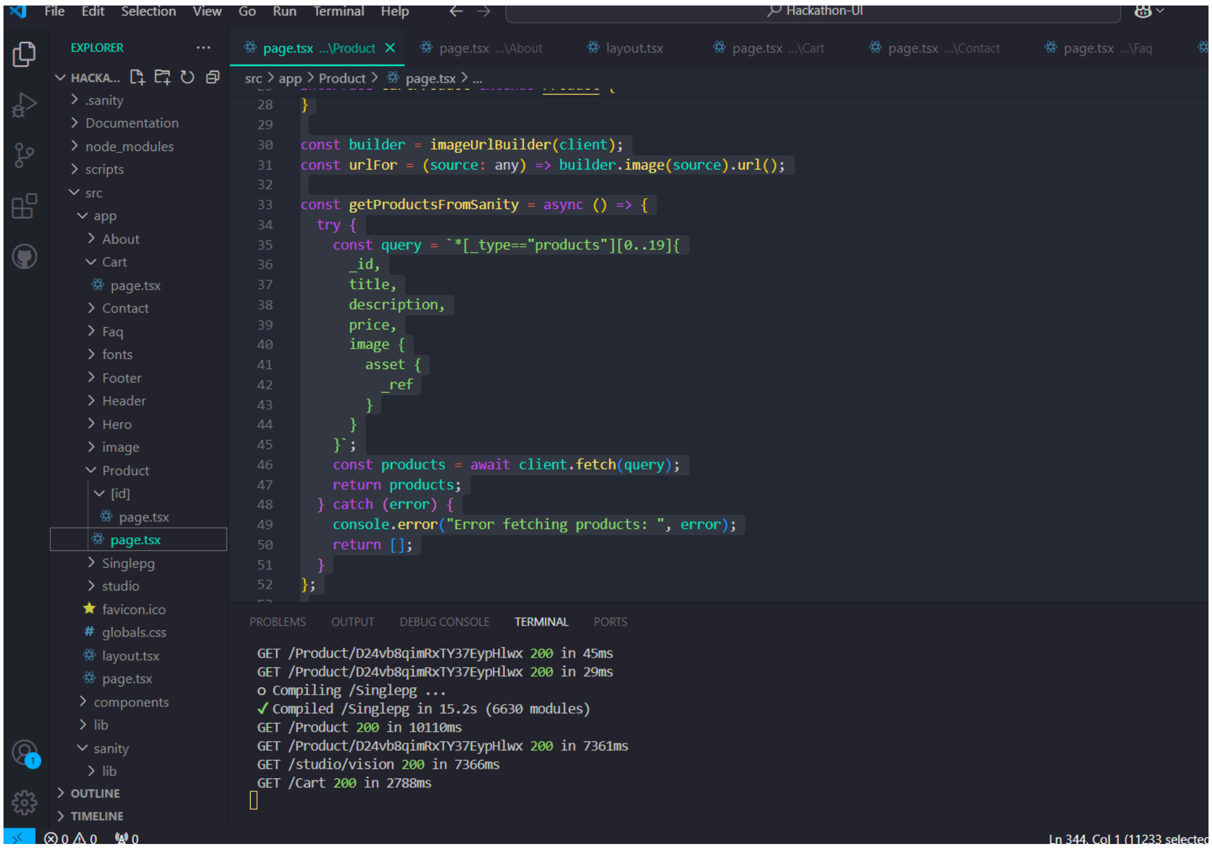
---

Introduction

In this document, we will explore the process of building dynamic frontend components for a marketplace application. The focus will be on fetching product data from **Sanity**, creating a product listing page, implementing dynamic routing for product details, and adding a search bar for filtering products. This guide will provide a theoretical understanding of each step involved in the development process.

