React JS-2

Hooks in react js

Hooks were added to React in version 16.8.

Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.

There are 3 rules for hooks:

- Hooks can only be called inside React function components.
- Hooks can only be called at the top level of a component.
- Hooks cannot be conditional (don't call hooks from inside loops, conditions, or nested statements so that the hooks are called in the same order each render.)

Why Use Hooks?
- Makes code **cleaner and reusable**
- Eliminates the need for **class components**
- Encourages **functional programming** in React

Import Hooks from react.

import { *Hook type } from "react";

1.To use the **useState** Hook (keep track of the application state)

import React,{ useState } from "react";

2.To use the **useState and useContext** Hook

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

React provides a bunch of standard in-built hooks:

Function/Hook	Purpose	Syntax
useState	Add and manage **local state** in function components	const [state, setState] = useState(initialValue)
useReducer	Manage **complex state logic** with a reducer function	const [state, dispatch] = useReducer(reducer, initialState)
createContext	Create a **global context** to share data across components	const MyContext = createContext(defaultValue)
useContext	Access the nearest **Context Provider's value**	const value = useContext(MyContext)
useEffect	Run **side effects** (fetching data, timers, API calls, subscriptions etc.)	useEffect(() => { /* effect */ }, [deps])

useState

The React useState Hook allows us to track state in a function component.
State generally refers to data or properties that need to be tracking in an application.

The useState Hook can be used to keep track of strings, numbers, booleans, arrays, objects, and any combination of these!

Initialize useState

We initialize our state by calling useState in our function component. It accepts an initial state and returns two values:

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

Syntax:

Const [current state, function to update state] =useState (initial state)

To import useState

```
import { useState } from "react";
```

Example 1: Write a program to build React app having a button which increase count by 1 while clicking it.

US1.js

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

function US1() {
 // Declare a new state variable, which we'll call "count"
 const [count, setCount] = useState(0);
//count is current state and setCount is function that is used to update our state.

 function handleCount () {
  setCount(count+1)
 }

 return (
  <div>
   <p>You clicked {count} times</p>
   <button onClick={handleCount}>
    Click me
   </button>
```

```
    </div>
 );
}
export default US1
```

```
import { useState } from "react";
function US1 () {
 const [count, setCount] = useState(0);
 return (
   <div>
     <p>You clicked {count} times</p>
     <button onClick={() => setCount(count + 1)}>Click me</button>
   </div>
 );
}export default US1
```

Example 2: Create a program to build React app having buttons to increment and decrement the number by clicking that respective button. Also, increment of the number should be performed only if number is less than 10 and decrement of the number should be performed if number is greater than 0.

US2.js

```
import { useState } from "react";
function US2 (){
   const [num,setnum]=useState(0)
   function increment(){
      if(num<10){
         setnum(num+1);
      }else{
         return false;
      }
   }
   function decrement(){
      if(num > 0){
         setnum(num-1);
      }else{
         return false;
      }
      }
   return (
      <div>
```

```
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
      <h1> {num} </h1>
    </div>
  )
}
export default US2
```

- **useState- In CSS**

Example 3: Write a program to build React app having a button which changes background color of a text by clicking it.

```
import React, { useState } from 'react';
function US3() {
        const [style,setstyle]=useState("red");
        function TextChange(){
                        setstyle("green")
                                }
 return (
   <div>
<button onClick={TextChange}>Change Text Color</button>
<h1 style={{backgroundColor:style}}>useState in CSS</h1>
   </div>
 ) }
export default US3
```

- **useState- In Image**

Example 4: Write a program to build React app having a button which changes image by clicking it.

```
import React, { useState } from 'react';
import img1 from "./img1.jpg";
import img2 from "./img2.jpg";
function US4() {
     const [pic,setpic]=useState(img1);
     function ImageChange(){ setpic(img2) }
return (
                <div>
                      <img src={pic} heigth="200px" width="200px" alt="logo" />
                      <button onClick={ImageChange}>Change Image</button>
                </div>
                ) }
export default US4
```

- **useState- with Input**

Example 5: Write a program having a button "show". By clicking a button, it will display text and button text will be changed as "Hide". By clicking Hide button, the text will be disappeared and button text will become "show" again.

```
import React,{useState} from 'react'
function US5 () {

const[buttontext,setButtontext]= useState("show")
const[text,setText]= useState("")
function showhide() {
   if(buttontext==="show")
    {
     setButtontext("hide");
     setText("Hello");
    }
    else{
     setButtontext("show");
     setText("");
   } }
 return (
  <div>
  <button onClick = {showhide}> {buttontext}</button>
  {text}
  </div>
 ) }
  export default US5;
```

Example: 6

Write a program to build React app to perform the tasks as asked below.

- Add three buttons "Change Text", "Change Color", "Hide/Show".
- Add heading "LJ University" in red color(initial) and also add "React Js Hooks" text in h2 tag.
- By clicking on "Change text" button text should be changed to "Welcome students" and vice versa.
- By clicking on "Change Color" button change color of text to "blue" and vice versa. This color change should be performed while double clicking on the button.
- Initially button text should be "Hide". While clicking on it the button text should be changed to "Show" and text "React Js Hooks" will not be shown.

US6.js

