**COMPLETE TUTORIAL TO BUILD E-COMMERCE SMALL APP USING**

1. **TailwindCSS**
2. **React Router DOM**
3. **React Icons**
4. **UseState And UseEffect HOOKS**
5. **CONTEXT API**
6. **Adding the products**
7. **Removing the products**
8. **Navigating to other pages**

//create react app

* create-react-app e-commerce
* cd e-commerce

//for icons

* npm install react-icons --save

//for react-router-dom

* npm i react-router-dom

//for tailwindcss

* npm install -D tailwind
* npx tailwindcssinit
* copy tailwind.config and past in tailwind.config.js
* copy content of index.css from website to your project

//start the APP

* npm start

let's start

Create the pages

* Create **Pages** folder

Create 3 files in this folder

* 1. Products.jsx
  2. Cart.jsx
  3. Success.jsx

Do rafce on each page

* Create **Components** folder

create 1 file in this folder

1. Navbar.jsx

Go to **index.js**

and Create **Routes**

import { createBrowserRouter, RouterProvider } from 'react-router-dom';

const router = createBrowserRouter([

  {

    path:'/',

    element:<App />,

    children:[

      {

        path:'/:category?',

        element:<Products />

      },

      {

        path:'/cart',

        element:<Cart />

      },

      {

        path:'/success',

        element:<Success />

      }

    ]

  }

])

//imports of index.js

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

//import pages

import Cart from './pages/Cart.jsx';

import Products from './pages/Products.jsx';

import Success from './pages/Success.jsx';

import reportWebVitals from './reportWebVitals';

import { createBrowserRouter, RouterProvider } from 'react-router-dom';

|  |  |
| --- | --- |
| Now instead of this </App> | Replace this |
| const root = ReactDOM.createRoot(document.getElementById('root'));  root.render(    <React.StrictMode>  </App>      </React.StrictMode>  ); | const root = ReactDOM.createRoot(document.getElementById('root'));  root.render(    <React.StrictMode>      <RouterProvider router={router}/>    </React.StrictMode>  ); |

Complete Index.js

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

//import pages

import Cart from './pages/Cart.jsx';

import Products from './pages/Products.jsx';

import Success from './pages/Success.jsx';

import reportWebVitals from './reportWebVitals';

import { createBrowserRouter, RouterProvider } from 'react-router-dom';

const router = createBrowserRouter([

  {

    path:'/',

    element:<App />,

    children:[

      {

        path:'/:category?',

        element:<Products />

      },

      {

        path:'/cart',

        element:<Cart />

      },

      {

        path:'/success',

        element:<Success />

      }

    ]

  }

])

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <React.StrictMode>

    <RouterProvider router={router}/>

  </React.StrictMode>

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

Now go to App.js and Add

**Outlet**

In React Router, the Outlet component is used as a placeholder for rendering child routes. It allows you to define a parent route that can have nested child routes, enabling you to create a hierarchy of routes in your application.

The Outlet component is crucial for creating nested routes in a React Router application. It allows for better organization of components and easier management of complex route structures, enhancing the overall navigation experience within a React app.

import './App.css';

import { Outlet } from 'react-router-dom';

function App() {

  return (

    <div>

      <Outlet/>

      </div>

  );

}

export default App;

Now routes are working fine

Check in browser

* localhost:3000/makeup
* localhost:3000/cosmetic
* localhost:3000/fragrance

Now let’s start with Navbar.jsx and import it in App.js

Open **App.js**

import './App.css';

import { Outlet } from 'react-router-dom';

import Navbar from './components/Navbar'

function App() {

  return (

    <div>

      <Navbar/>

      <Outlet/>

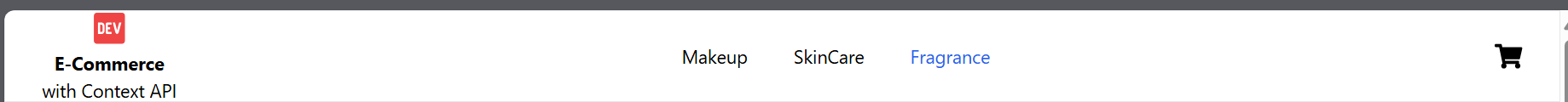
    </div>

  );

}

export default App;

let’s Design **Navbar.jsx**



import React, { useContext } from 'react';

import { BiLogoDevTo } from "react-icons/bi";

import { FaShoppingCart } from "react-icons/fa";

export default function Navbar() {

  return (

    <div className='w-full h-20 border shadow-lg flex items-center justify-between px-8 bg-white'>

  <BiLogoDevTo className='text-red-500 text-4xl' />

        <span className='font-bold'>E-Commerce</span>

        <span> with Context API</span>

      <ul className='flex items-center gap-10'>

<li><NavLink>Makeup</NavLink></li>

<li><NavLink>Cosmetic</NavLink></li>

<li><NavLink>Fragrance</NavLink></li>

</ul>

<div>

        <FaShoppingCart className='text-2xl' />

          </div>

    </div>

  )}

For data create a folder of **data**

Create a file in data folder named as products.js and add data

export const products\_data = [

    {

      id: 1,

      name: "Makeup Product",

      price: 10,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "makeup",

      image: "/images/m1.jpg"

    },

    {

      id: 2,

      name: "Makeup Product",

      price: 20,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "makeup",

      image: "/images/m2.jpg"

    },

    {

      id: 3,

      name: "Makeup Product",

      price: 30,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "makeup",

      image: "/images/m3.jpg"

    },

    {

      id: 4,

      name: "Makeup Product",

      price: 30,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "makeup",

      image: "/images/m4.jpg"

    },

    {

      id: 5,

      name: "Makeup Product",

      price: 10,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "makeup",

      image: "/images/m5.jpg"

    },

    {

      id: 6,

      name: "Cosmetic Product",

      price: 40,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "skincare",

      image: "/images/c1.jpg"

    },

    {

      id: 7,

      name: "Cosmetic Product",

      price: 99.99,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "skincare",

      image: "/images/c2.jpg"

    },

    {

      id: 8,

      name: "Cosmetic Product",

      price: 9.99,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "skincare",

      image: "/images/c3.jpg"

    },

    {

      id: 9,

      name: "Cosmetic Product",

      price: 7.99,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "skincare",

      image: "/images/c4.jpg"

    },

    {

      id: 10,

      name: "Cosmetic Product",

      price: 6,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "skincare",

      image: "/images/c5.jpg"

    },

    {

      id: 11,

      name: "Perfume Product",

      price: 7.5,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "fragrance",

      image: "/images/p1.jpg"

    },

    {

      id: 12,

      name: "Perfume Product",

      price: 4.99,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "fragrance",

      image: "/images/p1.jpg"

    },

    {

      id: 13,

      name: "Perfume Product",

      price: 9.99,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "fragrance",

      image: "/images/p1.jpg"

    },

    {

      id: 14,

      name: "Perfume Product",

      price: 4.99,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "fragrance",

      image: "/images/p1.jpg"

    },

    {

      id: 15,

      name: "Perfume Product",

      price: 9.99,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "fragrance",

      image: "/images/p1.jpg"

    },

    {

      id: 16,

      name: "Perfume Product",

      price: 5.99,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "fragrance",

      image: "/images/p1.jpg"

    },

    {

      id: 17,

      name: "Perfume Product",

      price: 9,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "fragrance",

      image: "/images/p1.jpg"

    },

    {

      id: 18,

      name: "Cosmetic Product",

      price: 10.99,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "skincare",

      image: "/images/c6.jpg"

    },

    {

      id: 19,

      name: "Cosmetic Product",

      price: 6.99,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "skincare",

      image: "/images/c7.jpg"

    },

    {

      id: 20,

      name: "Cosmetic Product",

      price: 17.99,

      smallDescription: "Glowing skin, Happy soul for our great customers.",

      category: "skincare",

      image: "/images/c1.jpg"

    }

  ];

    export const products\_categories = [{label:'Makeup',value:'makeup'},{label:'SkinCare',value:'skincare'},{label:'Fragrance',value:'fragrance'}];

Create a folder images and add images

Now map the products in Navbar.jsx file

Open **Navbar.jsx** and update

import React from 'react';

import { BiLogoDevTo } from "react-icons/bi";

import { Link, NavLink } from 'react-router-dom';

import { FaShoppingCart } from "react-icons/fa";

import { products\_categories } from '../data/products';

export default function Navbar() {

  return (

    <div className='w-full h-20 border shadow-lg flex items-center justify-between px-8 bg-white'>

      <NavLink className='flex flex-col items-center' to={'/'}>

        <BiLogoDevTo className='text-red-500 text-4xl' />

        <span className='font-bold'>E-Commerce</span>

        <span> with Context API</span>

      </NavLink>

      <ul className='flex items-center gap-10'>

        {

          products\_categories.map(category => {

            return (

**Mapping**

              <li key={category.value}>

<NavLinkto={`/${category.value}`}>{category.value}</NavLink>

</li>

            )

          })

        }

      </ul>

      <Link className=' relative' to={'/cart'}>

        <FaShoppingCart className='text-2xl' />

      </Link>

    </div>

  )

}

So check in browser localhost:3000/makeup working

Load all the data to home page

To manage the state of this Application here we use

**Context Api**

In SRC create folder Conext and file ProductContect.jsx

const { createContext } = require("react");

export const ProductContext = createContext([]);

export const ProductContextProvider = ({ children }) => {

    return (

        <ProductContext.Provider

              {children}

        </ProductContext.Provider>

    )

}

Now go to index.js

const root = ReactDOM.createRoot(document.getElementById('root'));

**Wrap**

**<RouterProvider router={router}/> in**

**<ProductContextProvider>**

**</ProductContextProvider>**

root.render(

  <React.StrictMode>

    <ProductContextProvider>

    <RouterProvider router={router}/>

    </ProductContextProvider>

  </React.StrictMode>

);

