**Batch: A-3 Roll No.: 16010122104**

**Experiment / assignment / tutorial No. 2**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| Title: Implementation of React Hooks. |

**AIM:** To Implement the React Hooks

**Problem Definition:**

To demonstrate the working of react hooks based on the following points and Apply this on assigned programming task

* useState
* useEffect
* useContext
* **useReducer**:
* useCallback
* useMemo

\*(Students have to perform the task assigned within group and demonstrate the same).

**Resources used:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**CO 1:**.Build full stack applications in JavaScript using the MERN technologies.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Shelly Powers Learning Node O’ Reilly 2 nd Edition, 2016.

**Pre Lab/ Prior Concepts:**

**Write details about the following content**

* useState
* useEffect
* useContext
* **useReducer**:
* useCallback
* useMemo

**1. useState**

**- Purpose:**

- `useState` is a React Hook that allows you to add state to functional components. It returns a state variable and a function to update that state.

**- Usage:**

- You call `useState` inside a functional component, passing the initial state as an argument. The hook returns an array with two elements: the current state and a function to update it.

**- Example:**

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

**- When to Use:**

- Use `useState` when you need to track simple data that changes over time, like form inputs, toggles, or counters.

**2. useEffect**

**- Purpose:**

- `useEffect` is a React Hook that allows you to perform side effects in your components, such as data fetching, subscriptions, or manually changing the DOM.

**- Usage:**

- You call `useEffect` in a functional component, passing a function (effect) as the first argument. This effect runs after every render by default. You can also pass a dependency array as the second argument to control when the effect runs.

**- Example:**

useEffect(() => {

document.title = `You clicked ${count} times`;

}, [count]);

**- The effect runs only when `count` changes.**

**- When to Use:**

- Use `useEffect` for tasks like fetching data, setting up subscriptions, or manually updating the DOM after a render.

**3. useContext**

**- Purpose:**

- `useContext` is a React Hook that allows you to access the value of a context directly in your functional components, bypassing the need to use the Context Consumer.

**- Usage:**

- You call `useContext` and pass the context object to it. The hook returns the current context value.

**- Example:**

const theme = useContext(ThemeContext);

**- `theme` would contain the value provided by `ThemeContext.Provider`.**

**- When to Use:**

- Use `useContext` when you need to consume a context value without wrapping your component in a Consumer component.

**4. useReducer**

**- Purpose:**

- `useReducer` is a React Hook that is used for state management in a component. It is an alternative to `useState` and is more suitable for complex state logic involving multiple sub-values or when the next state depends on the previous one.

**- Usage:**

- You call `useReducer` with two arguments: a reducer function and an initial state. The hook returns the current state and a dispatch function.

**- Example:**

const [state, dispatch] = useReducer(reducer, initialState);

**- `state` is the current state.**

**- `dispatch` is a function to send actions to the reducer.**

**- When to Use:**

- Use `useReducer` when you have complex state logic, such as multiple related state variables or when managing state transitions based on actions.

**5. useCallback**

**- Purpose:**

- `useCallback` is a React Hook that returns a memoized version of the callback function you pass to it, which only changes if one of the dependencies has changed.

**- Usage:**

- You call `useCallback` and pass a function and an array of dependencies. The hook returns a memoized version of the function.

**- Example:**

const memoizedCallback = useCallback(() => {

doSomething(a, b);

}, [a, b]);

**- The `memoizedCallback` function will only change if `a` or `b` changes.**

**- When to Use:**

- Use `useCallback` to avoid unnecessary re-renders or re-creation of functions, especially in cases where functions are passed as props to child components.

**6. useMemo**

**- Purpose:**

- `useMemo` is a React Hook that returns a memoized value. It computes the value only when one of the dependencies changes, thus avoiding expensive calculations on every render.

**- Usage:**

- You call `useMemo` with a function that computes the value and an array of dependencies. The hook returns the memoized value.

