<body>

    <div *id*="root"></div>

    <script>

*// Creating HTML tag using Javascript*

*const* heading = document.createElement("h1");

      heading.innerHTML = "Hello World From Javascript!";

*const* root = document.getElementById("root");

      root.appendChild(heading);

    </script>

  </body>

**Q:-How does browser understand what is document,createElement,getElementById all these things how browser can Understand??**

* Browser has Javascript Engine that exectute this JavaScript. Browser can understand Because of JS Engine.
* But Browser Don’t Understand React, so first we need to get react into our project

**# There is 2 way of adding react to our project**

CDN Links

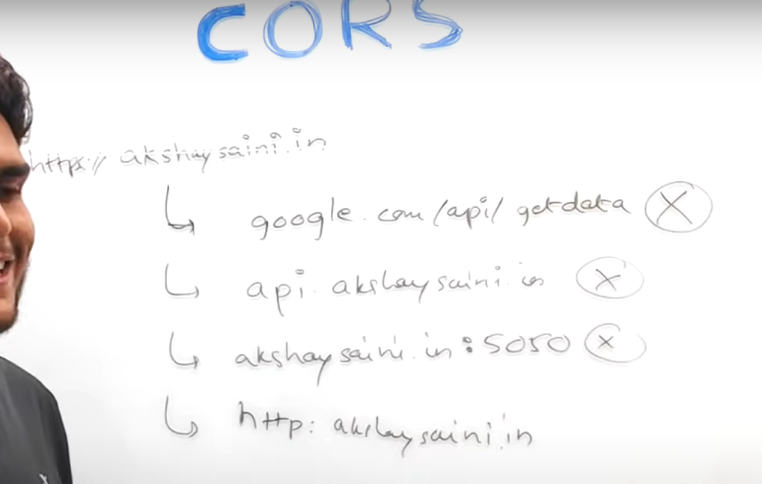
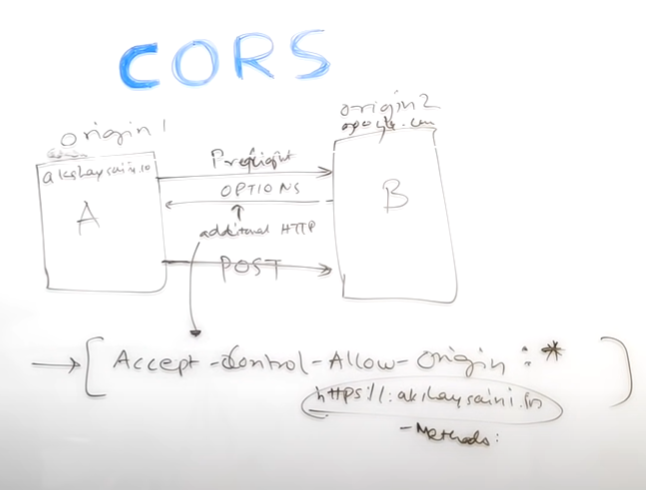
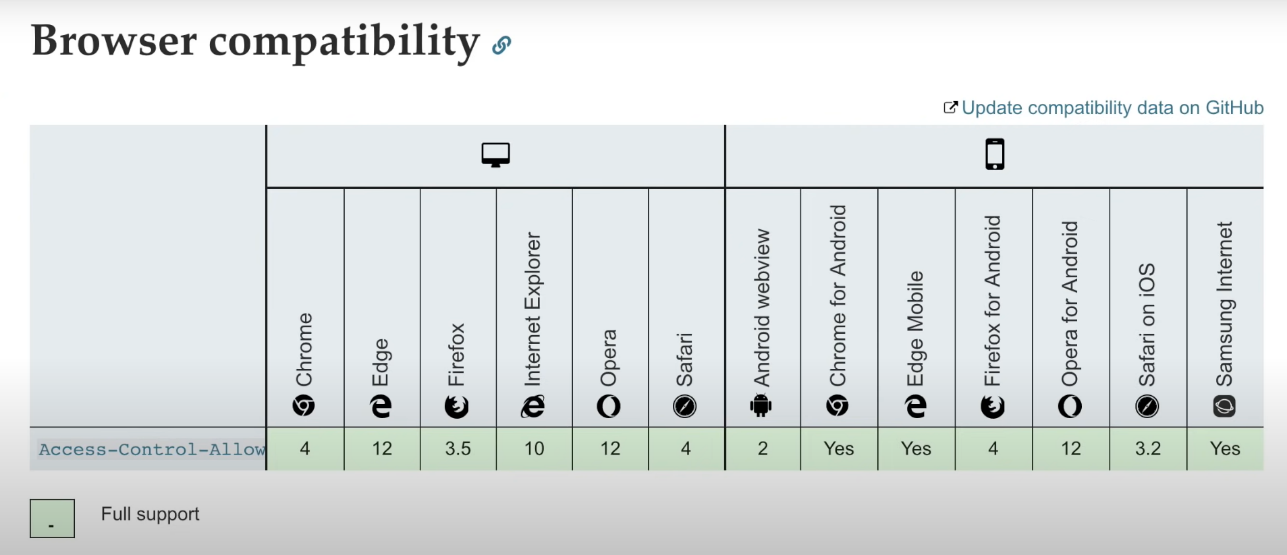
<script *crossorigin* *src*="https://unpkg.com/react@18/umd/react.development.js"></script>

<script *crossorigin* *src*="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

1 and 2 links are different for a reason check BookPage-11

* **Via CDN links**
* react.development.js
* react-dom.development.js
* fetching from CDN is costly Operation, it will make a network call to unpkg.com and it will get react from unpkg.com
* suppose we have already react in my node\_module [more on page 7]
* **Via npm**
* npm install react
* npm install react-dom