Go to **ProductContext.jsx** and add a **state**

import { products\_data } from "../data/products";

const { createContext, useState } = require("react");

export const ProductContext = createContext([]);

export const ProductContextProvider = ({ children }) => {

const [products, setProducts] = useState(products\_data)

    return (

        <ProductContext.Provider value={{products}}

              {children}

        </ProductContext.Provider>

    )}

Now go to the **Products.jsx**

import React, { useContext } from 'react'

import { ProductContext } from '../context/ProductContect';

export default function Products() {

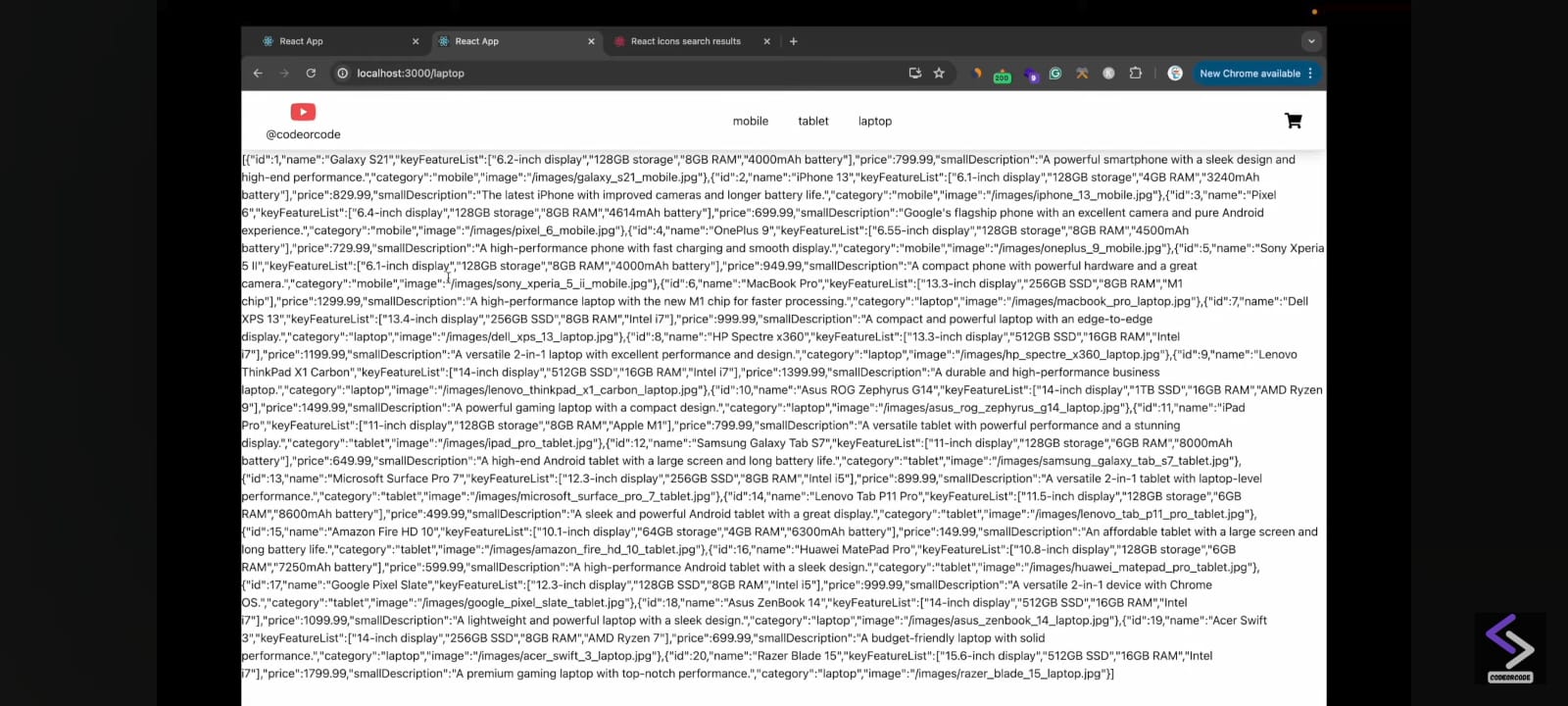
  const {products} = useContext(ProductContext) //destructure products

  return (

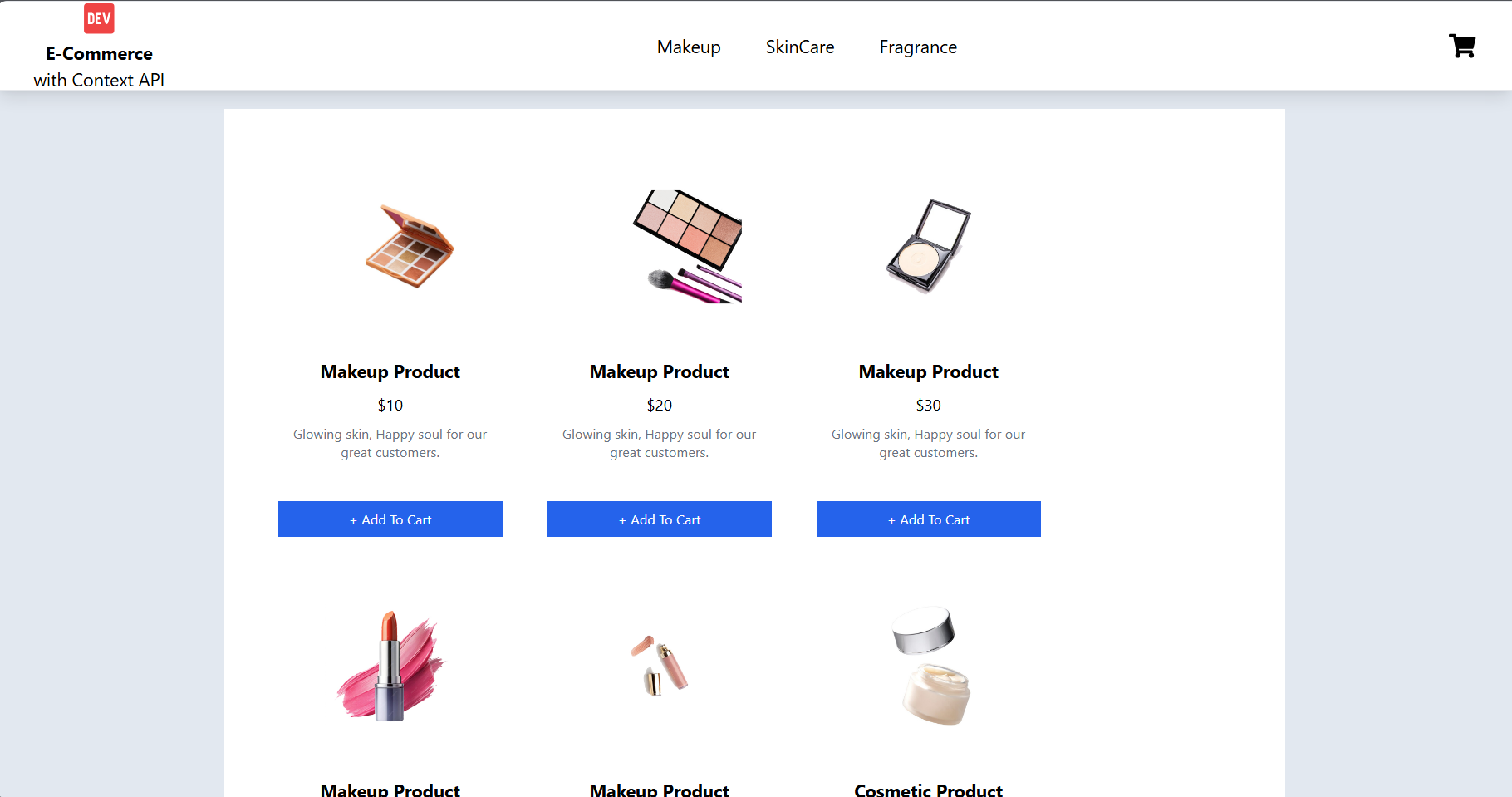
<div> {JSON.stringify(products)} </div>

  )}

So here we have the prducts data



Lets design this



Go to **App.jsx**

return (

    <div className='min-h-screen h-auto bg-slate-200 pb-40'>

      <Navbar/>

        <div className='w-[70%] m-auto p-10 bg-white my-4'>

      <Outlet/>

      </div></div>

  );

Now to **add products**

Go to **Products.jsx** and use **map function**

import React, { useContext } from 'react'

import { ProductContext } from '../context/ProductContect'

export default function Products() {

  const {products} = useContext(ProductContext)

  return (

    <div className='flex flex-wrap gap-6 text-center'>

      {

       products.map(product=>{

          return(

            <div key={product.id} className='w-[200px] rounded-md hover:shadow-lg m-2'>

              <img src={product.image} alt={product.namer} className='w-[200px] h-[150px] object-contain block m-auto p-4' />

              {/\* Product Info \*/}

              <div className='flex flex-col gap-2 my-4  h-[120px] p-2'>

                <p className='text-center font-bold'>{product.name}</p>

                <p className='text-center text-sm'>${product.price}</p>

                <p className='text-xs text-gray-500'>{product.smallDescription}</p>

              </div>

              <button className='w-full bg-blue-600 text-white text-center text-xs p-2' onClick={()=>addCart(product)}>+ Add To Cart</button>

            </div>

          )

        })

      }

    </div>

  )