**- Example:**

const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);

**- The `memoizedValue` will only be recomputed if `a` or `b` changes.**

**- When to Use:**

- Use `useMemo` for optimizing performance in components where expensive calculations are performed, and the result is needed across multiple renders without changes.

**Methodology:**

**User Profile Editor:**

import React, { useState } from "react";

function UserProfileEditor() {

  const initialProfile = { name: "", age: "", email: "" };

  const [profile, setProfile] = useState(initialProfile);

  const [showForm, setShowForm] = useState(true);

  const handleChange = (*e*) => {

    setProfile({ ...profile, [*e*.target.name]: *e*.target.value });

  };

  const handleReset = () => {

    setProfile(initialProfile);

  };

  return (

    <div>

      <button onClick={() => setShowForm(!showForm)}>

        {showForm ? "Hide Form" : "Show Form"}

      </button>

      {showForm && (

        <center>

          <form>

            <div>

              <label>Name:</label>

              <input type="text" name="name" value={profile.name} onChange={handleChange} />

            </div>

            <div>

              <label>Age:</label>

              <input type="text" name="age" value={profile.age} onChange={handleChange} />

            </div>

            <div>

              <label>Email:</label>

              <input type="email" name="email" value={profile.email} onChange={handleChange} />

            </div>

            <button type="button" onClick={handleReset}>Reset</button>

          </form>

        </center>

      )}

      <div>

        <center>

          <h3>Entered Information:</h3>

          <p>Name: {profile.name}</p>

          <p>Age: {profile.age}</p>

          <p>Email: {profile.email}</p>

        </center>

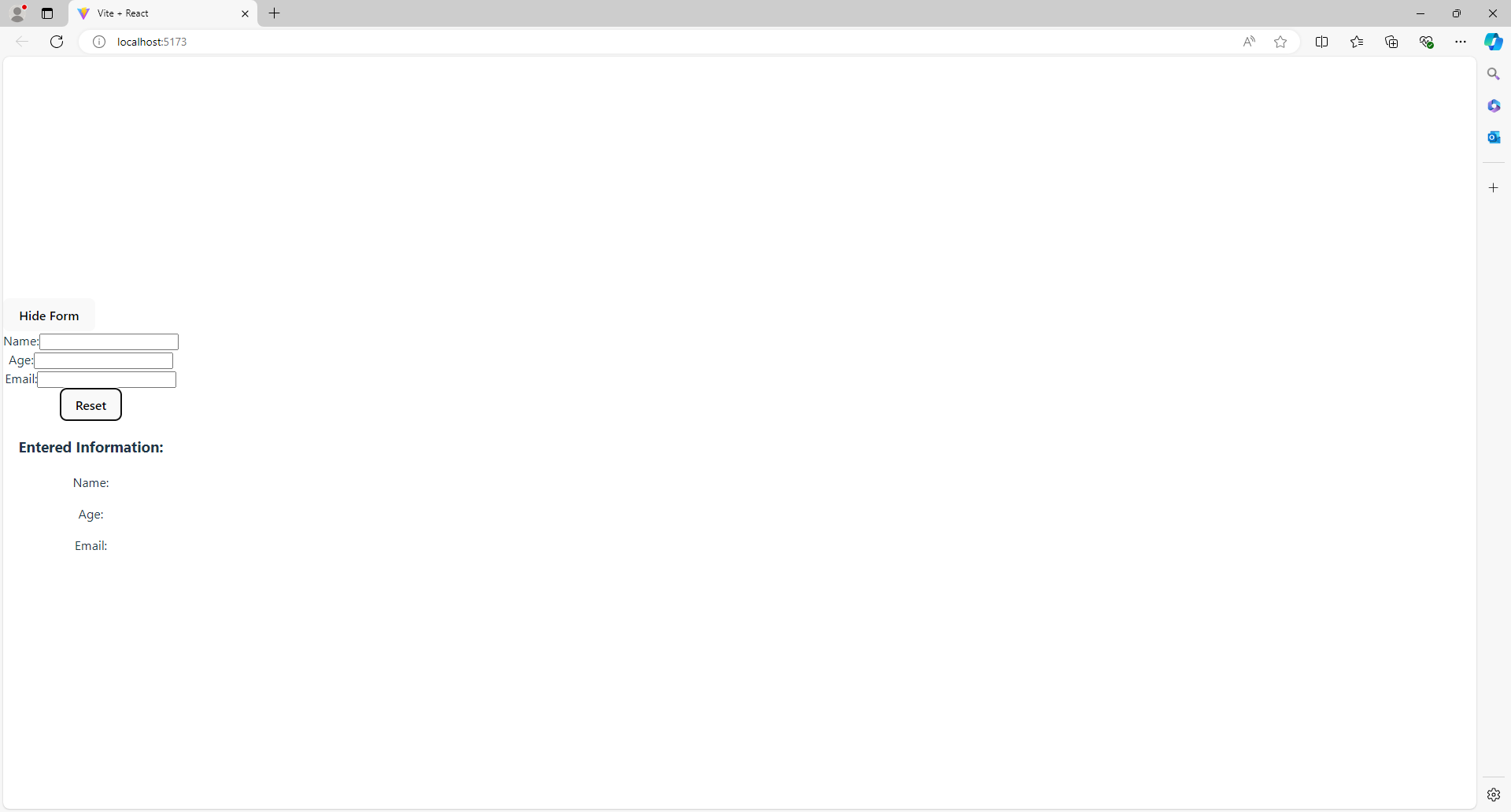
      </div>

    </div>

  );

}

export default UserProfileEditor;



**Task Manager:**

import React, { useState } from "react";

function TaskManager() {

  const [tasks, setTasks] = useState([]);

  const [task, setTask] = useState("");

  const [showCompleted, setShowCompleted] = useState(true);

  const addTask = () => {

    setTasks([...tasks, { text: task, completed: false }]);

    setTask("");

  };

  const toggleTaskCompletion = (*index*) => {

    const newTasks = tasks.map((*task*, *i*) =>

*i* === *index* ? { ...*task*, completed: !*task*.completed } : *task*

    );

    setTasks(newTasks);

  };

  const resetTasks = () => {

    setTasks([]);

  };

  return (

    <div>

      <input

        type="text"

        value={task}

        onChange={(*e*) => setTask(*e*.target.value)}

        placeholder="Add a new task"

      />

      <button onClick={addTask}>Add Task</button>

      <button onClick={() => setShowCompleted(!showCompleted)}>

        {showCompleted ? "Hide Completed" : "Show Completed"}

      </button>

      <button onClick={resetTasks}>Reset Tasks</button>

      <ul>

        {tasks

          .filter((*task*) => (showCompleted ? true : !*task*.completed))

