React JS

React Hooks

Agenda

01 Problems with Class Components

02 What are React hooks?

03 Functional Components with Hooks

04 Basic Hooks

05 useState() hook

06 useEffect() Hook

07 Rules to use Hooks in React Apps

08 Additional Hooks

09 Building Custom Hooks

Problems with Class Components

Problems with Class Components

Class Components have several issues with them

- Constructor
- Binding
- Sharing Non Visual Logic
- Duplicated Lifecycle Logic

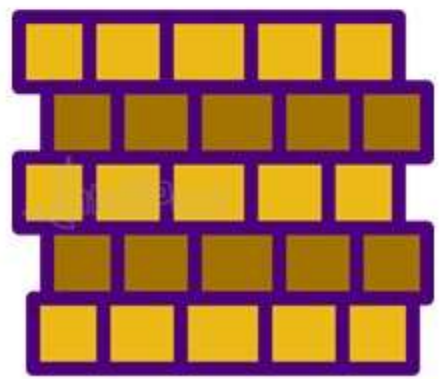
When using class components that uses state we have to create a constructor that calls super method with passed props

Problems with Class Components

Class Components have several issues with them

- Constructor
- Binding
- Sharing Non Visual Logic
- Duplicated Lifecycle Logic

When using React Class Component we would have to bind our methods using call to bind with this keyword passed as arguments to the call



Problems with Class Components

Class Components have several issues with them

- Constructor
- Binding
- Sharing Non Visual Logic
- Duplicated Lifecycle Logic

When we had non visual logic we would have to use complicated patterns like higher order components and render props

Problems with Class Components

Class Components have several issues with them

- Constructor
- Binding
- Sharing Non Visual Logic
- Duplicated Lifecycle Logic

Many a times when we override lifecycle methods we would have to duplicate code between different lifecycle methods

Hands On: Class Component Problems

What are React hooks?

What are React Hooks?

Hooks are React's solution to all the problems introduced by class components in building react components

What are React Hooks?

Before hooks we can only use functional components if we had no state or logic in our component

What are React Hooks?

With hooks we can now create functional components with state and code as lifecycle, as well as create our own hook to share non visual logic

What are React Hooks?

At their core react hooks are functions provided by react to help it know about apps internal state and lifecycle which can be used by functional components

Functional Components with Hooks

Functional Components with Hooks

Before hooks functional components were only used for presentational components which are also called dumb components

Functional Components with Hooks

But with hooks we have the ability to incorporate state and lifecycle in our functional components and reduce dependence on class components

Functional Components with Hooks

Using hooks we can build our react apps more declaratively and more effectively

Basic Hooks

Basic Hooks

Hooks are a new addition to react and are only available in react from version 16.8 and above

Basic Hooks

There are several hooks available in react ranging from using local state in component to creating state which can be shared globally among multiple components

Basic Hooks

Two of the most common and useful hooks are useState and useEffect hook

useState() hook

useState() Hook

In react use useState hook in order to enable our functional react components have internal state which is managed by react

useState() Hook

The useState function takes some internal state and returns a state variable and a function to set that variable in future, these are returned as an array

useState() Hook

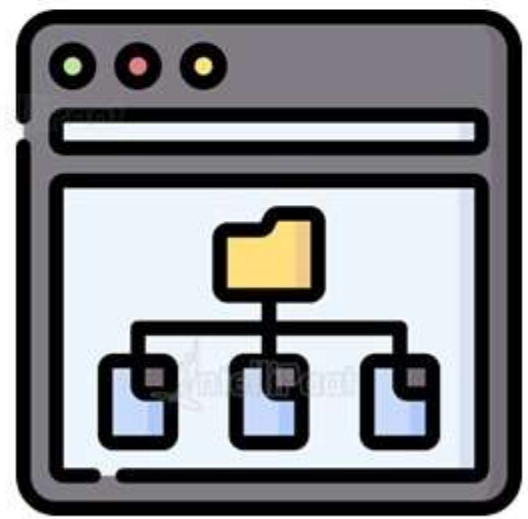
The state variable is monitored by react which will cause a re-render when it changes and it should only be changed by setter function returned by useState

Hands On: useState Hook

useEffect() Hook

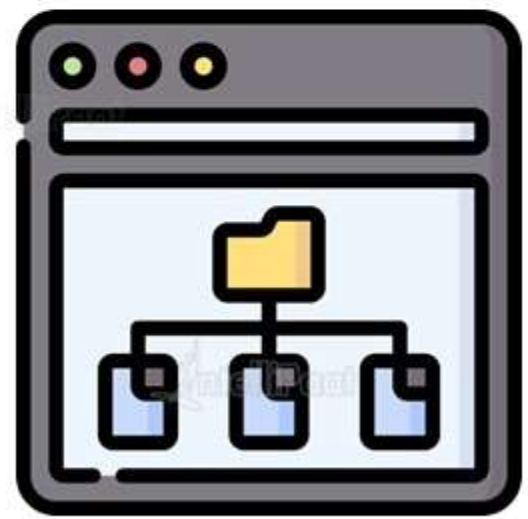
useEffect() Hook

In react use useEffect hook in order to enable our functional react components have lifecycle methods



