| | |
|---|---|
| **Batch:** D-2 | **Roll No.:** 16010122151 |
| **Experiment / assignment / tutorial No.____** | |
| **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:** Manages state in functional components.
- **Usage:** `const [state, setState] = useState(initialState);`
- **Example:** `const [count, setCount] = useState(0);`

### 2. useEffect

- **Purpose:** Handles side effects such as data fetching, subscriptions, or DOM updates.
- **Usage:** `useEffect(() => { /* effect */ }, [dependencies]);`
- **Example:** `useEffect(() => { document.title = "Hello"; }, []);`

### 3. useContext

- **Purpose:** Accesses the value from a React context.
- **Usage:** `const value = useContext(MyContext);`
- **Example:** `const theme = useContext(ThemeContext);`

### 4. useReducer

- **Purpose:** Manages complex state logic with a reducer function.
- **Usage:** `const [state, dispatch] = useReducer(reducer, initialState);`
- **Example:** `const [state, dispatch] = useReducer(reducer, { count: 0 });`

### 5. useCallback

- **Purpose:** Memoizes callback functions to prevent unnecessary re-renders.
- **Usage:** `const memoizedCallback = useCallback(() => { /* callback */ }, [dependencies]);`

- **Example:** `const handleClick = useCallback(() => { console.log('clicked'); }, []);`

## 6. useMemo

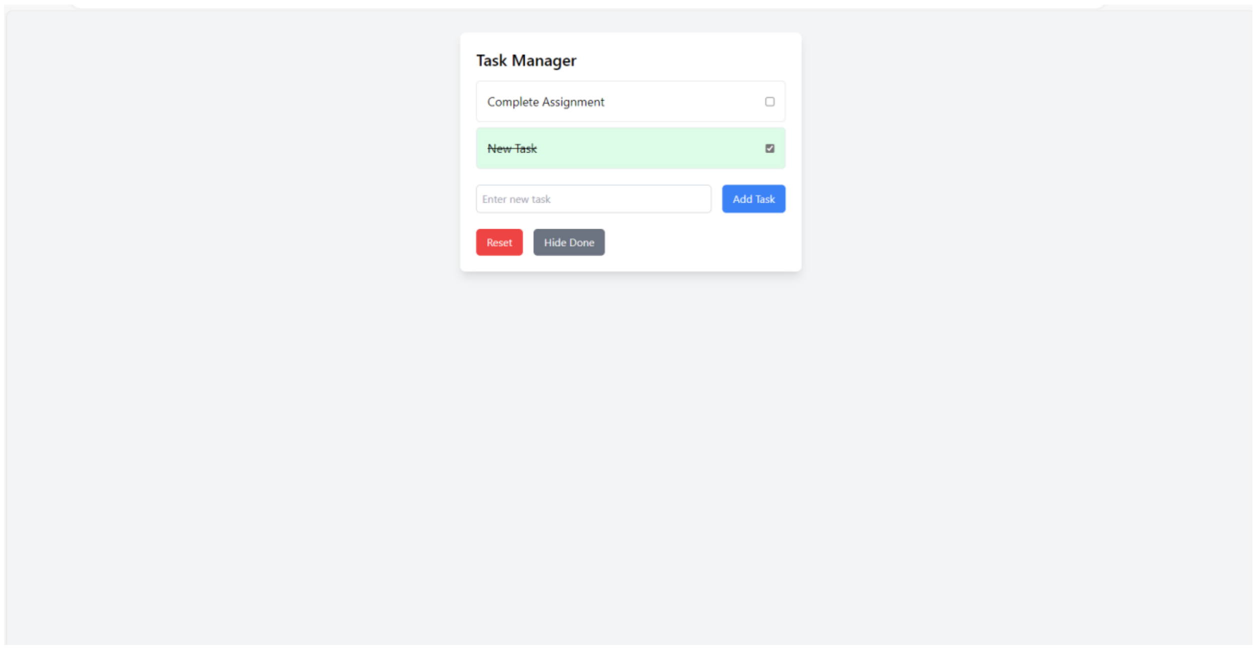
- **Purpose:** Memoizes computed values to optimize performance.
- **Usage:** `const memoizedValue = useMemo(() => computeValue(), [dependencies]);`
- **Example:** `const memoizedValue = useMemo(() => expensiveCalculation(), [input]);`