}

**Now we will implement when press on Makeup | Cosmetic | Fragrance then filtered images will be shown**

Go to **Navbar.jsx**

Replace category.value

<NavLinkto={`/${category.value}`}>{category.value}</NavLink>

With this category.label

<NavLinkto={`/${category.value}`}>{category.label}</NavLink>

**Now we want if we select makeup it will open Makeup Products**

Go to **ProductContent.jsx**

Add

    const filterProducts = (category) => {

        console.log(category)

        if (category) {

            const filteredProducts = products\_data.filter(product => {

                if (product.category === category) {

                    return product;

                }

            })

            setProducts(filteredProducts)

        }

        else {

            setProducts(products\_data)

        }

    }

And pass it as value

    return (

        <ProductContext.Provider value={{products, filterProducts}}

              {children}

        </ProductContext.Provider>

    )

Go to **App.jsx**

import { Outlet, useParams } from 'react-router-dom';

import Navbar from './components/Navbar'

import { useContext, useEffect } from 'react';

import { ProductContext } from './context/ProductContect';

  const {filterProducts} = useContext(ProductContext);

const {category} = useParams();

  useEffect(()=>{filterProducts(category)},[category])

  return (

    <div className='min-h-screen h-auto bg-slate-200 pb-40'>

      <Navbar/>

        <div className='w-[70%] m-auto p-10 bg-white my-4'>

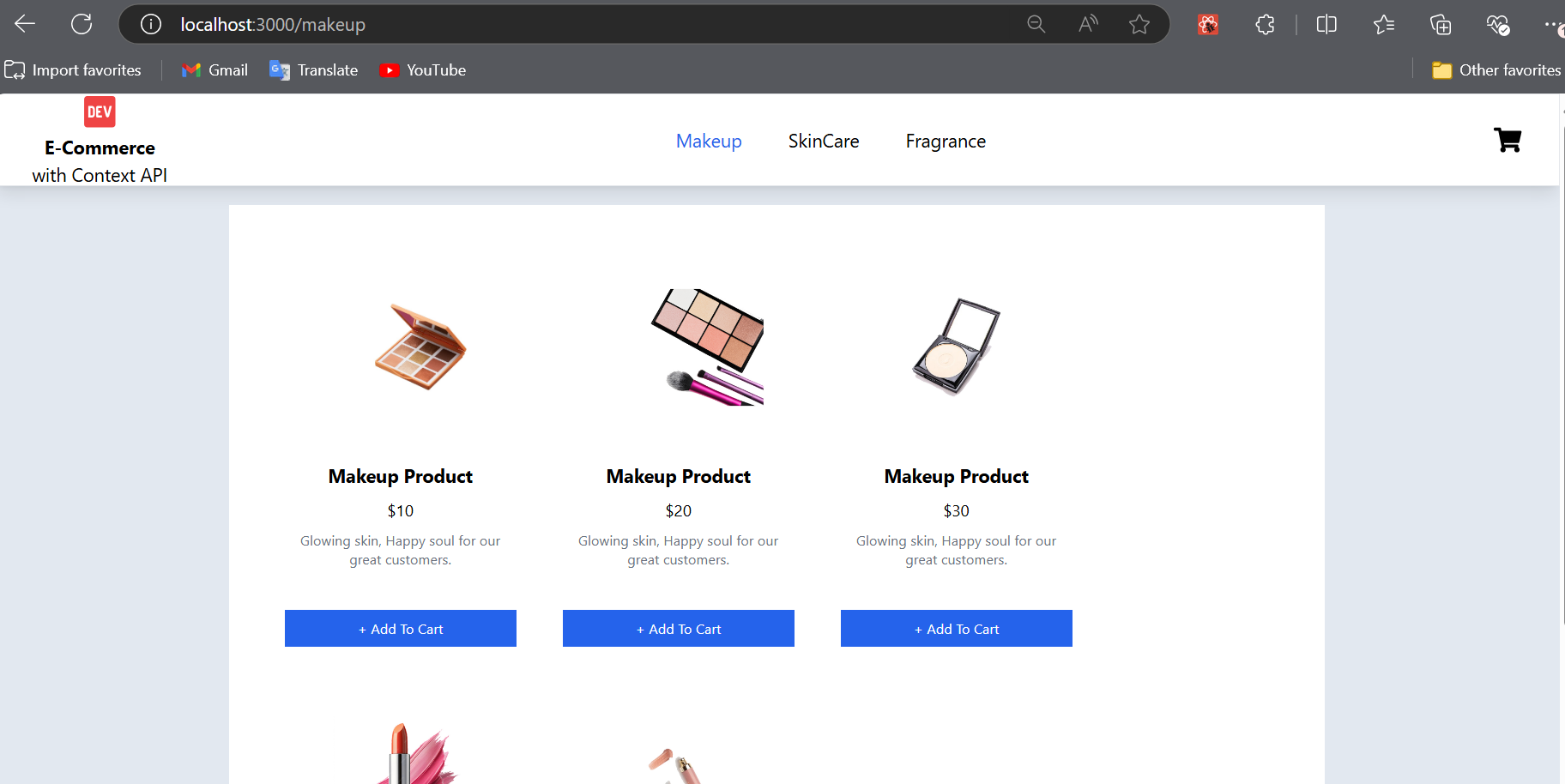
      <Outlet/>

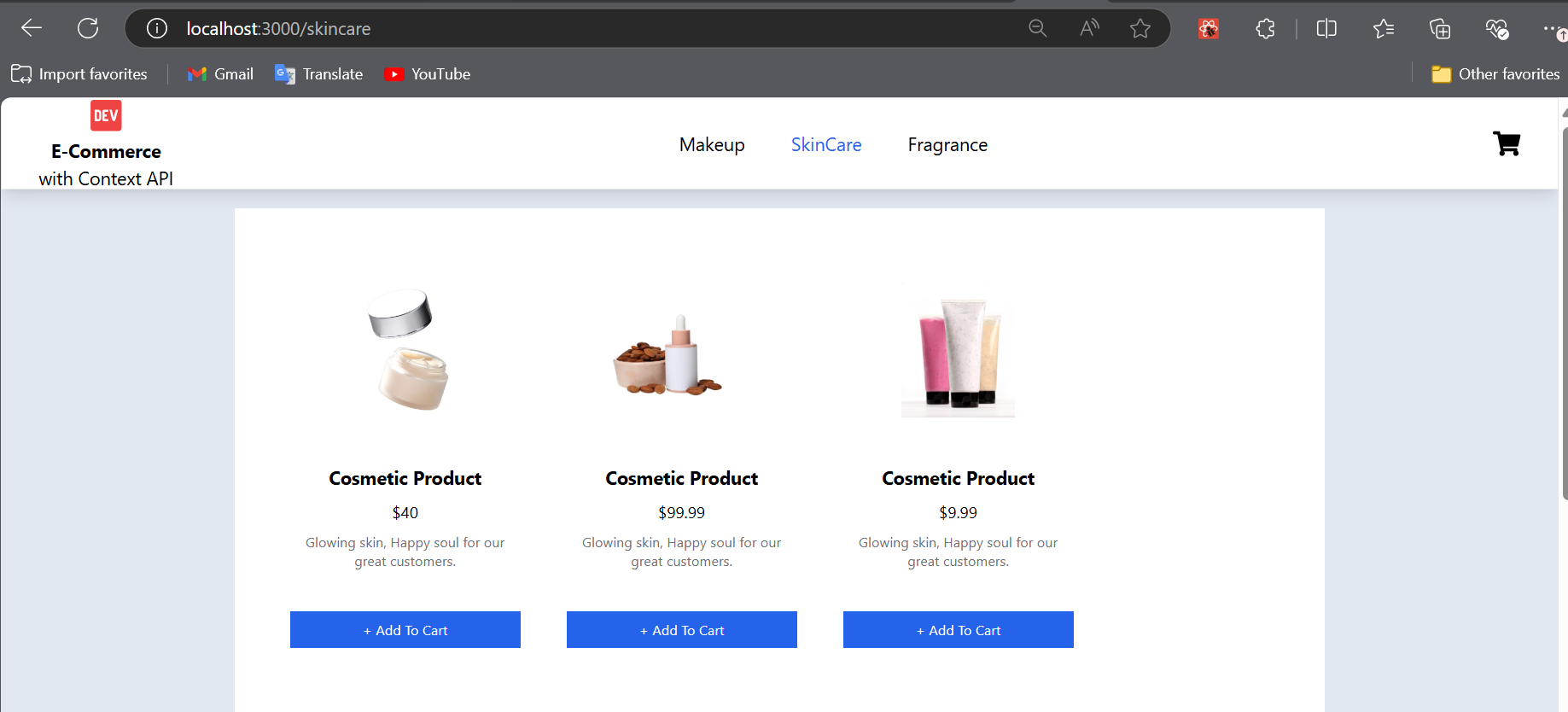
      </div>

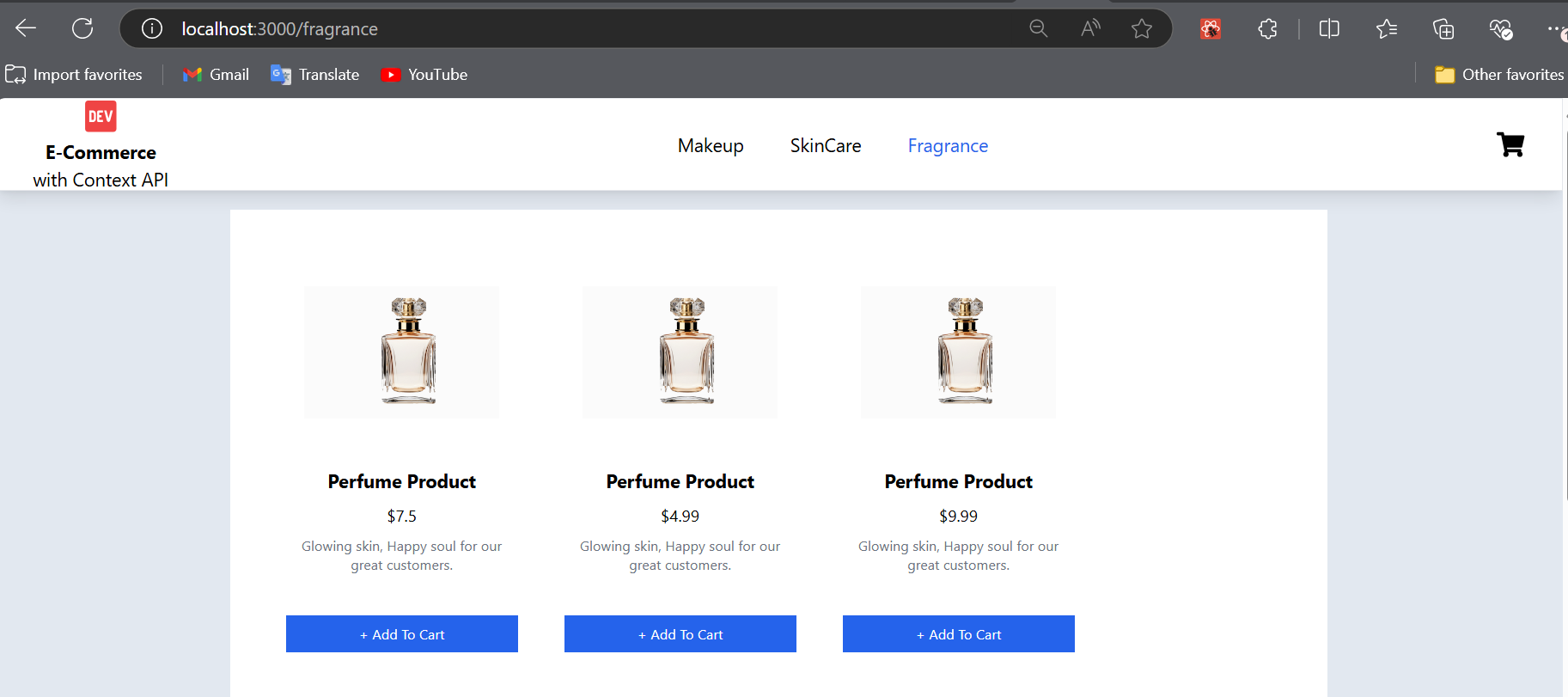
    </div>

  );}

Now see







Now **fix the Home button** to display all the content in one page

<NavLink className='flex flex-col items-center' to={'/'}>

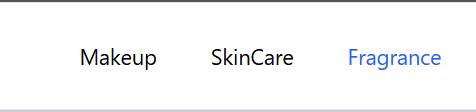
        <BiLogoDevTo className='text-red-500 text-4xl' />

        <span className='font-bold'>E-Commerce</span>

        <span> with Context API</span>

      </NavLink>

**Now next step is to change the color of navbar links when active**



**Explanation**

1. **Parameter element**:
   * element is the object you're passing into the function.
2. **Optional Chaining (?.)**:
   * element?.isActive uses optional chaining to safely check whether the element object exists and has the isActive property. If element is null or undefined, the expression will not throw an error and will return undefined instead.
   * If the isActive property is truthy (i.e., true), it will return 'text-blue-600', a CSS class often used in frameworks like Tailwind CSS to apply a blue text color.
3. **Return Value**:
   * If element?.isActive is true, the function returns the string 'text-blue-600'.
   * If element?.isActive is false, null, or undefined, it returns an empty string ''.

Open Navbar.jsx

 const isActive = (element) => {

    return element?.isActive ? 'text-blue-600' : ‘ '

  }

<ul className='flex items-center gap-10'>

        {

products\_categories.map(category => {

            return (

