GIT
https://docs.google.com/document/d/1z8RnQYfGqYrZMuKdZCOWspBCU5odUTyDK69xhC0nNwM/edit

Core Java document link

https://docs.google.com/document/d/15x_FDeZFA5SYM9rp6p_PsnBJRZzcPJgO80thNhccgUI/edit

Why Do We Need to Learn Web Designing ?

- To interact with the data .
- To Create the web pages.
- To Create Static websites.

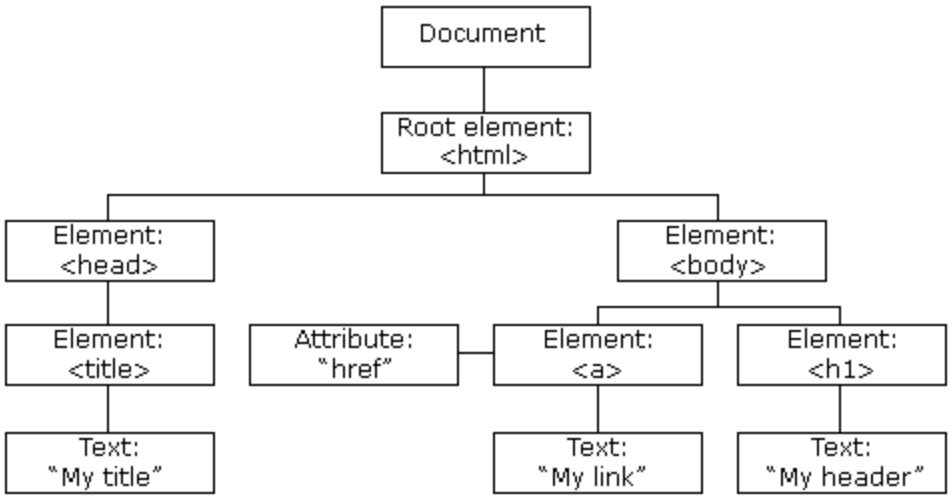
What is Web Designing ..?

- Navigations
- Content
- Web Friendly web pages.
- Visual Design

Web Designing	Web Development
Artistic	Analytical
Creative	Logical
Imaginative	Technical
Holistic (dealing with or treating the whole of something)	Progressive

What is the DOM ?

- DOM stands for ‘Document Object Model’.
- Created by the browser when a web page is loaded.
- In graphical form, it looks like a tree of elements /nodes.
- Programmatically, We can use JS read or change the DOM.
- In simple terms, it is a structured representation of the HTML elements that are present in a webpage or web app.
- DOM represents the entire UI of your application.
- The DOM is represented as a tree data structure.
- It contains a node for each UI element present in the web document.
- It is very useful as it allows web developers to modify content through JavaScript, also it being in structured format helps a lot as we can choose specific targets and all the code becomes much easier to work with.



Disadvantages of real DOM :

- Every time the DOM gets updated, the updated element and its children have to be rendered again to update the UI of our page.
- For this, each time there is a component update, the DOM needs to be updated and the UI components have to be re-rendered.

Example :

```
// Simple getElementById() method

document.getElementById('id').innerValue = 'updated value';
```

When writing the above code in the console or in the JavaScript file, these things happen:

- The browser parses the HTML to find the node with this id.
- It removes the child element of this specific element.
- Updates the element(DOM) with the ‘updated value’.
- Recalculates the CSS for the parent and child nodes.
- Update the layout.
- Finally, traverse the tree and paint it on the screen(browser) display.

Interacting with DOM

- Change / remove HTML elements in the DOM / On the page
- Change & add CSS styles to elements
- Read & Change element attributes (href, src,alt, custom)
- Create new HTML elements and insert them into the DOM / the page.
- Attach event listeners to elements (click, keypress, submit)

- Recalculating the CSS and changing the layouts involves complex algorithms, and they do affect the performance.
- So React has a different approach to dealing with this, as it makes use of something known as Virtual DOM.

Virtual DOM

- React uses Virtual DOM which is like a lightweight copy of the actual DOM(a virtual representation of the DOM).
- So for every object that exists in the original DOM, there is an object for that in React Virtual DOM.
- It is exactly the same, but it does not have the power to directly change the layout of the document.
- Manipulating DOM is slow, but manipulating Virtual DOM is fast as nothing gets drawn on the screen.
- So each time there is a change in the state of our application, the virtual DOM gets updated first instead of the real DOM.

How does virtual DOM actually make things faster?

- When anything new is added to the application, a virtual DOM is created and it is represented as a tree.
- Each element in the application is a node in this tree.
- So, whenever there is a change in the state of any element, a new Virtual DOM tree is created.
- This new Virtual DOM tree is then compared with the previous Virtual DOM tree and makes a note of the changes.
- After this, it finds the best possible ways to make these changes to the real DOM. **Now only the updated elements will get rendered on the page again.**

How does virtual DOM help React?

- In React, everything is treated as a **component**, be it a functional component or class component.
- A component can contain a state. Whenever the state of any component is changed, react updates its Virtual DOM tree.

- Though it may sound like it is ineffective the cost is not much significant as updating the virtual DOM doesn't take much time.

- React maintains two Virtual DOM at each time,
- one contains the updated Virtual DOM and
- one which is just the pre-update version of this updated Virtual DOM.

- Now it compares the pre-update version with the updated Virtual DOM and figures out what exactly has changed in the DOM like which components have been changed.

- This process of comparing the current Virtual DOM tree with the previous one is known as **‘diffing’**.

- Once React finds out what exactly has changed then it updates those objects only, on real DOM.

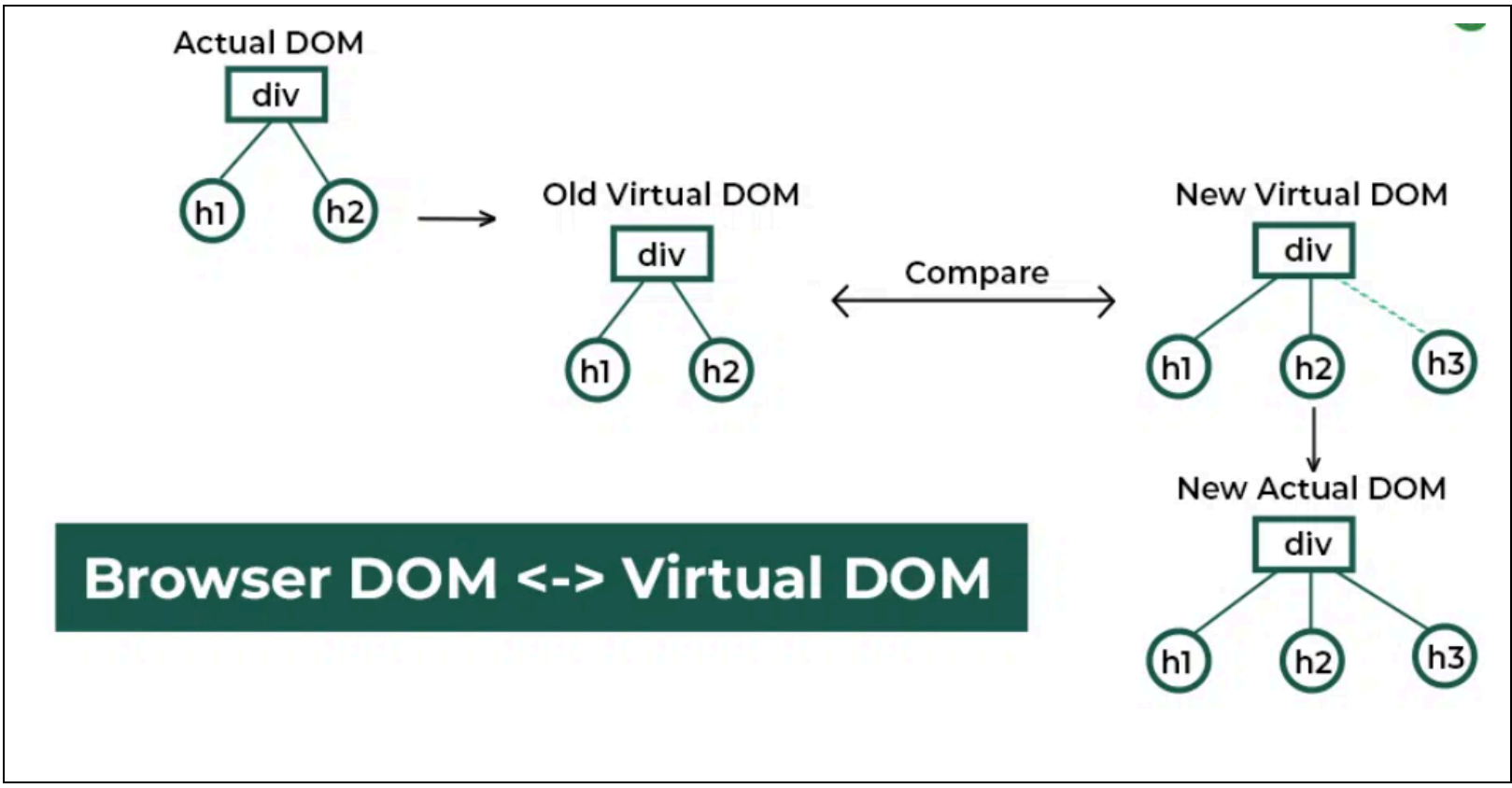
- React uses something called batch updates to update the real DOM.