          .map((*task*, *index*) => (

            <li key={*index*}>

              <span

                style={{

                  textDecoration: *task*.completed ? "line-through" : "none",

                }}

              >

                {*task*.text}

              </span>

              <button onClick={() => toggleTaskCompletion(*index*)}>

                {*task*.completed ? "Undo" : "Complete"}

              </button>

            </li>

          ))}

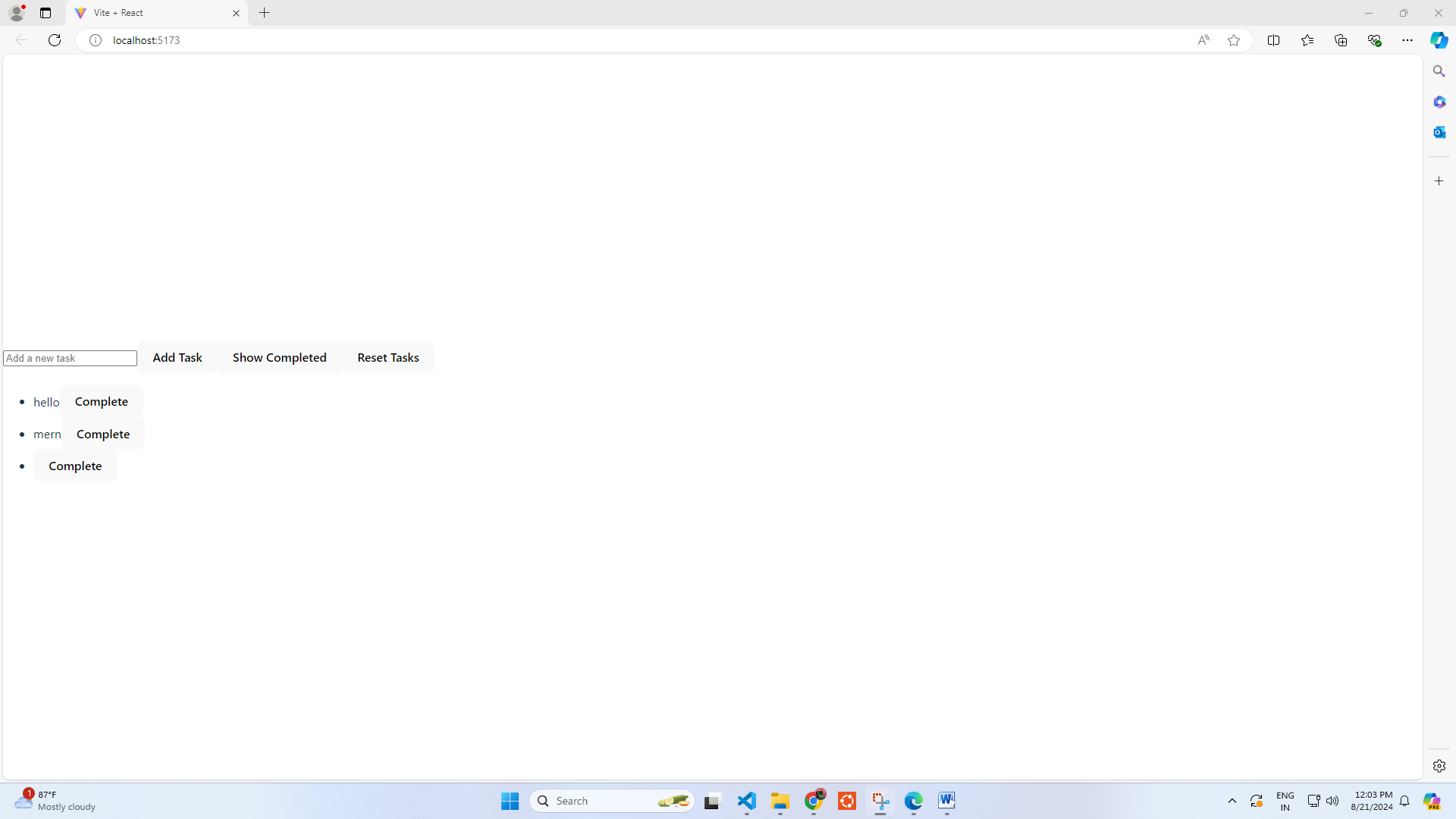
      </ul>

    </div>

  );

}

export default TaskManager;

****

**User Profile Manager:**

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

function UserProfileManager() {

  const initialProfile = { name: "", age: "", email: "" };

  const [profile, setProfile] = useState(initialProfile);

  useEffect(() => {

*// Simulate fetching initial profile data*

    const fetchProfile = () => {

      const initialData = { name: "John Doe", age: "30", email: "john.doe@example.com" };

      setProfile(initialData);

    };

    fetchProfile();

  }, []);

  const handleChange = (*e*) => {

    setProfile({ ...profile, [*e*.target.name]: *e*.target.value });

  };

  const handleSubmit = (*e*) => {

*e*.preventDefault();

    console.log("Updated profile data:", profile);

