**React**

**Day 63rd**

**Use Effect Hook**

The effect hook lets you perform side effects in function components. Like performing network calls interacting with DOM elements, set timeout, set interval, local storage. It is one of the basic hook used in various purpose like data fetching, setting up subscription and manually changing the DOM in react components are all examples of side effects or just effects, you’ve likely performed them in your component’s before.

In every ReactJS based application, a side-effect gets used in one way or another. It helps to manage the unexpected results from the system and provide and efficient output to the user. Also the code puts side effect components separately, especially away from the rendering process.

**When a side effect occurs**

A react side effect occurs when we use something that is outside of the scope of ReactJS in our react components e.g. Web APIs like local storage.

**Some important things**

📝 When we talk about side effects in the context of ReactJS, we are referring to anything that is outside the scope of React

📝 So calling any native Web APIs will be considered as a side effect as it’s not within the react universe.

📝 Making HTTPS request to an external API is another example of a side effect and the list goes on.

📝 We usually manage react side effects inside the useEffect hook (part of the React Hooks API).

**What does it mean by outside of the React Scope**

It means not part of the React framework, for example, the **localstorage** in your browser.

Localstorage is a Web API and not part of the react universe.

On a side note, localstorage is an essential tool for building web applications.

**What is useeffect and why it is called useeffect**

When the core React hooks were added to the library in 2018, many developers were confused by the name of this hook: “useEffect”

What exactly is an effect??

The word effect refers to a functional programming term called a side effect.

But to really understand what a side effect is, we first have to grasp the concept of a **pure function**.

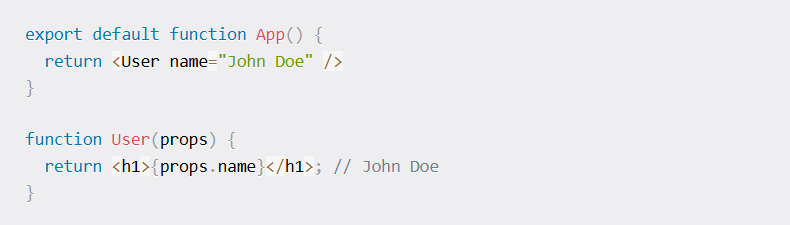
You may not know this, most React components are intended to be pure functions.

It may be strange to think about react components as functions but they are.

It helps to see that a regular react function component is declared like a JS function.

Most React components are pure functions, meaning they receive an inout and produce a predictable output of JSX.

The input of JS functions is arguments. What is the input of react component, however? Props!

Here we have a user component that has the prop name declared on it. Within user, the prop is value is displayed in a header element.

This is pure because, given the same input, it always return the same output.

If we pass user a name prop with value `John Doe`, our output will always be John Doe.

**What are side effects in React??**

Side effects are not predictable because they are actions which are performed with the “outside world”.

We perform a side effect when we need to reach outside of our react components to do something. Performing a side effect, however, will not give us a predictable result.

Think about if we were to request data like blog posts from a server that has failed and instead of our post data, gives us a 500 status code response.

Virtually all applications rely on side effects to work in one way or another, aside from the simplest applications.

Common side effects include:

* Making a request to an API for data from a backend server.
* To interact with browser APIs (that is, to use document or window directly)
* Using predictable timing functions like setTimeout or setInterval.

This is why useeffect exists: to provide a way to handle performing these side effects in what are otherwise pure react components.

**Syntax**

`**useEffect(setup, dependencies)**` here setup is a callback function and dependencies are optional.

**e.g.** import React, { useEffect, useState } from 'react'

const EffectHookExample = () => {

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

    console.log(document.title)

    useEffect(()=>{

        document.title = `This title is coming from functional ${count}`

        console.log(document.title)

    })

  return (

    <div>

        <h1>{count} from hook based component</h1>

        <button onClick={()=>setCount(count+1)}>Increment</button>

    </div>

  )

}

export default EffectHookExample

**Without dependencies equals to render method in class based component**