**Using useContext**

**By traltb@fe.edu.vn**

useContext là một Hook trong React cho phép các functional component truy cập trực tiếp vào giá trị của một Context. Context là một cách để chia sẻ dữ liệu giữa các component mà không cần phải truyền props qua nhiều tầng component con.

**Khi nào nên sử dụng Context?**

Context thường được sử dụng để chia sẻ dữ liệu mà nhiều component trong ứng dụng cần truy cập, chẳng hạn như:

* Theme (giao diện) của ứng dụng
* Ngôn ngữ hiện tại
* Thông tin người dùng đã đăng nhập
* Giỏ hàng

**Cách sử dụng useContext:**

1. **Tạo Context và cung cấp giá trị bằng Context.Provider**

Trước tiên, bạn cần tạo một context bằng createContext.

* + Sử dụng React.createContext() để tạo một đối tượng Context. Có thể truyền giá trị mặc định

const ThemeContext = React.createContext('light'); // Giá trị mặc định

1. **Cung cấp giá trị cho Context:**
   * Sử dụng component Provider để cung cấp giá trị cho Context.
   * Provider nhận một prop value chứa giá trị cần chia sẻ.

<ThemeContext.Provider value="dark">

{/\* Các component con \*/}

</ThemeContext.Provider>

1. **Truy cập giá trị Context:**
   * Trong functional component, sử dụng hook useContext để truy cập giá trị của Context.

function MyComponent() {

const theme = useContext(ThemeContext);

return (

<div>

Giao diện hiện tại: {theme}

</div>

);

}

**Exercise 1 (Exercise 14):** Tạo ứng dụng chuyển đổi Theme (chế độ sáng/tối) với useContext:

Sử dụng useContext để chia sẻ giá trị theme cho các component con mà không cần phải truyền props qua nhiều cấp.

**Tạo ứng dụng react:**

npx create-react-app exercise14-1

cd exercise14-1

npm start

**Bước 1**: Tạo ThemContext component: Dùng createContext() để tạo ra một context cho theme.

B1.1: Tạo 1 object **themes** là một object chứa hai theme: **light** và **dark**. Mỗi theme có hai thuộc tính: **foreground** (màu chữ) và **background** (màu nền).

export const themes = {

  light: {

    foreground: "#000000",

    background: "#eeeeee",

  },

  dark: {

    foreground: "#ffffff",

    background: "#61dafb",

  },

};

B1.2: **Tạo Context với createContext:**

* + **createContext** tạo ra một context mới, nơi bạn sẽ lưu trữ và chia sẻ thông tin về theme cho các component con.
  + **ThemeContext** là một đối tượng context mà các component khác có thể sử dụng để tiêu thụ giá trị của context.

const ThemeContext = createContext();

B1.3: Cung cấp giá trị context: tạo **ThemeProvider** để cung cấp giá trị theme và toggleTheme cho các component con thông qua ThemeContext.Provider.

export function ThemeProvider({ children }) {

  const [theme, setTheme] = useState(themes.light);

  const toggleTheme = () => {

    setTheme(theme === themes.light ? themes.dark : themes.light);

  };

  return (

    <ThemeContext.Provider value={{ theme, toggleTheme }}>

      {children}

    </ThemeContext.Provider>

  );

}

export default ThemeContext;

* ThemeProvider là một component cung cấp giá trị của theme và hàm toggleTheme cho các component con thông qua ThemeContext.Provider.
* useState(themes.light): Mặc định theme là light.
* toggleTheme: Hàm này dùng để chuyển đổi giữa hai theme (sáng và tối). Khi nhấn nút, theme sẽ thay đổi giữa light và dark.
* ThemeContext.Provider: Cung cấp giá trị cho các component con thông qua value={{ theme, toggleTheme }}, giúp các component con có thể truy cập giá trị này.
* Trong ThemeProvider, {children} đại diện cho bất kỳ nội dung nào được bao bọc bởi ThemeProvider. Điều này giúp chúng ta cung cấp giá trị context cho tất cả các component con của ThemeProvider.
* Khi bạn sử dụng {children}, React sẽ tự động thay thế {children} bằng các component được bao bọc trong ThemeProvider, giúp chúng có thể tiêu thụ giá trị của context (ở đây là theme và toggleTheme

**Bước 2**: Sử dụng useContext để tiêu thụ context trong component con

Tạo mới 1 component ThemeToggleButton

B2.1: Sử dụng useContext để tiêu thụ context:

const { theme, toggleTheme } = useContext(ThemeContext); // Lấy giá trị theme và hàm toggleTheme từ context

**useContext(ThemeContext)**: Hook này cho phép ThemeToggleButton truy cập vào giá trị context của ThemeContext. theme là giá trị theme hiện tại, và toggleTheme là hàm để thay đổi theme.