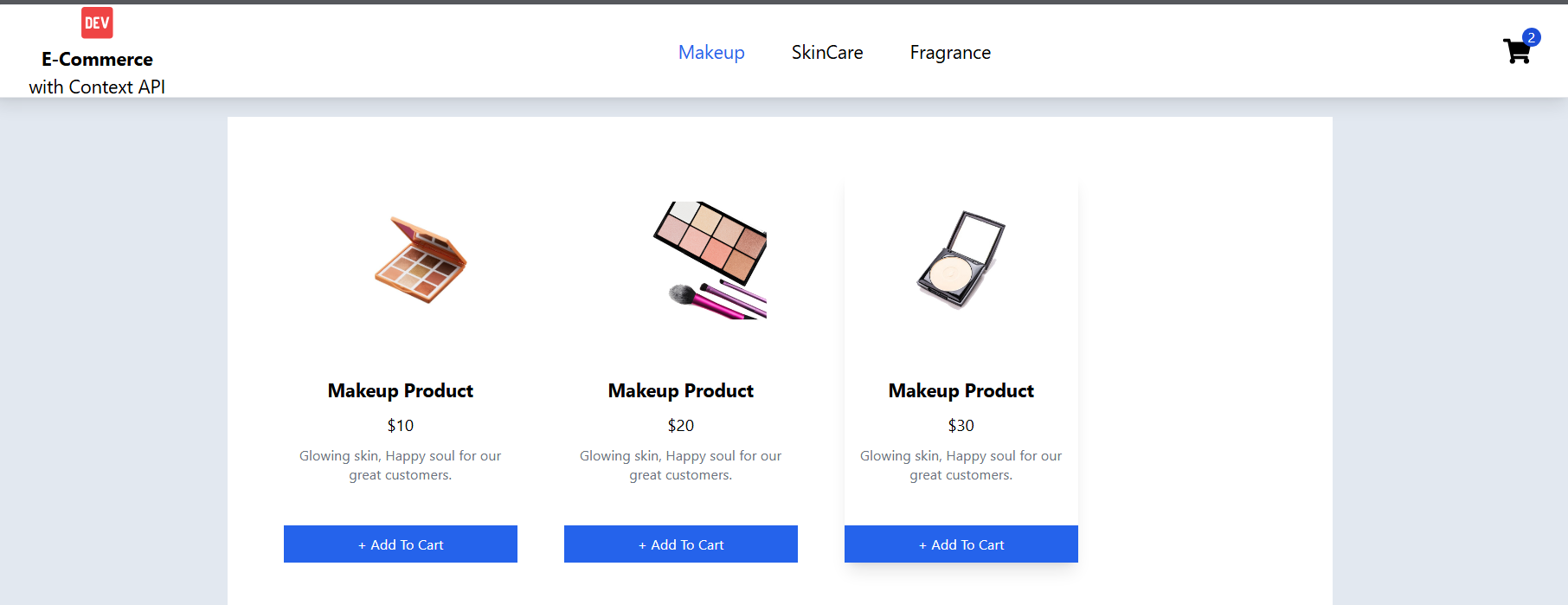
              <li key={category.value}><NavLink className={isActive} to={`/${category.value}`}>{category.label}</NavLink></li>

            ) }) }

</ul>

Next we are going to build the functionality

While pressing the Add to Cart Button there will be number icon on cart icon and things will be added in cart.



Go to ProductContext.js

Add one more state

const [cart,setCart] = useState([]);

 const addCart = (product)=>{

        setCart(oldCart=>{

            let previous = [...oldCart];

            if(previous.length<1){

                previous.push({...product,quantity:1});

            }

            else{

                const isProduct = previous.find(prod=>prod.id === product.id)

                if(!isProduct){

                    previous.push({...product,quantity:1});

                }

                else{

                    previous = previous.map(prod=>{

                        return prod.id === isProduct.id ? {...isProduct,quantity:isProduct.quantity + 1 } : prod;

                    })

                }  }  return previous;  }) }

Let’s pass this function addCart

 return (

        <ProductContext.Provider value={{ products, filterProducts, addCart, }}>

              {children}

        </ProductContext.Provider>

    )

Now

Go to Products.jsx

 const {products, addCart} = useContext(ProductContext)

 <button className='w-full bg-blue-600 text-white text-center text-xs p-2' onClick={()=>addCart(product)}>+ Add To Cart</button>

Now check using JSON.stringify that is products adding in the cart or not

Go to Productcontent.jsx and add

 return (

        <ProductContext.Provider value={{ products, filterProducts, addCart, }}>

{JSON.stringify(cart)}

              {children}

        </ProductContext.Provider>

    )

You can also use console.log to check if products are adding or not

In Productcontent.jsx

 const addCart = (product)=>{

console.log(product);

        setCart(oldCart=>{

            let previous = [...oldCart];

            if(previous.length<1){

and check in the browser for output

Now next step is to display the count on CART LOGO

ADD NEW STATE in PRODUCTCONTENT.JSX after const[cart, setCart] = useState([]);

const [invoice,setInvoice] = useState({count:0,subTotal:0});

Now add useEffect HOOK

const setInvoiceData = ()=>{

        setInvoice(previous=>{

            let newInvoice = {...previous,count:0,subTotal:0};

            cart.forEach(product=>{

                newInvoice.count += product.quantity;

                newInvoice.subTotal += product.quantity \* product.price;

            })

            return newInvoice;

        })

    }

 useEffect(()=>{

        const timeout = setTimeout(()=>{setMessage('')},600)

        setInvoiceData()

        return()=>{ clearTimeout(timeout)}

    },[cart])

hghvghv

This defines a function setInvoiceData, which is responsible for updating the invoice data based on the current state of the cart. Inside this function, the setInvoice function is called to update the invoice state.

