React Hooks

- No Breaking Changes
    - Completely optional
    - 100% backward compatibility with class
    - Hooks are available into function component from 16.8.0
- Purpose
    - Class components are heavy
    - Class is complex
    - Requires Lazy Loading
    - Classes confuse both people and machines
    - Reusability issued because of inheritance
    - Classes uses a state implicitly
    - Class state is complex in configuration and it is difficult to transport data across components.

Note: Hooks are provided as alternative for class component life cycle methods. They are not replacement for classes.

- Whats hooks can do? Where we can use Hooks?
    - React DOM manipulations
    - React DOM server
    - React Test Render
    - React Shallow Renderer
- React provides built-in hooks and also allows to create custom hooks.
- It is not mandatory to implement and use hooks.
- React provides several built-in hooks
    - useToggle
    - useFirestoreQuery
    - useMemoCompare
    - useAsync
    - useRequireAuth
    - useRouter
    - useAuth
    - useEventListener
    - useWhyDidYouUpdate
    - useDarkMode
    - useMedia
    - useLockBodyScroll
    - useTheme
    - useSpring
    - useHistory "react-router-dom"
    - useLocation "react-router-dom"
    - useParams "react-router-dom"
    - useScript
    - useKeyPress
    - useMemo
    - useDebounce
    - useOnScreen

- usePrevious
- useOnClickOutside
- useAnimation
- useEffect
- useState
- useLocalStore
- useHover etc.

useState Hook

- It defines a state for function component, so that your store values and use across requests.

Syntax:

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

const [product, setProduct] = useState({ })
```

useEffect Hook

- It defines the actions to perform on component "mount"
- You can also configure the action for "unmount"

Ex:

CycleDemoComponent.js

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



function SuccessComponent(){
  useEffect(()=> {
      alert("Success Component Mounted");
      return()=>{
        alert("Success Component Unmounted");
      }
    }, [])
  return(
    <h2>Login Success..</h2>
  )
```

```
}

function ErrorComponent(){
   useEffect(()=> {
      alert("Error Component Mounted");
      return()=>{
         alert("Error Component Unmounted");
      }
   }, [])
   return(
      <h2>Invalid Credentials</h2>
   )
}

export default class CycleDemoComponent extends React.Component
{
    constructor(props) {
       super(props);
       this.state = {
          msg: '',
          UserName: '',
          Password: ''
       }
       this.handleSuccessClick = this.handleSuccessClick.bind(this);
       this.handleErrorClick = this.handleErrorClick.bind(this);
       this.ChangeUserName = this.ChangeUserName.bind(this);
       this.PasswordChange = this.PasswordChange.bind(this);
       this.LoginClick = this.LoginClick.bind(this);

    }
    handleSuccessClick(){
```

```
    this.setState({
        msg: <SuccessComponent />
    })
}
handleErrorClick(){
    this.setState({
        msg: <ErrorComponent />
    })
}
ChangeUserName(e){
   this.setState({
       UserName: e.target.value,
       Password: this.state.Password
   })
}
PasswordChange(e){
   this.setState({
       UserName: this.state.UserName,
       Password: e.target.value
   })
}

LoginClick(){
   if(this.state.UserName=="john" && this.state.Password=="admin") {
       this.setState({
           msg: <SuccessComponent />
       })
   }
   else {
       this.setState({
           msg: <ErrorComponent />
```

```
        })
      }
    }

    render(){
      return(
        <div className="container-fluid mt-2">
          <dl>
            <dt>User Name</dt>
            <dd>
              <input type="text" name="UserName" onChange={this.ChangeUserName}/>
            </dd>
            <dt>Password</dt>
            <dd>
              <input type="password" name="Password" onChange={this.PasswordChange}/>
            </dd>
            <button onClick={this.LoginClick}>Login</button>
          </dl>
          <button onClick={this.handleSuccessClick} >Success</button>
          <button onClick={this.handleErrorClick}>Error</button>
          <div>
            <p>{this.state.msg}</p>
          </div>
        </div>
      )
    }
}
```

useContext

- It uses context memory
- Context memory is component memory.
- You can configure a context object to store values.