```jsx
import {useState} from "react";

function US6(){
  //useState to Change text
  const [name,setName] = useState("LJ University");
  //useState to Change Color
  const [textColor,setcolor] = useState("Red");
  //useState to Show Hide text
  const [hideText,setHide]=useState("React Js Hooks");
  const[buttontext,setButtontext]= useState("Hide")

  // Function to show and hide text and also button text
  function showhide() {
    if(buttontext==="Hide")
    {
      setButtontext("Show");
      setHide("")
    }
    else{
      setButtontext("Hide");
      setHide("React Js Hooks")
    }
  };
  // function to change text value
  function changeName(){
    if(name === "LJ University"){
    setName("Welcome Students")
    }else{
      setName("LJ University")
    }
  }
  // Function to change color of the text
  function changeColor(){
    if(textColor === 'red'){
    setcolor("blue")
    }else{
      setcolor("red")
    }
  }
return(
  <div>
    <button onClick={changeName}>Change Text</button>
    <button onDoubleClick={changeColor}>Change Color</button>
```

```
      <button onClick = {showhide}> {buttontext}</button>


      <h1 style={{color:textColor}}>{name}</h1>
      <h2>{hideText}</h2>
   </div>
)}
export default US6
```

Example -7 : Create a React component that randomly displays one image from a set of predefined images and changes the image when a button is clicked

```
import {useState} from "react";
import img1 from "./img1.jpg"
import img2 from "./img2.jpg"
import img3 from "./img3.png"
import img4 from "./img4.jpg"
import img5 from "./img5.jpg"

function US7()
{
   const arr = [img1,img2,img3,img4,img5]
   const [myimage,setimage] = useState(arr[0]);
   const changeImage = () => {
      const randomIndex = Math.floor(Math.random() * arr.length);
      setimage(arr[randomIndex]);
    };

    return (
     <div className="App">
      <header className="App-header">
       <h1>Random Image Generator</h1>
       <img src={myimage} alt="Random" width="500" height="500"/>
       <button onClick={changeImage}>Change Image</button>
      </header>
     </div>
    );


}
export default US7
```

Example 8: Write a program to build React app for task todo list.
- Add 1 input field and button and by clicking on button display entered task on the same page.
- Also, add delete button with each added task to delete the task.

Todo.js

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

function Todo() {
  const[Task, setTask]= useState("");
  const[Todolist, setTodoList]=useState([]);
  function handleChange(event) {
    setTask(event.target.value);
  }
  function addTask(event) {
    setTodoList([...Todolist,Task]);
  };

// Not to allow duplicate entry
function addTask(event) {

  if (!Todolist.includes(Task)) {
    setTodoList([...Todolist, Task]);
  } else {
    alert('Task already entered');
  }
  };

  // To Delete Task
  function deleteTask (taskName){
    setTodoList(
      Todolist.filter((task)=>{
        if (task!==taskName){
          return true;
        } else{
          return false;
        }
      })
    )
  }

return (
  <div>
  <h1> Enter Task </h1>
```

```
    <input onChange={handleChange}/>
      <button onClick={addTask}> Add Task </button>

      {/* Code If do not want to delete Task
      {
        Todolist.map((task)=>
        {
          return <h1>{task}</h1>
        })
      } */}

  {/* If want to add delete Task functionality */}
      {Todolist.map((task)=>{
        return(
          <div>
            <h1> {task}</h1>
            <button onClick={() => deleteTask(task)}>Delete</button>
          </div>
        );
      })}
    </div>
  );
}
export default Todo
```

Example 9: Write a program to build React app having 2 input fields and display the entered value on the same page.

Method -1 Using Serate useState Hook:

Advantages:

- Easier to manage for forms with very few fields.
- Clear separation between each input's state.

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

function US9() {
  const[firstName,setFirstName]=useState("");
  const[lastName,setLastName]=useState("");

  function handleChange(event)    {
    setFirstName(event.target.value);   }
```

```
    function handleChange1(event)
    {
        setLastName(event.target.value);
    }
  return (
    <div>
      <h3>Enter Your First Name</h3>
     <input name="firstName" value={firstName} onChange={handleChange}/>
      <br/>
      <h3>Enter Your Last Name</h3>
      <input name="lastName" value={lastName} onChange={handleChange1}/>
      <br/>
      <h1>First Name: {firstName} <br/>Lastname: {lastName}</h1>
    </div>
  ) }
export default US9
```

Or

Method – 2 Using Single useState Hook for All Fields

Advantages:

- Cleaner and scalable for forms with **many fields**.
- Easier to integrate with validations and form libraries.

For many field use Dynamic Property Update using Computed Property Names approach.

```
import React, { useState } from 'react'
function US9() {
  const[data,setdata]=useState({});

  const handleChange = (e) => {
    const { name, value } = e.target;
    setdata({...data,[name]: value});
  };
 return (
  <div>
    <div><input     type="text"    name="firstName"         onChange={handleChange}
placeholder='First Name'/></div>
    <div><input     type="text"    name="lastName"          onChange={handleChange}
placeholder='Last Name'/></div>
    <h1>First Name: {data.firstName} Lastname: {data.lastName}</h1>
  </div>
 ) }
export default US9
```

Example 10 Create react app which takes user defined inputs number 1 and number 2 and perform addition, subtraction, multiplication, division of the numbers. (Use useState hook)

