NAME: JANHVI SINGH

REGISTRATION NUMBER: 22BCE8544

Assignment 12

App.css

#root {

  max-width: 1280px;

  margin: 0 auto;

  padding: 2rem;

  text-align: center;

}

.logo {

  height: 6em;

  padding: 1.5em;

  will-change: filter;

  transition: filter 300ms;

}

.logo:hover {

  filter: drop-shadow(0 0 2em #646cffaa);

}

.logo.react:hover {

  filter: drop-shadow(0 0 2em #61dafbaa);

}

@keyframes logo-spin {

  from {

    transform: rotate(0deg);

  }

  to {

    transform: rotate(360deg);

  }

}

@media (prefers-reduced-motion: no-preference) {

  a:nth-of-type(2) .logo {

    animation: logo-spin infinite 20s linear;

  }

}

.card {

  padding: 2em;

}

.read-the-docs {

  color: #888;

}

App.jsx

// src/App.jsx

import React from 'react';

import UserDashboard from './UserDashboard'; // Import the UserDashboard component

import './styles.css'; // Import global styles

function App() {

  return (

    <div className="App flex items-center justify-center min-h-screen">

      {/\* Render the UserDashboard component \*/}

      <UserDashboard />

    </div>

  );

}

export default App;

UserDashboardjsx

// src/UserDashboard.jsx

import React, { useState, useEffect } from 'react';

const UserDashboard = () => {

  // Initialize state for name, status, and theme using useState

  // The initial name is "John Doe"

  const [name, setName] = useState("John Doe");

  // The initial status is true (Active)

  const [status, setStatus] = useState(true);

  // The initial theme is loaded from localStorage, or defaults to "light"

  const [theme, setTheme] = useState(() => {

    const savedTheme = localStorage.getItem('userTheme');

    return savedTheme ? savedTheme : 'light';

  });

  // useEffect to handle theme persistence to localStorage

  // This effect runs whenever the 'theme' state changes

  useEffect(() => {

    localStorage.setItem('userTheme', theme); // Save the current theme to localStorage

    // Apply or remove the 'dark' class to the document body

    // This allows global styling based on the theme

    if (theme === 'dark') {

      document.body.classList.add('dark-theme');

    } else {

      document.body.classList.remove('dark-theme');

    }

  }, [theme]); // Dependency array: rerun effect when 'theme' changes

  // Function to handle changes in the name input field

  const handleNameChange = (event) => {

    setName(event.target.value); // Update the name state with the input value

  };

  // Function to toggle the user's status

  const toggleStatus = () => {

    setStatus(prevStatus => !prevStatus); // Toggle the status boolean

  };

  // Function to toggle the theme between 'light' and 'dark'

  const toggleTheme = () => {

    setTheme(prevTheme => (prevTheme === 'light' ? 'dark' : 'light')); // Switch theme

  };

  return (

    // Main dashboard container with dynamic theme class and Tailwind styling

    <div

      className={`dashboard-container ${theme} p-8 rounded-lg shadow-xl max-w-md w-full transition-colors duration-300`}

    >

      <h1 className="text-3xl font-bold mb-6 text-center">User Dashboard</h1>

      {/\* User Information Section \*/}

      <section className="user-info mb-8">

        <h2 className="text-2xl font-semibold mb-4">User Information</h2>

        <div className="mb-4">

          <label htmlFor="name-input" className="block text-sm font-medium mb-1">

            Name:

          </label>

          <input

            id="name-input"

            type="text"

            value={name} // Bind input value to the 'name' state

            onChange={handleNameChange} // Call handler on input change

            className="w-full px-4 py-2 border rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500 transition-colors"

            placeholder="Enter user name"

          />

        </div>

        <div className="mb-4">

          <p className="text-lg">

            Status:{' '}

            <span className={`font-semibold ${status ? 'text-green-600' : 'text-red-600'}`}>

              {status ? 'Active' : 'Inactive'}

            </span>

          </p>

          {/\* Display message if status is inactive \*/}

          {!status && (

            <p className="text-sm text-red-500 mt-2">

              Your account is currently inactive. Please update your status.

            </p>

          )}

        </div>

        <button

          onClick={toggleStatus} // Call handler on button click

          className={`w-full py-2 px-4 rounded-md font-semibold transition-all duration-300 ${

            status

              ? 'bg-red-500 hover:bg-red-600 text-white'

              : 'bg-green-500 hover:bg-green-600 text-white'

          }`}

        >

          {status ? 'Deactivate Account' : 'Activate Account'}

        </button>

      </section>

      {/\* Theme Toggle Section \*/}

      <section className="theme-toggle">

        <h2 className="text-2xl font-semibold mb-4">Theme Preference</h2>

        <p className="text-lg mb-4">

          Current Theme:{' '}

          <span className="font-semibold capitalize">

            {theme}

          </span>

        </p>

        <button

          onClick={toggleTheme} // Call handler on button click

          className={`w-full py-2 px-4 rounded-md font-semibold transition-all duration-300 ${

            theme === 'light'

              ? 'bg-gray-800 hover:bg-gray-700 text-white'

              : 'bg-yellow-500 hover:bg-yellow-600 text-gray-900'

          }`}

        >

          {theme === 'light' ? 'Switch to Dark Mode' : 'Switch to Light Mode'}

        </button>

      </section>

      {/\* Tailwind CSS CDN for styling \*/}

      <script src="https://cdn.tailwindcss.com"></script>

    </div>

  );

};

export default UserDashboard;

User.css

/\* src/styles.css \*/

/\* Global styles for the body and root \*/

html, body, #root, .App {

  height: 100%; /\* Ensure full height \*/

  margin: 0;

  padding: 0;

  font-family: "Inter", sans-serif; /\* Use Inter font \*/

  box-sizing: border-box;