B2.2 Render button với style động dựa trên theme:

 <button

      onClick={toggleTheme}

      style={{

        background: theme === theme.background,

        color: theme.foreground,

      }}

    >

      Current Theme: {theme === themes.light ? "Light" : "Dark"}{" "}

      {/\* Hiển thị "Light" hoặc "Dark" \*/}

    </button>

* **Style động**: Dựa trên giá trị của theme, chúng ta sử dụng theme.background và theme.foreground để thay đổi màu nền và màu chữ của button.
* **Hiển thị tên theme**: Dựa vào theme, sẽ hiển thị "Light" nếu đang dùng theme sáng và "Dark" nếu đang dùng theme tối.

**Sự kiện onClick để thay đổi theme:**

* Khi nhấn nút, onClick={toggleTheme} sẽ gọi hàm toggleTheme từ ThemeContext, thay đổi theme từ sáng sang tối hoặc ngược lại.

**Bước 3**: Cung cấp giá trị context với ThemeProvider:

function App() {

  return (

    <ThemeProvider>

      <div>

        <h1>Welcome to React!</h1>

        <ThemeToggleButton />

      </div>

    </ThemeProvider>

  );

}

* + ThemeProvider bao bọc toàn bộ ứng dụng để cung cấp giá trị của context (theme, toggleTheme) cho các component con. Bất kỳ component nào bên trong ThemeProvider có thể truy cập và sử dụng context này.
  + Trong ví dụ này, component ThemeToggleButton nằm trong ThemeProvider, vì vậy nó có thể truy cập giá trị theme và hàm toggleTheme.

Code hoàn chỉnh:

File ThemeContext.js:

import React, { createContext, useState } from "react";

export const themes = {

  light: {

    foreground: "#000000",

    background: "#eeeeee",

  },

  dark: {

    foreground: "#ffffff",

    background: "#61dafb",

  },

};

const ThemeContext = createContext();

export function ThemeProvider({ children }) {

  const [theme, setTheme] = useState(themes.light);

  const toggleTheme = () => {

    setTheme(theme === themes.light ? themes.dark : themes.light);

  };

  return (

    <ThemeContext.Provider value={{ theme, toggleTheme }}>

      {children}

    </ThemeContext.Provider>

  );

}

export default ThemeContext;

File ThemeToggleButton.js:

import React, { useContext } from "react";

import ThemeContext, { themes } from "./ThemeContext"; //import themes để so sánh

function ThemeToggleButton() {

  const { theme, toggleTheme } = useContext(ThemeContext); // Lấy giá trị theme và hàm toggleTheme từ context

  return (

    <button

      onClick={toggleTheme}

      style={{

        background: theme === theme.background,

        color: theme.foreground,

      }}

    >

      Current Theme: {theme === themes.light ? "Light" : "Dark"}{" "}

      {/\* Hiển thị "Light" hoặc "Dark" \*/}

    </button>

  );

}

export default ThemeToggleButton;

File App.js:

import React from "react";

import ThemeToggleButton from "./ThemeToggleButton"; // Import ThemeToggleButton

import { ThemeProvider } from "./ThemeContext";

function App() {

  return (

    <ThemeProvider>

      <div>

        <h1>Welcome to React!</h1>

        <ThemeToggleButton />

      </div>

    </ThemeProvider>

  );

}

export default App;

A black text on a white background

AI-generated content may be incorrect.

**Exercise 2: Create a simple cart application using useContext in React**

**Cấu trúc ứng dụng**

**Tạo ứng dụng:**

npx create-react-app exercise14-2

cd exercise12-2

npm start

Ứng dụng sẽ có 3 phần chính:

1. **CartContext**: Cung cấp và quản lý trạng thái giỏ hàng toàn cục.
2. **DishesList**: Hiển thị danh sách món ăn và cho phép người dùng thêm món ăn vào giỏ.
3. **Cart**: Hiển thị các món ăn trong giỏ, tính tổng số lượng và giá trị, đồng thời cho phép xóa món ăn và làm sạch giỏ.

**Bước 1: Tạo CartContext**

Tạo một CartContext để quản lý giỏ hàng và cung cấp các hành động như thêm món ăn vào giỏ, xóa món ăn và làm sạch giỏ.