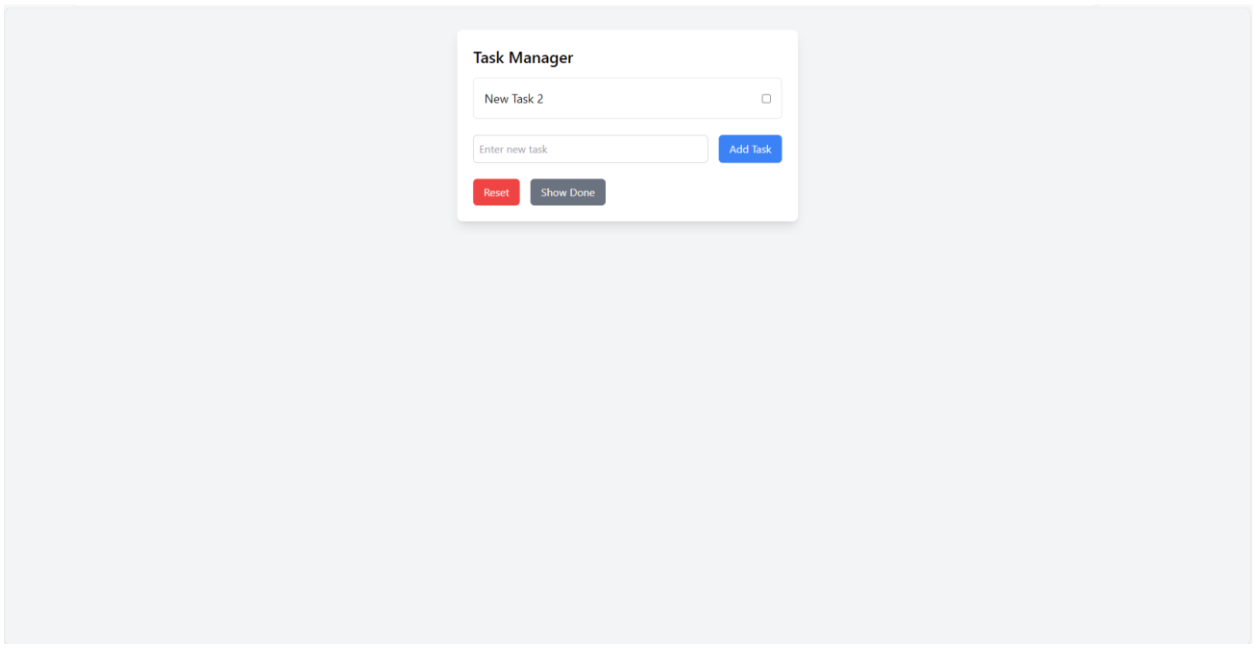
**Implementation Details:**

## Task: Task Manager

1. **Create a task manager with the following features:**
   - **An input field to add a new task.**
   - **A list to display added tasks.**
   - **A button to mark tasks as complete.**
   - **A button to toggle the visibility of completed tasks.**
   - **A button to reset the task list.**

```
import { useState } from 'react';
import './App.css';

function App() {
  const [tasks, setTasks] = useState([
    {
      task: 'Complete Assignment',
      done: false,
    },
  ]);

  const [taskD, setTasksD] = useState([]);
  const [newTask, setNewTask] = useState('');
  const [hide, setHide] = useState(false);

  const handleCheckboxChange = (index) => {
    const updatedTasks = tasks.map((task, i) =>
      i === index ? { ...task, done: !task.done } : task
    );

    const completedTask = tasks[index];
    completedTask.done = !completedTask.done;

    setTasks(updatedTasks);
    setTasksD(
      completedTask.done
        ? [...taskD, completedTask]
        : taskD.filter((_, i) => i !== index)
    );
  };

  const handleAddTask = (e) => {
    e.preventDefault();
    if (newTask.trim() === '') return;
    setTasks([...tasks, { task: newTask, done: false }]);
    setNewTask('');
  };

  return (
    <div className="min-h-screen bg-gray-100 p-8">
      <div className="container mx-auto max-w-lg bg-white p-6 rounded-lg shadow-lg">
        <h1 className="text-2xl font-semibold mb-4">Task Manager</h1>

        <form onSubmit={(e) => e.preventDefault()} className="mb-6">
          <div className="space-y-2">
            {tasks.map((task, index) => (
              !hide || !task.done ? (
                <div
                  key={index}
                  className={`flex items-center justify-between p-4 border rounded-md
mb-2 ${
                    task.done ? 'bg-green-100 line-through' : 'bg-white'
                  }`}
                >
                  <h2 className="text-lg">{task.task}</h2>
                  <input
                    type="checkbox"
                    checked={task.done}
                    onChange={() => handleCheckboxChange(index)}
                    className="form-checkbox"
                  />
                </div>
              ) : null
            ))}
          </div>
        </form>

        <form onSubmit={handleAddTask} className="flex gap-4 mb-6">
          <input
            type="text"
            id="task"
            className="flex-1 p-2 border border-gray-300 rounded-md"
            placeholder="Enter new task"
            value={newTask}
            onChange={(e) => setNewTask(e.target.value)}
          />
          <button
            type="submit"
            className="bg-blue-500 text-white px-4 py-2 rounded-md hover:bg-blue-600"
          >
            Add Task
          </button>
        </form>

        <div className="flex gap-4">
          <button
            className="bg-red-500 text-white px-4 py-2 rounded-md hover:bg-red-600"
            onClick={() =>{
              setTasks([]);
              setTasksD([])
            }}
          >
            Reset
          </button>
          <button
            className="bg-gray-500 text-white px-4 py-2 rounded-md hover:bg-gray-600"
            onClick={() => setHide(!hide)}
          >
            {hide ? 'Show Done' : 'Hide Done'}
          </button>
        </div>
      </div>
    </div>
  );
}

export default App;
```