}

/\* Default body theme (light) \*/

body {

  background-color: #f0f2f5; /\* Light grey background \*/

  color: #333; /\* Dark text \*/

  transition: background-color 0.3s ease, color 0.3s ease;

}

/\* Dark theme for the body \*/

body.dark-theme {

  background-color: #1a202c; /\* Dark background \*/

  color: #e2e8f0; /\* Light text \*/

}

/\* Styling for the main dashboard container \*/

.dashboard-container {

  border-radius: 1rem; /\* Rounded corners \*/

  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1); /\* Soft shadow \*/

  transition: background-color 0.3s ease, color 0.3s ease;

}

/\* Light theme for the dashboard container \*/

.dashboard-container.light {

  background-color: #ffffff; /\* White background \*/

  color: #333333; /\* Dark text \*/

}

/\* Dark theme for the dashboard container \*/

.dashboard-container.dark {

  background-color: #2d3748; /\* Darker background \*/

  color: #f8f8f8; /\* Lighter text \*/

}

/\* Input field basic styling \*/

input[type="text"] {

  border: 1px solid #ccc;

  padding: 0.75rem;

  border-radius: 0.5rem;

  font-size: 1rem;

  width: 100%; /\* Full width \*/

}

/\* Focus state for input field \*/

input[type="text"]:focus {

  outline: none;

  border-color: #3b82f6; /\* Blue border on focus \*/

  box-shadow: 0 0 0 3px rgba(59, 130, 246, 0.3); /\* Blue glow on focus \*/

}

/\* Button general styling \*/

button {

  padding: 0.75rem 1.5rem;

  border-radius: 0.5rem;

  font-size: 1rem;

  cursor: pointer;

  border: none;

  outline: none;

  transition: background-color 0.3s ease, transform 0.2s ease;

}

/\* Button hover effect \*/

button:hover {

  transform: translateY(-2px);

}

/\* Specific button styles for status toggle \*/

button.bg-red-500:hover {

  background-color: #dc2626; /\* Darker red on hover \*/

}

button.bg-green-500:hover {

  background-color: #16a34a; /\* Darker green on hover \*/

}

/\* Specific button styles for theme toggle \*/

button.bg-gray-800:hover {

  background-color: #2d3748; /\* Darker gray on hover \*/

}

button.bg-yellow-500:hover {

  background-color: #d97706; /\* Darker yellow on hover \*/

}

/\* Ensuring font Inter is used for all elements \*/

body \* {

  font-family: "Inter", sans-serif;

}

README

React User Dashboard

This project demonstrates state management in a React functional component using the useState hook. It simulates a user dashboard where information like name, status, and theme preference can be dynamically updated based on user interactions.

Assignment Tasks & Implementation Details

How useState is Used

The useState hook is central to managing the dynamic data in this application.

name: A string state initialized to "John Doe". It's managed by useState("John Doe"). The setName function updates this state when the user types in the name input field.

status: A boolean state initialized to true (representing "Active"). It's managed by useState(true). The setStatus function toggles this boolean value when the "Activate/Deactivate Account" button is clicked.

theme: A string state initialized to "light" or a value retrieved from localStorage. It's managed by useState(() => { /\* logic to get from localStorage \*/ }). The setTheme function switches this string between "light" and "dark" when the "Switch Theme" button is clicked.

Dynamic User Interactions

Changing Name:

An <input type="text" /> element is used for the user's name.

The value prop of the input is bound to the name state (value={name}). This creates a controlled component, meaning React controls the input's value.

The onChange event listener (onChange={handleNameChange}) is attached to the input. Whenever the user types, handleNameChange is called, which in turn calls setName(event.target.value), updating the name state.

As soon as name state updates, React re-renders the component, and the displayed name immediately reflects the new input.

Toggling Status:

A button is provided to toggle the user's status.

The button's text dynamically changes between "Deactivate Account" and "Activate Account" based on the current status state (status ? 'Deactivate Account' : 'Activate Account').

An onClick event listener (onClick={toggleStatus}) is attached to the button.

The toggleStatus function uses setStatus(prevStatus => !prevStatus) to flip the boolean value of the status state.

Upon state change, the component re-renders, updating the displayed status text ("Active" or "Inactive") and the button's text.

Additionally, a conditional message "Your account is currently inactive. Please update your status." is displayed when status is false.

Toggling Theme:

A button is provided to switch between light and dark modes.

The button's text dynamically indicates the next theme to switch to (theme === 'light' ? 'Switch to Dark Mode' : 'Switch to Light Mode').

An onClick event listener (onClick={toggleTheme}) is attached.

The toggleTheme function updates the theme state by toggling between "light" and "dark".

A useEffect hook is used to react to changes in the theme state:

It saves the current theme preference to localStorage (localStorage.setItem('userTheme', theme)).

It adds or removes the dark-theme class from the document.body based on the theme state. This global class allows the styles.css file to apply appropriate background and text colors to the entire page.

On initial render, the theme state is initialized by attempting to retrieve a saved preference from localStorage. If no preference is found, it defaults to "light".

Challenges Faced

Initial useEffect for Local Storage: Ensuring the theme is loaded from localStorage only once on component mount, and subsequently saved whenever the theme state changes, required careful use of useEffect with its dependency array and the functional update for useState's initial value.

Applying Global Theme Styles: Deciding whether to apply theme classes to the main dashboard container or to the document.body. Applying it to document.body (using document.body.classList.add/remove) allows for more comprehensive page-wide theme changes, which was a good fit for this scenario, alongside local container styling.

This assignment provided excellent practice in leveraging useState for robust, dynamic UI management, a crucial skill for building interactive React applications.

Screenshot



