Dynamic Routing in Next.Js

Dynamic routing in Next.js enables you to create flexible and dynamic applications by defining routes that adapt to varying parameters. This document outlines the dynamic routing implementation based on the provided code for the product page.

overview

The code demonstrates how to implement dynamic routing to display individual product details based on a unique identifier (_id). Each product's details are fetched from an API and displayed on a dynamically generated route.

Key Concepts

1. Dynamic Route Definition

Next.js uses the file-based routing system, where dynamic routes are defined by enclosing the parameter name in square brackets (e.g., [id]). For this implementation, the dynamic route would look like this:

```
/pages/products/[product._id].tsx
```

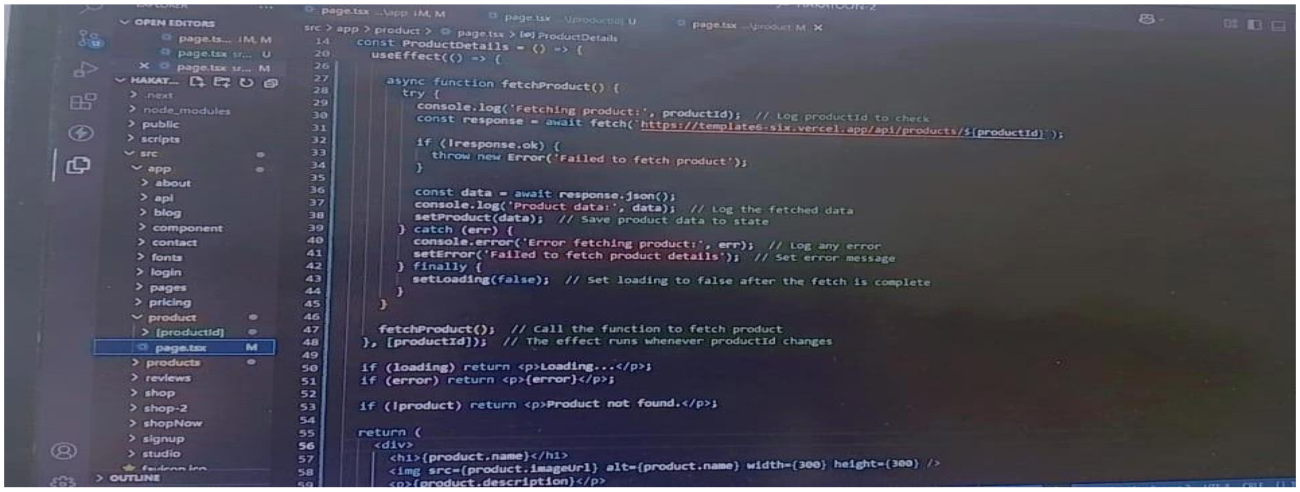
This file will handle requests to routes like /products/, where product_id is the dynamic id of a product.

2. Fetching Data for Dynamic Routes

The code fetches product data from an external API endpoint:

```
useEffect(() => {
  async function fetchData() {
    try {
      const response = await fetch('https://template6-six.vercel.app/api/products');
      const result = await response.json();
      setData(result);
    } catch {
      setError('Failed to fetch products');
    } finally {
      setLoading(false);
    }
  }
 fetchData();
}, []);
```

here is the picture of the code



```
const ProductDetails = () => {
  useEffect(() => {

    async function fetchProduct() {
      try {
        console.log('Fetching product:', productId);  // Log productId to check
        const response = await fetch(`https://template6-six.vercel.app/api/products/${productId}`);

        if (!response.ok) {
          throw new Error('Failed to fetch product');
        }

        const data = await response.json();
        console.log('Product data:', data);  // Log the fetched data
        setProduct(data);  // Save product data to state
      } catch (err) {
        console.error('Error fetching product:', err);  // Log any error
        setError('Failed to fetch product details');  // Set error message
      } finally {
        setLoading(false);  // Set loading to false after the fetch is complete
      }
    }

    fetchProduct();  // Call the function to fetch product
  }, [productId]);  // The effect runs whenever productId changes

  if (loading) return <p>Loading...</p>;
  if (error) return <p>{error}</p>;

  if (!product) return <p>Product not found.</p>;

  return (
    <div>
      <h1>{product.name}</h1>
      <img src={product.imageUrl} alt={product.name} width={300} height={300} />
      <p>{product.description}</p>
```