---

1. Fetching Products from Sanity



```tsx
}

const builder = imageUrlBuilder(client);
const urlFor = (source: any) => builder.image(source).url();

const getProductsFromSanity = async () => {
  try {
    const query = `*[_type=="products"][0..19]{
      _id,
      title,
      description,
      price,
      image {
        asset {
          _ref
        }
      }
    }`;
    const products = await client.fetch(query);
    return products;
  } catch (error) {
    console.error("Error fetching products: ", error);
    return [];
  }
};
```

```
GET /Product/D24vb8qimRxTY37EypHlwx 200 in 45ms
GET /Product/D24vb8qimRxTY37EypHlwx 200 in 29ms
o Compiling /Singlepg ...
✓ Compiled /Singlepg in 15.2s (6630 modules)
GET /Product 200 in 10110ms
GET /Product/D24vb8qimRxTY37EypHlwx 200 in 7361ms
GET /studio/vision 200 in 7366ms
GET /Cart 200 in 2788ms
```

Overview

Sanity is a headless CMS that allows developers to manage content in a structured way. To display products in our marketplace, we need to fetch product data from Sanity's dataset. This involves writing a query to retrieve the necessary fields for each product.

Key Concepts

- **Sanity Client**: A JavaScript client that allows us to interact with the Sanity API.
- **GROQ**: Sanity's query language, which is used to fetch data from the dataset.

Steps Involved

1. **Set Up Sanity Client**:
   - Connect to the Sanity project using the Sanity client.
2. **Write a GROQ Query**:
   - Define the query to retrieve the necessary product information, such as the product name, description, price, and image.
3. **Fetch Data**:
   - Execute the query to retrieve product data from Sanity and make it available for use in the frontend.

Expected Outcome

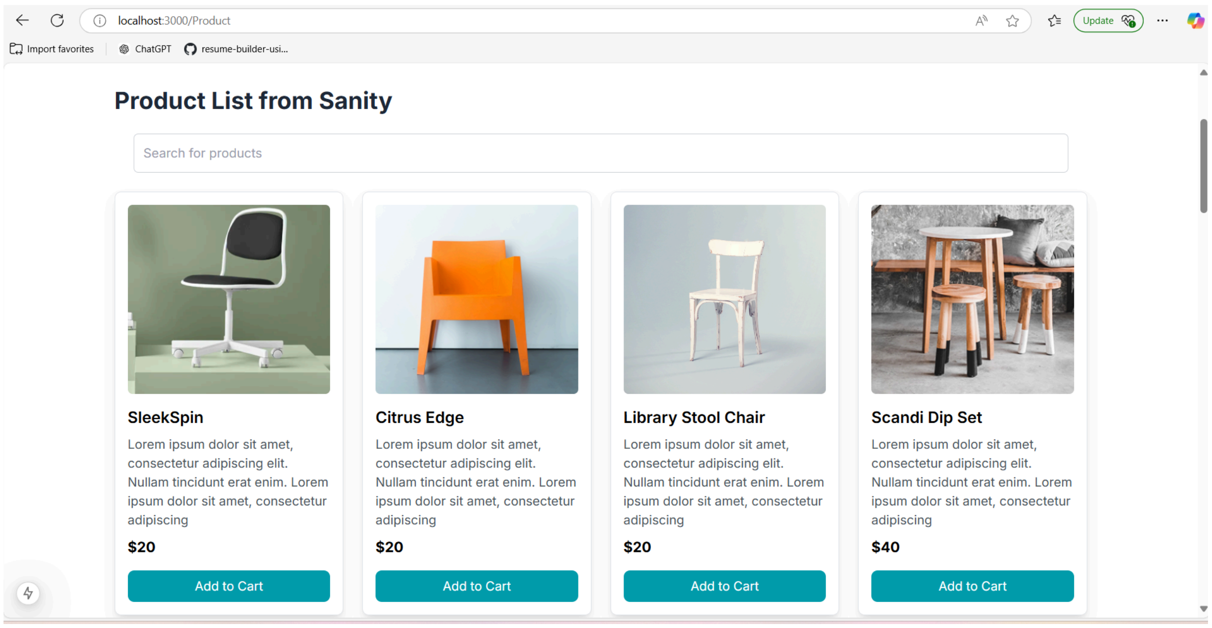
By the end of this step, you will be able to fetch product data from Sanity, which will be used to populate your marketplace.

