**Task-1**

**Aim:**

**1. Experiments with React Hooks**

**1) Theoretical Background:**

* useState : We initialize our state by calling useState in our function component.

useState accepts an initial state and returns two values:

* The current state.
* A function that updates the state.

**Source Code:** import { useState } from 'react'; export default function Counter() {

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

setCount(count + 1);

} return (

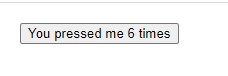
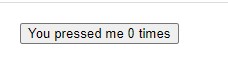
<button onClick={handleClick}> You pressed me {count} times

</button>

);

}

**Output:**



**Learning Outcome:In this practical i learned about how to use and implement code using useState hook.**

**2)Theoretical Background:**

* useEffect :The useEffect Hook allows you to perform side effects in your components.
* Some examples of side effects are: fetching data, directly updating the DOM, and timers.
* useEffect accepts two arguments. The second argument is optional.
* useEffect(<function>, <dependency>)

**Source Code:**

import {useState, useEffect } from 'react';

import { createConnection } from './chat.js';

function ChatRoom({ roomId }) {

const [serverUrl, setServerUrl] = useState('https://localhost:1234');

useEffect(() => {

const connection = createConnection(serverUrl, roomId);

connection.connect(); return () => {

connection.disconnect();

};

}, [roomId, serverUrl]);

return (

<>

<label>

Server URL:{' '}

<input value={serverUrl}

onChange={e => setServerUrl(e.target.value)}

/>

</label>

<h1>Welcome to the {roomId} room!</h1>

</>

);

}

export default function App() {

const [roomId, setRoomId] = useState('general'); const [show, setShow] = useState(false); return (

<>

<label>

Choose the chat room:{' '}

<select value={roomId}

onChange={e => setRoomId(e.target.value)}

>

<option value="general">general</option>

<option value="travel">travel</option>

<option value="music">music</option>

</select>

</label>

<button onClick={() => setShow(!show)}>

{show ? 'Close chat' : 'Open chat'}

</button>

{show && <hr />}

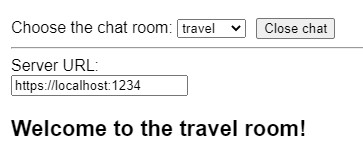
{show && <ChatRoom roomId={roomId} />}

</>

);

}

**Output:**



**Learning Outcome: In this practical i learned about how to use the useEffect hook.**

**3)Theoretical Background:**

* useRef : The useRef Hook allows you to persist values between renders.
* It can be used to store a mutable value that does not cause a re-render when updated.
* It can be used to access a DOM element directly.

**Source Code:** import { useRef } from 'react';

export default function Counter() {

let ref = useRef(0);

function handleClick() {

ref.current = ref.current + 1;

alert('You clicked ' + ref.current + ' times!');

}

return (

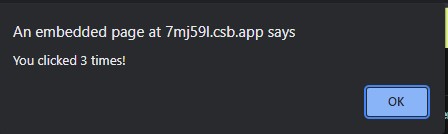
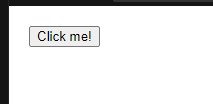
<button onClick={handleClick}> Click me!

</button>

);

}

**Output:**



**Learning Outcome: I learned about how we can implement the useRef hook and it’s uses.**

**4)Theoretical Background:**

* useContext : React Context is a way to manage state globally.
* It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.

**Source Code:**

import { createContext, useContext } from 'react'; const ThemeContext = createContext(null);

export default function MyApp() {

return (

<ThemeContext.Provider value="dark">

<Form />

</ThemeContext.Provider>

)

}

function Form() {

return (

<Panel title="Welcome">

<Button>Sign up</Button>

<Button>Log in</Button>

</Panel>

);

}

function Panel({ title, children }) { const theme = useContext(ThemeContext); const className = 'panel-' + theme;

return (

<section className={className}>

<h1>{title}</h1>

{children}

</section>

)

}

function Button({ children }) {

const theme = useContext(ThemeContext); const className = 'button-' + theme;

return (

<button className={className}>

{children}

</button>

);

}

**Output:**



**Learning Outcome:**

In this practical I learned about implementation and uses of useContext hook.