**Q:- What is CORS (Cross-Origin Resource Sharing)**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>
* CORS and CORS Headers : <https://developer.mozilla.org/en-US/docs/Glossary/CORS>
* CORS is a mechanism which uses additional HTTP Headers to tell the browser whether a specific web app can share resource with another web app
* But but but the major important point to note over here is that both the web apps should have different origin.
* So suppose if they have same origin then they can share resources very easily but if they don’t have the same origin then they need to follow the CORS mechanism.
* Longback when the CORS was not standardized browser never allowed web applications to share resources between different origins.
* How the resource sharing between two web applications work, suppose we have two apps on different domain/origin
* Now they want to share resources, So Basically a CORS preflight mechanism is followed. So what do I mean by CORS pre-flight is?
* A pre-flight option call is made before the actual API call made
* So suppose this **A** wants to make a post call to **B,** so what will happen? First is a pre-flight call will be made and the CORS uses additional HTTP headers to verify this request first so suppose **A** wants to make a actual post call so browser itself will make a pre-flight call first which is also called as an **options call** and then that’s our work which is **B** over here takes their responsibility of verifying whether this call is valid or not.
* So if this call is valid then **B** will set some additional HTTP headers which will let the client know or the browser know that okay this is safe. And then the actual post call or actual call is made
* List of Browsers which are supporting CORS mechanism and almost all are their so there shouldn’t be an issue

**# Creating HTML Tag Element in React**

// Before root and after root all element work fine only affected those inside the root by react (but what is rendering matter, I’m just giving example of root)

    <div *id*="root">

*//Whatever inside the id root will be replaced after root.render*

</div>

    <script *crossorigin src*="https://unpkg.com/react@18/umd/react.development.js"></script>

    <script *Crossorigin* *src*="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

    <script>

*// Its the Job of 1st CDN Link react*

*const* heading = React.createElement("h1", {}, "Hello World From React!");

*// Its the Job of 2nd CDN Link react-dom*

*const* root = ReactDOM.createRoot(document.getElementById("root"));

      root.render(heading);

    </script>

* *const* heading = React.createElement("h1", {}, "Hello World From React!");
* this heading Element at the end of the is Javascript Object
* root.render(heading);
* this render method is basically responsible to take this “heading” object and put it up, Convert it into the “HTML heading tag” and put it up(root i.e mention in code) on the DOM

# NPM is not Node Package Manager, in npm’s official website there is no place where it has written that npm is Node Package Manager.

NPM Does not have Full Form 😅, so basically npm manages packages but it does not stand for Node Package Manager.

**Chapter 01 - Inception**

Link to my Code(<https://bitbucket.org/namastedev/namaste-react-live/commits/928e6faee91549739adc0c3c97c8f0352a55607e> )

Theory –

● What is Emmet? => <https://www.bugpilot.com/guides/en/type-html-faster-in-react-with-emmet-and-vs-code-1893>

● Difference between a Library and Framework? => <https://www.interviewbit.com/blog/framework-vs-library/>

● What is CDN? Why do we use it? => <https://www.cloudflare.com/learning/cdn/what-is-a-cdn/>

● Why is React known as React? => <https://www.linkedin.com/pulse/why-react-called-muhammad-irfan-wfrwc/>

● What is crossorigin in script tag? => <https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/crossorigin>

● What is diference between React and ReactDOM => <https://stackoverflow.com/questions/34114350/react-vs-reactdom>

● What is difference between react.development.js and react.production.js files via CDN? => <https://stackoverflow.com/questions/75791204/the-difference-between-react-development-js-and-react-production-js-files-via-cd#:~:text=In%20production%20mode%2C%20compression%20and,when%20compared%20to%20development%20mode>.

● What is async and defer? - see my Youtube video ;) => <https://www.youtube.com/watch?v=IrHmpdORLu8&t=4s&ab_channel=AkshaySaini>

=> <https://www.geeksforgeeks.org/explain-asynchronous-vs-deferred-javascript/>

Coding –

● Set up all the tools in your laptop

○ VS Code

○ Chrome

○ Extensions of Chrome

● Create a new Git repo

● Build your first Hello World program using,

○ Using just HTML

○ Using JS to manipulate the DOM

○ Using React

■ use CDN Links

■ Create an Element

■ Create nested React Elements

■ Use root.render

● Push code to Github (Theory as well as code)

● Learn about Arrow Functions before the next class

References:

- <https://beta.reactjs.org/apis/react/createElement>

- <https://www.youtube.com/watch?v=IrHmpdORLu8>

**# Initializing a New Project**

To create a new project, navigate to your desired project folder and run the following command:

$ npm init

This command will prompt you to enter some basic information about your project, such as the name, version, description, and entry point. Once you've provided the required information, ***npm will generate a package.json file***, which will contain all your project's metadata and dependencies.

**# Understanding package.json**

The package.json file is the heart of your project, as it stores all the necessary information about your project, such as its name, version, description, dependencies, and more. Here's a simple example of a package.json file:

{

"name": "my-project",

"version": "1.0.0",

"description": "A simple example project",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"dependencies": {

"express": "^4.17.1"

}

}

In this example, the project is named "my-project" and has a single dependency: the Express.js framework. The dependencies object lists all the packages required for your project to run correctly.

**# what-is-npm-and-how-does-it-work**

( <https://reintech.io/blog/what-is-npm-and-how-does-it-work> )

**# Package.json is Configuration for npm**

(<https://heynode.com/tutorial/what-packagejson/#:~:text=Your%20package.,dependencies%20required%20by%20the%20application>)

* **Sometimes Packages also known as Dependencies**

**# React Bundler**

(<https://www.dhiwise.com/post/embark-an-enlightening-journey-with-react-bundler> )

* When we have HTML CSS and JS file our whole code needs to be bundled together, our whole code needs to be minified, whole code needs to be compressed and needs to be cleaned before it can be sent to production so Bundler helps you to do all that things(webpack, parcel, vite)
* These bundlers are the job of to basically bundles our app it packages our app properly so that it can be shipped to production
* That “create-react-app” behind the scene uses “webpack” bundler
* But in our project we are using “parcel” bundler