**CartContext.js**

import React, { createContext, useState, useEffect } from "react";

// Tạo context cho Cart

export const CartContext = createContext();

// CartProvider cung cấp các giá trị cho các component con

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

  const [cartItems, setCartItems] = useState([]);

  // Hàm thêm sản phẩm vào giỏ hàng

  const addToCart = (dish) => {

    setCartItems((prevItems) => [...prevItems, dish]);

  };

  // Hàm xóa sản phẩm khỏi giỏ hàng

  const removeFromCart = (id) => {

    setCartItems((prevItems) => prevItems.filter((item) => item.id !== id));

  };

  // Hàm xóa tất cả sản phẩm trong giỏ hàng

  const clearCart = () => {

    setCartItems([]);

  };

  // Tổng giá trị giỏ hàng

  const totalValue = cartItems

    .reduce((acc, item) => acc + parseFloat(item.price), 0)

    .toFixed(2);

  // Save cartItems to localStorage to persist across page reloads

  useEffect(() => {

    const savedCart = JSON.parse(localStorage.getItem("cartItems"));

    if (savedCart) {

      setCartItems(savedCart);

    }

  }, []);

  useEffect(() => {

    localStorage.setItem("cartItems", JSON.stringify(cartItems));

  }, [cartItems]);

  return (

    <CartContext.Provider

      value={{ cartItems, addToCart, removeFromCart, clearCart, totalValue }}

    >

      {children}

    </CartContext.Provider>

  );

};

**Giải thích:**

* Trong phương thúc addToCart: prevItems là mảng các món ăn hiện có trong giỏ hàng, ...prevItems: tạo ra bản sao của mảng prevItems, dish là món ăn được thêm vào cuối mảng. kết quả mảng giỏ mới sẽ chứa tất cả món ăn trước đó, cộng thêm món ăn mới vừa chọn.
* Trong phương thức removeFromCart: phương thức filter sẽ tạo ra mảng mới chứa các phần tử của mảng prevItems mà không có món chứa id trùng với id bạn muốn xóa. Mảng giỏ hàng mới sẽ chứa tất cả các món ăn trừ món ăn có id trùng với id đã cho
* Phương thức reduce:

reduce là một phương thức mảng trong JavaScript, cho phép chúng ta tính toán một giá trị duy nhất từ một mảng.

Cấu trúc cơ bản của reduce là:

array.reduce((accumulator, currentValue) => { ... }, initialValue)

* + **accumulator**: Đây là giá trị tích lũy mà chúng ta muốn tính toán qua mỗi vòng lặp. Ban đầu, giá trị này là initialValue.
  + **currentValue**: Đây là phần tử hiện tại trong mảng mà phương thức reduce đang xử lý.
  + **initialValue**: Giá trị bắt đầu cho accumulator. Trong trường hợp này, giá trị khởi tạo là 0, tức là tổng giá trị ban đầu của giỏ hàng là 0.

cartItems.reduce((acc, item) => acc + parseFloat(item.price), 0).toFixed(2);

 **acc + parseFloat(item.price)**:

* acc: Là giá trị tích lũy, bắt đầu từ 0 (theo initialValue).
* item: Là một đối tượng trong mảng cartItems, đại diện cho một món ăn.
* item.price: Là giá của món ăn (dưới dạng chuỗi, ví dụ "4.99").
* parseFloat(item.price): Chuyển đổi giá trị chuỗi item.price thành số thực (ví dụ, "4.99" thành 4.99).
* **acc + parseFloat(item.price)**: Cộng giá trị price của món ăn vào tổng tích lũy (accumulator).

 **Kết quả**: Phương thức reduce sẽ duyệt qua tất cả các món ăn trong giỏ hàng, cộng dồn giá của từng món vào acc và trả về tổng giá trị giỏ hàng, định dạng kết quả với 2 chữ số thập phân.

**Ví dụ:**

Giả sử cartItems là:

cartItems = [

{ id: 1, name: "Uthappizza", price: "4.99" },

{ id: 2, name: "Zucchipakoda", price: "1.99" },

{ id: 3, name: "Vadonut", price: "1.99" }

];

Quá trình của reduce sẽ như sau:

* Vòng 1: acc = 0, item = { id: 1, name: "Uthappizza", price: "4.99" }, tổng = 0 + 4.99 = 4.99
* Vòng 2: acc = 4.99, item = { id: 2, name: "Zucchipakoda", price: "1.99" }, tổng = 4.99 + 1.99 = 6.98
* Vòng 3: acc = 6.98, item = { id: 3, name: "Vadonut", price: "1.99" }, tổng = 6.98 + 1.99 = 8.97

Sau khi hoàn thành, reduce sẽ trả về 8.97.

 **toFixed(2)**: Đây là phương thức của kiểu dữ liệu số trong JavaScript, dùng để định dạng số thành chuỗi với số chữ số thập phân cố định (2 chữ số trong trường hợp này).