---

2. Creating the Product Listing Page

Overview

The product listing page is the main interface where users can view all available products. This page will display a grid of product cards, each containing an image, name, description, and price. Each product card will link to its respective detail page.



Product List from Sanity

SleekSpin
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam tincidunt erat enim. Lorem ipsum dolor sit amet, consectetur adipiscing
$20

Citrus Edge
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam tincidunt erat enim. Lorem ipsum dolor sit amet, consectetur adipiscing
$20

Library Stool Chair
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam tincidunt erat enim. Lorem ipsum dolor sit amet, consectetur adipiscing
$20

Scandi Dip Set
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam tincidunt erat enim. Lorem ipsum dolor sit amet, consectetur adipiscing
$40

Key Concepts

- **React Components**: Building reusable UI components to display product information.
- **State Management**: Using React's state to manage the list of products fetched from Sanity.

Steps Involved

1. **Create a Product List Component**:
   - Define a component that fetches and displays the list of products.
2. **Map Through Products**:
   - Iterate over the fetched products and display them in a grid of cards.
3. **Link to Product Details**:
   - Set up routing to link each product card to its corresponding product detail page.

Expected Outcome

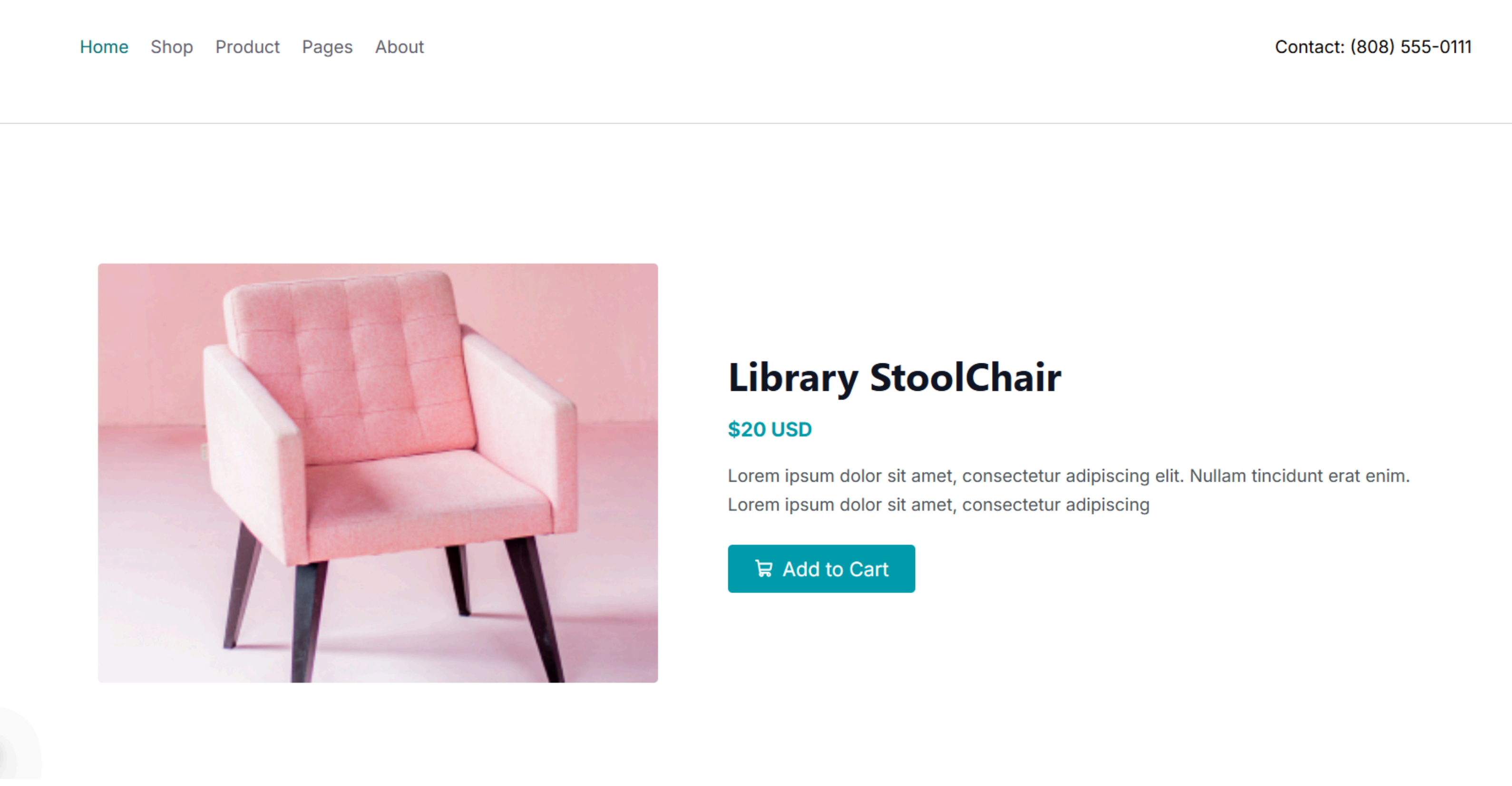
By the end of this step, you will have a functional product listing page displaying all products fetched from Sanity, each linking to its detail page.