**# parcel Bundler**

* Parcel Bundler Ignite our App
* npm install -D parcel
* here -D is devDependency(search on net for more info)
* These are your development dependencies. Dependencies that you need at some point in the development workflow but not while running your code (e.g. Babel or Flow).
* ( <https://github.com/parcel-bundler/parcel> ) / (<https://parceljs.org/>)

**# Dependencies vs devDependency**

* (<https://medium.com/@reemshakes/devdependencies-vs-dependencies-in-reactjs-db7261e13012#:~:text=When%20your%20project%20needs%20code,list%20of%20your%20project's%20dependencies> )

# **whats the difference between tilde(~) and caret(^) in package.json?**

* ~version **“Approximately equivalent to version”**, will update you to all future patch versions, without incrementing the minor version. ~1.2.3 will use releases from 1.2.3 to <1.3.0.
* ^version **“Compatible with version”**, will update you to all future minor/patch versions, without incrementing the major version. ^1.2.3 will use releases from 1.2.3 to <2.0.0.
* (<https://stackoverflow.com/questions/22343224/whats-the-difference-between-tilde-and-caret-in-package-json> )

**# Package.json vs Package-lock.json**

* What is the purpose of package.json?
* Tilde (~) and carat (^), and their difference
* What is package-lock.json?
* What is the purpose of package-lock.json?
* Comparing package.json and package-lock.json
* What is the role of npm-shrinkwrap.json in versioning?
* (<https://www.atatus.com/blog/package-json-vs-package-lock-json/> )
* "node\_modules/@babel/code-frame": {
* "version": "7.23.5",
* "resolved": "https://registry.npmjs.org/@babel/code-frame/-/code-frame-7.23.5.tgz",
* "integrity": "sha512-CgH3s1a96LipHCmSUmYFPwY7MNx8C3avkq7i4Wl3cfa662ldtUe4VM1TPXX70pfmrlWTb6jLqTYrZyT2ZTJBgA==",
* "dev": true,
* "dependencies": {
* "@babel/highlight": "^7.23.4",
* "chalk": "^2.4.2"
* },
* "engines": {
* "node": ">=6.9.0"
* }
* },
* Above data is from package-lock.json, **here the integrity meaning**
* This is Hash, have you heard of that thing it is working on my Local, I Don’t know how it break / it’s not working on Production
* So basically to avoid that package-lock.json keeps hash to verify that whatever is there in my machine is the same version which is being deployed onto the production.
* It’s very important file package-lock.json it keeps a track of all the exact versions of all dependencies
* When we install parcel there is one more thing that was created i.e node\_modules

Q:- What is node\_modules folder in react (BookPage-15)

* When creating a new React app your project will be populated with a bunch of new folders and files, and you may have noticed a **node\_modules** folder that contains an insane amount of folders.
* Because our needs parcel / project has dependency parcel. Now parcel as a project has it’s own dependencies and those dependencies can have it’s own dependencies, those dependencies can gave their own dependencies, this is known as “**Transitive Dependencies**”, Now this parcel can itself be dependent on lot of things. Parcel cannot do all these things on its own, parcel needs help of a lot of other packages, example parcel needs help of Babel also.(*Akshay Saini*)
* node\_modules are one of the most important directories in your React project as React requires node\_modules to run. The node\_modules directory is where all the dependancies packages are stored that are used to build and run your react project.
* So this is you'll find packages like React and React-DOM, your build packages like Vite, Babal or Webpack, and linters like ESLint or Prettier to name just a few. This directory can contain hundreds of dependancies!
* Q:- **Why aren't they included in version control?**
* => The main reason is the sheer size of this directory. Rather than including hundreds of package dependancies in version control, we can instead track a file called **package.json** which contains information about the project, and a list of dependencies required by the app. Other developers can use the package.json file and **npm install** to regenerate the node\_modules.

# **😱 when accidentally tracked node\_modules**

* Best practice would be to include the node\_modules in the **.gitignore** file in your project before pushing any code,
* which will prevent version control from tracking this folder - but if you've accidentally tracked and pushed the node\_modules to GitHub, like I did in my first project 🤦🏻‍♀️, simply follow the steps below.
* **Remove node\_modules from version control**
* Create a .gitignore file in your project and add node\_modules
* **Remove the node\_modules:**
* ***git rm -r --cached node\_modules***
* Commit and push without the node\_modules. The node\_modules should now be deleted from your repository.

**# Igniting Our App / Running Our Application On a Development Server**

* npx parcel src/index.html
* Output
* ❯ npx parcel src/index.html
* Server running at http://localhost:1234
* ✨ Built in 5ms
* Parcel’s built-in development server is now running. The npx parcel command takes your entry point of src/index.html and builds your application with the necessary assets. The output also indicates that the application is running on <http://localhost:1234>.
* Just like we have npm similarly we have something known as npx that means executing a package, npm if we have to install a package we need to write npm install
* ***# How to bundle a web app with parcel***
* <https://www.digitalocean.com/community/tutorials/how-to-bundle-a-web-app-with-parcel-js>

**# installing React from npm**

* Suppose if we already have react in my node\_modules, how easy would it be to use inside our code
* So first thing is that we don’t want to make another network call to get react
* We will have it already in our node\_modules that is why we will install it
* The second thing is today <https://unpkg.com/react@18/umd/react.development.js> react@18, tomorrow React@19 came in, and some other version of react came in, what will happen is we will have to keep changing this ***URL.***
* So it’s better to just have it inside our package.json it is easier to manage all are dependecies and it is very easy to manage react also.
* As one of the dependency inside our npm package.json
* Command => npm install react or npm i react
* Command => npm install react-dom or npm i react-dom
* After this we got "react": "^18.2.0" and "react-dom": "^18.2.0"dependencies in package.json file
* See is react over here react version ^18.2.0 and ^ carret, suppose tomorrow 18.2.3 comes in it will be there in our code
* Package-lock.json will also have
* Now we no longer need CDN links
* Now we have to import React and ReactDOM
* In our HTML we are injecting this App.js, Browser thinks it’s normal javascript file and it does not understand what is import, normal javascript does not need import.
* <script *type*="module" *src*="/App.js"></script>
* That is why we need to tell the browser that this file is not a normal browser file, it’s module.
* We have to write *type*="module" over here