  };

  return (

    <div>

      <form onSubmit={handleSubmit}>

        <div>

          <label>Name:</label>

          <input type="text" name="name" value={profile.name} onChange={handleChange} />

        </div>

        <div>

          <label>Age:</label>

          <input type="text" name="age" value={profile.age} onChange={handleChange} />

        </div>

        <div>

          <label>Email:</label>

          <input type="email" name="email" value={profile.email} onChange={handleChange} />

        </div>

        <button type="submit">Update Profile</button>

      </form>

      <div>

        <h3>Profile Information:</h3>

        <p>Name: {profile.name}</p>

        <p>Age: {profile.age}</p>

        <p>Email: {profile.email}</p>

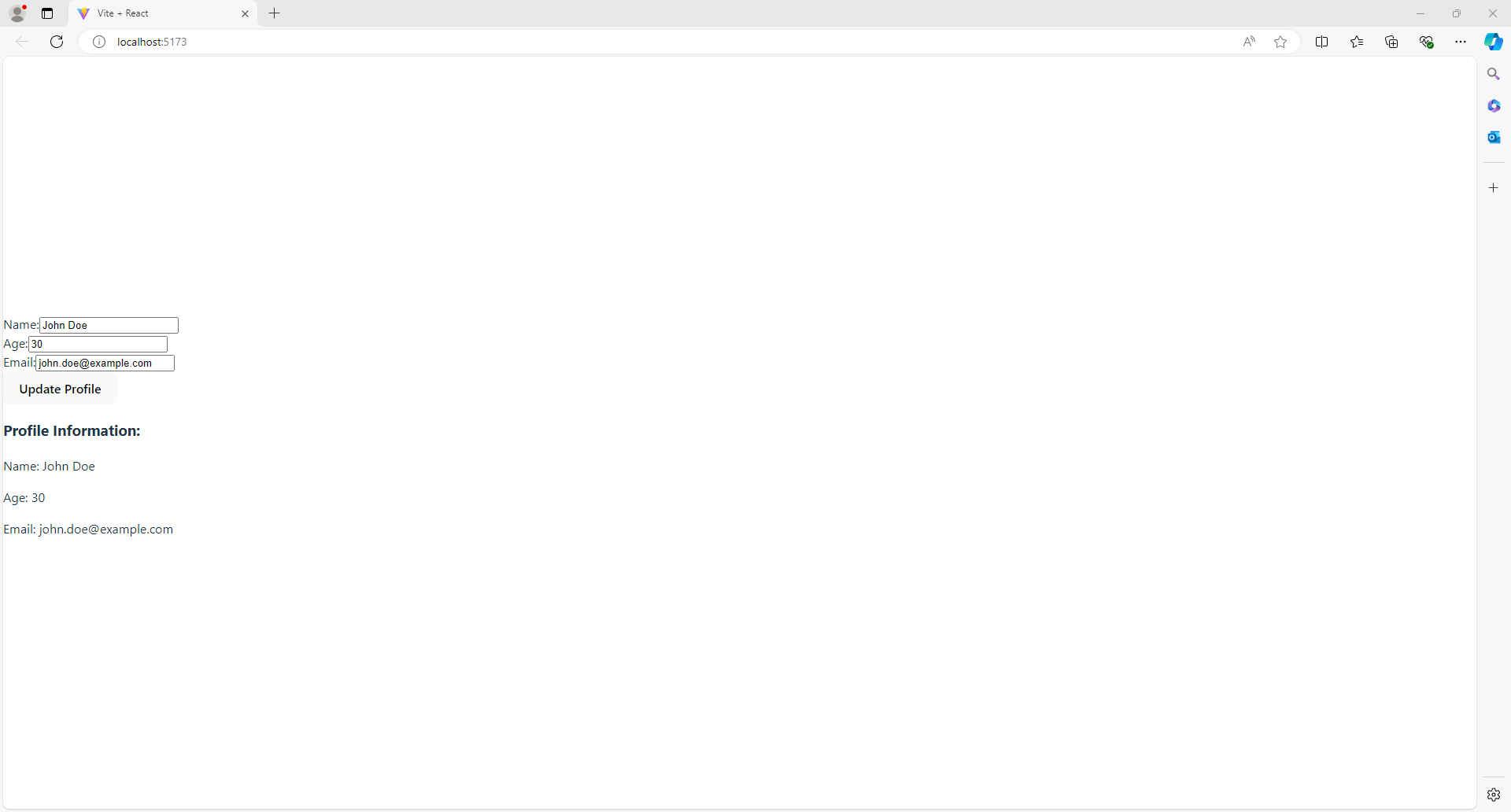
      </div>

    </div>

  );

}

export default UserProfileManager;

****

**Implementation Details:**

**Task: User Profile Editor**

**Implementation Details:**

1. **Form Creation:**
   * Create a form with input fields for the user's name, age, and email.
   * Use React's useState hook to manage the state of each input field.
2. **Displaying Entered Information:**
   * After the user fills in the form, the entered information is displayed below the form.
   * Bind the input fields to the state variables and display the values in a separate section below the form.
3. **Toggle Form Visibility:**
   * Add a button that toggles the visibility of the entire form. Use useState to manage the visibility state.
4. **Reset Form Fields:**
   * Add a button to reset the form fields to their initial values. This can be done by resetting the state variables that hold the input values.

**Steps of Execution:**

1. Render the form with input fields for name, age, and email.
2. As the user types in the fields, update the state variables with the current input values.
3. Display the current state values below the form.
4. Implement a button to toggle the visibility of the form by changing the visibility state.
5. Implement a reset button that resets the state variables to their initial values.

**Task: Task Manager**

**Implementation Details:**

1. **Input Field for New Task:**
   * Create an input field to allow users to add new tasks.
   * Use useState to manage the task list and input field.
2. **List to Display Added Tasks:**