- It just means that the changes to the real DOM are sent in batches instead of sending any update for a single change in the state of a component.

- We have seen that the re-rendering of the UI is the most expensive part and React manages to do this most efficiently by ensuring that the Real DOM receives batch updates to re-render the UI. This entire process of transforming changes to the real DOM is called **Reconciliation**.

- This significantly improves the performance and is the main reason why React and its Virtual DOM are much loved by developers all around.

- The diagrammatic image below briefly describes how the virtual DOM works in the real browser environment



Virtual DOM Key Concepts :

- Virtual DOM is the virtual representation of Real DOM
- React update the state changes in Virtual DOM first and then it syncs with Real DOM.
- Virtual DOM is just like a blueprint of a machine, can do changes in the blueprint but those changes will not directly apply to the machine.
- Virtual DOM is a programming concept where a virtual representation of a UI is kept in memory synced with “Real DOM ” by a library such as ReactDOM and this process is called **reconciliation**
- Virtual DOM makes the performance faster, not because the processing itself is done in less time.
- The reason is the amount of changed information – rather than wasting time on updating the entire page, you can dissect it into small elements and

interactions.

Why React JS

- Created and Maintained by Facebook.
- More than 100k starts on GitHub.
- Huge Community.(you will find most of the articles to find your problems.)
- Very Demand Skill Set.
- React is Component Based Architecture.
- React lets you compose complex UI from small and isolated pieces of code called components.
- (Header, Main, Content, SideNav, Footer)
- Composing in the right way and right usable code.
- Reusable code.
- We can use it in the articles below.

  React JS Angular JS Vue JS
- Seamlessly integrate react into any of your applications.
- Portion of your page or a complete page or even an entire application itself.
- (React will fit in right place)
- React native for Mobile applications.

  The bottomline is that React has a great skill set.

Ex: Now, React also makes no assumptions about the rest of the technology stack you're using for your project.

React Introduction

- React = **Declarative** UI Programming.
- React also known as React.js or ReactJS
- React is a free and open-source **front-end JavaScript library** for building user interfaces based on **components**.
- It is maintained by Meta (formerly Facebook) and a community of individual developers and companies.
- React can be used to develop single-page, mobile, or server-rendered applications with frameworks like Next.js.
- Because React is only concerned with the user interface and rendering components to the DOM, React applications often rely on libraries for routing and other client-side functionality.
- A key advantage of React is that it **only rerenders those parts of the page that have changed**, avoiding unnecessary rerendering of unchanged DOM elements.
- React is a **declarative**, efficient and Flexible JavaScript Library.
- React Lifecycle is Major important concept in UI.

Prerequisites

- HTML, CSS and Javascript fundamentals.
- ES6.(ECMAScript) (knowledge on modern Javascript is essential)
- javascript - 'this' keyword, filter, map and reduce methods.

- ES6 - let & cost, arrow functions, template literals, default parameters, Object Literals, rest and spread operators and destructuring assignments.
- We will be Learning React from Scratch anyway.

Node JS Interactive (Node + IDE)

Step 1: check in cmd prompt with the command like node -v
Step 2: Downloading the Node.js '.msi' installer.
Step 3: Running the Node.js installer.
Step 4: Verify that Node.js was properly installed or not again.

PATH :
C:\Users\{username}\AppData\Roaming\npm
C:\Program Files\{path to the node js folder}

PATH :
C:\Users\admin\AppData\Roaming\npm
C:\Program Files\nodejs

Note: After adding to the PATH, restart the command line, because PATH is only loaded when initializing new command line sessions.]
Step 5: Updating the Local npm version

- The final step in node.js installed is the update of your local npm version(if required) – the package manager that comes bundled with Node.js.

You can run the following command, to quickly update the npm

npm install npm –global // Updates the 'CLI' client

Step 6 : Download and install Visual Studio code

https://code.visualstudio.com/

Hello World React app.

- We will be using **create-react-app** (is a Command Line Interface tool that allows you quickly to create and run react applications with no configuration, if you simply run a command and it will use entire project is create for you)
- Steps to Create helloworld app.
- Below are the Commands to Create React app.

1) To Check version	node -v
2) To Check npm version	npm -version
3) To install npm with nx	npm install nx -g
4) To Create React app	npx create-react-app <project_name>
5) To Redirect your workspace	cd helloworld
6) To start npm	npm start.

The above commands we must do in Terminal in visual studio code.
webpack compiled successfully !!

Update edit environment variables with path :
C:\Users\91997\AppData\Roaming\npm

1st approach
-npx is an npm package runner, which gets installed when you installed node and that is how we are directly able to run create-react-app without having to install it, npx takes care of it.

2nd approach

- npm
- In this approach we are creating apps globally and then used to package generate projects globally.
- npm install create-react-app -g
- create-react-app <project_name>

Recommended approach is the 1st one, because we do not need to create the packages globally !!

After strats npm -- it will open Browser with local host as like --> http://localhost:3000/
and it will show --> Edit src/App.js and save to reload.

Folder Structure

- package.json
  These files contain dependencies and the scripts.
- node_modules
- which is the place where all the dependencies are installed, it is generated

when you run **create-react-app** or when you run npm install.

- public folder
- manifest.json-- is out of scope.
- favicon.ico -- Its Icon from react.
- index.html -- this is only html you are going to have in your application, you are building single page applications.

- The view changes dynamically in the browser but this html file serves out.
- Typically we are not going to write any code to this file, there may be some changes in the **head tag** but definitely not in the **body tag**.
- you want to use React to control the UI and for that purpose we have one "**div**" tag.

JavaScript Variables :

- users can declare a variable using three keywords that are **var, let, and const**.
- The behavior and the scope of a variable are also based on the keyword used to define it.