```jsx
import  { useState } from 'react'
function US10() {
   const[data,setdata]=useState({});
   const[result,setresult]=useState();

   const handleChange = (e) => {
     const { name, value } = e.target;
     setdata({...data,[name]: value});
   };

     function addition(){
      setresult(parseInt(data.num1) + parseInt(data.num2))
     }
     function sub(){
      setresult(parseInt(data.num1) - parseInt(data.num2))
     }
     function mult(){
      setresult(parseInt(data.num1) * parseInt(data.num2))
     }
     function division(){
      setresult(parseInt(data.num1) / parseInt(data.num2))
     }

   return (
    <div>
      <div><input type="number" name="num1"  onChange={handleChange} placeholder='First Name'/></div>
      <div><input type="number" name="num2"  onChange={handleChange} placeholder='Last Name'/></div>
       <button onClick={addition}>addition</button>
       <button onClick={sub}>Subtraction</button>
       <button onClick={mult}>Multiplication</button>
       <button onClick={division}>Division</button>
       <h1> {result}</h1>
    </div>
   ) }
export default US10
```

Hook in React Forms

Forms are an integral part of any modern web application. It allows the users to interact with the application as well as gather information from the users.

Forms can perform many tasks that depend on the nature of your business requirements and logic such as authentication of the user, adding user, searching, filtering, booking, ordering, etc. A form can contain text fields, buttons, checkbox, radio button, etc.

In React, form data is usually handled by the components. When the data is handled by the components, all the data is stored in the component state. You can control changes by adding event handlers in the onChange attribute and that event handler will be used to update the state of the variable.

Adding Forms in React like any other element:

```
function MyForm() {
   return (
   <form>
     <label>Enter your name:
         <input type="text" />
    </label>
   </form>
   ) }
export default MyForm
```

Handling Forms

Handling forms is about how you handle the data when it changes value or gets submitted.

In React, form data is usually handled by the components. When the data is handled by the components, all the data is stored in the component state. You can control changes by adding event handlers in the onChange attribute. We can use the useState Hook to keep track of each inputs.

```
import { useState } from 'react';
function MyForm() {
 const [name, setName] = useState("");
 return (
 <form>
 <label>Enter your name:
  <input type="text"  value={name}   onChange={(e) => setName(e.target.value)} />
 </label>
{/*<br/> <h1> Entered Value: {name}</h1>  To check entered values*/}
 </form>
 ) }
export default MyForm
```

Prof. Khushbu Patel

Submitting Forms

You can control the submit action by adding an event handler in the **onSubmit** attribute for the **<form>:**

```
import { useState } from 'react';
function MyForm() {
        const [name, setName] = useState("");
        const handleSubmit = (event) => {
            event.preventDefault();
            alert(`The name you entered was: ${name}`)
        }
 return (
 <form onSubmit={handleSubmit}>
  <label>Enter your name:
      <input type="text" value={name} onChange={(e) => setName(e.target.value)} />
 </label>
{/*<br/> <h1> Entered Value: {name}</h1>  */}
        <input type="submit" />
        </form>
 ) }
export default MyForm
```

Output:
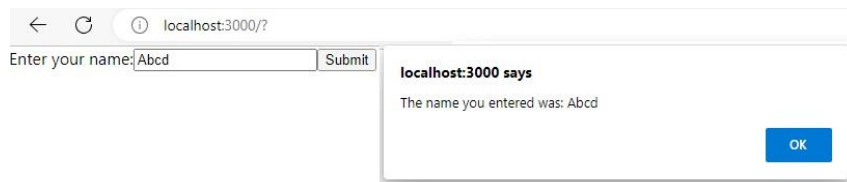
Form Fields

- **Textarea**

The textarea element in React is slightly different from ordinary HTML. In React the value of a textarea is placed in a value attribute. We'll use the useState Hook to manage the value of the textarea:

```
import { useState } from 'react';
function MyForm() {
const [txtarea, setTextarea] = useState(
"The content of a textarea goes in the value attribute"
);
const handleChange = (event) => {
setTextarea(event.target.value)
}
return (
```

```
<form>
<textarea value={txtarea} onChange={handleChange} />
</form>
)} export default MyForm
```

- ## Select

A drop-down list, or a select box, in React is also a bit different from HTML.

In React, the selected value is defined with a **value** attribute on the **select** tag:

```
import { useState } from 'react';
function MyForm() {
   const [myCar, setMyCar] = useState("Volvo");
     const handleChange = (event) => {
     setMyCar(event.target.value)
   }
   return (
   <form>
       <select value={myCar} onChange={handleChange}>
       <option value="Ford">Ford</option>
       <option value="Volvo">Volvo</option>
       <option value="Fiat">Fiat</option>
       </select>
   </form>
   ) }
export default MyForm
```

- ## Radio Button

Example: - Create a React Form for select any of pizza size using radio button.