Ví dụ: 8.973.toFixed(2) sẽ trả về 8.97.

**Bước 2: Tạo các Component**

**B2.1 Tạo component DishesList.js:**

Đây là component để hiển thị danh sách món ăn và cho phép người dùng thêm món vào giỏ hàng.

import React, { useContext } from "react";

import { CartContext } from "./CartContext";

import PropTypes from "prop-types";

// DishesList component renders the list of dishes and adds them to the cart

const DishesList = ({ dishes }) => {

  const { addToCart } = useContext(CartContext);

  return (

    <div>

      <h2>Danh sách món ăn</h2>

      <div className="dishes">

        {dishes.map((dish) => (

          <div key={dish.id} className="dish-item">

            <img src={dish.image} alt={dish.name} />

            <h3>{dish.name}</h3>

            <p>{dish.description}</p>

            <p>{`Price: $${parseFloat(dish.price).toFixed(2)}`}</p>

            <button onClick={() => addToCart(dish)}>Add to Cart</button>

          </div>

        ))}

      </div>

    </div>

  );

};

// Prop validation to ensure proper data structure

DishesList.propTypes = {

  dishes: PropTypes.arrayOf(

    PropTypes.shape({

      id: PropTypes.number.isRequired,

      name: PropTypes.string.isRequired,

      image: PropTypes.string.isRequired,

      price: PropTypes.string.isRequired,

      description: PropTypes.string.isRequired,

    })

  ).isRequired,

};

export default DishesList;

**B2.2: Tạo Cart.js:**

Component này hiển thị các món trong giỏ hàng, tổng số món, tổng giá trị, và các nút để xóa món hoặc làm sạch giỏ hàng.

import React, { useContext } from "react";

import { CartContext } from "./CartContext";

const Cart = () => {

  const { cartItems, removeFromCart, clearCart, totalValue } =

    useContext(CartContext);

  return (

    <div>

      <h2>Giỏ hàng</h2>

      {cartItems.length === 0 ? (

        <p>Giỏ hàng của bạn đang trống.</p>

      ) : (

        <div>

          <ul>

            {cartItems.map((item) => (

              <li key={item.id}>

                {item.name} - ${item.price}

                <button onClick={() => removeFromCart(item.id)}>Remove</button>

              </li>

            ))}

          </ul>

          <div>

            <p>{`Tổng số món: ${cartItems.length}`}</p>

            <p>{`Tổng giá trị: $${totalValue}`}</p>

            <button onClick={clearCart}>Clear Cart</button>

          </div>

        </div>

      )}

    </div>

  );

};

export default Cart;

B2.3: **Tích hợp và sử dụng CartContext trong ứng dụng**

Sau khi tạo các component, bạn cần tích hợp chúng vào ứng dụng chính và sử dụng CartProvider để cung cấp context cho các component con.

Lưu ý tạo thư mục: images trong thư mục public và thêm 3 ảnh vào thư mục này.

import React from "react";

import { CartProvider } from "./CartContext";

import DishesList from "./DishesList";

import Cart from "./Cart";

import "./styles.css";

// Sample dishes array

const dishes = [

  {

    id: 0,

    name: "Uthappizza",

    image: "images/uthappizza.png",

    price: "4.99",

    description: "A unique combination of Indian Uthappam and Italian pizza.",

  },

  {

    id: 1,

    name: "Zucchipakoda",

    image: "images/zucchipakoda.png",

    price: "1.99",

    description: "Deep fried Zucchini with chickpea batter.",

  },

  {

    id: 2,

    name: "Vadonut",

    image: "images/vadonut.png",

    price: "1.99",

    description: "A combination of vada and donut.",

  },

  {

    id: 3,

    name: "ElaiCheese Cake",

    image: "images/elaicheesecake.png",

    price: "2.99",

    description: "New York Style Cheesecake with Indian cardamoms.",

  },

];

function App() {

  return (

    <CartProvider>

      <div className="App">

        <DishesList dishes={dishes} />

        <Cart />

      </div>

    </CartProvider>

  );

}

export default App;

**Thêm mới file styles.css**

.dishes {

  display: flex;

  flex-wrap: wrap;

  gap: 16px;

}

.dish-item {

  border: 1px solid #ccc;

  padding: 16px;

  width: 200px;

  text-align: center;

}

.dish-item img {

  width: 100%;

  height: auto;

}

.dish-item button {

  margin-top: 10px;

  padding: 8px 16px;

  background-color: #4caf50;

  color: white;

  border: none;