Task Manager
Complete Assignment
~~New Task~~
Enter new task
Add Task
Reset
Hide Done


Task Manager
New Task 2
Enter new task
Add Task
Reset
Show Done

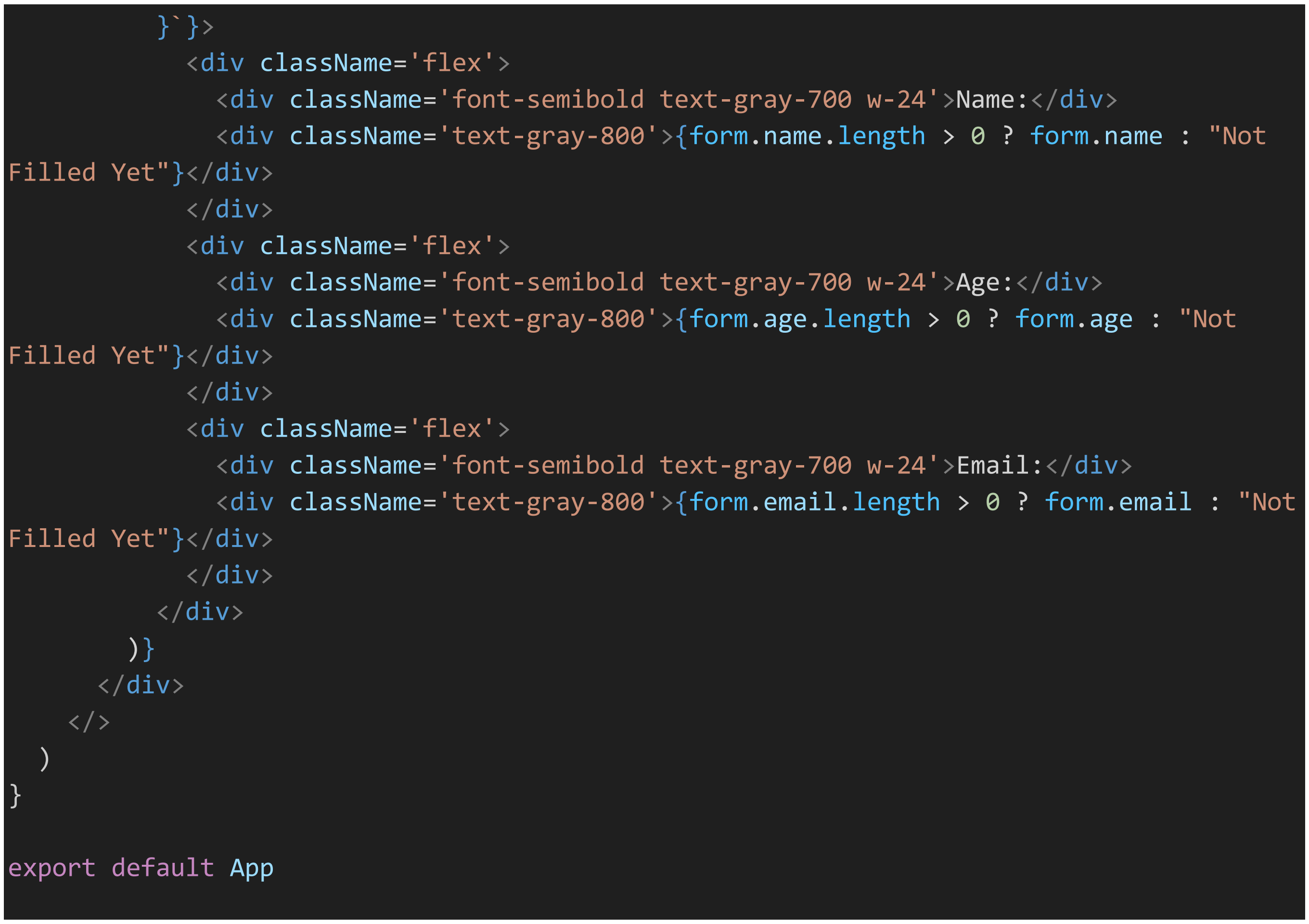
## Task: User Profile Editor

1. **Create a form with fields for the user's name, age, and email.**
2. **Display the entered information below the form.**
3. **Add a button to toggle the visibility of the form.**
4. **Add a button to reset the form fields to their initial values.**

```
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'

function App() {
  const [form, setForm] = useState({
    name: "",
    age: "",
    email: ""
  })

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

  return (
    <>
      <div className='w-full h-screen flex justify-center items-center flex-col bg-
gray-100 p-6'>
       {showForm &&  <form className='w-full max-w-md bg-white p-8 rounded-lg shadow-lg
space-y-6' onSubmit={(e) => {
          e.preventDefault();
          if(form.name.length>0 && form.email.length>0 && form.age.length>0  )
          setShow(true)
        }}>
          <div>
            <label htmlFor="Name" className='block text-lg font-semibold text-gray-700
mb-1'>Name</label>
            <input
```

```
              type="text"
              name="name"
              id="name"
              value={form.name}
              className='w-full p-3 border-2 border-gray-300 rounded-lg focus:border-
blue-500'
              onChange={(e) => setForm({ ...form, name: e.target.value })}
            />
          </div>

          <div>
            <label htmlFor="Age" className='block text-lg font-semibold text-gray-700
mb-1'>Age</label>
            <input
              type="number"
              name="age"
              id="age"
              value={form.age}
              className='w-full p-3 border-2 border-gray-300 rounded-lg focus:border-
blue-500'
              onChange={(e) => setForm({ ...form, age: e.target.value })}
            />
          </div>

          <div>
            <label htmlFor="Email" className='block text-lg font-semibold text-gray-700
mb-1'>Email</label>
            <input
              type="email"
              name="email"
              id="email"
              value={form.email}
              className='w-full p-3 border-2 border-gray-300 rounded-lg focus:border-
blue-500'
              onChange={(e) => setForm({ ...form, email: e.target.value })}
            />
          </div>

