**Batch: TY A2 Roll No.: 16010122041**

**Experiment / assignment / tutorial No.\_\_3\_\_**

**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**

The useState hook is a fundamental React hook that allows you to add state to functional components. It takes the initial state as an argument and returns an array containing the current state and a function to update it.

**Syntax:**

const [state, setState] = useState(initialState);

**Example:**

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

In this example, count is the current state, and setCount is the function used to update it.

**2. useEffect**

The useEffect hook lets you perform side effects in functional components, like fetching data, setting up subscriptions, or manually changing the DOM. By default, it runs after every render, but you can control when it runs by passing dependencies.

**Syntax:**

useEffect(() => {

// Your code here (e.g., fetch data)

}, [dependencies]);

**Example:**

useEffect(() => {

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

}, [count]);

Here, useEffect runs after every time the count state changes.

**3. useContext**

useContext allows you to access values from the nearest matching Context.Provider in the component tree without needing to pass props through every level.

**Syntax:**

const value = useContext(MyContext);

**Example:**

const theme = useContext(ThemeContext);

This lets you access the theme value that was set in the ThemeContext provider, avoiding prop drilling.

**4. useReducer**

useReducer is an alternative to useState for managing more complex state logic. It takes a reducer function and an initial state, returning the current state and a dispatch function for triggering state updates.

**Syntax:**

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

**Example:**

const reducer = (state, action) => {

switch (action.type) {

case 'increment':

return { count: state.count + 1 };

default:

return state;

}

};

const [state, dispatch] = useReducer(reducer, { count: 0 });

Here, the reducer manages the state transitions based on the action type.

**5. useCallback**

useCallback is used to memoize functions in React, preventing them from being recreated on every render. It returns a memoized version of a callback function, which only changes if one of the dependencies has changed.

**Syntax:**

const memoizedCallback = useCallback(() => {

// Your code here

}, [dependencies]);

**Example:**

const handleClick = useCallback(() => {

console.log("Button clicked!");

}, []);

This memoized handleClick function will remain the same across renders unless the dependencies change.

**6. useMemo**

useMemo is used to memoize expensive computations, ensuring that a function is only re-executed when its dependencies change. It can improve performance by avoiding unnecessary recalculations.

**Syntax:**

const memoizedValue = useMemo(() => {

// Expensive computation

return value;

}, [dependencies]);

**Example:**

const expensiveCalculation = useMemo(() => {

return computeExpensiveValue(a, b);

}, [a, b]);

Here, useMemo ensures that computeExpensiveValue is only recomputed when a or b changes.