- The **var** is the oldest keyword to declare a variable in JavaScript.
- It has the Global scoped or function scoped which means

variables defined outside the function can be accessed globally, and variables defined inside a particular function can be accessed within the function.

var a = 10	function f() {	var a = 10

function f() {
  var b = 20
  console.log(a, b)
}
f();
console.log(a);
	// It can be accessible any
	// where within this function
	var a = 10;
	console.log(a)
	}
	f();
	
	// A cannot be accessible
	// outside of function
	console.log(a);
		// User can re-declare
		// variable using var
		var a = 8
		
		// User can update var variable
		a = 7
		console.log(a);
		
		—----------
		console.log(a);
		var a = 10

The let keyword is an improved version of the var keyword.
It was introduced in ES6 or EcmaScript 2015.
These variables have the block scope.
It can't be accessible outside the particular code block ({block}).

```
let a = 10;
function f() {
    let b = 9
    console.log(b);
    console.log(a);
}
f();


//9
//10

—----------------

console.log(a);
let a = 10;

Uncaught
ReferenceError:
Cannot access 'a'
before initialization
```

```
let a = 10;
function f() {
    if (true) {
        let b = 9

        // It prints 9
        console.log(b);
    }

    // It gives error as it
    // defined in if block
    console.log(b);
}
f()

// It prints 10
console.log(a)

//9
ReferenceError: b is not
defined
```

```
let a = 10

// It is not allowed
let a = 10

// It is allowed
a = 10

//Uncaught SyntaxError:
Identifier 'a' has already
been declared

—----------------------

let a = 10
if (true) {
    let a = 9
    console.log(a) // It prints
9
}
console.log(a) // It prints
10
```

Notable features

- Declarative
  - React adheres to the declarative programming paradigm.
  - (a pattern or model)
  - Developers design views for each state of an application, and React updates and renders components when data changes.
  - This is in contrast with imperative programming.

imperative	**Declarative**
How to do things.	What to do.
Statements.	Expressions.
Defining variables and changing their values.	Evaluate results based on input.

- **Components**
  - React code is made of entities called components.
  - These components are modular and reusable.
  - React applications typically consist of many layers of components.
  - The components are rendered to a root element in the DOM using the React DOM library.
  - When rendering a component, values are passed between components through **props** (short for "properties").
  - Values internal to a component are called its **state**.
  - The two primary ways of declaring components in React are through

function components and

class components.

- **Function components**
- Function components are declared with a function (using JavaScript function syntax or an arrow function expression) that accepts a single "**props**" argument and returns JSX.
- From React v16.8 onwards, function components can use **state** with the **useState Hook.**
- Unlike Class Components (which utilize the ES6 Classes that act as
- "special function" to simulate the classes in OO programming),
- Functional Components are just plain old functions and can be written two different ways.

- Functional components are easier to read and test because they are just plain JavaScript functions without state or lifecycle-methods.

- We cannot call setState inside of them but can use hooks, like
  - useState and useEffect instead.
  - They are also known as stateless components

```
function Meow(props) {
  return (
    <div>
      {props.cat} says, "Meow"
    </div>
  )
```

```
}
//or, using the arrow functions introduced in ES6
const Meow = (props) => {
  return (
    <div>
      {props.cat} says, "Meow"
    </div>
  )
}
```

- **React Hooks**
- On February 16, 2019, React 16.8 was released to the public, introducing React Hooks.
- Hooks are functions that let developers "hook into" React state and lifecycle features from function components.
- Notably, Hooks do not work inside classes — they let developers use more features of React without classes.
- React provides several built-in hooks such as
    - **useState**,
    - **useContext**,
    - **useReducer**,
    - **useMemo** and **useEffect**.
    - useState and useEffect, which are the most commonly used, are for controlling state and side effects, respectively.

Rules of hooks

- There are two rules of hooks, which describe the characteristic code patterns that hooks rely on:

- "Only call hooks at the top level" — **don't call hooks from inside loops**, conditions, or nested statements so that the hooks are called in the same order each render.
- "Only call hooks from React functions" — **don't call hooks from plain JavaScript functions** so that stateful logic stays with the component.
-

  Although these rules can't be enforced at runtime, code analysis tools such as linters can be configured to detect many mistakes during development.

- The rules apply to both usage of Hooks and the implementation of custom Hooks which may call other Hooks.

- Server components
- **React server components** or "RSC"s are function components that run exclusively on the server.

- The concept was first introduced in the talk Data Fetching with Server Components Though a similar concept to Server Side Rendering, RSCs do not send corresponding JavaScript to the client as no hydration occurs.

- As a result, they have no access to hooks.
- However, they may be asynchronous function, allowing them to directly perform asynchronous operations:

```
async function MyComponent() {
```

```
 const message = await fetchMessageFromDb();

 return (

   <div>Message: {message}</div>

 );

}
```

Currently, server components are most readily usable with Next.js

Class components

- Class components are declared using ES6 classes.
- They behave the same way that function components do, but instead of using Hooks to manage state and lifecycle events, they use the lifecycle methods on the React.Component base class.

```
class Meow extends Component {
  constructor(props){
    super(props)
  }

  render() {
    return (
      <div>
        {props.cat} says, "Meow"
      </div>
```

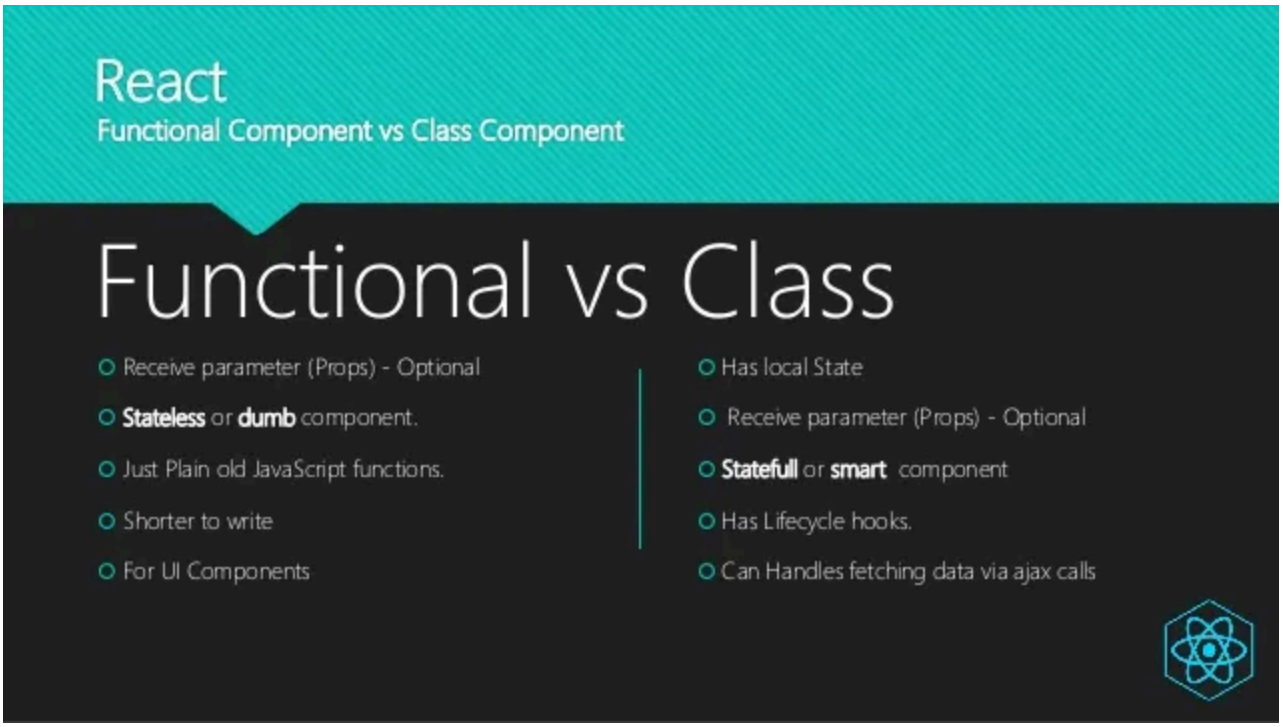
```
    )

  }
}
```

- The introduction of React Hooks with React 16.8 in February 2019 allowed developers to manage state and lifecycle behaviors within functional components, reducing the reliance on class components.
- This trend aligns with the broader industry movement towards functional programming and modular design.
- As React continues to evolve, it's essential for developers to consider the benefits of functional components and React Hooks when building new applications or refactoring existing ones.

- A functional component is just a plain JavaScript pure function that accepts props as an argument and returns a React element(JSX).