The function setInvoice takes a callback as an argument. This callback receives the previous state of the invoice. Using this pattern ensures that you're updating the state based on its latest value, which is important when state updates are asynchronous in React.

let newInvoice = { ...previous, count: 0, subTotal: 0 };

This creates a new object newInvoice by copying the existing previous state using the spread operator (...previous). This is necessary because in React, state updates should always be immutable (i.e., you should avoid directly modifying the existing state). It then resets the count and subTotal fields to 0, preparing to recalculate them.

cart.forEach((product) => {

newInvoice.count += product.quantity;

newInvoice.subTotal += product.quantity \* product.price;

});

Here, you loop through each item in the cart array using forEach. For each product in the cart:

* newInvoice.count is incremented by the product's quantity, summing up the total number of items in the cart.
* newInvoice.subTotal is incremented by the product's quantity \* price, calculating the total cost of all items.

js

Copy code

return newInvoice;

});

};

After the loop finishes, the updated newInvoice object (with the recalculated count and subTotal) is returned. This will be used to update the state inside the setInvoice function.

#### useEffect Hook

js

Copy code

useEffect(() => {

This is the useEffect hook. It will run its contents whenever the dependency array ([cart]) changes. In this case, it runs when the cart changes, either due to an item being added, removed, or modified.

js

Copy code

const timeout = setTimeout(() => {

setMessage(''); // Clear the message after 600ms

}, 600);

This sets a timeout that will execute after 600 milliseconds. Inside the setTimeout callback, setMessage('') is called, which clears any message by setting it to an empty string after 600ms. The purpose of this seems to be to show a temporary message that will disappear after a short period.

js

Copy code

setInvoiceData();

After setting the timeout, the setInvoiceData function is called, which recalculates and updates the invoice based on the cart's current contents.

#### Cleanup Function

js

Copy code

return () => {

clearTimeout(timeout);

};

This is the cleanup function for the useEffect. It will run before the next effect is executed or when the component unmounts. In this case, it clears the timeout if the component re-renders or is removed, preventing memory leaks from lingering timeouts.

#### Dependency Array

js

Copy code

}, [cart]);

The useEffect hook will re-run whenever the value of cart changes. The dependency array [cart] ensures the effect only triggers when there's an actual change in cart.

**Key Concepts:**

1. **useEffect Hook**: Runs side effects (like setting timeouts) and handles re-running when cart changes.
2. **Timeout Logic**: Temporarily shows a message, then clears it after 600ms.
3. **Invoice Calculation**: Iterates through the cart to calculate the total count and subtotal of products.
4. **Cleanup**: Removes the timeout on unmount or before the next render to avoid potential issues.

This code ensures that whenever the cart changes, both the message and the invoice are updated accordingly, with a temporary message shown and cleared after 600ms.

Now pass this **invoice in productcontent.jsx**

  return (

        <ProductContext.Provider value={{ products, filterProducts, addCart, invoice}}>

              {children}

        </ProductContext.Provider>

    )

Now go to Navbar.jsx

      <Link className=' relative' to={'/cart'}>

        <FaShoppingCart className='text-2xl' />

          <div className=' absolute -top-2 -right-2 w-4 h-4 text-xs bg-blue-700 text-white flex items-center justify-center rounded-full'>

          </div>

      </Link>

  const { invoice } = useContext(ProductContext);

      <div className=' relative' >

        <FaShoppingCart className='text-2xl' />

        {

          invoice?.count > 0 &&

          <div className=' absolute -top-2 -right-2 w-4 h-4 text-xs bg-blue-700 text-white flex items-center justify-center rounded-full'>

            {invoice?.count}

          </div>

        }

      </div>

import React, { useContext } from 'react';

import { BiLogoDevTo } from "react-icons/bi";

import { Link, NavLink } from 'react-router-dom';

import { FaShoppingCart } from "react-icons/fa";

import { products\_categories } from '../data/products';

import { ProductContext } from '../context/ProductContect';

export default function Navbar() {

  const { invoice } = useContext(ProductContext)

  const isActive = (element) => {

    return element?.isActive ? 'text-blue-600' : ''

  }

  return (

    <div className='w-full h-20 border shadow-lg flex items-center justify-between px-8 bg-white'>

      <NavLink className='flex flex-col items-center' to={'/'}>

        <BiLogoDevTo className='text-red-500 text-4xl' />

        <span className='font-bold'>E-Commerce</span>

        <span> with Context API</span>

      </NavLink>

      <ul className='flex items-center gap-10'>

        {

          products\_categories.map(category => {

            return (

              <li key={category.value}><NavLink className={isActive} to={`/${category.value}`}>{category.label}</NavLink></li>

            )

          })

        }

      </ul>

      <div className=' relative' >

        <FaShoppingCart className='text-2xl' />

        {

          invoice?.count > 0 &&

          <div className=' absolute -top-2 -right-2 w-4 h-4 text-xs bg-blue-700 text-white flex items-center justify-center rounded-full'>

            {invoice?.count}

          </div>

        }

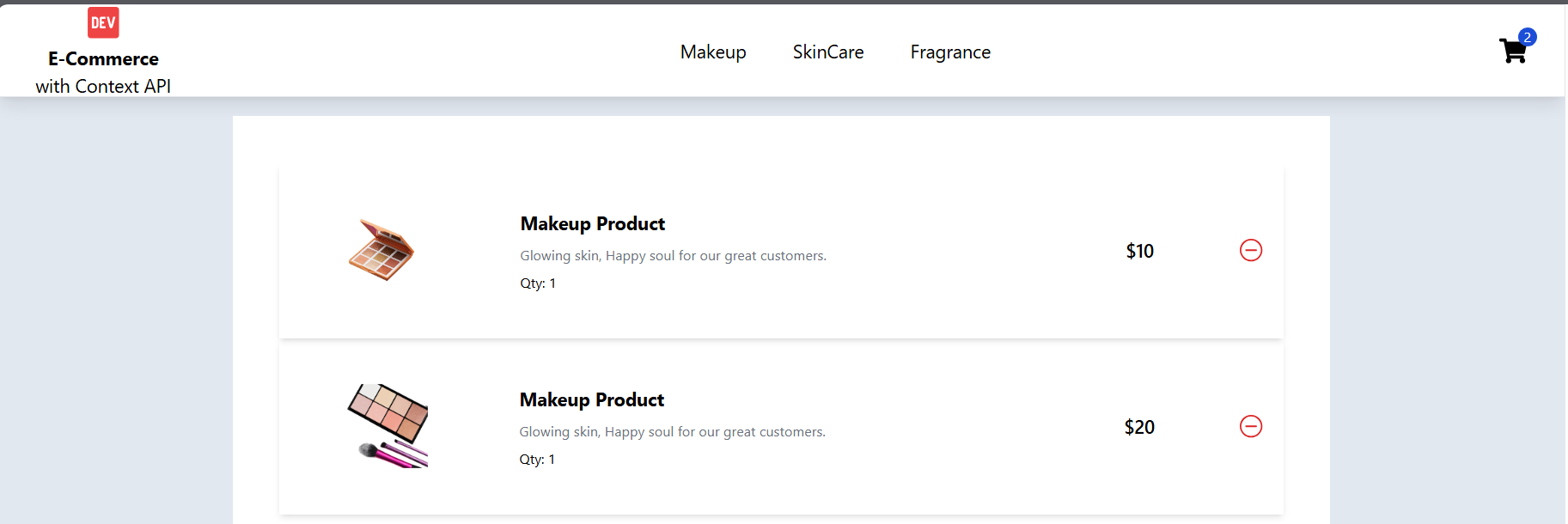
      </div>

    </div>

  )

}

**Next Step is**



When we click the cart icon the products should display on the page.

      <Link className=' relative' to={'/cart'}>

        <FaShoppingCart className='text-2xl' />

        {

          invoice?.count > 0 &&

          <div className=' absolute -top-2 -right-2 w-4 h-4 text-xs bg-blue-700 text-white flex items-center justify-center rounded-full'>