```
import React from 'react';
import { useState } from 'react';
function Myform() {
       const [size, setsize] = useState("Medium");
       function onOptionChange(e){
        setsize(e.target.value);
             };
 return (
 <div>
    <h3>Select Pizza Size</h3>
  <form>
  <input    type='radio'   name='size'   value='Regular'   checked={size   ===   'Regular'}
onChange={onOptionChange} />Regular
```

Select Pizza Size

○ Regular ● Medium ○ Large

Selected size **Medium**

```
   <input type='radio' name='size' value='Medium' checked={size === 'Medium'}
onChange={onOptionChange} />Medium
   <input  type='radio'  name='size'  value='Large'  checked={size  ===  'Large'}
onChange={onOptionChange} />Large
   </form>
   <p>   Selected size <strong>{size}</strong>   </p>
 </div>
 ) }
export default Myform
```

Example 1:

Create react app which contains form with following fields.
• First Name(Input type text)
• Lastname(Input type text)
• Email(Input type email)
• Password(Input type password)
• Message (Textarea)
• Gender(Radio Button)
• City (Dropdown)
Display submitted values in alert box. (Using useState Hook)

Myform2.js

```
import React, { useState } from 'react'
function Myform2() {
 const[formdata,setformdata]=useState({});

function handlechange(event){

 const name = event.target.name;
 const value = event.target.value;

 setformdata({...formdata, [name]: value})
 }
function handlesubmit(e) {
 e.preventDefault();
 alert("Your form has been submitted.\nName: " + formdata.fname +"\nEmail: " + formdata.eid
+ "\nCity: "+ formdata.city +"\nGender:"+formdata.gender)
 }

return (
 <div>
 <form className="form-data" onSubmit={handlesubmit}>
```

```
 <label>First Name:</label>
 <input type="text" name="fname" onChange={handlechange} /><br/>

 <label>Email Id:</label>
 <input type="email" name="eid" onChange={handlechange} /><br/>

 <label>Password:</label>
 <input type="password" name="pass" onChange={handlechange}required/><br/>

 <label>Message : </label>
 <textarea name="msg" onChange={handlechange} /><br/>

 <select onChange={handlechange} name='city'>
 <option value="Ahmedabad">Ahmedabad</option>
 <option value="Rajkot">Rajkot</option>
 </select>

 <input type="radio" name="gender" value="Male" onChange={handlechange} />Male
 <input type="radio" name="gender" value="Female" onChange={handlechange}/>Female

 <button type="submit">Submit</button> <br/>
 </form>
 </div>
 )
}
export default Myform2
```

Example 2:
Create react app which contains form with following fields.
• Email(Input type email)
• Password(Input type password)
• Confirm Password(Input type password)

[For Ref.: Add validation using regex to validate email id and password (Must contain at least 8 characters and must contain at least 1 uppercase, 1 lowercase and 1 digit).]
Also values of password and confirm password must be same.

Display email in alert box. (Using useState Hook)

```
import React, { useState } from 'react'
function Myform3 () {
const[formdata,setformdata]=useState({ });
```

```
function handlechange(event){
const name = event.target.name;
const value = event.target.value;
setformdata({...formdata, [name]: value})
}

function handlesubmit(e){
e.preventDefault();
const emailregex=/^([a-zA-Z0-9._-]+@[a-zA-Z]+\.[a-zA-Z]{2,4}$)/;
const passregex=/(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z]).{8,}/;
if(!emailregex.test(formdata.eid))
{
alert("Please enter a valid Email ID");
}
else if(!passregex.test(formdata.pass))
{
alert("Password must contain atleast 8 characters ( Atleast 1 digit, 1lowercase & 1 uppercase
alphabets )");
}
else if(formdata.pass !== formdata.cpass){
alert("password and confirm password must be same");
}
else{
alert("Your form has been submitted.\nEmail: " + formdata.eid )
}
}

return (
<div>
<form onSubmit={handlesubmit}>

<label>Email Id:</label>
<input type="email" name="eid" onChange={handlechange} /><br/>
<label>Password:</label>
<input type="password" name="pass" onChange={handlechange}/><br/>
<label>Confirm Password:</label>
<input type="password" name="cpass" onChange={handlechange} /><br/>
<button type="submit">Submit</button> <br/>
</form>
</div>
)
}
export default Myform3
```

• ^ represents the starting of the string.
• (represents the starting of the group.
• [A-Za-z0-9._-] represents the domain name should be a-z or A-Z or
0-9 and .,_,-.
• \. represents the string followed by a dot.
•)+ represents the ending of the group, this group must appear at least
1 time, but allowed multiple times for subdomain.
• [A-Za-z]{2, 6} represents , must be A-Z or a-z between 2 and 4
characters long.
• $ represents the ending of the string.

Task 13 : Create react app which contains form with fields Name, Email Id, Password and Confirm Password. When the form submitted the values of password and confirm password fields must be same else it will give an error message in alert box. If form submitted successfully then display entered name and email id in alert box.

Form2.js

```
import React, { useState } from 'react'
function Form2(){
  const[formdata,setformdata]=useState({});
  const handlechange = (event) => {
    const name = event.target.name;
    const value = event.target.value;
    setformdata({...formdata, [name]: value})
  }

        const handlesubmit=(e)=>

  {
    e.preventDefault();
    if(formdata.pass !== formdata.cpass){
       alert("Values of Password and Confirm password must be same")
    }else{
       alert("Welcome "+formdata.fname+"\n Your Email id is: "+ formdata.eid)
    }
  }
  return (
  <div>
    <form className="form-data" onSubmit={handlesubmit}>

      <label>Name:</label>
      <input type="text" name="fname" onChange={handlechange} /><br/>

      <label>Email Id:</label>
      <input type="email" name="eid" onChange={handlechange}
required/><br/>

      <label>Password :</label>
```

```
        <input type="password"  name="pass" onChange={handlechange}
required/><br/>