**# Create Prod Build**

* *Command=> npx parcel build index.html*
* Only need to add build for prod build
* But if we run this command we will get error
* If we are using parcel we need to remove **"main": "App.js"** from package.json file
* **PS C:\Users\riyaz\OneDrive\Desktop\React-Practice> npx parcel build index.html**
* **npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.**
* **✨ Built in 3.06s**
* **dist\index.html 378 B 1.20s**
* **dist\index.8d566482.css 84 B 785ms**
* **dist\index.47db806b.js 138.77 KB 1.56s**
* Suppose if we our app will have like 10, 20 files it will compress everything and minify everything to these 3 files and now these 3 files will contain all the code that we write these 3 files are the production ready code of our app
* It will bundle, it will minify it will put all those files inside new folder **“dist”** , but before production build the **“dist”**  folder contains the development build files
* When we write npx parcel index.html it generate a development build of our project and it host it onto **localhost:1234** and put it up into this **“dist”**  folder
* If we delete dist and .parcel-cache these are the temporary folders
* If we do *npx parcel index.html* or *npx parcel build index.html* these dist and .parcel-cache folder automatically regenerated
* So these things any code that we can automatically generate we don’t have to put in GitHub
* Put this things in .gitignore

**# What is Browserslist**

* [**Browserslist**](https://browsersl.ist/) can specify which browsers your web application can run in, it provides a configuration for specifying browsers range. Browserslist has become a standard in the industry, it is used by libraries such as Autoprefixer, Babel, ESLint, PostCSS, SWC and Webpack.
* If we specify some browser’s list that means that listed browser’s definitely 100 of the times will work on these, it might or might not be work in rest of the browser’s
* (<https://modernjs.dev/builder/en/guide/advanced/browserslist> ) / (<https://github.com/browserslist/browserslist?tab=readme-ov-file#queries> )

**Chapter 02 - Assignment - Igniting our App Please**

Note: Write the answers and code on your own while finishing your assignments. Try to put down your thoughts into words by yourself in your own words. (This will help you develop muscle memory and you will remember all the concepts properly) ✌

# Theory Assignment:

● - What is `NPM`?

● - What is `Parcel/Webpack`? Why do we need it?

● - What is `.parcel-cache`

● - What is `npx` ?

● - What is difference between `dependencies` vs `devDependencies`

● - What is Tree Shaking?

● - What is Hot Module Replacement?

● - List down your favourite 5 superpowers of Parcel and describe any 3 of them in your own words.

● - What is `.gitignore`? What should we add and not add into it?

● - What is the difference between `package.json` and `package-lock.json`

● - Why should I not modify `package-lock.json`?

● - What is `node\_modules` ? Is it a good idea to push that on git?

● - What is the `dist` folder?

● - What is `browserlists` Read about dif bundlers: vite, webpack, parcel

● Read about: ^ - caret and ~ - tilda

● Read about Script types in html (MDN Docs)

# Project Assignment: - In your existing project

● - intialize `npm` into your repo

● - install `react` and `react-dom`

● - remove CDN links of react

● - install parcel

● - ignite your app with parcel

● - add scripts for “start” and “build” with parcel commands

● - add `.gitignore` file

● - add `browserlists`

● - build a production version of your code using `parcel build`

# References

● Creating your own create-react-app(<https://medium.com/@JedaiSaboteur/creating-a-react-app-from-scratch-f3c693b84658> )

● Parcel Documentation (<https://parceljs.org/getting-started/webapp/> )

● Parcel on Production (<https://parceljs.org/features/production/> )

● BrowsersList: <https://browserslist.dev/>

**# Creating script for run server**

* Earlier we doing like this npx parcel index.html now instead of writing this command again and again we just create a simple script
* It is an npm script we need to create that script in our Package.json file
* So we can create different script for development and production build
* "scripts": {
* "start": "parcel index.html", // Development script
* "build": "parcel build index.html", // Production script
* "test": "jest"
* },
* Now when we have created these scripts we no longer have to write npx parcel index.html to run our code
* Now we can use npm run start / npm run build to start our script (here start and build are script name)
* if you go to company and you don’t know how to start the project, just goto their package.json find this script and you will get the exact command to run the code

**# React Element**

*// Creating React Element using React*

*- const* heading = React.createElement("h1",{ id: "heading" },"Namaste React 🚀");

* Lot’s of developer thinks this is React.createElement is basically a HTML element, NO it’s not an HTML element
* How its work => When we do React.createElement it gives React Element and this React Element is basically an Object. So React Element at the end of the day is an javascript Object. Then this JS Object is rendered as an HTML Element.
* *const* root = ReactDOM.createRoot(document.getElementById("root"));
* root.render(heading);
* This ReactDOM takes this Object and convert it to HTML and push it to the browser
* It will replace not append everything that is inside this root whatever it is it will be replace when rendering.

**# JSX Syntax eXtension for JavaScript**

* JSX is a JavaScript syntax which is easier to create react elements
* Lot’s of people think’s JSX is a part of React, NO it’s not a part of React.
* React is Different JSX is Different, we can write React without JSX also But JSX makes our Developer life easy that is why we use JSX
* *JSX is not HTML* inside Javascript, JSX is Different than HTML. JSX is a HTML like Syntax
* JSX is a convention where we kind of merge HTML and JavaScript together
* JSX is just a **Syntax**
* *// Creating React Element using JSX*
* *const* jsxHeading = <h1 *id*="heading">Namaste React 🚀 from JSX</h1>;
* How its work => When we write JSX code is *Transpiled*(converting)to React.createElement it gives React Element and this React Element is basically an Object. So React Element at the end of the day is an javascript Object. Then this JS Object is rendered as an HTML Element.
* Babel is Converting our JSX to React.createElement
* jsxHeading Element is same as above heading Element both are object before Rendering