---

3. Implementing Dynamic Routing for Product Details

Overview

Dynamic routing allows us to create pages that can display different content based on the URL. In our marketplace, each product will have its own detail page that shows more information about the product when clicked.



Contact: (808) 555-0111

Library StoolChair

$20 USD

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam tincidunt erat enim. Lorem ipsum dolor sit amet, consectetur adipiscing

Key Concepts

- **Dynamic Routes**: Create pages based on URL parameters (e.g., product ID).
- **Fetching Product Details**: Similar to fetching the product list, fetch details for a specific product based on its ID.

Steps Involved

1. **Create a Dynamic Route**:
   - Set up a route that captures the product ID from the URL.
2. **Fetch Product Data**:
   - Use the captured product ID to fetch the specific product's details.
3. **Display Product Information**:
   - Render the product's name, description, price, and image on the detail page.

Expected Outcome

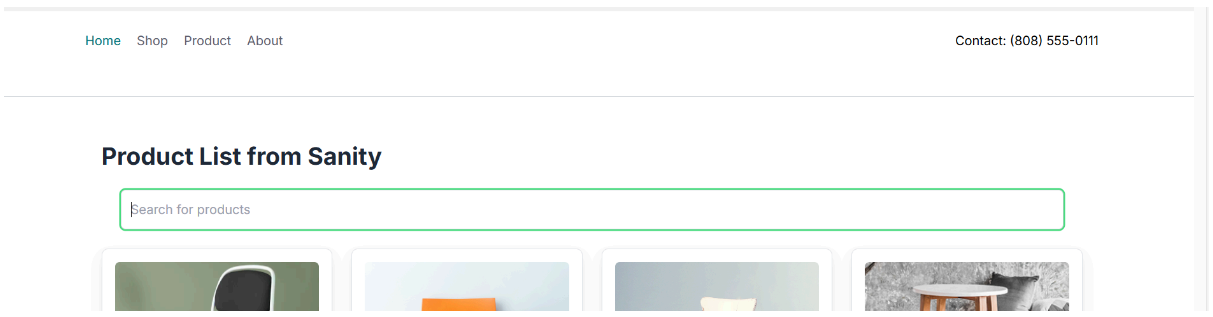
By the end of this step, each product will have a unique detail page, and users will be able to view more detailed information about products through dynamic routing.

---

4. Adding a Search Bar

Overview

A search bar enhances user experience by allowing users to filter products based on their names or descriptions. This feature can be implemented by maintaining a state for the search term and filtering the displayed products accordingly.