        <label>Confirm Password :</label>
        <input type="password"  name="cpass" onChange={handlechange} /><br/>

    <button type="submit">Submit</button> <br/>
    </form>

  </div>
 )
}
export default Form2
```

Form Just for the reference without using useState

```
  const handlesubmit=(e)=>
  {
    e.preventDefault();
    if(e.target.pass.value !== e.target.cpass.value){
      alert("password and confirm password must be same");
    }
    else{
      alert("form Submitted with below data.\n First name: "+ e.target.fname.value +" Last
Name : "+ e.target.lname.value +" Email: "+ e.target.email.value);
```

<u>useReducer</u>

The useReducer Hook is the better alternative to the <u>useState</u> hook and is generally more preferred over the useState hook when you have complex state-building logic or when the next state value depends upon its previous value or when the components are needed to be optimized.

You can add a reducer to your component using the useReducer hook. Import the useReducer method from the library like this:

```
import { useReducer } from 'react'
```

The useReducer method gives you a state variable and a dispatch method to make state changes. You can define state in the following way:

```
const [state, dispatch] = useReducer(reducerFunction, initialValue)
```

The reducer function contains your state logic. You can choose which state logic to call using the dispatch function. The state can also have some initial value similar to the useState hook.

Example:1 Perform incement using useReducer

The useReducer(reducer, initialState) hook accepts 2 arguments: the reducer function and the initial state. The hook then returns an array of 2 items: the current state and the dispatch function.

```
import React, { useReducer } from 'react';
const initialstate = 0;
function UR1() {
  const [state, dispatch] = useReducer(reducer, initialstate);
  function reducer(state,action){
    if(action.type==='increment'){
      return state+1;
    }
  }
  return (
      <button onClick={()=> dispatch({type:"increment"})}>
    Click me ({state})
   </button>
 );
}
export default UR1
```

A. Initial state
The initial state is the value the state is initialized with. Here initialized with 0.

B. Reducer function
The reducer is a pure function that accepts 2 parameters: **the current state and an action object.** Depending on the action object, the reducer function must update the state in an immutable manner, and return the new state.

C. Action object
An action object is an object that describes how to update the state.

Typically, the action object has a property **type** — a string describing what kind of state update the reducer must do.

If the action object must carry some useful information to be used by the reducer, then you can add additional properties to the action object. **Here we have defined type "increment".**

D. Dispatch function
The dispatch is a special function that dispatches an action object.

The dispatch function is created for you by the useReducer() hook:

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

Whenever you want to update the state (usually from an event handler or after completing a fetch request), you simply call the dispatch function with the appropriate action object: dispatch(actionObject).

Like as below in above example

```
<button onClick={()=> dispatch({type:"increment"})}>
```

In simpler terms, dispatching means a request to update the state.

What Is a Reducer and Why do You Need It?

Let's take an example of a To-Do app. This app involves adding, deleting, and updating items in the todo list. The update operation itself may involve updating the item or marking it as complete.

When you implement a todo list, you'll have a state variable todoList and make state updates to perform each operation. However, these state updates may appear at different places, sometimes not even inside the component.
To make your code more readable, you can move all your state updates into a single function that can exist outside your component. While performing the required operations, your component just has to call a single method and select the operation it wants to perform.

The function which contains all your state updates is called the reducer. This is because you are reducing the state logic into a separate function. The method you call to perform the operations is the dispatch method.

Example 2: Create react js app to increase value by 5 while clicking on button. Initialize value with 20. Use useReducer hook to perform the task.

UR2.js

```
import React, { useReducer } from 'react'
function UR2(){
   const [state,dispatch]=useReducer(reducer ,20);
   function reducer(state,action){
      return state+action;
   }
 return (
   <div align="center">
      <h1 align="center">{state}</h1>
      <button onClick={()=>dispatch(5)}>Add</button>
   </div>
 )}
export default UR2
```

Example 3: Create react js app to increase value by 1 while clicking on button "Increment" and decrease value by 1 while clicking on button "Decrement". While clicking on reset button value will reset to initial value. Initialize value with 0. Use useReducer hook to perform the task.

UR3.js

```
import React, { useReducer } from "react";
function UR3(){
   const[state,dispatch] = useReducer(reducer,0);
   function reducer(state,action){
      //console.log(state,action)
      if(action.type==='increment'){ return state+1; }
      if(action.type==='decrement'){ return state-1; }
if (action.type === "reset") {return 0; }} // Reset state to 0
return(
   <>
      <h1>{state}</h1>
      <button onClick={()=> dispatch({type:"increment"})}> Increment </button>
      <button onClick={()=> dispatch({type:"decrement"})}> Decrement </button>
     <button onClick={() => dispatch({ type:"reset"})}>Reset</button>
   </>
   ) }
export default UR3
```

useContext

In React, useContext is a hook that allows you to access and consume values from the React Context API. The Context API provides a way to pass data down the component tree without having to pass props explicitly at every level. It's useful when you want to share data between multiple components without the need for prop drilling.

Context provides a way to pass data or state through the component tree without having to pass props down manually through each nested component. It is designed to share data that can be considered as global data for a tree of React components.

How to use the context

Using the context in React requires 3 simple steps:

- creating the context,
- providing the context,
- consuming the context.

Example 1: Write a reactJS program to perform the tasks as asked below.

- Create one main file (parent file) name Main.js and other 2 component files UC1.js and UC2.js
- Pass First name and Last name from Main.js file to UC2.js file. And display Welcome ABC XYZ (suppose firstname is ABC and Last name is XYZ) in browser.

A. Creating the context

The built-in function createContext(default) creates a context instance:

```
import React, { createContext } from "react"
const Variablename = createContext();
```

B. Providing the context

Context.Provider component available on the context instance is used to provide the context to its child components, no matter how deep they are.

To set the value of context use the **value prop**

```
< Variablename.Provider value=”Test” />
```

Main.js

```
import React, { createContext } from "react"
import UC1 from "./UC1"
// To create context for firstname and lastname
const Fname = createContext();
const Lname = createContext();

function Main(){
    return (
```

```
    <>
        <Fname.Provider value="ABC">  // to provide the value which should be passed
            <Lname.Provider value="XYZ">
                <UC1/>
            </Lname.Provider>
        </Fname.Provider>
    </>
    ) }
export default Main
export {Fname,Lname} //Each component must have one default export, so the context
should be exported using { } as object.
```

Again, what's important here is that all the components that'd like later to consume the context have to be wrapped inside the provider component.
If you want to change the context value, simply update the value prop.

C. Consuming the context

Consuming the context can be performed by the useContext(Context) React hook:

The hook returns the value of the context: value = useContext(Context). The hook also makes sure to re-render the component when the context value changes.

Suppose, we have one parent component and we want to access its data in child component 3. Then, we can use createContext for creating a context and usecontext to use the context.

Parent Component	Child Component 1	Child Component 2
Main.js	UC1.js	UC2.js
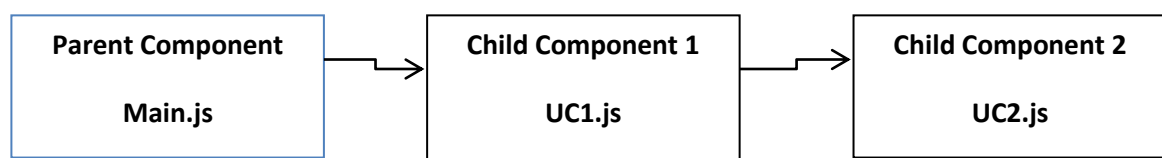
UC1.js

```
import React from "react"
import UC2 from "./UC2"
function UC1(){
    return (
        <UC2/>
    ) }
export default UC1
```

UC2.js

```
import React, { useContext } from "react"
import { Fname,Lname } from "./Main"
function UC2(){
    const firstname = useContext(Fname)
    const surname = useContext(Lname)
    return (
    <h1>Welcome {firstname} {surname}</h1>
    ) }
export default UC2
```

*Call only Main.js in App.js
App.js <- Main.js <- UC1.js <- UC2.js
So, First Name and Last Name are defined in Main.js component and are used directly in UC2.js component without passing data to all other hierarchical components.

Example 2: Write a reactJS program to perform the tasks as asked below.

- Create one main file (parent file) name Comp.js and other 3 component files Comp1.js, Comp2.js, Comp3.js.
- Pass Number1 and Number 2 from Comp.js file to Comp3.js file. Calculate multiplication of the numbers using useContext.

Comp.js

```
import React, { createContext } from "react"
import Comp1 from "./comp1"
const Num1 = createContext();
const Num2 = createContext();
function Comp(){
    return (
    <>
        <Num1.Provider value="20">
            <Num2.Provider value="5">
                <Comp1/>
            </Num2.Provider>
        </Num1.Provider>
    </>
    )}
export default Comp
export {Num1,Num2}
```

Comp1.js

```
import React from "react"
import Comp2 from "./comp2"

function Comp1(){
   return ( <Comp2/> )
}
export default Comp1
```

Comp2.js

```
import React from "react"
import Comp3 from "./comp3"

function Comp2(){
   return (
        <Comp3/>
   )
}
export default Comp2
```

Comp3.js

```
import React, { useContext } from "react"
import { Num1,Num2 } from "./comp"

function Comp3(){
   const num1 = useContext(Num1)
   const num2 = useContext(Num2)
   return (
   <h1>Multiplication of numbers in component-3: {num1 * num2}</h1>
   )
}
export default Comp3
```

App.js file:

```
import Comp from "./Comp"
function App() {
 return (
  <div>
        <Comp/>
  </div>
 ) }
export default App;
```

Prof. Khushbu Patel

Example 3 Use multiple contexts in a React application by creating and consuming them across different components.

Comp1.js: Creates a context for CSS styling and provides it to Comp2.

Comp2.js: Creates a context for a string value ("Students") and provides it to Comp3.

Comp3.js: Consumes both contexts and displays a message with the provided styles and string.

Comp1.js

```
import { createContext } from "react"
import Comp2 from "./Comp2"
const CC = createContext();
const mycss={backgroundColor:'yellow',color:'red',fontSize:"45px"}
function Comp1(){
   return (
     <CC.Provider value={mycss}>
         <Comp2/>
     </CC.Provider>
   )}
export default Comp1
export {CC}
```

Comp2.js

```
import { createContext } from "react"
import Comp3 from "./Comp3"
const CC1 = createContext();
function Comp2(){
   return (
     <CC1.Provider value="Students">
     <Comp3/>
     </CC1.Provider>
   )}
export default Comp2
export {CC1}
```

Comp3.js