**# Babel**

* *const* jsxHeading = <h1 *id*="heading">Namaste React 🚀 from JSX</h1>
* So this JSX is a not a pure valid Javascript code.
* Any piece of Javascript code that JS Engine can understand.
* So the JS Engine not understand the Above JSX code, then how this code is working?
* Bundler(parcel) is doing the job behind the scene, Even before the whole code goes to Browser/JS-Engine it is Transpiled(it means this code is converted to the code that browser can understand).
* Then JS-Engine receives that Transpiled code so that browser can understand
* Then who is Transpiling the code Parcel itself? No no no Parcel is Gives the Responsibility of this Transpilation to a Package which is known as **Babel**
* Babel is a Package, Who is install Babel? We did not install then who install? => it’s a Parcel who install Babel
* Babel is present in node\_modules
* As soon as we write and save, So babe converted this code quickly to a code that JS engine will understand, Babel is transpiling ou code.
* *Babel in react is a crucial tool for React developers because it allows them to write code using the latest syntax and features while still ensuring that their code is compatible with all environments. This is particularly important because not all browsers support the latest JavaScript syntax, which could cause compatibility issues if developers tried to write React components using the latest syntax.*
* ( <https://www.scaler.com/topics/react/what-is-babel-in-react/> ) must visit and understand

**#** **Differece Between JSX and HTML**

* If we have to give an attributes to JSX we have to use camelCase (eg: className)
* (<https://www.freecodecamp.org/news/html-vs-jsx-whats-the-difference/> )

Q:- How to write JSX in multiple line

*// JSX in multipe line use () round braces*

*const* jsxHeading = (

  <h1 *id*="heading">

  Namaste React 🚀 from JSX

  </h1>

);

* We wrap JSX inside round brackets because Babel needs to understand from where JSX is starting and where is JSX ending.

**#Component**

* Everything is a component in React yes this statement is true
* A component is an independent, reusable code block which divides the UI into smaller pieces.
* For example, if we were building the UI of Twitter with React:
* 
* Rather than building the whole UI under one single file, we can and we should divide all the sections (marked with red) into smaller independent pieces. In other words, these are components.
* If we will see/create a webpage so a Button is a component, a Header is a component, a Footer is a component and a card, title, input box, search bar is component’s.
* There is 2 Types of Components
  + 1.Class Based Components (Old way of writing code)
  + 2.Function Based Components (new way of writing code)

**#Functional Component**

* React Functional Component is just a normal Javascript Function that must returns a React element (JSX).
* *// Functional Component*
* *const* HeadingComponent = () => {
* return <h1>Namaste React 🚀 From Functional Component</h1>;
* };
* *const* root = ReactDOM.createRoot(document.getElementById("root"));
* root.render(<HeadingComponent />);
* components are rendered like this *root.render(<HeadingComponent />);*
* this syntax that differentiate it’s a ReactElement or a Component that Babel understand
* always starts with a capital letter (naming convention)
* According to React's official docs, the function below is a valid functional component:
* function Welcome(props) {
* return <h1>Hello, {props.name}</h1>;
* }
* export default Welcome;
* Or
* const Welcome = (props) => {
* return <h1>Hello, {props.name}</h1>;
* }
* export default Welcome;
* To be able to use a component later, you need to first export it so you can import it somewhere else.
* (<https://www.freecodecamp.org/news/react-components-jsx-props-for-beginners/>) / (<https://codersera.com/blog/react-functional-components/> )

Q:- what is component Composition?

* Putting component inside a Component(nested) is known as Component Composition

*const* HeadingComponent = () => {

  return <h1>Namaste React 🚀 From Functional Component</h1>;

};

*const* MainComponent = () => (

  <div *className*="Container">

    <HeadingComponent />

    <h1>Namaste React from MAIN-COMPONENT 🚀🚀🚀🚀</h1>

  </div>

); // Javascript Expression

Q:- How to write Javascript inside JSX?

* We can run any piece of Javascript expression inside this { } curly braces
* *const* MainComponent = () => (
* <div *className*="Container">
* {heading} // Javascript Variable
* <h1>{10 + 2 + 196}</h1> // Javascript Expression
* {jsxHeading} // JSX Element
* <HeadingComponent /> // functional Component
* <h1>Namaste React from MAIN-COMPONENT 🚀🚀🚀🚀</h1>
* </div>
* );

Q:- What if API sent some Malicios Data.

* JSX is so amazing takes care of this injection attacks / will escape it
* When we getting some malicios data in {data}, whenever any data is wrapped inside these curly braces JSX wouldn’t blindly run it. It will sanitize your data whatever coming and then passing. It prevent cross-site scripting attacks for you.

Q:- How many ways of writing components inside component

*const* MainComponent = () => (

  <div *className*="Container">

    <HeadingComponent /> // 1

    <HeadingComponent></HeadingComponent> // 2

    {HeadingComponent()} // 3 calling functional component [at the end of the day it’s a function]

    <h1>Namaste React from MAIN-COMPONENT 🚀🚀🚀🚀</h1>

  </div>

);

**#Props in React Component**

* (<https://www.simplilearn.com/what-is-reactjs-props-article#:~:text=In%20ReactJS%2C%20the%20props%20are,components%20are%20read%2Donly%20components> )

**Chapter 03 - Laying the Foundation Topics**

● JSX ● React.createElement vs JSX

● Benefits of JSX

● Behind the Scenes of JSX

● Babel & parcel role in JSX

● Components

● Functional Components

○ Composing Components Assignment

● What is JSX?

● Superpowers of JSX

● Role of type attribute in script tag? What options can I use there?