Key Concepts

- **Controlled Components**: Managing the search input value using state.
- **Filtering Logic**: Implementing logic to filter the list of products based on the search term.

Steps Involved

1. **Create a Search Input**:
   - Add an input field on the product listing page for users to type in search queries.
2. **Manage Search State**:
   - Track the search term entered by the user.
3. **Filter Products**:
   - Implement logic to filter the products based on the search term, updating the displayed products in real-time as the user types.

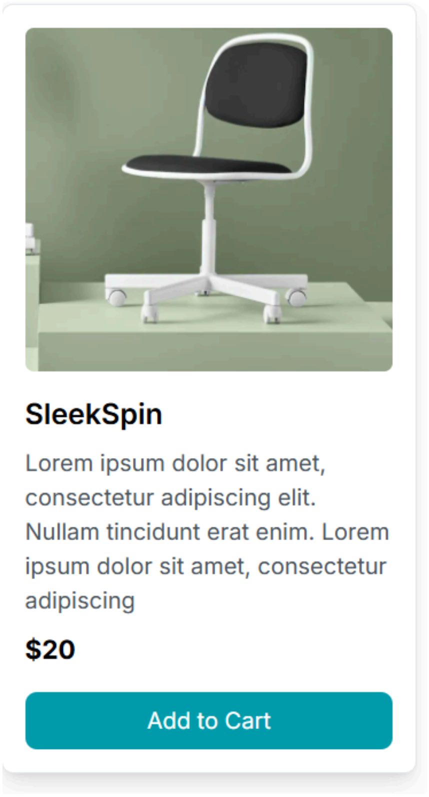
Expected Outcome

By the end of this step, users will be able to search for products by name or description, enhancing the usability and interactivity of your marketplace application.

Overview of the "Add to Cart" Feature

The **"Add to Cart"** feature is an essential part of any e-commerce platform.

It allows users to select products they wish to purchase and store them in a virtual shopping cart This helps users keep track of the items they are interested in, while also preparing for checkout.