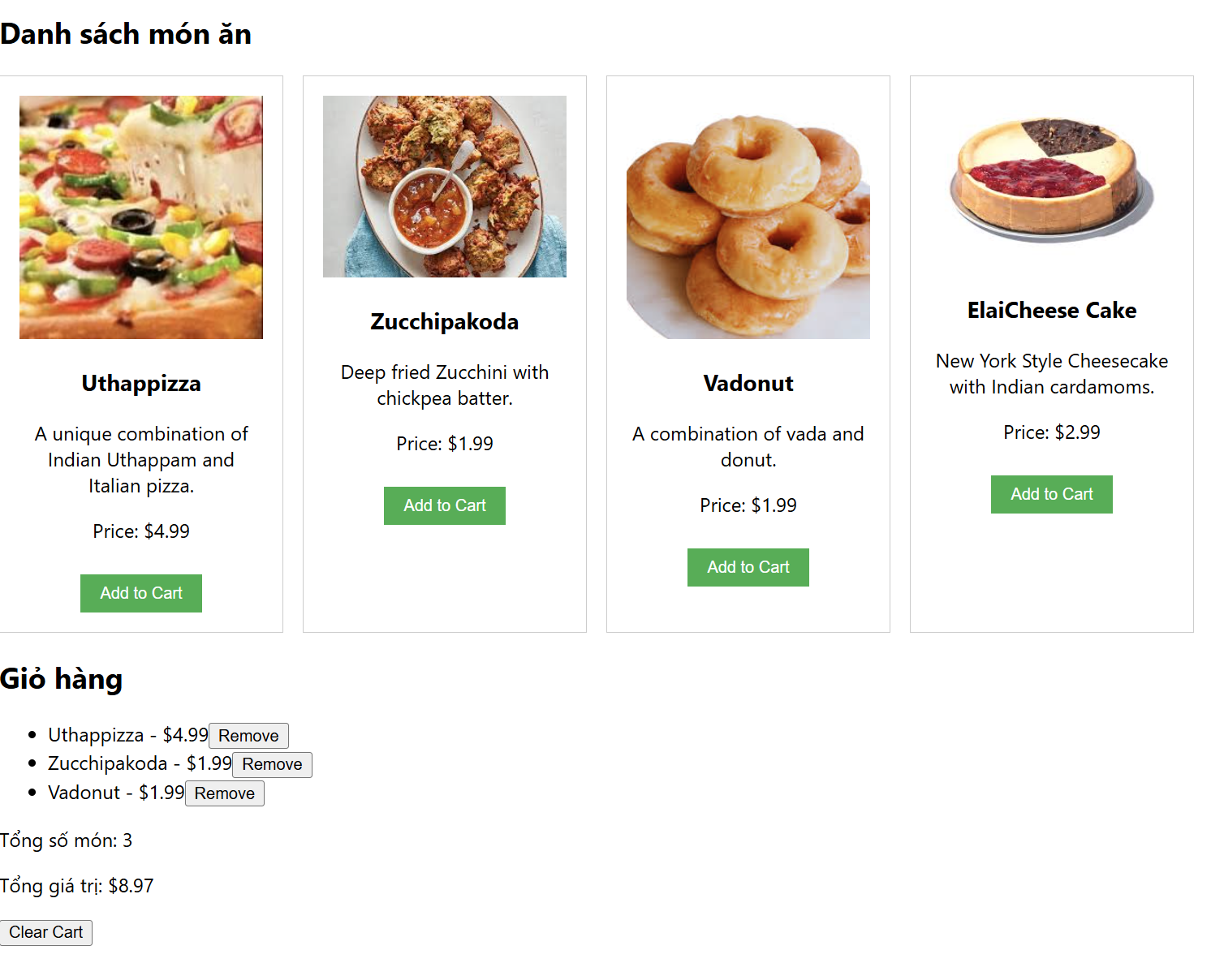
  cursor: pointer;

}

.dish-item button:hover {

  background-color: #45a049;

}

****

Bài tập làm thêm:

* + Cập nhật thêm tính năng “Tìm kiếm món ăn” khi người dùng nhập 1 món ăn vào input để lọc theo tên hoặc mô tả
  + **Thêm tính năng "Xác nhận đơn hàng" và "Thanh toán"**

Người dùng có thể xác nhận đơn hàng và hiển thị thông báo thành công khi đơn Tạo một nút "Xác nhận đơn hàng" để người dùng có thể hoàn tất thanh toán.

Hiển thị một thông báo sau khi thanh toán thành công.

* + Thêm "Chế độ Dark Mode" cho giao diện để giúp người dùng dễ dàng sử dụng: thêm một button để chuyển đổi giao diện sáng và tối dùng useState