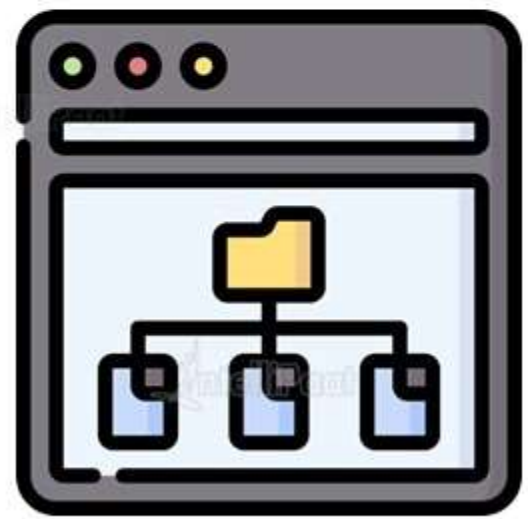
useEffect() Hook

useEffect hook takes in a function and an optional array of variables which when changed will cause the effect function to be executed again



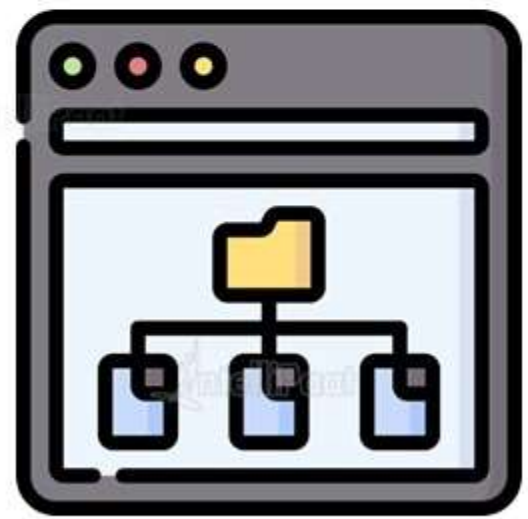
useEffect() Hook

If an empty array is passed as second argument it runs only when component is added to screen, like componentDidMount



useEffect() Hook

Optionally from the function you can return a function to perform any cleanup necessary, like unregistering any event handlers etc.



Hands On: useEffect Hook

Rules to use Hooks in React Apps

Rules to use Hooks in React Apps

To use React Hooks you need to follow certain rules

Top Level Calls

Only call a react hook from top level of a react functional component i.e. do not call hooks from inside a conditional, loop etc.

Calls from React Function

Rules to use Hooks in React Apps

To use React Hooks you need to follow certain rules

Top Level Calls

Call Hooks either from React function components. or from custom Hooks otherwise react will not be able to manage these hooks

Calls from React Function

Additional Hooks

Additional Hooks

There are quite a number of hooks available for us to use other than useState and useEffect, a few of those are:

useContext

useRef

useReducer

In react a context object is special object managed by react which can be used to share data between sibling components

Additional Hooks

There are quite a number of hooks available for us to use other than useState and useEffect, a few of those are:

useContext

useRef

useReducer

The useContext hook allows us to use data from context objects in our components much more easily

Additional Hooks

There are quite a number of hooks available for us to use other than useState and useEffect, a few of those are:

- useContext
- useRef
- useReducer

When we re render a component all the variables declared inside the component will reset

Additional Hooks

There are quite a number of hooks available for us to use other than useState and useEffect, a few of those are:

useContext

useRef

useReducer

useRef allows us to create objects that will persist even after component re-renders and will persist for full lifetime of the component

Additional Hooks

There are quite a number of hooks available for us to use other than useState and useEffect, a few of those are:

useContext

useRef

useReducer

useReducer can be thought of an advanced version of useState, as it allows us to define initial state of an application as well as a reducer function

Additional Hooks

There are quite a number of hooks available for us to use other than useState and useEffect, a few of those are:

useContext

useRef

useReducer

A Reducer function is just a simple function that takes in current state and an action and manipulates state to create new state based on the action

Additional Hooks

There are quite a number of hooks available for us to use other than useState and useEffect, a few of those are:

useContext

useRef

useReducer

For example if initial state is 0 and action is to increment then out function will return 1

Building Custom Hooks

Building Custom Hooks

We can even create custom hooks which will allow us to perform certain tasks and integrate them with react

Building Custom Hooks

We can build these custom hooks by using already available hooks like useState, useEffect etc.

Hands On: Building Custom Hooks

India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

support@intellipaat.com

24/7 Chat with Our Course Advisor