● {TitleComponent} vs {} vs {} in JSX Coding Assignment:

● Create a Nested header Element using React.createElement(h1,h2,h3 inside a div with class “title”)

○ Create the same element using JSX

○ Create a functional component of the same with JSX

○ Pass attributes into the tag in JSX

○ Composition of Component(Add a component inside another)

○ {TitleComponent} vs {} vs {} in JSX

● Create a Header Component from scratch using Functional Components with JSX

○ Add a Logo on left ○ Add a search bar in middle

○ Add User icon on right ○ Add CSS to make it look nice References

● Babel: <https://babeljs.io/>

● Attribute Type: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script#attr-type>

● JS Modules: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules>

● Babel Playground: [https://babeljs.io/repl#](https://babeljs.io/repl)

● React without JSX: <https://reactjs.org/docs/react-without-jsx.html>

**#Config Driven UI (understand this concept properly)**

* Configuration-driven UI, also known as config-driven UI or configuration-based UI - When you build real a world application so you want should your website work in many country or many place we control our fontend it is known as using Config-Driven UI . API or Backend Driven which is data Comming from API.
* or
* In a configuration-driven UI, the layout, styles, and other properties of UI elements are defined in a configuration file or database, which can be easily modified without requiring changes to the codebase. This approach makes it easier to customize the UI for different use cases or user groups, without the need for extensive coding.
* The configuration file or database may also define the data sources and the data to be displayed in the UI, as well as the interactions and behavior of the UI components. This allows for greater flexibility and adaptability of the UI to different use cases, as the configuration data can be easily modified or replaced without affecting the underlying application logic
* (<https://portal.gitnation.org/contents/config-driven-ui-using-reactjs>)

**FOOD ORDERING APP**

* First of all do planning before doing anything
* Don’t just blindly write code, plan it that how our app will look like

#Components in our App

* Header

. Logo

. Nav Items

* Body

. Search

. RestaurantContainer

. RestaurantCard

* Footer

. Copyright

. Links

. Address

. Contact

<div *className*="res-container">

          { resList.map( (*restaurant*) => ( <RestaurantCard *key*={restaurant.data.id} *resData*={restaurant} /> ) ) }

</div>

* resList.map function then this map function basically takes
* restList is an array so it will Loop over all of these restaurants.
* so for each restaurant we have to return a RestaurantCard
* we are passing dynamically data into this RestaurantCard *resData*={restaurant}
* ***whenever we are looping on to anything we have to always give a key over there***
* ***But why?***
* ***However, if you check the console log, you will see that there is a warning like, “Warning: Each child in a list should have a unique key prop.” Whenever you use a loop it is important to provide a unique key attribute. The reason is that React uses these keys to track if items were changed, added, or removed.***
* ( <https://www.telerik.com/blogs/beginners-guide-loops-in-react-jsx#:~:text=However%2C%20if%20you%20check%20the,changed%2C%20added%2C%20or%20removed>. )

**#Why not to use index as a key?**

* **React Official Page =>** We don’t recommend using indexes for keys if the order of items may change. This can negatively impact performance and may cause issues with component state. Check out Robin Pokorny’s article for an [in-depth explanation on the negative impacts of using an index as a key](https://robinpokorny.com/blog/index-as-a-key-is-an-anti-pattern/). If you choose not to assign an explicit key to list items then React will default to using indexes as keys.
* Here is an [in-depth explanation about why keys are necessary](https://legacy.reactjs.org/docs/reconciliation.html#recursing-on-children) if you’re interested in learning more.

**#Why ReactJs is Fast?**

* When we say react is fast. React is fast in DOM Manipulation, React is efficient in DOM Manipulation.
* This is the exact problem that react is solving, suppose we have to keep our data and UI layer Consistent with each other tied to each other that is where react comes into picture and all the other Frameworks
* Angular, React, Vue all the other frameworks are trying to solve this. That UI Layer and Data Layer are work each with each other right properly sync in.
* If my data changes my UI Layer changes too, this is the thing which all these frameworks are like trying to solve updating the DOM efficiently
* React is fast because it can do faster DOM Manipulation it can do Efficient Dom Manipulation there is something known as Virtul-DOM, Diff Algorithm, Reconciliation and many more..

**Chapter 04 - Talk is cheap, show me the code!**

Assignment

● Is JSX mandatory for React?

● Is ES6 mandatory for React?

● {TitleComponent} vs {} vs {} in JSX

● How can I write comments in JSX?

● What is and <> ?

● What is Virtual DOM?

● What is Reconciliation in React?

● What is React Fiber?

● Why we need keys in React? When do we need keys in React?

● Can we use index as keys in React?

● What is props in React? Ways to

● What is a Config Driven UI ?

Coding Assignment:

● Build a Food Ordering App

○ Think of a cool name for your app

○ Build a AppLayout

○ Build a Header Component with Logo & Nav Items & Cart

○ Build a Body Component

○ Build RestaurantList Component

○ Build RestaurantCard Component

○ Use static data initially

○ Make your card dynamic(pass in props)

○ Props - passing arguments to a function - Use Destructuring & Spread operator

○ Render your cards with dynamic data of restaurants

○ Use Array.map to render all the restaurants PS.

Basically do everything that I did in the class, in the same sequence. Don't skip small things.

References

● Code Link: <https://bitbucket.org/namastedev/namaste-react-live/src/master/>

● React without JSX: <https://reactjs.org/docs/react-without-jsx.html>

● Virtual DOM: <https://reactjs.org/docs/faq-internals.html>

● Reconciliation: <https://reactjs.org/docs/reconciliation.html>

● React Fiber Architecture: <https://github.com/acdlite/react-fiber-architecture>

● React Without ES6: <https://reactjs.org/docs/react-without-es6.html>

● Index Keys as Anti-Pattern: <https://robinpokorny.com/blog/index-as-a-key-is-an-anti-pattern/>

**#Hooks in ReactJs**

* Normal Javascript Utility Function written by Facebook Developer.
* They have write this utility function inside React(node\_module – react), when we did npm install react we got all these utility function into our code
* We have to import these utility functions, there are multiple React Hooks and there are 2 very important React Hook
* useState()
* useEffect()
* (<https://www.freecodecamp.org/news/react-hooks-fundamentals/> )