- A class component requires you to extend from React. Component and create a render function that returns a React element.
- There is no render method used in functional components.



Routing

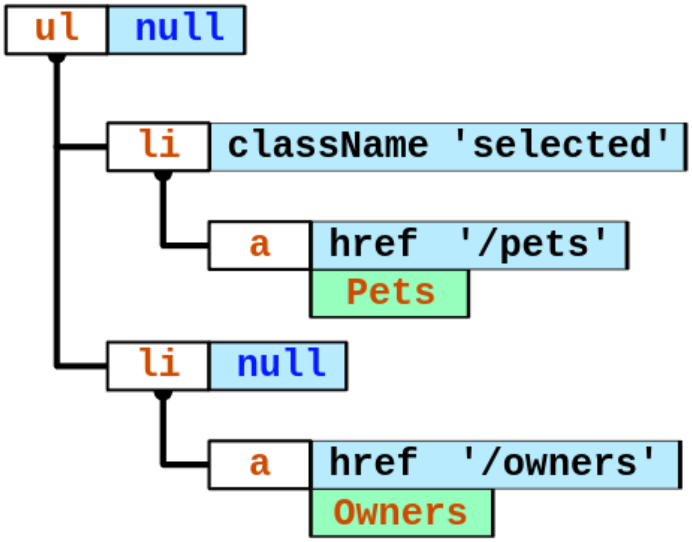
- React itself does not come with built-in support for routing.
- React is primarily a library for building user interfaces, and it doesn't include a full-fledged routing solution out of the box.
- Third-party libraries can be used to handle routing in React applications.
- It allows the developer to define routes, manage navigation, and handle URL changes in a React-friendly way.

Virtual DOM

- Another notable feature is the use of a virtual Document Object Model, or Virtual DOM.
- React creates an in-memory data-structure cache, computes the resulting differences, and then updates the browser's displayed DOM efficiently.
- This process is called reconciliation.

- This allows the programmer to write code as if the entire page is rendered on each change, while React only renders the components that actually change.
  This selective rendering provides a major performance boost.
- There is a Virtual DOM that is used to implement the real DOM.

Virtual DOM



Real DOM

```
<ul>
  <li class="selected">
    <a href="/pets">Pets</a>
  </li>
  <li>
    <a href="/owners">Owners</a>
  </li>
</ul>
```

Updates

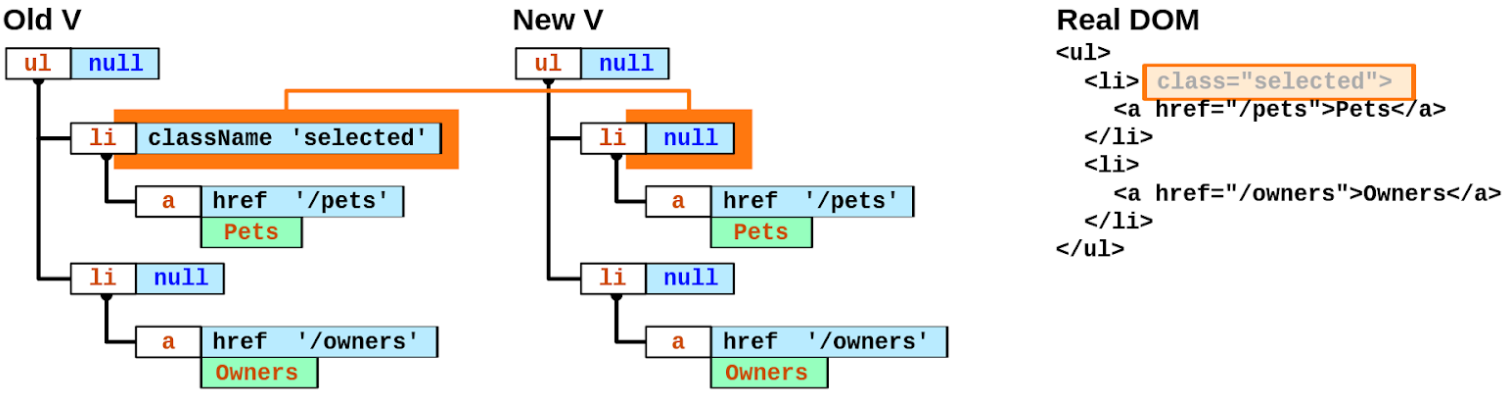
- When ReactDOM.render is called again for the same component and target, React represents the new UI state in the Virtual DOM and determines which parts (if any) of the living DOM needs to change.

Lifecycle methods

- Lifecycle methods for class-based components use a form of hooking that allows the execution of code at set points during a component's lifetime.

- **ShouldComponentUpdate** allows the developer to prevent unnecessary re-rendering of a component by returning false if a render is not required.
- (if it is wide ball or no ball as an example, so by default it will return true like)
- **componentDidMount** is called once the component has "mounted"
- (Will keep one time activities info (all overs info in cricket))
- (the component has been created in the user interface, often by associating it with a DOM node).
- This is commonly used to trigger data loading from a remote source via an API.
- **componentDidUpdate** is invoked immediately after updating occurs.
- (Update instantly like example : every ball in cricket)
- **componentWillUnmount** is called immediately before the component is torn down or "unmounted".
- (After completing the match and updating the final state.)
- This is commonly used to clear resource-demanding dependencies to the component that will not simply be removed with the unmounting of the component (e.g., removing any setInterval() instances that are related to the component, or an "eventListener" set on the "document" because of the presence of the component)
- render is the most important lifecycle method and the only required one in any component. It is usually called every time the component's state is updated, which should be reflected in the user interface.

(For every change of the state of the component and execute render function)



- The virtualDOM will update the realDOM in real-time effortlessly

JSX

- JSX, or JavaScript Syntax Extension, is an extension to the JavaScript language syntax.
- Similar in appearance to HTML,JSX provides a way to structure component rendering using syntax familiar to many developers.
- React components are typically written using JSX, although they do not have to be (components may also be written in pure JavaScript).
- JSX is similar to another extension syntax created by Facebook for PHP called XHP.

```
class App extends React.Component {
  render() {
    return (
      <div>
```

```
      <p>Header</p>
      <p>Content</p>
      <p>Footer</p>
    </div>
  );
 }
}
```

Architecture beyond HTML

- The basic architecture of React applies beyond rendering HTML in the browser.
- For example, Facebook has dynamic charts that render to <canvas> tags,and Netflix and PayPal use universal loading to render identical HTML on both the server and client.

Npm bootstrap

npm i react-bootstrap (install this in terminal and take below examples.)

```
import Jumbotron from 'react-bootstrap/Jumbotron';
import Toast from 'react-bootstrap/Toast';
import Container from 'react-bootstrap/Container';
import Button from 'react-bootstrap/Button';
```

Application 01 (Small app to start!!)

→ Follow the below steps.

→ Create below JS files

index.js
Counter.js
TestCount.js
App.js -> Inside this call Counter app and call TestCount Components.

Step 1 : Create a project from VSC, select Terminal and select new Terminal.

Step 2 : Create project with the name "hello" like below
ex : npx create-react-app

step 3 : And then start the application with the below command inside the hello folder.
ex: npm start

step 4 : Update the changes in App.js and render that in index.js,

index.js root was configured in index.html

step 5 : If you want to create any new components, create one folder name called components and keep all components in the same folder like Counter and TestComponent.

step 6 : In those components use a **useState** hook for initializing the values and use **props** in components.

step 7 : Call the components in App.js instead of changing index.js.

step 8 : start the application and components will be called in one place.