**5)Theoretical Background:**

* useMemo: The React useMemo Hook returns a memoized value.
* Think of memoization as caching a value so that it does not need to be recalculated.
* The useMemo Hook only runs when one of its dependencies update.
* This can improve performance.

**Source Code:**

import { useMemo } from 'react'; import List from './List.js'; import { filterTodos } from './utils.js'

export default function TodoList({ todos, theme, tab }) {

const visibleTodos = useMemo(

() => filterTodos(todos, tab),

[todos, tab]

);

return (

<div className={theme}>

<p><b>Note: <code>List</code> is artificially slowed down!</b></p>

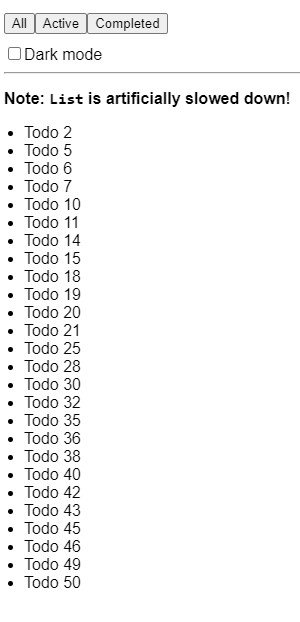
<List items={visibleTodos} />

</div>

);

}

**Output:**



**Learning Outcome:** In this practical i learned about use and how to implement useMemo hook.

**Task-2**

**Aim:**

**1. Create a Custom React Hook to Fetch PUBLIC API data and Display it on the webpage.**

**Theoretical Background:**

Hooks are reusable functions.

When you have component logic that needs to be used by multiple components, we can extract that logic to a custom Hook.

**Source Code:** import { useEffect, useState } from "react"; import axios from "axios";

const useFetch = (method, url, body) => { const [isLoading, setIsLoading] = useState(false); const [apiData, setApiData] = useState(null); const [serverError, setServerError] = useState(null);

useEffect(() => { setIsLoading(true); const fetchData = async () => { try { const resp = await axios({ method:

method, url: url, data:

body

}); const data = await resp?.data;

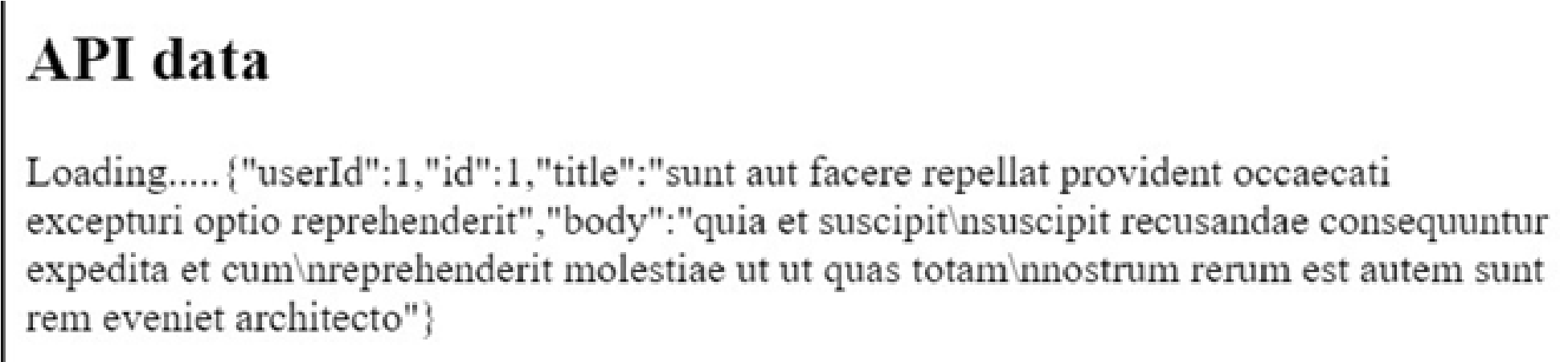
setApiData(data); setIsLoading(false); } catch (error) { setServerError(error); setIsLoading(false);

}

}; fetchData(); }, [url, method, body]);

return { isLoading, apiData, serverError };

}; export default **Output:**



**Learning Outcome:** In this practical i learned about how to use different hooks and how to create custom hook.