Portfolio: Week 7 – React Hook API

**Q1: Reflective Learning on React Hook API** This week, I learned about React Hook APIs, including useState and how it manages state in functional components. I also explored using event handlers like onClick to interact with elements and update the state dynamically. By building the emoji counter, I learned how to create multiple components and pass data as properties (props). It was also interesting to see how modular and reusable React components can be, making the development process clean and organized.

**Q2: EmojeeCounters.js Analysis**

**What is the Name of the Component?** The name of the component is **EmojiCounter**.

**Identify the Line of Code That Uses the EmojiCounter in index.js:**

<EmojiCounter type="Love" />

<EmojiCounter type="Like" />

<EmojiCounter type="Sad" />

**States Declared in EmojeeCounters.js:** The state is declared using useState to keep track of the click count.

const [count, setCount] = useState(0);

* count is the variable that holds the current value.
* setCount is the function used to update count.

**Lines of Code That Associate the Event Handler:** The onClick event handler is associated with the button to update the count:

<button onClick={handleClick}>

**Explain <EmojiCounter pic='Love' />:**

* pic='Love' is a **prop** passed to the EmojiCounter component.
* It allows the component to dynamically render the correct emoji image based on the value of pic.

**What is useEffect and Why It’s Used?** useEffect is a hook used to handle **side effects** in functional components, such as updating the DOM, fetching data, or setting up subscriptions.

* In this case, useEffect can be used to log changes in state or perform updates whenever count changes.

**Explanation of Return Statement in EmojiCounter.js:**

return (

    <div className="App">

      <p>

        {props.pic}

        <button onClick={ClickHandle}>

          {count}

          <img src={pic} alt="" />

        </button>

      </p>

    </div>

  );

* <div className="App"> wraps the component for styling purposes.
* {props.pic} displays the emoji name (e.g., "Love").
* <button onClick={ClickHandle}> handles the click event to increment count.
* {count} dynamically shows the current value of the counter.
* <img src={pic} alt="" /> displays the emoji image based on the pic prop.

**Q3: Code for Component With Text Box and Label**

**Code: TextImageDisplay.js** Here’s a functional component that displays an image based on user input:

import React, { useState } from "react";

import Happy from "./Happy.png";

import Like from "./Like.png";

import Sad from "./Sad.png";

function TextImageDisplay() {

  const [input, setInput] = useState(""); // To store text box input

  const [image, setImage] = useState(""); // To store the corresponding image

  const handleInputChange = (e) => {

    const value = e.target.value.toLowerCase();

    setInput(value);

    // Update the image based on input

    if (value === "happy") setImage(Happy);

    else if (value === "like") setImage(Like);

    else if (value === "sad") setImage(Sad);

    else setImage(""); // Clear image if input doesn’t match

  };

  return (

    <div className="App">

      <h1>Dynamic Image Display</h1>

      <input

        type="text"

        placeholder="Type 'Happy', 'Like', or 'Sad'"

        value={input}

        onChange={handleInputChange}

      />

      <label>

        {image && <img src={image} alt={input} style={{ width: "100px" }} />}

      </label>

    </div>

  );

}

export default TextImageDisplay;

**Code in index.js:**

import React from "react";

import ReactDOM from "react-dom/client";

import "./App.css";

import TextImageDisplay from "./TextImageDisplay";

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

root.render(

  <React.StrictMode>

    <TextImageDisplay />

  </React.StrictMode>

);

**How I Developed This Component:**

* I created a functional component called TextImageDisplay that uses the useState hook to manage input and images dynamically.
* I added an input box to accept user input and updated the state whenever the input value changes.
* I compared the input value to predefined conditions ("happy", "like", "sad") and updated the image state to display the relevant emoji image.
* I used a label to show the image dynamically with the <img> tag.

**Screenshot of the Output:**

* **Input:** Happy → Displays Happy face image.
* **Input:** Like → Displays Like emoji.
* **Input:** Sad → Displays Sad emoji.

A screenshot of a computer

Description automatically generated