Exercise 14: React Hook (useContext)

Objectives and Outcomes

useContext is a React hook that allows you to consume a Context within a functional component. Context provides a way to pass data through the component tree without having to pass props manually at every level. These exercises will help you understand how to use useContext in React

Exercises

1. Create a theme using the useContext hook in React.

const themes = {

light: {

foreground: "#000000",

background: "#eeeeee"

},

dark: {

foreground: "#ffffff",

background: "#61dafb"

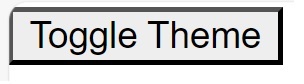
}

};

Based on themes variable, you create ThemeContext, ThemeProvider and Theme component that can change background color of a button when clicked.

**Hint:**

* Create a ThemeContext using createContext and define the themes object with light and dark themes. The ThemeProvider component provides the theme state and toggle function to child components using the ThemeContext.Provider.
* The Theme component uses the useTheme hook to access the theme context and renders a button that changes the theme when clicked. The background color of the button and the container are set dynamically based on the current theme.
* Finally, in the App component, we wrap the Theme component with the ThemeProvider to make the theme context available.



1. Create a simple cart application using useContext in React

When you run your application, the DishesList component will display a list of dishes, and you can add them to the cart by clicking the "Add to Cart" button. The Cart component will show the items in the cart, allow you to remove items, display the total number of items, total value, and provide a "Clear Cart" button to empty the cart.

By using the CartContext and useContext hook, you can manage the cart state and actions across multiple components in your application.

"dishes": [

    {

      "id": 0,

      "name": "Uthappizza",

      "image": "images/uthappizza.png",

      "category": "mains",

      "label": "Hot",

      "price": "4.99",

      "featured": true,

      "description": "A unique combination of Indian Uthappam (pancake) and Italian pizza, topped with Cerignola olives, ripe vine cherry tomatoes, Vidalia onion, Guntur chillies and Buffalo Paneer."

    },

    {

      "id": 1,

      "name": "Zucchipakoda",

      "image": "images/zucchipakoda.png",

      "category": "appetizer",

      "label": "",

      "price": "1.99",

      "featured": false,

      "description": "Deep fried Zucchini coated with mildly spiced Chickpea flour batter accompanied with a sweet-tangy tamarind sauce"

    },

    {

      "id": 2,

      "name": "Vadonut",

      "image": "images/vadonut.png",

      "category": "appetizer",

      "label": "New",

      "price": "1.99",

      "featured": false,

      "description": "A quintessential ConFusion experience, is it a vada or is it a donut?"

    },

    {

      "id": 3,

      "name": "ElaiCheese Cake",

      "image": "images/elaicheesecake.png",

      "category": "dessert",

      "label": "",

      "price": "2.99",

      "featured": false,

      "description": "A delectable, semi-sweet New York Style Cheese Cake, with Graham cracker crust and spiced with Indian cardamoms"

    }]

1. Update the cart count in exercise 2 and value in real-time without refreshing the page

Hint: By updating the state within the CartProvider using setCartItems, the cart count and value will be dynamically recalculated. Then, by accessing the updated values using useContext in the DishesList and Cart components, you can display the updated cart count and value in real-time without refreshing the page.

Conclusion

In conclusion, the useContext hook in React provides a straightforward way to consume a Context within functional components.