SleekSpin

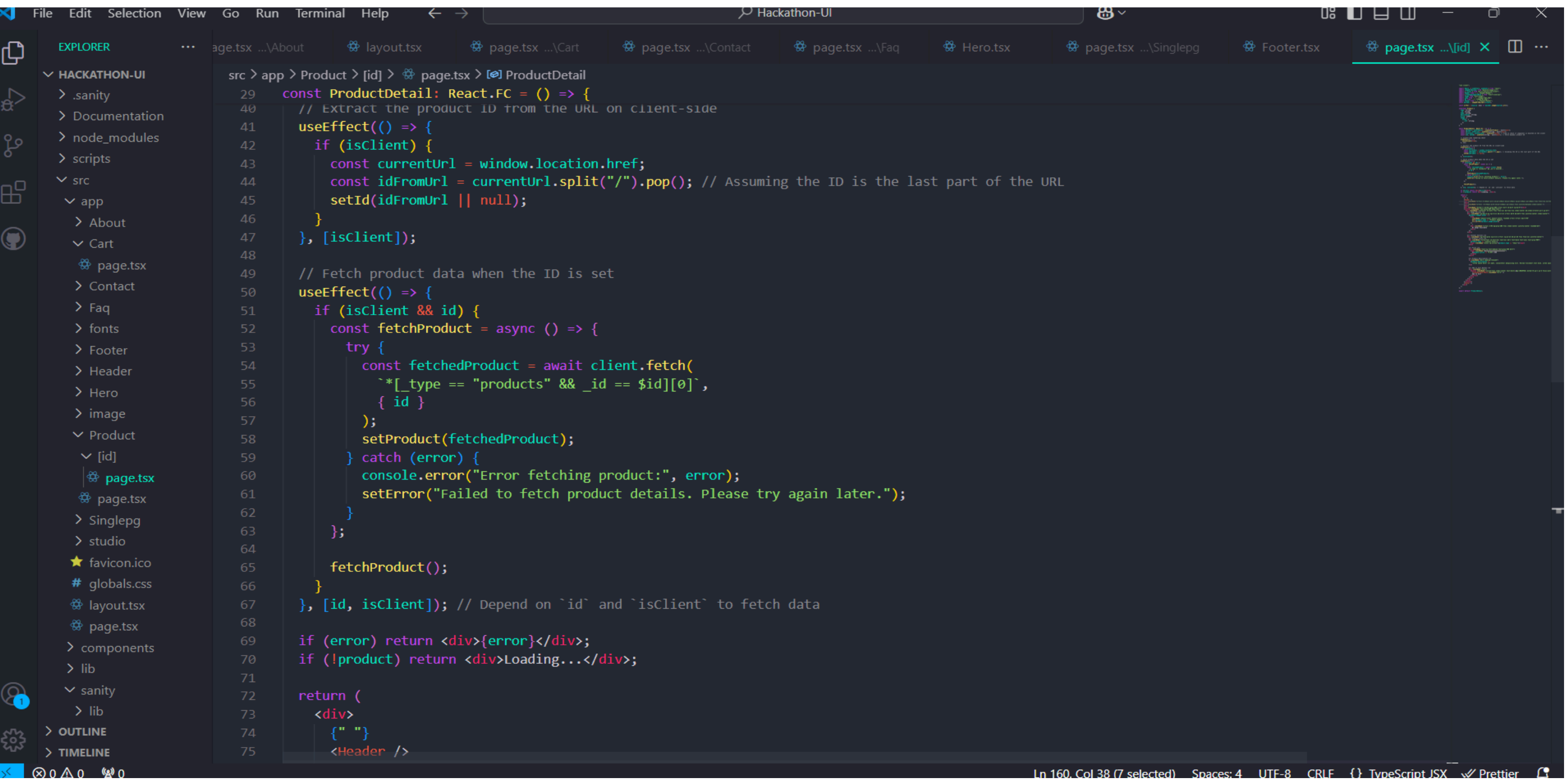
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam tincidunt erat enim. Lorem ipsum dolor sit amet, consectetur adipiscing

$20

Problem Solved

Key Problems Solved by Dynamic Routing in Product Details

1. **Efficiency**: Instead of having to create and maintain separate pages for each product, dynamic routes enable the reuse of a single page template for all products. The content is loaded based on the product ID in the URL.
2. **Scalability**: As the marketplace grows, dynamic routing allows for an unlimited number of products without the need for manual intervention in the routing configuration. The same page template can display any product's details dynamically.
3. **URL Management**: Dynamic URLs allow easy mapping between the product's unique identifier (ID) and the content displayed. The structure is easy to manage and doesn't require creating new routes for every new product.
4. **Maintaining Content Consistency**: When a product's information changes, you only need to update the data source (like a CMS or API), not the entire page or route, ensuring consistency and reducing maintenance.



```
const ProductDetail: React.FC = () => {
  // Extract the product ID from the URL on client-side
  useEffect(() => {
    if (isClient) {
      const currentUrl = window.location.href;
      const idFromUrl = currentUrl.split("/").pop(); // Assuming the ID is the last part of the URL
      setId(idFromUrl || null);
    }
  }, [isClient]);

  // Fetch product data when the ID is set
  useEffect(() => {
    if (isClient && id) {
      const fetchProduct = async () => {
        try {
          const fetchedProduct = await client.fetch(
            `*[_type == "products" && _id == $id][0]`,
            { id }
          );
          setProduct(fetchedProduct);
        } catch (error) {
          console.error("Error fetching product:", error);
          setError("Failed to fetch product details. Please try again later.");
        }
      };

      fetchProduct();
    }
  }, [id, isClient]); // Depend on `id` and `isClient` to fetch data

  if (error) return <div>{error}</div>;
  if (!product) return <div>Loading...</div>;

  return (
    <div>
      {" "}
      <Header />
```