          <div className='flex gap-4'>
            <button
              type="submit"
              className='bg-blue-500 text-white py-2 px-4 rounded-lg hover:bg-blue-600
focus:outline-none focus:ring-2 focus:ring-blue-500 focus:ring-opacity-50'
```

```
            >
              Submit
            </button>
            <button
              type="button"
              className='bg-gray-500 text-white py-2 px-4 rounded-lg hover:bg-gray-600
focus:outline-none focus:ring-2 focus:ring-gray-500 focus:ring-opacity-50'
              onClick={() => {
                setShow(false)
                setForm({ name: "", age: "", email: "" })
            }}

            >
              Reset
            </button>
          </div>
        </form> }

        <div className='mt-6'>
          <button
            onClick={() => setShow(!show)}
            className='bg-indigo-500 text-white py-2 px-4 rounded-lg hover:bg-indigo-
600 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:ring-opacity-50'
          >
            Check the filled data
          </button>
        </div>

        {show && <div className='mt-6'>
          <button
            onClick={() => setShowForm(!showForm)}
            className='bg-indigo-500 text-white py-2 px-4 rounded-lg hover:bg-indigo-
600 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:ring-opacity-50'
          >
            Toggle form
          </button>
        </div>}


        {show && (
          <div className={`mt-6 p-6 w-full max-w-md bg-white rounded-lg shadow-lg
space-y-4 transition-opacity duration-500 ${
            show ? "popup-enter" : "popup-exit opacity-0"
```



```
          }`}>
            <div className='flex'>
              <div className='font-semibold text-gray-700 w-24'>Name:</div>
              <div className='text-gray-800'>{form.name.length > 0 ? form.name : "Not
Filled Yet"}</div>
            </div>
            <div className='flex'>
              <div className='font-semibold text-gray-700 w-24'>Age:</div>
              <div className='text-gray-800'>{form.age.length > 0 ? form.age : "Not
Filled Yet"}</div>
            </div>
            <div className='flex'>
              <div className='font-semibold text-gray-700 w-24'>Email:</div>
              <div className='text-gray-800'>{form.email.length > 0 ? form.email : "Not
Filled Yet"}</div>
            </div>
          </div>
        )}
      </div>
    </>
  )
}

export default App
```