App.js

```
import logo from './logo.svg';
import './App.css';

import Counter from './components/Counter';
import TestComponent from './components/TestComponent';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
```

```
        {/* Accessing Counter component in App.js */}
        <div><Counter /></div>

        {/* Accessing TestComponent in component in App.js */}
        {/* passing the parameter values to the component */}
        <div><TestComponent username="Srikanth" password="React@123"
/></div>
      </header>
    </div>
  );
}

export default App;
```

Counter.js

```
import React, { useState } from 'react';
const Counter = () => {
   //Initializing the state or count variable with 0 value
   const [count, setCount] = useState(5);

   //Initializing the state or mul variable with 1 value
   const [mul, setMul] = useState(1);
```

```
//Creating function for Increment, decrement and multiply count
const add = () => {
    setCount(count + 1);
}

const sub = () => {
    setCount(count - 1);
}

const multiply = () => {
    setMul(mul * 2);
}

return (
    <div>
        <h1>Application info</h1>
        <div>
            <h3>Counter APP</h3>
        </div>
        <div>{count}</div>
        <div>{mul}</div>
```

```
        <div>
          <div> <button onClick={add}>Increment</button></div>
          <div> <button onClick={sub}>Decrement</button></div>
          <div> <button onClick={multiply}>Multiply</button></div>
        </div>
      </div>
  );
}export default Counter;
```

TestComponent.js

```
function TestComponent(props) {
  return (
    <div>
      {/* Accessing information from props */}
      <h1>Welcome {props.username}</h1>
      <h1>Your password is  {props.password}</h1>
    </div>
  );

}
export default TestComponent;
```

Questions from class room 01/05 2024

Q) Can we Create using var or let instead of const for functional components ?

Ans: Var is No but let and const we can use.

- var is not used in modern Javascript and var is not suggestable to use, in react due to the scope being available in function scope, not in block scope.
- It's a drawback from the developer perspective, it is unnecessarily available for remaining blocks so not recommended to use.
- And const is final. We cannot reassign the reference so better to have const is always recommended for the arrow function of const.
- If we use let, there might be a chance of hacking those functional components easily so let is also not recommended until and unless there will be a must situation.

Q) Can we pass parameters directly without using props ? Yes

Ans: Yes we can say, using destructuring props and state.

- The useState hook is a powerful addition to React, introduced in version 16.8.

- It allows you to manage state within functional components without the need for class-based components.
- In this useState Hooks article, we'll learn about useState, its syntax, usage and best practices with examples.

What is useState Hook?

- In React, **useState** is a special function that lets you **add state** to functional components.
- It provides a way **to declare and manage state variables** directly within a function component.

- It should be noted that **one use of useState() can only be used to declare one state variable**. It was introduced in version 16.8.

Why Use useState?

- Functional components are some of the more commonly used components in ReactJS.

- Most developers prefer using **functional components** over class-based components for the simple reason that functional components require less coding (on the developer's part).

However, two main features for the class are lost when one goes with a functional component – a dedicated state that persists through render calls as well as the use of lifecycle functions to control how the component looks and behaves at separate stages of its lifecycle.

1. **Simplicity**: Functional components are preferred due to their concise code. useState simplifies state management.
2. **Avoiding Class Components**: With useState, you can handle state

without converting your component into a class.

Importing the useState Hook

To import the useState hook, write the following code at the top level of your component.

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

Structure of React useState hook

- This hook takes some initial state and returns two values.
- The **first value contains the state** and the **second value is a function that updates the state**.
- The value passed in useState will be treated as the default value.

Syntax:

```
const [var, setVar] = useState(0);
```

Internal working of useState hook

- useState() creates a new cell in the functional component's memory object.
- New state values are stored in this cell during renders.
- The stack pointer points to the latest cell after each render.
- Deliberate user refresh triggers stack dump and fresh allocation.
- The memory cell preserves state between renders, ensuring persistence.

```
// Filename - App.js
```

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

function App() {
    const click = useState('Hello Srikanth');
    return (
        <h1>Welcome  {click}</h1>
    );
}

export default App;
```

Explanation:

- useState() returns an array with initial state value and a function to update it.
- Array destructuring simplifies simultaneous assignment.
- Separate assignment is possible using the array indices.
- Updating state using useState Hook
- To update states using useState we will use the second value passed in useState hook which updates the first value.

Syntax to Update State in useState hook:
setVar(newState);

Example 1

```
// Filename - App.js
Updating React useState Hook State

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

```
function App() {
    const [click, setClick] = useState(0);
    // using array destructuring here
    // to assign initial value 0
    // to click and a reference to the function
    // that updates click to setClick
    return (
        <div>
            <p>You clicked {click} times</p>

            <button onClick={() => setClick(click + 1)}>
                Click me
            </button>
        </div>
    );
}

export default App;
```

Example 2

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

function App() {
    const [click, setClick] = useState(0);
    return (
        <div>
            <p>You've clicked {click} times!</p>
```

```
        <p>The number of times you have clicked
          is {click % 2 == 0 ? 'even!' : 'odd!'}</p>

        <button onClick={() => setClick(click => click + 1)}>
          Click me
        </button>
      </div>
    );
}

export default App;
```

Example 3

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

function App() {
  const [click, setClick] = useState([]);

  const addNumber = () => {
    setClick([
      ...click,
      {
        id: click.length,
        value: Math.random() * 10
      }
    ]);
  };
```

```
  return (
    <div>
      <ul>
        {click.map(item => (
          <li key={item.id}>{item.value}</li>
        ))}
      </ul>
      <button onClick={addNumber}>
        Click me
      </button>
    </div>
  );
}

export default App;
```

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

function App() {
  const [data, setData] = useState({
    username: '',
    password: ''
  });
  const [form, setForm] = useState({
```

```
    username: '',
    password: ''
});
const [submit, submitted] = useState(false);

const printValues = e => {
    e.preventDefault();
    setForm({
        username: data.username,
        password: data.password
    });
    submitted(true);
};

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

return (
    <div>
        <form onSubmit={printValues}>
            <label>
                Username:
                <input
                    value={data.username}
                    name="username"
                    onChange={updateField}
```

```
            />
          </label>
          <br />
          <label>
            Password:
            <input
              value={data.password}
              name="password"
              type="password"
              onChange={updateField}
            />
          </label>
          <br />
          <button>Submit</button>
        </form>

        <p>{submit ? form.username : null}</p>

        <p>{submit ? form.password : null}</p>
      </div>
    );
}

export default App;
```

Simple Application 02

1) App.js

```
import logo from './logo.svg';
import {useState} from 'react';
import './App.css';

function App() {

  const content = [
    [
      "React is extremely popular",
      "It makes building complex, interactive UIs a breeze",
      "It's powerful & flexible",
      "It has a very active and versatile ecosystem"
    ],
    [
      "Components, JSX & Props",
      "State",
      "Hooks (e.g., useEffect())",
      "Dynamic rendering"
    ],
```

```
  [
    "Official web page (react.dev)",
    "Next.js (Fullstack framework)",
    "React Native (build native mobile apps with React)"
  ],
  [
    "Java is Simple",
    "Java is Robust",
    "Java is Multithreaded"
  ]
  ];


const [activeContentIndex, setActiveContentIndex] = useState(0);
return (
  <div className="App">
    <header className="App-header">
      <img src={logo} className="App-logo" alt="logo" />
      <div>
        <h1>React.js</h1>
        <p>i.e., using the React library for rendering the UI</p>
      </div>
```

```
<div id="tabs">
  <menu>
    <button
      className={activeContentIndex === 0 ? "active" : ""}
      onClick={() => setActiveContentIndex(0)}
    >
      Why React?
    </button>

    <button
      className={activeContentIndex === 1 ? "active" : ""}
      onClick={() => setActiveContentIndex(1)}
    >
      Core Features
    </button>
    <button
      className={activeContentIndex === 2 ? "active" : ""}
      onClick={() => setActiveContentIndex(2)}
    >
      Related Resources
    </button>
```

```
      <button
        className={activeContentIndex === 3? "active" : ""}
        onClick={() => setActiveContentIndex(3)}
      >
        What is Java ?
      </button>
    </menu>

    <div id="tab-content">
      <ul>
        {content[activeContentIndex].map((item) => (
          <li key={item}>{item}</li>
        ))}
      </ul>
    </div>
  </div>



