**Marketplace Builder Hackathon 2025**

**Day 4: Building Dynamic Frontend Components for Your Marketplace**

**Introduction**

Welcome to Day 4 of the Marketplace Builder Hackathon 2025! In this tutorial, we'll focus on building dynamic frontend components for your marketplace application using Next.js. The goal is to create reusable, dynamic UI elements that fetch and display real-time data to provide an interactive shopping experience.

This guide covers key Next.js features like dynamic routing, API integration, and reusable components to improve the functionality and usability of your e-commerce platform.

**Table of Contents**

1. [Prerequisites](#prerequisites)
2. [Setting Up the Project](#setting-up-the-project)
3. [Dynamic Routing](#dynamic-routing)
4. [Building the Product Component](#building-the-product-component)
5. [Displaying Real-Time Data](#displaying-real-time-data)
6. [Reusable Component Design](#reusable-component-design)
7. [Final Touches and Testing](#final-touches-and-testing)
8. [Conclusion](#conclusion)

**1. Prerequisites**

Before starting with this tutorial, ensure you have the following:

* Basic knowledge of JavaScript and React
* Familiarity with Next.js and its core concepts
* An active Next.js project set up for your marketplace

If you’re new to Next.js, we recommend reviewing the official documentation: Next.js Documentation.

**2. Setting Up the Project**

Start by setting up a new Next.js project if you haven’t done so already:

bash

CopyEdit

npx create-next-app@latest my-marketplace

cd my-marketplace

Next, install any required dependencies for fetching data, such as Axios or a similar HTTP client:

bash

CopyEdit

npm install axios

**3. Dynamic Routing**

Dynamic routing allows you to create pages that are dynamically generated based on parameters (e.g., product ID or category).

In Next.js, create dynamic routes by using square brackets ([]). For instance, to generate a page for individual products, create a new file inside the pages directory:

bash

CopyEdit

mkdir pages/products

touch pages/products/[id].js

In [id].js, you'll fetch the product data based on the product ID:

js

CopyEdit

import { useRouter } from 'next/router'

import axios from 'axios'

const ProductPage = ({ product }) => {

const { query } = useRouter()

return (

<div>

<h1>{product.name}</h1>

<p>{product.description}</p>

<p>Price: ${product.price}</p>

</div>

)

}

export async function getServerSideProps({ params }) {

const { id } = params

const response = await axios.get(`https://api.example.com/products/${id}`)

const product = response.data

return { props: { product } }

}

export default ProductPage

This code fetches product details based on the id and displays them on the page.

**4. Building the Product Component**

The next step is to build a product component to display on the homepage or any product listing page.

Create a new component in the components folder:

bash

CopyEdit

mkdir components

touch components/ProductCard.js

Inside ProductCard.js, create a reusable component to display product details:

js

CopyEdit

const ProductCard = ({ product }) => {

return (

<div className="product-card">

<img src={product.imageUrl} alt={product.name} />

<h3>{product.name}</h3>

<p>{product.description}</p>

<p>Price: ${product.price}</p>

</div>

)

}

export default ProductCard

**5. Displaying Real-Time Data**

To make your components dynamic, you'll need to fetch real-time data. This could be data from an API, a database, or any external service.

For example, to display a list of products, you can fetch data from your API endpoint:

js

CopyEdit

import axios from 'axios'

import ProductCard from '../components/ProductCard'

const HomePage = ({ products }) => {

return (

<div className="product-list">

{products.map((product) => (

<ProductCard key={product.id} product={product} />

))}

</div>

)

}

export async function getServerSideProps() {

const response = await axios.get('https://api.example.com/products')

const products = response.data

return { props: { products } }

}

export default HomePage

This will display a list of products on the homepage, dynamically fetching data from the API.

**6. Reusable Component Design**

When building components, it's essential to make them reusable. For example, the ProductCard component can be used across different pages like the homepage, category pages, and product details page.

Consider making your components flexible by passing props for different use cases:

js

CopyEdit

const ProductCard = ({ product, showPrice = true }) => {

return (

<div className="product-card">

<img src={product.imageUrl} alt={product.name} />

<h3>{product.name}</h3>

<p>{product.description}</p>

{showPrice && <p>Price: ${product.price}</p>}

</div>

)

}

Here, the showPrice prop allows you to decide whether or not to display the price.

**7. Final Touches and Testing**

Once your components are set up and dynamic routing is working, it’s time to test your app.

* Ensure that data is being correctly fetched and displayed.
* Test navigation between product pages.
* Verify that your components are responsive and optimized for different devices.

You can also use the Next.js development server to test your app locally:

bash

CopyEdit

npm run dev

Visit http://localhost:3000 to preview your marketplace.

**8. Conclusion**

Congratulations! You've successfully built dynamic frontend components for your marketplace using Next.js. You’ve learned how to use dynamic routing, fetch real-time data, and create reusable components that can be integrated across your platform.

With these foundations, you’re well on your way to building a robust and interactive marketplace. Continue exploring advanced Next.js features to improve performance, SEO, and user experience!

**Next Steps**

* Explore styling solutions like CSS modules or styled-components for your marketplace.
* Implement user authentication to allow customers to log in and make purchases.
* Add payment gateway integration for a fully functional e-commerce experience.