**#useState in ReactJs [this content after episode 6]**

* if we want to make our component Dynamic, if we want something should change in our component then here is where the State Variable comes into the picture
* Whenever a state variable Changes/updates, React will re-render’s the component. React kind of refresh this compononet and all the updated values will be there.
* Whenever state variable update, React triggers a Reconciliation cycle(re-render the component)
* This is the power of state variable
* Example :- When we click on the login then it changed to logout, Because as soon as we click on this button React Updated this button and also referesh the Component, it triggered the *Reconciliation cycle*.
* it calculating the difference between VirtulDOM and the old state of VirtualDOM and the new state of VirtualDOM and then updating the UI everything is happening right now, as soon as we click on this button.
* It’s amazing how fast it is happening
* **Also refer Note on useState at Episode 7**
* Q:- What do you mean by Rendering a Component?
* => it will just call this function/component or it will trigger once again
* *const* Header = () => {
* *const* [logBtn, setLogBtn] = useState("Login");
* return (
* <div *className*="header">
* <div *className*="logo-container">
* <img *className*="logo" *src*={LOGO\_URL} *alt*="logo" />
* </div>
* <div *className*="nav-items">
* <ul>
* <li>Home</li>
* <li>About Us</li>
* <li>Contact Us</li>
* <li>Cart</li>
* <button *className*="login" *onClick*={() => { setLogBtn("Logout")}}>{logBtn}</button>
* </ul>
* </div>
* </div>
* );
* };
* export default Header;
* Q:- it rendering the whole header again or it just modifying that button?
* => it will re-render the whole header coomponent, initilly the header is render but it again also rendered when as soon as we click on login (*logBtn) Button* the Header is rendered once again, the whole component was refreshed quickly and was called quickly once again.
* So what happened as soon as we clicked on this button [**onClick**] as soon as this onClcik was called this callback function **<button *className*="login" *onClick*={() => { setLogBtn("Logout")}}>{logBtn}</button>** was called and this **setLogBtn("Logout")** was called, so React was keeping a track of this [logBtn, setLogBtn] logBtn variable and this logBtn variable react will update this variable and then it will call Header Component it will render Header Component once again
* (<https://www.freecodecamp.org/news/usestate-hook-3-different-examples/> )
* **Q:- How this const variable is getting updated? *const* [logBtn, setLogBtn] = useState("Login");**
* => Rendering this component once again but this time when we invoke this function this logBtn is a new variable (than it was before). On the intial time the state variable was login right? But as soon as I setLogBtn it will call the Header Component once again it will create a new instance this logBtn is different than that older logBtn which was printed and now when logBtn is created it is not created with the default value but it is created with the updated value(Logout). All this happening behind the scenes
* As soon as we call setLogBtn it will update logBtn value reference and it will render Header Component once again and it will find the Diff between the older version and the newer version. And it will see that in that diff only, this button is getting updated, there is no other changes in Header Component, so their nothing will change But only this Button will be changed, this is happening with this diff algorithm this is the Reconciliation process.

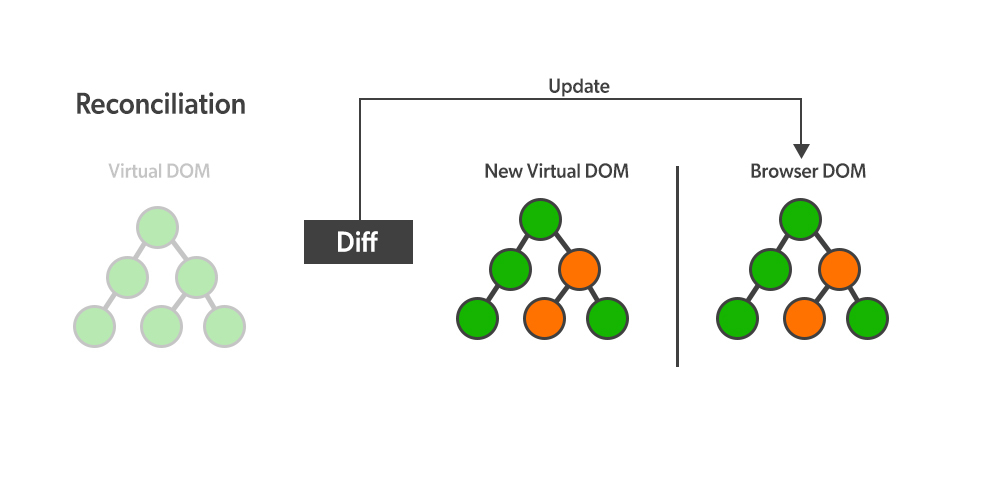
**#What is Reconciliation in ReactJs/React Fiber Architecture**

* **Introduction**
* **What is reconciliation?**
* **Reconciliation versus rendering**
* **Scheduling**
* **What is a fiber?**
* **Structure of a fiber(type & key, Child & Sibling, return, pendingProps & memoizedProps, pendingWorkPriority)**
* **(**[**https://github.com/acdlite/react-fiber-architecture**](https://github.com/acdlite/react-fiber-architecture) **)**
* React uses Reconciliation Algorithm is Also Known As React-Fiber, this has comes up in React16
* The Reconciliation Algorithm which finds out the difference between two virtualDOM, Updates the DOM only when there is required and only the portion of DOM which required.
* Fiber is a reconciliation algorithm used in the popular JavaScript library React to efficiently update a web application’s user interface (UI). It was introduced in [version 16.0 of React](https://reactjs.org/blog/2017/09/26/react-v16.0.html) in 2017 and has significantly improved the performance of React applications.
* In React, when the state of a component changes, the component needs to update its UI to reflect the new state. This process of updating the UI is called reconciliation.
* React uses a [Virtual DOM (VDOM)](https://reactjs.org/docs/faq-internals.html) to perform reconciliation, which is used to compare a component’s current and previous states.

**What is the Virtual DOM?**

* Virtual-DOM is not Actual-DOM, Virtual-DOM is representation of an Actual-DOM, VirtualDOM basically an Object
* The VDOM is a lightweight in-memory representation of the actual DOM.
* When the state of a component changes, React compares the VDOM of the last and current states and calculates the minimum number of DOM operations required to update the actual DOM to match the current VDOM.