  </header>
</div>
```

```
  );
}

export default App;
```

App.CSS

```
* {
  box-sizing: border-box;
}

body {
  font-family: sans-serif;
  background-color: #181c1f;
  color: #bdd1d4;
  margin: 2rem;
}

header {
  margin: 2rem 0;
```

```
  display: flex;
  gap: 1.5rem;
  align-items: center;
}

header img {
  width: 3rem;
  object-fit: contain;
}

header h1 {
  margin: 0;
  color: #48d9f3;
}

header p {
  margin: 0;
  color: #82c2ce;
}

#tabs {
  max-width: 32rem;
```

```
  margin: 2rem 0;
  overflow: hidden;
}

#tabs menu {
  margin: 0;
  padding: 0;
  display: flex;
  gap: 0.25rem;
}

#tabs button {
  font: inherit;
  font-size: 0.85rem;
  background-color: #282f33;
  border: none;
  border-bottom-color: #48d9f3;
  color: #e0eff1;
  border-radius: 4px 4px 0 0;
  padding: 0.75rem 1rem;
  cursor: pointer;
  transition: all 0.2s ease-out;
```

```
}

#tabs button:hover,
#tabs button.active {
  background-color: #48d9f3;
  color: #273133;
}

#tab-content {
  background-color: #2d3942;
  border-radius: 0 4px 4px 4px;
  padding: 1rem;
}

#tab-content li {
  margin: 0.75rem 0;
}
.App {
  text-align: center;
}

.App-logo {
```

```
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
```

```
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }

  to {
    transform: rotate(360deg);
  }
}
```

After Updating with Reusable Components

```
import { useState } from "react";
import '../App.css';

const menuOptions = [
    {
      id: 0,
      name: 'Why React?'
    },
    {
      id: 1,
      name: 'J2SE'
    },
    {
```

```
    id: 2,
    name: 'J2EE'
  },
  {
    id: 3,
    name: 'Frameworks'
  },
  {
    id: 4,
    name: 'Web UI'
  }

];

const content = [
  [
    "React is extremely popular",
    "It makes building complex, interactive UIs a breeze",
    "It's powerful & flexible",
    "It has a very active and versatile ecosystem"
  ],
  [
    "Core Java ",
    "Opps Through all concepts"
  ],
  [
    "Servlets",
    "JSP"
  ],
  [
    "Spring",
    "Spring Boot with Micro Services Architecture"
  ],
  [
    "HTML",
    "CSS",
    "JS",
    "Bootstrap",
```

```
      "React"
    ]
];

const Menu = ({ option, activeContentIndex, setActiveContentIndex }) => {
  return (
    <button
      className={activeContentIndex === option.id ? "active" : ""}
      onClick={() => setActiveContentIndex(option.id)} >
      {option.name}
    </button>
  );
}

const DisplayContent = ({ activeContentIndex }) => {
  return (
    <div id="tab-content">
      <ul>
        {content[activeContentIndex].map(
          (item) => (
            <li key={item}>{item}</li>
          ))}
      </ul>
    </div>
  );
}

const Header = () => {
  return (
    <div>
      <h1>React.js</h1>
      <p>i.e., using the React library for rendering the UI</p>
    </div>
  );
}

const Tabs = ({ activeContentIndex, setActiveContentIndex }) => {
  return (
```

```
      <div id="tabs">
        <menu>
          {
            menuOptions.map((option) => <Menu
              option={option}
              activeContentIndex={activeContentIndex}
              setActiveContentIndex={setActiveContentIndex}
            />)
          }

        </menu>
      </div>
    );
}

const Home = () => {
    const [activeContentIndex, setActiveContentIndex] = useState(0);
    return (
      <div className="App">
        <Header />
        <Tabs activeContentIndex={activeContentIndex} setActiveContentIndex={setActiveContentIndex} />
        <DisplayContent activeContentIndex={activeContentIndex} />
      </div>
    );
}

export default Home;
```

— — — End of Simple app —- —

ListComponent.JSX

```
import React from 'react'
const ListComponent = () => {

  const dummydata = [
    {
      "id": "1",
      "firstname": "Srikanth",
      "lastname": "C",
      "email": "srkntjva@gmail.com"
    },
    {
      "id": "2",
      "firstname": "Venkat",
      "lastname": "C",
      "email": "Venkat@gmail.com"
    },
    {
      "id": "3",
      "firstname": "Nag",
      "lastname": "C",
```

```
        "email": "Nag@gmail.com"
    }

]
return (
    <div className='container'>
        ListComponent

        <h1>List of employees </h1>
        <table className='table table-striped table-bordered'>
            <thead>
                <tr>
                    <th>Employee ID</th>
                    <th>Employee First Name</th>
                    <th>Employee Last Name</th>
                    <th>Employee Email</th>
                </tr>
                <tr>
                </tr>
            </thead>
            <tbody>
                {
```

```
                dummydata.map(s =>
                    <tr key={s.id}>
                        <td>{s.id}</td>
                        <td>{s.firstname}</td>
                        <td>{s.lastname}</td>
                        <td>{s.email}</td>

                    </tr>
                )
            }
        </tbody>
    </table>
</div>
)
}

export default ListComponent
```

What is the useEffect hook in React?

- The useEffect in ReactJS is used to handle the side effects such as fetching data and updating DOM.

- This hook runs on every render but there is also a way of using a **dependency array** using which we can control the effect of rendering.

Why choose **useEffect** hook?
useEffect hook is used to handle side effects in functional components, such as fetching data, updating the DOM, and setting up subscriptions or timers.

It is used to mimic the lifecycle methods of class-based components.

The motivation behind the introduction of useEffect Hook is to eliminate the side effects of using class-based components.

For example, tasks like updating the DOM, **fetching data from API end-points**,
setting up subscriptions or timers, etc can lead to unwarranted side effects.

Since the render method is too quick to produce a side-effect, one needs to use life cycle methods to observe the side effects.

How does it work?
You call useEffect with a **callback function** that contains the side effect logic.

By default, this function runs after every render of the component.

You can optionally provide a **dependency array** as the second argument.

The effect will only run again if any of the values in the dependency array change.

Importing useEffect hook

- To import the useEffect hook, write the following code at the top level of your component

import { useEffect } from "react";

Structure of useEffect hook

The useEffect hook syntax accepts two arguments where the second argument is optional.

React useEffect Hook Syntax:
useEffect(<FUNCTION>, <DEPENDENCY>)

React useEffect Hook ShortHand for:

FUNCTION: contains the code to be executed when useEffect triggers.

DEPENDENCY: is an optional parameter, useEffect triggers when the given dependency is changed.
Controlling side effects in useEffect :
1. To run useEffect **on every render** do not pass any dependency

```
useEffect(()->{
   // Example Code
})
```

2. To run useEffect **only once on the first render pass any empty array in the dependency

```
useEffect(()->{
   // Example Code
}, [] )
```

3. To run useEffect **on change of a particular value**. Pass the state and props in the dependency array

```
useEffect(()->{
   // Example Code
}, [props, state] )
```

---

useEffect Example HookCounterOne

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

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

```
    useEffect(() => {
        document.title = `You clicked ${count} times`;
    }, [count]);

    return (
        <div>
            <button onClick={() => setCount((prevCount) => prevCount + 1)}>
                Click {count} times{" "}
            </button>
        </div>
    );
}
```

Output: Initially, the document title reads “You clicked 0 times”.
when you click on the button, the count value increments, and the document title is updated.

---

useEffect triggers a function on every component render, leveraging React to execute specified tasks efficiently.

Positioned within the component, it grants easy access to state and props without additional coding.

For replicating lifecycle methods in functional components, copy and customize the provided code snippet according to your needs.
export default HookCounterOne;

Ways to mimic lifecycle methods using useEffect hook

We know that the useEffect() is used for causing side effects in functional components and it is also capable of handling componentDidMount(), componentDidUpdate(), and componentWillUnmount() life-cycle methods of class-based components into the functional components.

For **componentDidMount**