   * Use a map function to display each task in a list format.
   * Store tasks in an array and update the state as new tasks are added.
3. **Mark Tasks as Complete:**
   * Add a checkbox or button next to each task to mark it as complete.
   * Use useState to manage the completed state of each task.
4. **Toggle Visibility of Completed Tasks:**
   * Implement a button that toggles the visibility of completed tasks.
   * Use useState to control the visibility of completed tasks.
5. **Reset Task List:**
   * Add a button to clear all tasks and reset the task list.

**Steps of Execution:**

1. Create an input field where users can type in new tasks.
2. Use a state variable to store and display the list of tasks.
3. Add functionality to mark tasks as complete and update the task list accordingly.
4. Implement a toggle button to show or hide completed tasks based on their completion status.
5. Provide a reset button to clear the task list and reset the state.

**Task: User Profile Manager**

**Implementation Details:**

1. **Form Creation:**
   * Create a form with fields for the user's name, age, and email.
   * Use useState to manage the state of each input field.
2. **Fetching Initial Profile Data:**
   * Use useEffect to fetch the initial profile data when the component mounts.
   * Populate the form fields with the fetched data.
3. **Updating Profile Information:**
   * Allow users to update their profile information by modifying the input fields.
   * Use useState to update the profile state as the user makes changes.
4. **Display Updated Information:**
   * Display the updated profile information below the form after the user submits the changes.
5. **Logging Updates:**
   * Use useEffect to log a message to the console whenever the profile data is updated.

**Steps of Execution:**

1. Render the form with input fields for name, age, and email.
2. Fetch initial profile data when the component mounts and populate the form fields.
3. As the user updates the form fields, update the state variables with the new input values.
4. Display the updated profile information below the form upon submission.
5. Implement a logging mechanism to log updates to the console whenever the profile data changes.

**Conclusion:**

**Learnt to implement react hooks.**

**Postlab questions:**

1)