**Implementation Details:**

**Expt 3**

**Task 1**

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

**import './UserProfileEditor.css';**

**const UserProfileEditor = () => {**

**const [formVisible, setFormVisible] = useState(true);**

**const [formData, setFormData] = useState({**

**name: '',**

**age: '',**

**email: ''**

**});**

**const initialFormData = {**

**name: '',**

**age: '',**

**email: ''**

**};**

**const handleChange = (e) => {**

**const { name, value } = e.target;**

**setFormData({ ...formData, [name]: value });**

**};**

**const toggleFormVisibility = () => {**

**setFormVisible(!formVisible);**

**};**

**const resetForm = () => {**

**setFormData(initialFormData);**

**};**

**return (**

**<div className="profile-editor">**

**<button onClick={toggleFormVisibility}>**

**{formVisible ? 'Hide Form' : 'Show Form'}**

**</button>**

**{formVisible && (**

**<form className="profile-form">**

**<div className="form-group">**

**<label htmlFor="name">Name:</label>**

**<input**

**type="text"**

**id="name"**

**name="name"**

**value={formData.name}**

**onChange={handleChange}**

**/>**

**</div>**

**<div className="form-group">**

**<label htmlFor="age">Age:</label>**

**<input**

**type="number"**

**id="age"**

**name="age"**

**value={formData.age}**

**onChange={handleChange}**

**/>**

**</div>**

**<div className="form-group">**

**<label htmlFor="email">Email:</label>**

**<input**

**type="email"**

**id="email"**

**name="email"**

**value={formData.email}**

**onChange={handleChange}**

**/>**

**</div>**

**<button type="button" onClick={resetForm}>Reset</button>**

**</form>**

**)}**

**<div className="profile-info">**

**<h2>Profile Information</h2>**

**<p><strong>Name:</strong> {formData.name}</p>**

**<p><strong>Age:</strong> {formData.age}</p>**

**<p><strong>Email:</strong> {formData.email}</p>**

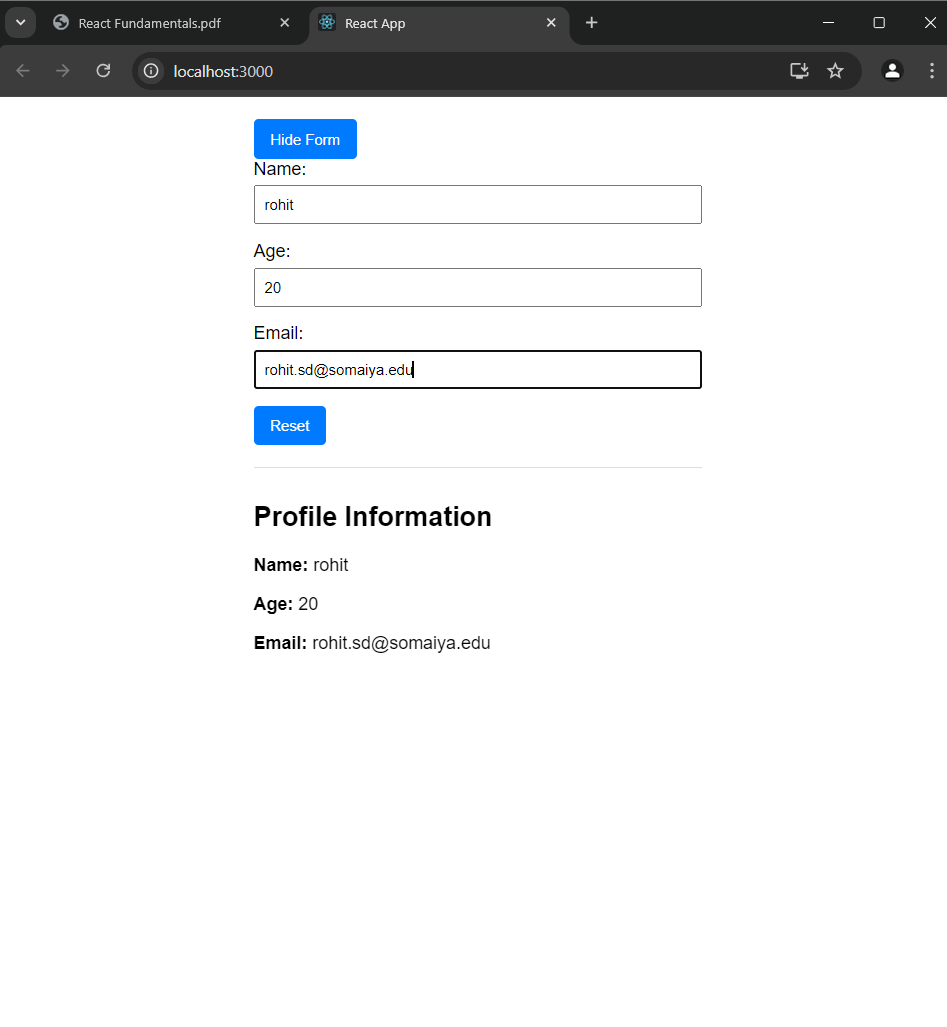
**</div>**

**</div>**

**);**

**};**

**export default UserProfileEditor;**

****

**Conclusion:**

Learnt about implementing different react hooks