```
useEffect(()=>{
   //You can add your code here for mounting phase of component
   console.log("Mounting in Functional Component")
},[])
// adding an empty array ensures that the useEffect is only triggered once
// (when the component mounts)
```

For **componentDidUpdate**

```
useEffect(()=>{
   //You can add your code for updating phase of component
   console.log("Updating in Functional Component")
},[values])
//values triggers re render whenever they are updated in your program,
//you can add multiple values by separating them by commas
```

For **componentWillUnmount**

```
useEffect(()=>{
   return()=>{
   //You can add your code for unmounting phase of component
   console.log("Functional Component Removed ")
```

```
    }
},[])
//Write all the code of unmounting phase only inside the callback function
```

Using useEffect

Simple app 02 using Spring Boot Data.

HelloSpringBootController.java

```
package com.srikanthsb.myfirstsbapp.controller;

import java.util.ArrayList;
import java.util.List;

import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.srikanthsb.myfirstsbapp.model.Employee;

@RestController
@CrossOrigin("*")
public class HelloSpringBootController {

	// http://localhost:9090
	@RequestMapping
	String helloSpringBoot() {
		String str = "Hello Spring Boot Application !!";
```

```
		return str;
	}

//	http://localhost:9090/hi-sb
	@GetMapping("/hi-sb")
	public String hello() {
		return "Hello Spring Boot app or Micro Sercices !!";
	}

//	/http://localhost:9090/getItems
	@GetMapping("/getItems")
	public String getItems() {
		return " [\r\n" + "    {\r\n" + "        \"id\": 0,\r\n" + "        \"name\":
\"Why React?\",\r\n"
				+ "        \"content\": [\r\n" + "            \"React is extremely
popular\",\r\n"
				+ "            \"It makes building complex, interactive UIs a
breeze\",\r\n"
				+ "            \"It's powerful & flexible\",\r\n"
				+ "            \"It has a very active and versatile
ecosystem\"\r\n" + "        ]\r\n" + "    },\r\n"
				+ "    {\r\n" + "        \"id\": 1,\r\n" + "        \"name\": \"Core
Features\",\r\n"
				+ "        \"content\": [\r\n" + "            \"Components, JSX &
Props\",\r\n"
				+ "            \"State\",\r\n" + "            \"Hooks (e.g.,
useEffect())\",\r\n"
				+ "            \"Dynamic rendering\"\r\n" + "        ]\r\n" + "
},\r\n" + "    {\r\n"
				+ "        \"id\": 2,\r\n" + "        \"name\": \"Related
```

```
Resources\",\r\n"
                        + "       \"content\": [\r\n" + "           \"Official web page
(react.dev)\",\r\n"
                        + "           \"Next.js (Fullstack framework)\",\r\n"
                        + "           \"React Native (build native mobile apps with
React)\"\r\n" + "       ]\r\n"
                        + "    },\r\n" + "       {\r\n" + "       \"id\": 3,\r\n" + "
\"name\": \"Java\",\r\n"
                        + "       \"content\": [\r\n" + "           \"Java is Simple\",\r\n"
                        + "           \"Java id Robust\",\r\n" + "           \"Java is
Secure\"\r\n" + "       ]\r\n"
                        + "    }\r\n" + "]";
        }

//
//      @GetMapping("/employee")
//      Employee getEmployee() {
//              return new Employee("Rahul", "Reddy ");
//      }
//
        @GetMapping("/employees")
        List<Employee> getEmployees() {
                List<Employee> empList = new ArrayList<Employee>();
                empList.add(new Employee("Aditya", "A"));
                empList.add(new Employee("Kruthik", "k"));
                empList.add(new Employee("Naina", "Shaik"));
                empList.add(new Employee("Zaheer", "s"));
                empList.add(new Employee("Manga", "m"));
                return empList;
        }
```

```
}
```

App.jsx

```
import logo from './logo.svg';
import { useEffect, useState } from "react";
import './App.css';


function App() {

  const [activeItemId, setActiveItemId] = useState(0);
  const [items, setItems] = useState([]);

  //useEffect will take first parameter as arrow function or any function,
  // second parameter empty array will take and it will execute only once.
  //it is a replica of componentDidMount which executes only once in this
situation.

  // componentDidupdate - for every update it will render, added activeItemId - },
[activeItemId],)
```

```
useEffect(() => {
  //need to connect service only once
  fetch('http://localhost:9090/getItems').then((response) => {
    response.text().then((items) => {
      console.table(JSON.parse(items));
      setItems(JSON.parse(items));
    });
  })
}, [])

return (
  <div className="App">
    <header className="App-header">
      <img src={logo} className="App-logo" alt="logo" />
      <div>
        <h1>React.js</h1>
        <p>i.e., using the React library for rendering the UI</p>
      </div>
    </header>

    <div id="tabs">
```

```
    <menu>
      {
        items && items.map(item => {
          return (
            <button className={activeItemId === item.id ? "active" : ""}
              onClick={() => setActiveItemId(item.id)}> {item.name} </button>
          );
        })
      }

    </menu>
  </div>

  <div id="tab-content">
    <ul>
      {
        items && items.find((item) => item.id === activeItemId)
          ?.content?.map(item => <li key={item.id}> {item} </li>)
      }
    </ul>
  </div>
</div>
```

```
  );
}

export default App;
```

– End of Application 2 —-

What is React Context?

React Context is a method to pass props from parent to child component(s), by storing the props in a store and using these props from the store by child component(s) without actually passing them manually at each level of the component tree.

Why React Context? We have Redux!!

Using Redux to interact with states from parent to child components is not only quite difficult to understand but also gives you a more complex code.
Through the usage of Context, the understanding of concept and code is far easier than that of Redux.

Example:

If we have three components in our app, A->B->C where A is the parent of B and B is the parent of C.

To change a state from C and pass it to A, keep the state of A in a store, then extract the state from the store and use it in C. This completely eliminates the necessity of the state to pass through B. So the flow is like A->C.

//The Problem

State should be held by the highest parent component in the stack that requires access to the state.

To illustrate, we have many nested components.
The components at the top and bottom of the stack need access to the state.

To do this without Context, we will need to pass the state as "props" through each nested component. This is called "**prop drilling**".

// Even though components 2-4 did not need the state, they had to pass the state along so that it could reach component 5.

```
import { useState } from "react";
import ReactDOM from "react-dom/client";

function Component1() {
  const [user, setUser] = useState("Hello Srikanth");

  return (
    <>
      <h1>{`Hello ${user}!`}</h1>
      <Component2 user={user} />
```

```
    </>
  );
}

function Component2({ user }) {
  return (
    <>
      <h1>Component 2</h1>
      <Component3 user={user} />
    </>
  );
}

function Component3({ user }) {
  return (
    <>
      <h1>Component 3</h1>
      <Component4 user={user} />
    </>
  );
}
```

```
function Component4({ user }) {
  return (
    <>
      <h1>Component 4</h1>
      <Component5 user={user} />
    </>
  );
}

function Component5({ user }) {
  return (
    <>
      <h1>Component 5</h1>
      <h2>{`Hello ${user} again!`}</h2>
    </>
  );
}

export default Component1;

// Even though components 2-4 did not need the state, they had to pass the state along so that it could reach component 5.
```

The Solution

```
import { useState, createContext, useContext } from "react";
import ReactDOM from "react-dom/client";

const UserContext = createContext();

function UcComponent1() {
  const [user, setUser] = useState("Hello Srikanth");

  return (
    <UserContext.Provider value={user}>
      <h1>{`Welcome ${user}!`}</h1>
      <UcComponent2 />
    </UserContext.Provider>
  );
}

function UcComponent2() {
  return (
```

```
      <>
        <h1>Component 2</h1>
        <UcComponent3 />
      </>
    );
}

function UcComponent3() {
  return (
      <>
        <h1>Component 3</h1>
        <UcComponent4 />
      </>
    );
}

function UcComponent4() {
  return (
      <>
        <h1>Component 4</h1>
        <UcComponent5 />
      </>
```