            {invoice?.count}

          </div>

        }

      </Link>

Now go to **Cart.jsx** Page

Now pass this **cart in productcontent.jsx**

  return (

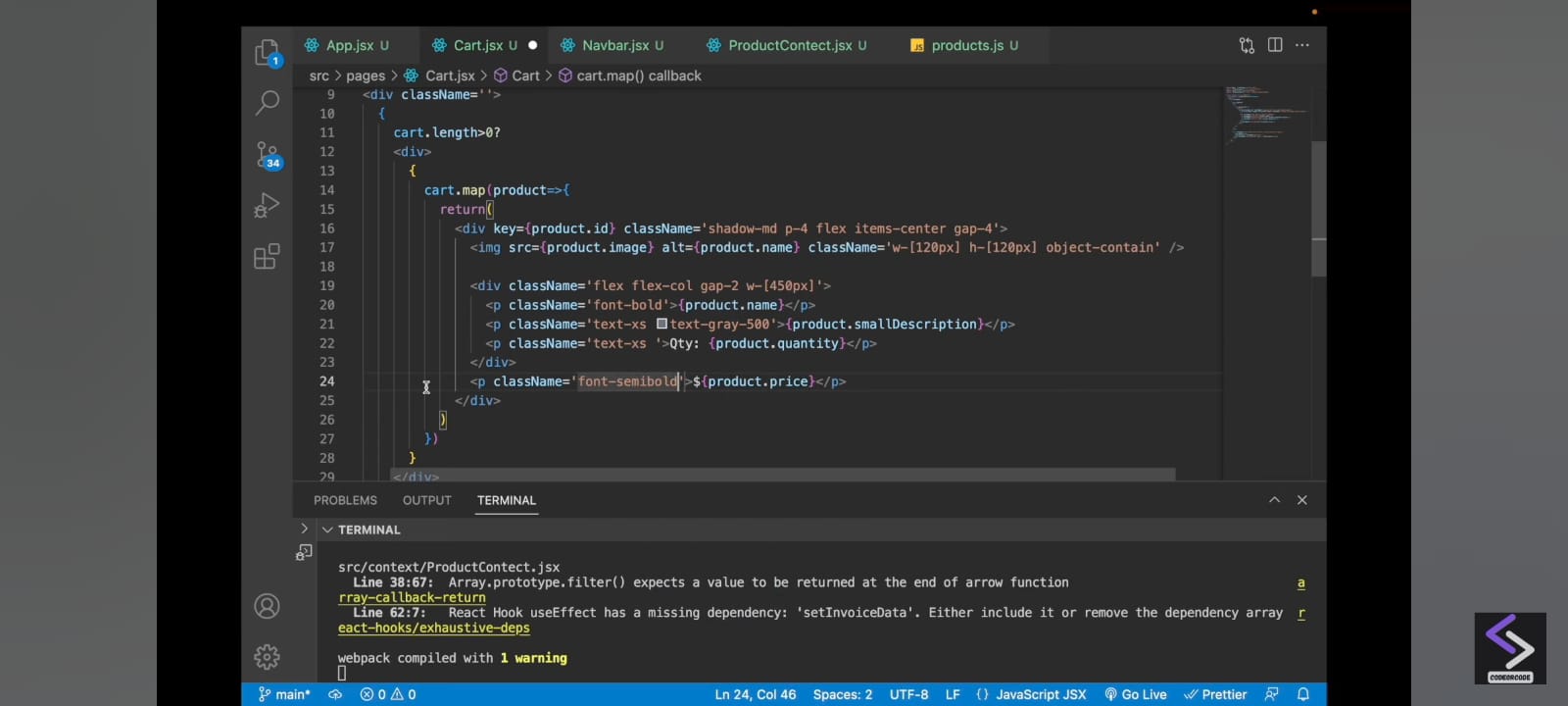
        <ProductContext.Provider value={{ products, filterProducts, addCart, invoice, cart}}>

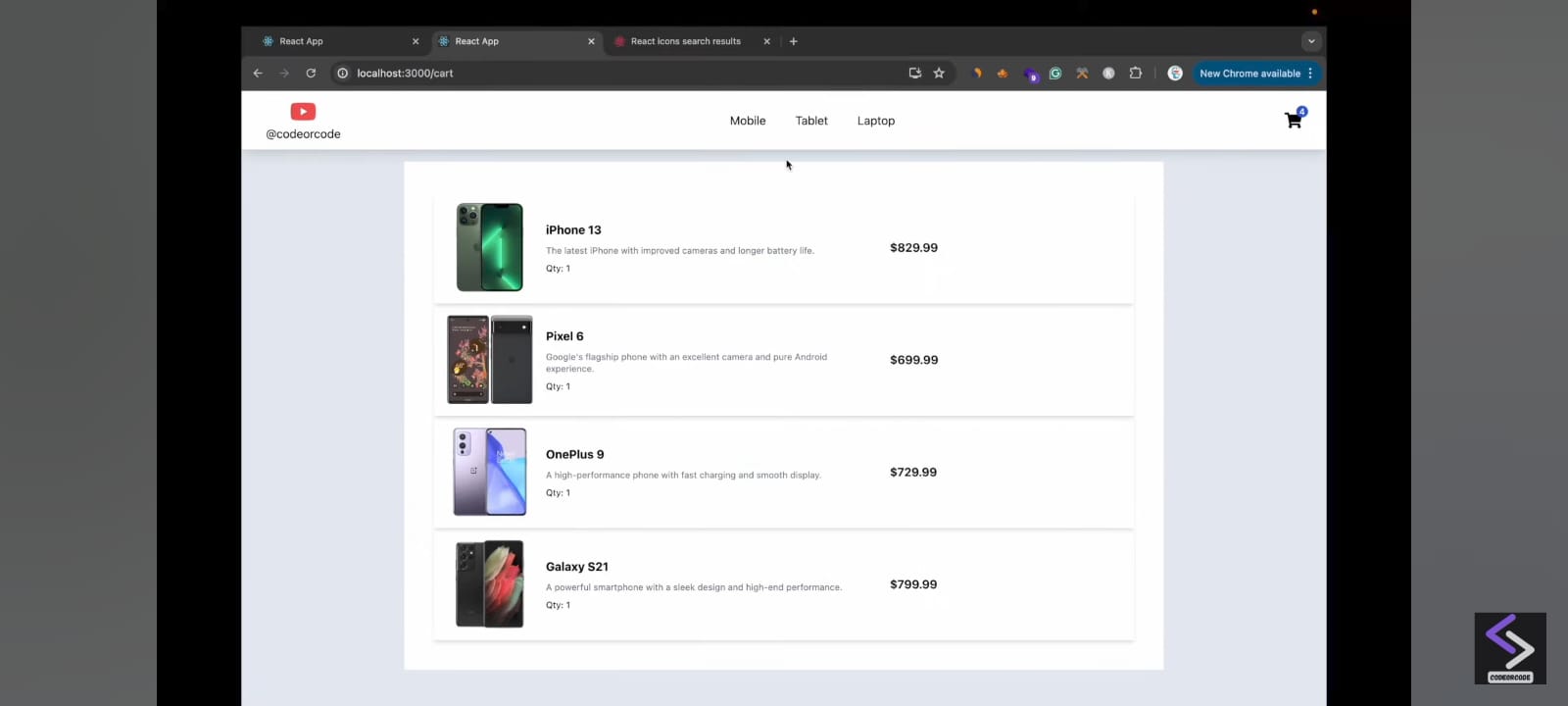
              {children}

        </ProductContext.Provider>

    )

**Cart.jsx**

****

****

**Now we Need – Minus button**

<IoIosRemoveCircleOutline className='text-red-600 text-2xl cursor-pointer' />

**Now Add subtottle**

<div className='flex flex-col items-end gap-3 py-4'>

            <p className='font-bold'>Subtotal({invoice.count} {invoice.count > 1 ? 'items':'item'}): ${invoice.subTotal.toFixed(2)}</p>

          </div>

**Add place order button**

<div className='flex flex-col items-end gap-3 py-4'>

            <p className='font-bold'>Subtotal({invoice.count} {invoice.count > 1 ? 'items':'item'}): ${invoice.subTotal.toFixed(2)}</p>

            <button className='bg-blue-600 text-sm text-white p-2 w-[200px] rounded-md'>Place Order</button>

          </div>

**Now Implement Remove Items from the Cart**

**Go to ProductContent.jsx**

**Copy Paste addCart function and remane with removeCart**

 const removeCart = (product)=>{

        setMessage(`${product.name} removed from the cart`)

        setCart(oldCart=>{

            let previous = [...oldCart];

            const isProduct = previous.find(prod=>prod.id == product.id)

            if(isProduct){

                const index = previous.indexOf(isProduct);

                previous.splice(index,1)

            }

            return previous;

        })

    }

**Explanation**

The function removeCart removes a product from the cart and sets a message informing the user about the removal. Here’s a breakdown of the code:

**1. Function Definition and Parameter:**

js

Copy code

const removeCart = (product) => {

* removeCart is an arrow function that takes product as a parameter.
* product is expected to be an object representing a product with at least an id and name.

**2. Set Removal Message:**

js

Copy code

setMessage(`${product.name} removed from the cart`)

* This sets a message in the UI using the setMessage function.
* The message is constructed using template literals (`${}`) to display the product's name.

**3. Update the Cart State:**

js

Copy code

setCart(oldCart => {

* setCart is used to update the state of the cart. It accepts a callback function where oldCart represents the previous state of the cart (an array of products).
* The callback allows us to safely update the state based on the current value of oldCart.

**4. Copying the Previous Cart State:**

js

Copy code

let previous = [...oldCart];

* A shallow copy of the oldCart array is made into the variable previous. This is important because we don't want to mutate the state directly.

**5. Finding the Product in the Cart:**

js

Copy code

const isProduct = previous.find(prod => prod.id == product.id)

* This line searches for the product in the copied previous array by checking if any product's id matches the id of the product to be removed.
* If the product is found, isProduct will hold that product; otherwise, it will be undefined.

**6. Removing the Product (if it Exists):**

js

Copy code

if (isProduct) {

const index = previous.indexOf(isProduct);

previous.splice(index, 1)

}

* If isProduct is not undefined, meaning the product exists in the cart, the code:
  1. Finds the index of the product in the previous array.
  2. Uses splice to remove that product from the array by its index.

**7. Returning the Updated Cart:**

js

Copy code

return previous;

* The updated previous array (which no longer contains the removed product) is returned, and the cart state is updated accordingly.

**Summary:**

* The function removes a product from the cart and updates the state to reflect the change.
* It first sets a message notifying the user of the removal.
* Then it checks if the product exists in the cart and, if found, removes it from the cart.

Now pass this **removeCart in productcontent.jsx**

  return (

        <ProductContext.Provider value={{ products, filterProducts, addCart, invoice, cart, removeCart}}>

              {children}

        </ProductContext.Provider>

    )

**Now go to Cart.jsx**

export default function Cart() {

  const {cart,invoice,removeCart} = useContext(ProductContext);

**Add in the button**

<IoIosRemoveCircleOutline className='text-red-600 text-2xl cursor-pointer' onClick={()=>removeCart(product)}/>

import React, { useContext } from 'react'

import { FaShoppingCart } from 'react-icons/fa'

import { Link, useNavigate } from 'react-router-dom'

import { ProductContext } from '../context/ProductContect'

import { IoIosRemoveCircleOutline } from "react-icons/io";

export default function Cart() {

  const {cart,invoice,removeCart,setCart,setInvoice} = useContext(ProductContext);

  const navigate = useNavigate();

  const placeOrder = ()=>{

    setCart([])

    setInvoice({count:0,subTotal:0});

    navigate('/success');

  }

  return (

    <div className=''>

      {

        cart.length>0?