```
import { useContext } from "react"
import { CC } from "./Comp1"
import { CC1 } from "./Comp2"
function Comp3(){
   const mycss = useContext(CC)
   const data = useContext(CC1)
   return (
   <h1 style={mycss}>Welcome to useContext tutorial {data}</h1>
   )}
export default Comp3
```

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])**

To import useEffect

> import { useEffect } from "react";

Below is the example to understand concept of empty array and array with value of useEffect

Example 1: Write react js app to perform the tasks as asked below.
- Add two buttons and increment count by one with each click.
- Display alert as an effect on specified conditions.
  - Effect will be triggered only when page rendered for the 1st time. (empty array)
  - Effect will be triggered every time the button A is clicked. (array with value)
  - When the Page is Rendered for the First Time and On Every Update/Event Triggered

UE1.js

```
import React, {useState,useEffect } from 'react'
function UE1 (){
  const[countA,setcountA]=useState(0);
  const[countB,setcountB]=useState(0);
//Triggered when the page is rendered first time and every time when Button A(count) clicked.
  useEffect(()=>
  {
    alert("Clicked")
  },[countA]);
//Triggered only once when the page is rendered first time
//useEffect(()=>
  //{
    //alert("Clicked")
  //},[]);
//Triggered everytime when the page is rendered
//useEffect(()=>
  //{
    //alert("Clicked")
  //});

  function changeCountA(){
```

```
    setcountA(countA+1);
  }
  const changeCountB=()=>{
    setcountB(countB+1);
  }
 return (
  <div>
    <button onClick={changeCountA}>Button A {countA}</button><br/>
    <button onClick={ changeCountB}>Button B { countB}</button>
  </div>
 ) }
export default UE1
```

Explanation of useEffect Usage

When the Page is Rendered for the First Time and When Button A (CountA) is Clicked:

```
useEffect(() => {
    alert("Clicked");
}, [countA]);
```

This useEffect runs whenever the count state changes, including the initial render. So, every time Button A is clicked and countA is updated, the effect runs, showing the alert "Clicked".

When the Page is Rendered for the First Time Only:

```
useEffect(() => {
    alert("Clicked");
}, []);
```

This useEffect with an empty dependency array runs only once when the component mounts. It won't run again unless the component is unmounted and remounted.

When the Page is Rendered for the First Time and On Every Update/Event Triggered:

```
useEffect(() => {
    alert("Clicked");
});
```

This useEffect runs on every render, including the initial render and any subsequent updates. This is because there is no dependency array, so it runs every time the component renders.

If your alert is showing twice initially, it is likely because of React's Strict Mode. In development mode, React's Strict Mode intentionally invokes certain lifecycle methods (like useEffect) twice to help identify potential issues.

Explanation

React Strict Mode is a tool for highlighting potential problems in an application. It does so by intentionally invoking certain methods, such as component mounts and updates, twice. This

behavior is designed to help developers find bugs related to side effects that might be dependent on certain lifecycle methods.

How to Check if Strict Mode is the Cause
You can check if this is the issue by looking at your index.js file where the React application is initialized. If you see React.StrictMode wrapping your application, that's the reason for the double invocation.

Example2: Write a ReactJS script to create a digital clock running continuously. (useEffect)

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

function USE1() {
 const [date, setDate] = useState(new Date());
 useEffect(() => {
  setInterval(() => {
   setDate(new Date());
  }, 1000)
 }, [])
 return (
  <h1>
   Time using Localtimestring - {date.toLocaleTimeString()}<br/>
   Hour-{date.getHours()}:Min-{date.getMinutes()}:Sec-{date.getSeconds()}
  </h1>
 )}
export default USE1;
```

Explanation:

   ✓ useEffect(..., []) ensures the interval is set **only once** when the component mounts.
   ✓ setInterval updates the time every second.

Without useEffect Issues
   ✓ Every time the component re-renders, a new setInterval is created.
   ✓ You'll end up with **multiple intervals**, causing performance issues and rapid updates.

Prof. Khushbu Patel

API integration with Axios

What is Axios?

It is a library which is used to make requests to an API, return data from the API, and then do things with that data in our React application.

Axios is a very popular (over 78k stars on Github) HTTP client, which allows us to make HTTP requests from the browser.

Syntax:

```
const getPostsData = () => {
 axios
 .get("API URL")
 .then(data => console.log(data.data))
 .catch(error => console.log(error));
 };
getPostsData();
```

Installing Axios

> • In order to use Axios with React, we need to install Axios. It does not come as a native JavaScript API, so that's why we have to manually import into our project.

> • Open up a new terminal window, move to your project's root directory, and run any of the following commands to add Axios to your project.

Using npm:

```
npm install axios
```

To perform this request when the component mounts, you use the useEffect hook. This involves importing Axios, using the .get() method to make a GET request to your endpoint, and using a .then() callback to get back all of the response data.

.catch() callback function : What if there's an error while making a request? For example, you might pass along the wrong data, make a request to the wrong endpoint, or have a network error.

In this case, instead of executing the .then() callback, Axios will throw an error and run the .catch() callback function. In this function, we are taking the error data and putting it in state to alert our user about the error. so if we have an error, we will display that error message.