- You can access and use the values across multiple requests of same component or accessible to child components at any level of hierarchy.
- You can create a context object by using **React.createContext()**
- Context is used as a service for your component.
- Service uses a software design pattern called “Single Ton”
- Component uses service by using “DI” dependency injection mechanism.
- **DI** requires an “**Injector and Provider”**
- **Injector** is used to inject a service into component.
- **Provider [Locator]** is used to provide a value for service.

FAQ: What is difference between a factory and service?

- Factory is a set of functions and values.
- Service is a set of factories.

FAQ: Why can’t you use a factory directly in your component?

- Factory uses a “**Single Call”** mechanism
- Service uses a “**Single Ton”** mechanism

Ex: UseContext

ContextDemoComponent.js

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


var userDetailsContext = React.createContext(null);


export default function ContextDemoComponent(){
    var [userDetails] = useState({
      name: 'john',
      email: 'john@gmail.com'
    })
    return(
      <userDetailsContext.Provider value={userDetails} >
        <div className="container-fluid">
        <h1>Main Component</h1>
         <HomeComponent />
        </div>
      </userDetailsContext.Provider>
```

```
      )
}
function HomeComponent(props) {
    var contextData = useContext(userDetailsContext);
    return(
       <div>
          <h3>Home Component - Hello ! {contextData.name} </h3>
          <InboxComponent />
       </div>
    )
}
function InboxComponent(props) {
   var contextData = useContext(userDetailsContext);
   return(
      <div>
         <h3>Inbox Component</h3>
         <dl>
            <dt>User Name</dt>
            <dd>
               {contextData.name}
            </dd>
            <dt>Email</dt>
            <dd>
               {contextData.email}
            </dd>
         </dl>
      </div>
   )
}
```

Use Reducer

- It is an alternative for “useState”

- It makes data available across requests and components.
- It is an alternative for “Redux”, which is external state library used in JavaScript based application.
- Use this “Reducer” hook to manage data across multi-level requests and at application level.
- Reducer deals with
    - action
    - dispatch
- “action” defines the action to perform.
- “dispatch” defines the data to store.

Ex: ReducerDemoComponent.js

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



const intialState = {count:0};


function reducer(state, action) {
   switch(action.type)
   {
      case 'increment':
      return {count: state.count + 1}
      case 'decrement':
      return {count: state.count - 1}
      default:
      console.log(`Unable to Execute`);
   }
}


export default function ReducerDemoComponent(){
   const [state, dispatch] = useReducer(reducer, intialState);
   return(
      <>
       <div className="container-fluid">
          [{state.count}] Likes
```

```
        <button onClick={()=> dispatch({type:'increment'})} >Like</button>
        <button onClick={()=> dispatch({type:'decrement'})}>Dislike</button>
      </div>
    </>
  )
}
```

Use Reference

- It is used to configure and create persisted mutable value.
- It is mostly used to manage DOM elements dynamically
    - Disable and Enable
    - Read-Only and Read Write
    - Focus etc.

Syntax:

```
const reference = useRef(initalValue);
const handler() {
    const value = reference.current;
    reference.current = newValue;
}
```

Ex:

```
import { useRef, useEffect } from "react";

export default function ReferenceDemoComponent(){
  const inputRef = useRef();

  useEffect(()=> {
    inputRef.current.focus();
  },[]);

  function handleDisable(){
    inputRef.current.disabled = true;
  }
```

```
  function handleEnable(){
     inputRef.current.disabled = false;
  }

  return(
    <div className="container-fluid">
       Name:
       <input type="text" ref={inputRef} />
       <button onClick={handleDisable} >Disable</button>
       <button onClick={handleEnable}> Enable</button>
    </div>
  )
}
```