        <div>

          {

            cart.map(product=>{

              return(

                <div key={product.id} className='shadow-md p-4 flex items-center gap-4 justify-between'>

                  <img src={product.image} alt={product.name} className='w-[120px] h-[120px] object-contain' />

                  <div className='flex flex-col gap-2 w-[450px]'>

                    <p className='font-bold'>{product.name}</p>

                    <p className='text-xs text-gray-500'>{product.smallDescription}</p>

                    <p className='text-xs '>Qty: {product.quantity}</p>

                  </div>

                  <p className='font-semibold'>${product.price}</p>

                 <IoIosRemoveCircleOutline className='text-red-600 text-2xl cursor-pointer' onClick={()=>removeCart(product)}/>

                </div>

              )

            })

          }

          <div className='flex flex-col items-end gap-3 py-4'>

            <p className='font-bold'>Subtotal({invoice.count} {invoice.count > 1 ? 'items':'item'}): ${invoice.subTotal.toFixed(2)}</p>

            <button className='bg-blue-600 text-sm text-white p-2 w-[200px] rounded-md' onClick={placeOrder}>Place Order</button>

          </div>

        </div>

        :

        <div className='flex items-center text-2xl justify-center p-4 gap-2 '>

           <span>Empty</span>

           <FaShoppingCart className='text-2xl' />

          <Link className='text-blue-600' to={'/'}>Add Products</Link>

        </div>

      }

    </div>

  )

}

Let's break down the React component step by step:

**1. Imports**

js

Copy code

import React, { useContext } from 'react';

import { FaShoppingCart } from 'react-icons/fa';

import { Link, useNavigate } from 'react-router-dom';

import { ProductContext } from '../context/ProductContect';

import { IoIosRemoveCircleOutline } from "react-icons/io";

* **React and useContext**: Importing React as it's needed for JSX, and useContext to access the context (in this case, ProductContext).
* **FaShoppingCart**: Importing a shopping cart icon from the react-icons/fa library.
* **Link and useNavigate**: These are from react-router-dom. Link helps create navigable links in the app, and useNavigate is a hook to programmatically navigate to other routes.
* **ProductContext**: Importing ProductContext to access the shared state and functions related to products and the shopping cart.
* **IoIosRemoveCircleOutline**: Importing a remove (delete) icon from react-icons/io.

**2. Cart Component**

js

Copy code

export default function Cart() {

const {cart, invoice, removeCart, setCart, setInvoice} = useContext(ProductContext);

const navigate = useNavigate();

* **Cart component**: The main functional component for the cart page.
* **useContext(ProductContext)**: Accesses the shared values (cart, invoice, removeCart, setCart, and setInvoice) from the ProductContext.
  + cart: The array of products in the cart.
  + invoice: Holds the details of the total number of items and the subtotal.
  + removeCart: Function to remove an item from the cart.
  + setCart: Function to update the cart (e.g., when placing an order and resetting the cart).
  + setInvoice: Function to update the invoice details.
* **useNavigate()**: Provides navigation functions for programmatic routing. In this case, it is used to navigate to a success page after placing an order.

**3. placeOrder function**

js

Copy code

const placeOrder = () => {

setCart([]);

setInvoice({count: 0, subTotal: 0});

navigate('/success');

}

* **placeOrder**: This function is triggered when the user clicks the "Place Order" button.
  + **setCart([])**: Resets the cart to an empty array (removing all items).
  + **setInvoice({count: 0, subTotal: 0})**: Resets the invoice details (item count and subtotal) to zero.
  + **navigate('/success')**: Navigates to the /success route, which could be a confirmation page.

**4. Returning JSX (UI)**

js

Copy code

return (

<div className=''>

{

cart.length > 0 ?

<div>

* **Conditional rendering**: Checks if there are items in the cart (cart.length > 0). If so, the cart items are displayed; otherwise, an "Empty" message appears.

**5. Rendering Cart Items**

js

Copy code

{

cart.map(product => {

return (

<div key={product.id} className='shadow-md p-4 flex items-center gap-4 justify-between'>

<img src={product.image} alt={product.name} className='w-[120px] h-[120px] object-contain' />

* **cart.map()**: Loops over the items in the cart and returns JSX for each product.
* **key={product.id}**: Assigns a unique key to each product, based on its id, which helps React track elements efficiently.
* **Product display**:
  + **product.image**: Displays the product image.
  + **product.name**: The name of the product.
  + **CSS classes** (like w-[120px], h-[120px], flex, etc.) are Tailwind CSS classes for styling.

<div className='flex flex-col gap-2 w-[450px]'>

<p className='font-bold'>{product.name}</p>

<p className='text-xs text-gray-500'>{product.smallDescription}</p>

<p className='text-xs '>Qty: {product.quantity}</p>

</div>

<p className='font-semibold'>${product.price}</p>

<IoIosRemoveCircleOutline className='text-red-600 text-2xl cursor-pointer' onClick={() => removeCart(product)} />

</div>

)

})

}

* **Product details**:
  + **Product name**: Rendered in a bold font (product.name).
  + **Small description**: Displayed below the name, with a smaller and lighter font (product.smallDescription).
  + **Quantity**: Displays how many of each item is in the cart (product.quantity).
  + **Price**: Rendered with bold styling (product.price).
  + **Remove icon (IoIosRemoveCircleOutline)**: A clickable icon to remove the item from the cart. The removeCart(product) function is called when the user clicks this icon.

**6. Cart Summary and "Place Order" Button**

js

Copy code

<div className='flex flex-col items-end gap-3 py-4'>

<p className='font-bold'>Subtotal({invoice.count} {invoice.count > 1 ? 'items' : 'item'}): ${invoice.subTotal.toFixed(2)}</p>

<button className='bg-blue-600 text-sm text-white p-2 w-[200px] rounded-md' onClick={placeOrder}>Place Order</button>

</div>

</div>

* **Subtotal**: Displays the total item count and the subtotal amount from the invoice object.
  + **{invoice.count}**: The number of items in the cart.
  + **${invoice.subTotal.toFixed(2)}**: The subtotal price formatted to two decimal places.
* **"Place Order" Button**: Calls the placeOrder function when clicked, clearing the cart and navigating to the success page.

**7. Empty Cart Message**

js

Copy code

:

<div className='flex items-center text-2xl justify-center p-4 gap-2 '>

<span>Empty</span>

<FaShoppingCart className='text-2xl' />

<Link className='text-blue-600' to={'/'}>Add Products</Link>

</div>

}

</div>

)

}

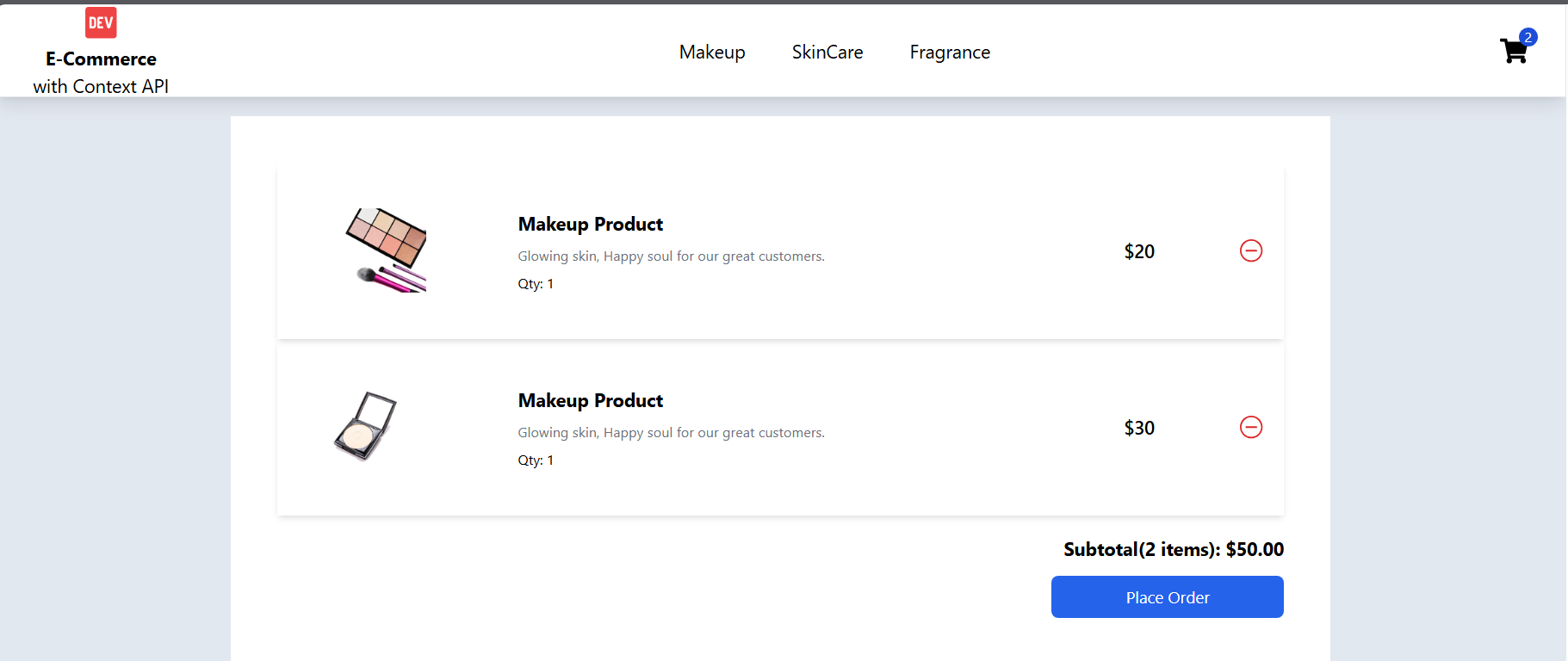
* **Empty cart**: If the cart is empty (cart.length === 0), the user sees an "Empty" message with a shopping cart icon and a link to add products (navigating to the home page).