Write React component that fetches a random joke from the API and displays it when the "Generate Joke" button is clicked.

```
import React,{ useState,useEffect } from 'react'
import axios from "axios";
const Randomjokeapi = () =>{
   const[joke,setJoke]=useState('');

   function fetchJoke(){
      axios
      .get("https://official-joke-api.appspot.com/random_joke")
      .then((response)=>{setJoke(response.data);})
      .catch((error)=>{console.error(error);})
   }
    useEffect(fetchJoke,[])

   return(
      <div>
      <h1>{joke.setup}</h1>
      <h3>{joke.punchline}</h3>
      <button onClick={fetchJoke} >Generate Joke </button>
      </div>
   )
}
export default Randomjokeapi
```

https://official-joke-api.appspot.com/random_joke link contains object as shown below. We have used setup and punchline properties for our webpage.

{"type":"general","setup":"What has ears but cannot hear?","punchline":"A field of corn.","id":238}

useEffect(fetchJoke, []) : This fetches a joke once when the component loads and lets the user fetch more with the button•

joke: to store the joke data, specifically the setup and punchline properties.

- fetchjoke uses axios.get to fetch a random joke from the API.
- On success, updates the joke state with the setup and punchline from the fetched data.
- On failure, logs the error to the console and updates the error state with a user-friendly message.

Example: Create react app to display API data by clicking on button using Axios (UseEffect + SetInterval).

```
import React,{ useState,useEffect } from 'react'
import axios from "axios";
const Randomjokeapi = () =>
{
   const[joke,setJoke]=useState("");

   function fetchJoke(){
      axios
      .get("https://official-joke-api.appspot.com/random_joke")
      .then((response)=>{setJoke(response.data);})
      .catch((error)=>{console.error(error); }  )
  }
useEffect(() => {
      setInterval(() => { setJoke(joke);}, 8000)
      },[])

   return(
      <div>
      <h1>{joke.setup}</h1>
      <h3>{joke.punchline}</h3>
      <button onClick={fetchJoke} >Generate Joke </button>

      </div>
   )}
export default Randomjokeapi
```

Note: OnClick : Joke will fetch(*take some time) Then observe the effect of SetInterval.

Example: Create a react program to generate random Dog Image using API.

```
import React,{ useState,useEffect } from 'react'
import axios from "axios";
const Randomimage = () =>
{
   const[myimg,setimg]=useState("");

   useEffect(() => {
     setInterval(() => {
        axios
        .get('https://dog.ceo/api/breeds/image/random')
        .then((response)=>{console.log(response.data);setimg(response.data);})
        .catch((error)=>{console.error(error);})
     }, 10000)
   },[])

   return(
     <div>
        <img src={myimg.message} alt='image' height={300} width={300}/>
     </div>
   )
}
export default Randomimage
```

- **useEffect**: This hook runs side effects in functional components.
- The empty dependency array [] means the effect runs only once after the initial render.
- **setInterval:** Sets up a function to run every 2 seconds (2000 milliseconds).
- axios.get('https://dog.ceo/api/breeds/image/random'): Makes a GET request to the Do CEO API to fetch a random dog image.
- **then():** Executes when the API call is successful. The response data, which contains the image URL, is logged to the console and stored in the myimg state.
- **catch():** Executes if there's an error with the API call, logging the error to the console.

Miscellaneous Tasks

Task1 : Write a React code to pass message "My {brand}. It is a {color} {model} from {year}." And message will convert details of car by clicking button.

```
import { useState } from "react";

function US6() {
  const [data, setdata] = useState({brand:"Ford",model:"Mustang",year:1964,color:"red"});



  function carChange()
  {
     setdata({brand:"Hyundai",model:"Creta",year:2024,color:"TitanGray"})
//setdata( pr=>({…pr,color: "blue"})) to change only color value from object
  }
  return (
    <>
     <h1>My {data.brand}.</h1>
     <p>
      It is a {data.color} {data.model} from {data.year}.
     </p>
     <button onClick={carChange}>
      Click me
     </button>
    </>
  )
}
export default US6
```

Task2: Change message according to checkbox selection

```
import { useState } from 'react';

export default function MyCheckbox() {
  const [liked, setLiked] = useState(true);
  function handleChange(e) {
    setLiked(e.target.checked);
  }
  return (
    <>
```

```
    <label>
     <input
      type="checkbox"
      checked={liked}
      onChange={handleChange}
     />
     I liked this
    </label>
    <p>You {liked ? 'liked' : 'did not like'} this.</p>
  </>
 );}
```

Task 3 : Declare more than one state variable in the same component. Each state variable is completely independent.

```
import { useState } from 'react';

export default function Form() {
 const [name, setName] = useState('Taylor');
 const [age, setAge] = useState(42);

 return (
  <>
   <input   value={name}  onChange={e => setName(e.target.value)}  />
   <button onClick={() => setAge(age + 1)}>
    Increment age
   </button>
   <p>Hello, {name}. You are {age}.</p>
  </>
 );
}
```

Task 4: This example passes the updater function, so the "+3" button works.

```
import { useState } from 'react';

export default function Counter() {
 const [age, setAge] = useState(42);

 function increment() {
  setAge(a => a + 1);
 }

 return (
  <>
    <h1>Your age: {age}</h1>
    <button onClick={() => {
     increment();
     increment();
     increment();
    }}>+3</button>
    <button onClick={() => {
     increment();
    }}>+1</button>
  </>
 );
}
```