```
  );
}

function UcComponent5() {
  const user = useContext(UserContext);

  return (
    <>
      <h1>Component 5</h1>
      <h2>{`Welcome Back  ${user} again!`}</h2>
    </>
  );
}

export default UcComponent1;
```

Use the useContext Hook

In order to use the Context in a child component, we need to access it using the useContext Hook.

First, include the useContext in the import statement:

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

Then you can access the user Context in all components:

```
function Component5() {
  const user = useContext(UserContext);

  return (
    <>
      <h1>Component 5</h1>
      <h2>{`Hello ${user} again!`}</h2>
    </>
  );
}
```

useContextExample

```
import { useState, useContext, createContext } from 'react';
import '../App.css';

const Activecontext = createContext();

const menuOptions = [
    {
        id: 0,
        name: 'Why React?'
    },
    {
        id: 1,
        name: 'Core Features'
    },
    {
        id: 2,
        name: 'Related Resources'
    },
    {
```

```
        id: 3,
        name: 'What is Java ?'
    },
    {
        id: 4,
        name: 'J2SE '
    }

];


const content = [
    [
        "React is extremely popular",
        "It makes building complex, interactive UIs a breeze",
        "It's powerful & flexible",
        "It has a very active and versatile ecosystem"
    ],
    [
        "Components, JSX & Props",
        "State",
        "Hooks (e.g., useEffect())",
        "Dynamic rendering"
    ],
    [
        "Official web page (react.dev)",
        "Next.js (Fullstack framework)",
        "React Native (build native mobile apps with React)"
    ],
    [
        "Java is Simple",
        "Java is Robust",
        "Java is Multithreaded"
    ],
    [
        "Java ",
        "Java ",
        "Java "
    ]
```

```
];

const Menu = ({ option }) => {
    const { activeContentIndex, setActiveContentIndex } = useContext(Activecontext);
    return (
        <button
            className={activeContentIndex === option.id ? "active" : ""}
            onClick={() => setActiveContentIndex(option.id)} >
            {option.name}
        </button>
    );
}

const DisplayContent = () => {
    const { activeContentIndex } = useContext(Activecontext);
    return (
        <div id="tab-content">
            <ul>
                {content[activeContentIndex].map(
                    (item, i) => (
                        <li key={i}>{item}</li>
                    ))}
            </ul>
        </div>
    );
}

const Tabs = () => {
    return (
        <div id="tabs">
            <menu>
                {
                    menuOptions.map(
                        (option) =>
                            <Menu
                                option={option}
                            />)
                }
            </menu>
```

```
        </div>
    );

}


const Home = () => {
    const [activeContentIndex, setActiveContentIndex] = useState(0);
    return (
        <Activecontext.Provider value={{ activeContentIndex, setActiveContentIndex }}>

            <div className="App">
                <header className="App-header">

                    <Tabs />
                    <DisplayContent />

                </header >
            </div >
        </Activecontext.Provider>
    );
}


export default Home;
```

useReducer

The useReducer Hook is similar to the useState Hook.

It allows for custom state logic.

If you find yourself keeping track of multiple pieces of state that rely on complex logic, useReducer may be useful.

Syntax
The useReducer Hook accepts two arguments.
useReducer(<reducer>, <initialState>)

The **reducer function** contains your custom state logic and the **initialState** can be a **simple value** but generally will contain an **object**.

The **useReducer** Hook returns the **current state** and a **dispatch** method.

Here is an example of useReducer in a counter app:
The reducer function contains your custom state logic and the initialState can be a simple value but generally will contain an object.

The useReducer Hook returns the current state and a dispatch method.
Here is an example of useReducer in a counter app:
Example:Get your own React.js Server

UseRedcuerApp.jsx

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

export default function LoginUseState() {

  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [isLoading, showLoader] = useState(false);
  const [error, setError] = useState('');
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  const onSubmit = async (e) => {
    e.preventDefault();
    setError('');
    showLoader(true);
    try {
      await new Promise((resolve, reject) => {
        setTimeout(() => {
          if (username === 'Srikanth' && password === 'Srikanth123') {
            resolve();
          } else {
```

```
            reject();
          }
        }, 1000);
      });
        setIsLoggedIn(true);

  } catch (error) {
    setError('Incorrect username or password!');
    showLoader(false);
    setUsername('');
    setPassword('');
  }
};
return (
  <div className='App'>
    <div className='login-container'>
      {isLoggedIn ? (
        <>
          <h1>Welcome {username}!</h1>
          <button onClick={() => setIsLoggedIn(false)}>Log Out</button>
        </>
      ) : (
```

```
    <form className='form' onSubmit={onSubmit}>
      {error && <p className='error'>{error}</p>}
      <p>Please Login!</p>
      <input
        type='text'
        placeholder='username'
        value={username}
        onChange={(e) => setUsername(e.currentTarget.value)}
      />
      <input
        type='password'
        placeholder='password'
        autoComplete='new-password'
        value={password}
        onChange={(e) => setPassword(e.currentTarget.value)}
      />
      <button className='submit' type='submit' disabled={isLoading}>
        {isLoading ? 'Logging in...' : 'Log In'}
      </button>
    </form>
  )}
</div>
```

```
    </div>
  );
}
```

UseRedcuerApp1.jsx

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

function loginReducer(state, action) {
  switch (action.type) {
    case 'field': {
      return {
        ...state,
        [action.fieldName]: action.payload,
      };
    }
    case 'login': {
      return {
        ...state,
        error: '',
        isLoading: true,
      };
```

```
  }
  case 'success': {
    return {
      ...state,
      isLoggedIn: true,
      isLoading: false,
    };
  }
  case 'error': {
    return {
      ...state,
      error: 'Incorrect username or password!',
      isLoggedIn: false,
      isLoading: false,
      username: '',
      password: '',
    };
  }
  case 'logOut': {
    return {
      ...state,
      isLoggedIn: false,
```

```
      };
    }
    default:
      return state;
  }
}
const initialState = {
  username: '',
  password: '',
  isLoading: false,
  error: '',
  isLoggedIn: false,
};
export default function LoginUseReducer() {
  const [state, dispatch] = useReducer(loginReducer, initialState);
  const { username, password, isLoading, error, isLoggedIn } = state;

  const onSubmit = async (e) => {
    e.preventDefault();
    dispatch({ type: 'login' });
    try {
      await new Promise((resolve, reject) => {
```

```
        setTimeout(() => {
          if (username === 'Srikanth' && password === 'java') {
            resolve();
          } else {
            reject();
          }
        }, 1000);
      });
    dispatch({ type: 'success' });
  } catch (error) {
    dispatch({ type: 'error' });
  }
};
return (
  <div className='App'>
    <div className='login-container'>
      {isLoggedIn ? (
        <>
          <h1>Welcome {username}!</h1>
          <button onClick={() => dispatch({ type: 'logOut' })}>
            Log Out
          </button>
```

```
    </>
  ) : (
    <form className='form' onSubmit={onSubmit}>
      {error && <p className='error'>{error}</p>}
      <p>Please Login!</p>
      <input
        type='text'
        placeholder='username'
        value={username}
        onChange={(e) =>
          dispatch({
            type: 'field',
            fieldName: 'username',
            payload: e.currentTarget.value,
          })
        }
      />
      <input
        type='password'
        placeholder='password'
        autoComplete='new-password'
        value={password}
```

```
            onChange={(e) =>
              dispatch({
                type: 'field',
                fieldName: 'password',
                payload: e.currentTarget.value,
              })
            }
          />
          <button className='submit' type='submit' disabled={isLoading}>
            {isLoading ? 'Logging in...' : 'Log In'}
          </button>
        </form>
      )}
    </div>
  </div>
 );
}
```