## Task: User Profile Manager

1. **Create a form with fields for the user's name, age, and email.**
2. **Fetch initial profile data when the component mounts.**
3. **Display the fetched profile data in the form fields.**
4. **Allow the user to update their profile information.**
5. **Display the updated profile information below the form.**
6. **Log a message to the console whenever the profile data is updated.**

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

function App() {

  const [profile, setProfile] = useState({ name: '', age: '', email: '' });
  const [updatedProfile, setUpdatedProfile] = useState(null);


  useEffect(() => {

    const fetchProfile = async () => {
      const data = await new Promise((resolve) =>
        setTimeout(() => resolve({ name: 'Minav Karia', age: '30', email: 'minav.karia
@example.com' }), 1000)
      );
      setProfile(data);
    };

    fetchProfile();
  }, []);


  const handleChange = (e) => {
    const { name, value } = e.target;
    setProfile((prevProfile) => ({
      ...prevProfile,
      [name]: value,
    }));
  };
  const handleSubmit = (e) => {
    e.preventDefault();
    setUpdatedProfile(profile);
    console.log('Profile updated:', profile);
  };

  return (
    <div className="container mx-auto p-6 max-w-lg bg-white shadow-md rounded">
      <h1 className="text-2xl font-bold mb-4">User Profile</h1>

      <form onSubmit={handleSubmit} className="space-y-4">
        <div>
          <label htmlFor="name" className="block text-sm font-medium text-gray-
700">Name</label>
          <input
            type="text"
            id="name"
            name="name"
            value={profile.name}
            onChange={handleChange}
            className="mt-1 block w-full px-3 py-2 border border-gray-300 rounded-md
shadow-sm focus:outline-none focus:ring-indigo-500 focus:border-indigo-500 sm:text-sm"
          />
        </div>

        <div>
          <label htmlFor="age" className="block text-sm font-medium text-gray-
700">Age</label>
          <input
            type="text"
            id="age"
            name="age"
            value={profile.age}
            onChange={handleChange}
            className="mt-1 block w-full px-3 py-2 border border-gray-300 rounded-md
shadow-sm focus:outline-none focus:ring-indigo-500 focus:border-indigo-500 sm:text-sm"
          />
        </div>

        <div>
          <label htmlFor="email" className="block text-sm font-medium text-gray-
700">Email</label>
          <input
```

```
            type="email"
            id="email"
            name="email"
            value={profile.email}
            onChange={handleChange}
            className="mt-1 block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none focus:ring-indigo-500 focus:border-indigo-500 sm:text-sm"
          />
        </div>

        <button
          type="submit"
          className="bg-blue-500 text-white px-4 py-2 rounded-md hover:bg-blue-600"
        >
          Update Profile
        </button>
      </form>

      {updatedProfile && (
        <div className="mt-6 p-4 border border-gray-200 rounded-md bg-gray-50">
          <h2 className="text-xl font-semibold mb-2">Updated Profile</h2>
          <p><strong>Name:</strong> {updatedProfile.name}</p>
          <p><strong>Age:</strong> {updatedProfile.age}</p>
          <p><strong>Email:</strong> {updatedProfile.email}</p>
        </div>
      )}
    </div>
  );
}

export default App;
```

.

**Conclusion:**

We learned about hooks and how to implement it with a use case like to list, user manager profile and editor