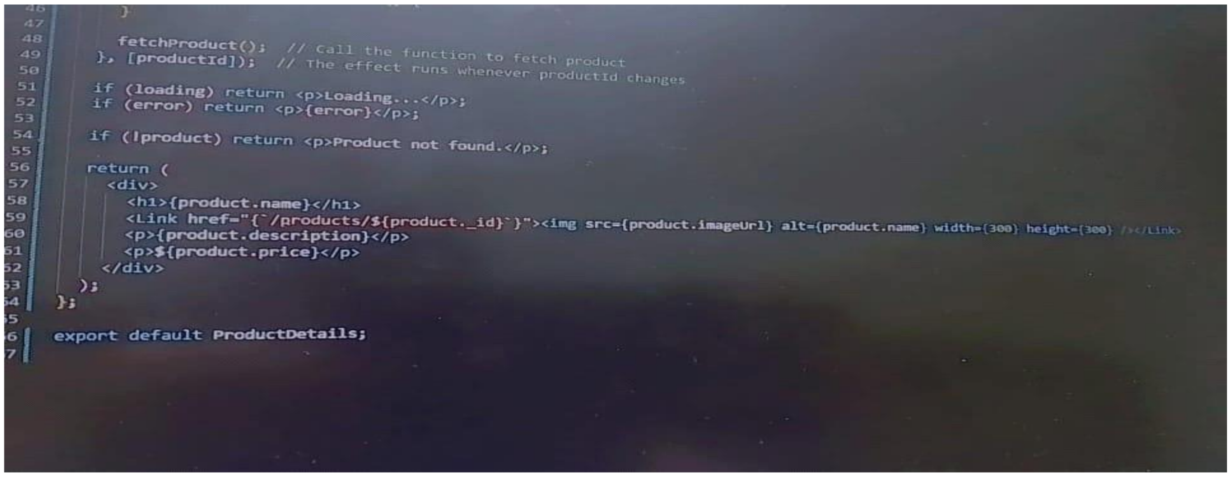
The product data is dynamically loaded and stored in the state using the useState hook.

3. Linking to Dynamic Routes

The Link component from Next.js is used to navigate to the dynamic product pages:

```
<Link href={`/products/${product._id}`}>
  <img
    src={product.imageUrl}
    alt={product.name}
    className="w-full h-48 sm:h-64 object-cover rounded-lg"
  />
</Link>
```

hereis the picture of the code



```
    }

    fetchProduct();  // Call the function to fetch product
  }, [productId]);  // The effect runs whenever productId changes

  if (loading) return <p>Loading...</p>;
  if (error) return <p>{error}</p>;

  if (!product) return <p>Product not found.</p>;

  return (
    <div>
      <h1>{product.name}</h1>
      <Link href="{`/products/${product._id}`}"><img src={product.imageUrl} alt={product.name} width={300} height={300} /></Link>
      <p>{product.description}</p>
      <p>${product.price}</p>
    </div>
  );
};

export default ProductDetails;
```

4. Dynamic Route Handling

The [id].tsx file would use the useRouter hook from Next.js to retrieve the dynamic id parameter:

```
import { useRouter } from 'next/router';

const ProductPage = () => {

  const router = useRouter();

  const { id } = router.query;


  // Fetch the product data using the `id`

  useEffect(() => {

    if (id) {

      // Fetch product details based on the `id`

    }

  }, [id]);

 return <div>Product Details for ID: {id}</div>;
};

export default ProductPage;
```

IMPLEMENTATION DETAILS

1. Search Functionality

The code allows users to filter products using a search bar:

```
const filteredProducts = data.filter(
  (product) =>
    product.title.toLowerCase().includes(searchQuery.toLowerCase()) ||
    product.description.toLowerCase().includes(searchQuery.toLowerCase())
);
```

This feature dynamically updates the displayed products based on the user's input

2. Cart Management

The code provides functions to manage a shopping cart, including adding, removing, and adjusting product quantities. For example:

- **Adding a product:**

```
const addToCart = (product, quantity) => {
  const existingProductIndex = cart.findIndex((item) => item.product._id === product._id);
  if (existingProductIndex >= 0) {
    const updatedCart = [...cart];
    updatedCart[existingProductIndex].quantity += quantity;
    setCart(updatedCart);
  } else {
    setCart([...cart, { product, quantity }]);
  }
};
```

- **Removing a product:**

```
const removeFromCart = (productId) => {
  const updatedCart = cart.filter((item) => item.product._id !== productId);
  setCart(updatedCart);
};
```

3. Product Description Toggle

The toggleDescription function dynamically shows or hides additional product details:

```
const toggleDescription = (productId) => {
  setExpandedDescription((prevId) => (prevId === productId ? null : productId));
};
```

Styling and UI Enhancements

The UI uses Tailwind CSS for responsive and clean designs. For example:

```
<div className="search-bar mb-4">
  <input
    type="text"
    placeholder="Search products..."
    className="border p-2 rounded-md w-full md:w-[30rem] hover:bg-slate-200 font-sans"
    value={searchQuery}
    onChange={(e) => setSearchQuery(e.target.value)}
  />
</div>
```

Summary

This implementation demonstrates dynamic routing in Next.js by:

- Creating routes that dynamically adapt to unique product IDs.
- Fetching data and rendering it dynamically based on user input.

- Enhancing user experience with features like search, cart management, and expandable product descriptions.

Dynamic routing is a powerful feature in Next.js that allows developers to create scalable and user-friendly applications. This code provides a robust foundation for a dynamic e-commerce platform.