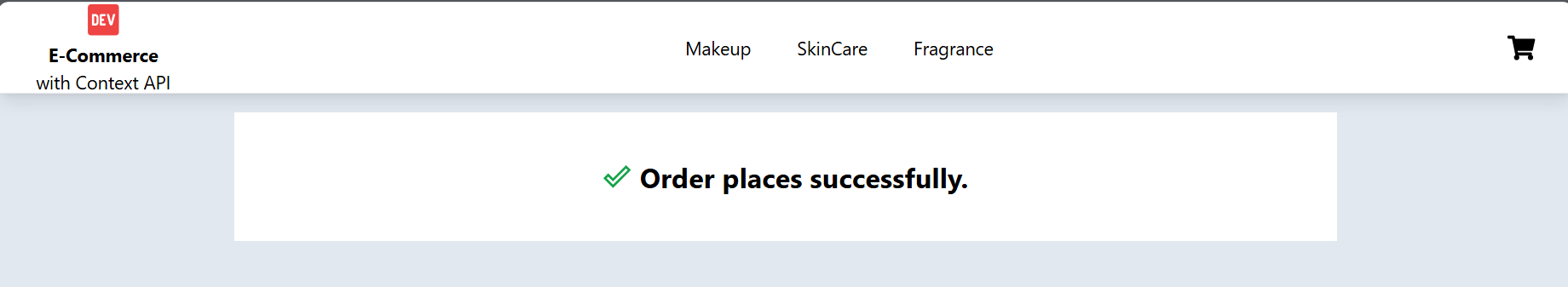
**Summary**

This component manages the display of the user's shopping cart. It checks if there are products in the cart:

* If products exist, it maps over them and displays each product's details (image, name, quantity, price). Users can remove products via a remove icon. The subtotal is displayed, and users can place their order.
* If the cart is empty, a message prompts users to add products to the cart.

**Now implement this**

****

****

**Let’s make Success page**

import React from 'react';

import { MdDoneOutline } from "react-icons/md";

export default function Success() {

  return (

    <div className='flex items-center justify-center font-bold gap-2 text-2xl'><MdDoneOutline className='text-green-600'/>Order places successfully.</div>

  )

}

**Now we should pass the two methods in Productcontent.jsx**

 return (

        <ProductContext.Provider value={{ products, filterProducts, addCart, removeCart,invoice,setInvoice,cart,setCart }}>

            {message && <div className=" fixed rounded-md shadow-lg right-0 top-20 bg-green-600 text-white min-w-[300px] p-2 text-center">{message}</div>}

              {children}

        </ProductContext.Provider>

    )

**And get these values here in Cart.jsx**

const {cart,invoice,removeCart,setCart,setInvoice} = useContext(ProductContext);

**Now Add PlaceOrder function in Cart.jsx page**

const placeOrder = ()=>{

    setCart([])

    setInvoice({count:0,subTotal:0});

  }

Now add navigate to success page

  const navigate = useNavigate();

update the function

 const placeOrder = ()=>{

    setCart([])

    setInvoice({count:0,subTotal:0});

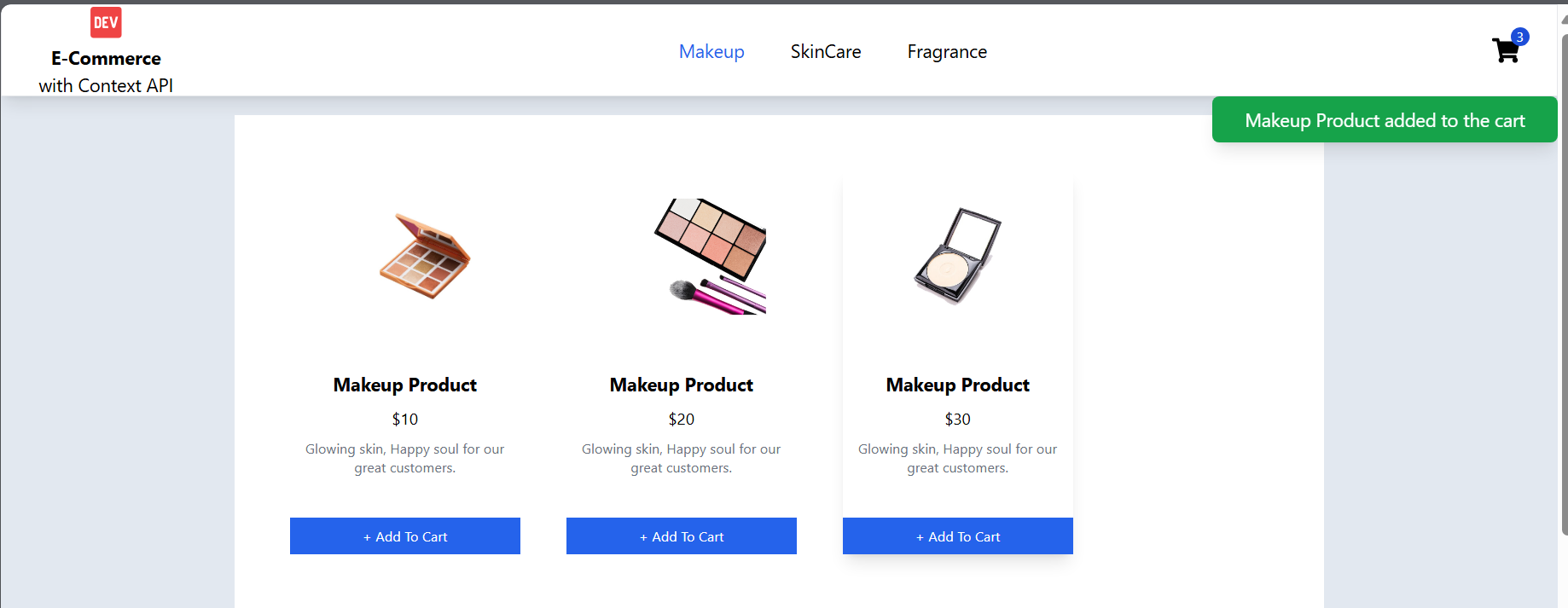
    navigate('/success');

  }

**Now add placeorder function on button**

 <button className='bg-blue-600 text-sm text-white p-2 w-[200px] rounded-md' onClick={placeOrder}>Place Order</button>

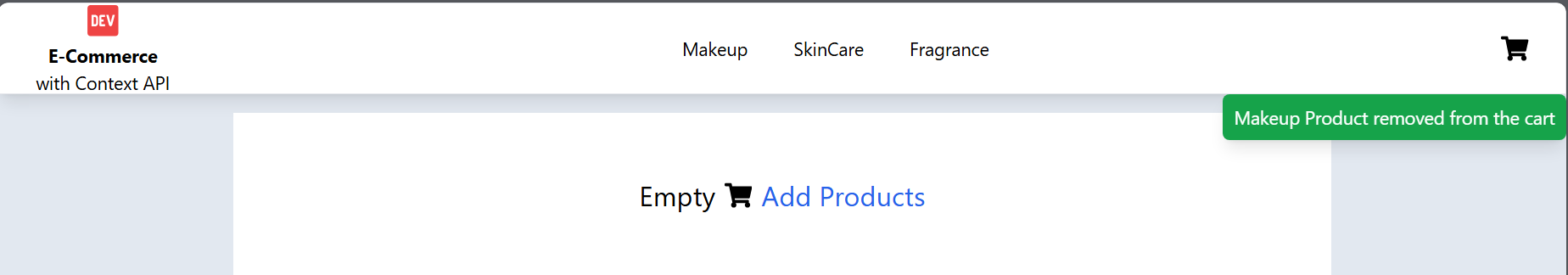
**Next thing to do is**

****

**When we press Add to Cart**

**A message will display**

**Product is added to the cart and also on removal of product**

****

**In productcontent.jsx add state**

   const [message,setMessage]=useState('');

 return (

        <ProductContext.Provider value={{ products, filterProducts, addCart, removeCart,invoice,setInvoice,cart,setCart }}>

            {message && <div className=" fixed rounded-md shadow-lg right-0 top-20 bg-green-600 text-white min-w-[300px] p-2 text-center">{message}</div>}

              {children}

        </ProductContext.Provider>

    )

Now add this to **addCart** function

 const addCart = (product)=>{

**setMessage(`${product.name} added to the cart`)**

        setCart(oldCart=>{

            let previous = [...oldCart];

            if(previous.length<1){

                previous.push({...product,quantity:1});

            }

            else{

                const isProduct = previous.find(prod=>prod.id === product.id)

                if(!isProduct){

                    previous.push({...product,quantity:1});

                }

                else{

                    previous = previous.map(prod=>{

                        return prod.id === isProduct.id ? {...isProduct,quantity:isProduct.quantity + 1 } : prod;

                    })

                }

            }

Now add this to **removeCart** function

 const removeCart = (product)=>{

**setMessage(`${product.name} removed from the cart`)**

        setCart(oldCart=>{

            let previous = [...oldCart];

            const isProduct = previous.find(prod=>prod.id == product.id)

            if(isProduct){

                const index = previous.indexOf(isProduct);

                previous.splice(index,1)

            }

            return previous;

        })

    }

**Now after 1 or 2 second it should disappear**

 useEffect(()=>{

**const timeout = setTimeout(()=>{setMessage('')},600)**

        setInvoiceData()

**return()=>{ clearTimeout(timeout)}**

    },[cart])