**What is Diffing Algorithm in React ?**

* Diffing Algorithm in React JS differentiates the updated and previous DOM of the application. DOM stores the components of a website in a tree structure. React uses virtual DOM which is a lightweight version of the DOM. The only difference is the ability to write the screen like the real DOM, in fact, a new virtual DOM is created after every re-render.
* Diffing short for Differences Algorithm is used to differentiate the DOM Tree for efficient updates. React utilize diffing algorithm to identify the changes in the newly created virtual dom and previous version of dom after any changes are made.
* (<https://www.geeksforgeeks.org/what-is-diffing-algorithm/> )
* What is Diff Algorithm => it Basically finds out the difference between two virtual DOM’s, Updated VDOM and the previous VDOM

**The Catch**

* This helps reduce the number of DOM manipulations and improve the application’s performance. However, there are specific scenarios where the reconciliation process can become inefficient.
* *For example, suppose a component has many elements that need to be updated. In that case, the reconciliation process can take a long time and cause the UI to become unresponsive – This is where Fiber comes in.*

**What is Fiber?**

* Fiber is a new reconciliation algorithm introduced in React 16.0 that aims to improve the performance of React applications by making the reconciliation process more efficient.
* It does that by allowing the reconciliation process to be broken down into smaller chunks and scheduled over multiple frames rather than being completed in a single frame.
* Fiber divides the reconciliation work into smaller units called “fibers“.

**What is a Fiber?**

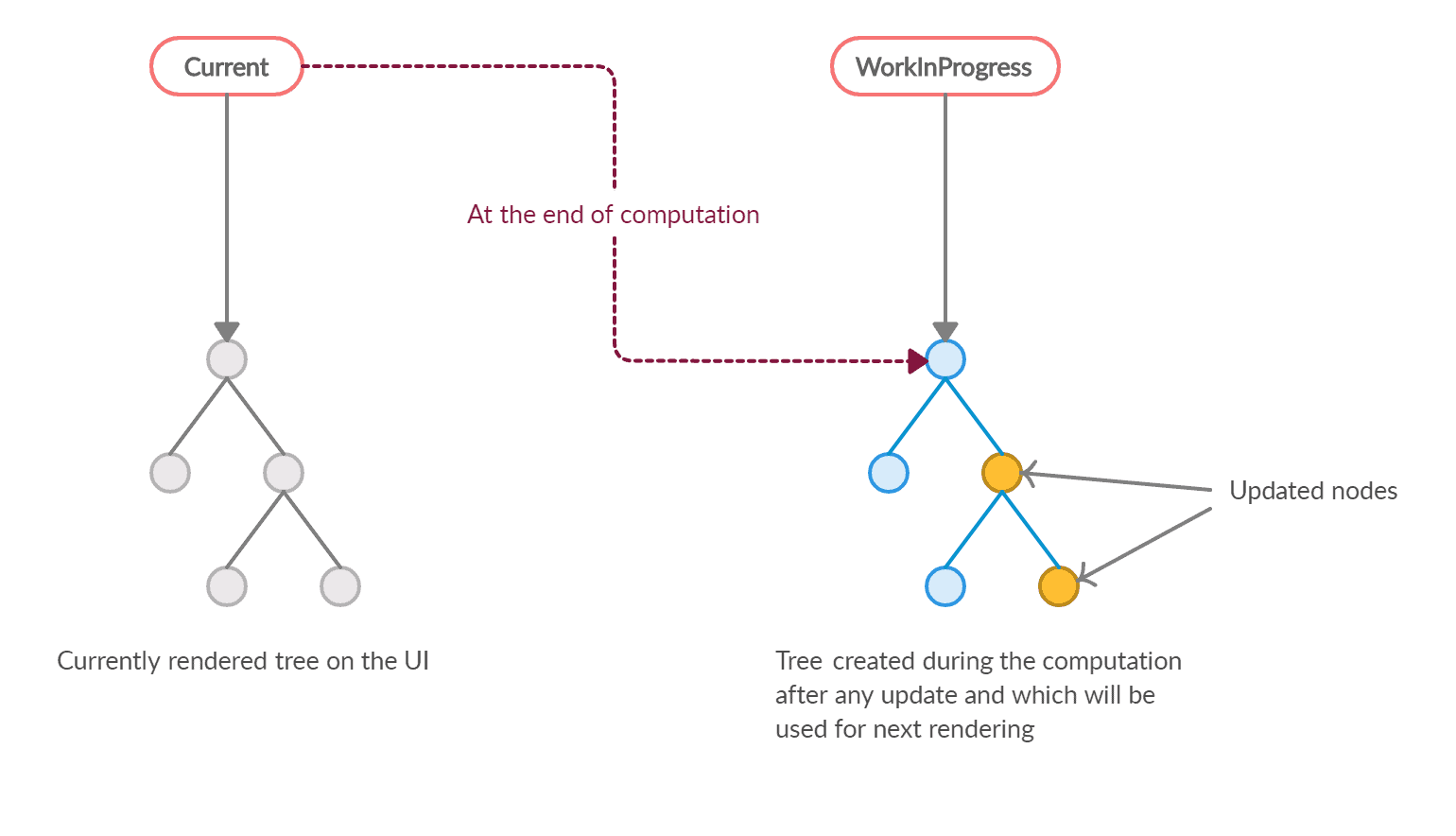
* Each Fiber represents a single element in the VDOM tree, and the reconciliation process is performed on each Fiber individually.
* This allows React to prioritize the reconciliation of certain fibers over others, depending on the importance of the updates.
* For example, suppose a component has many elements that need to be updated. In that case, React can prioritize the reconciliation of the elements that are visible to the user while deferring the reconciliation of the other elements until later.
* This helps ensure that the UI remains responsive even when there are a large number of updates.

**Fiber in Asynchronous Programming**

Fiber also introduces a new concept called “suspense”. Suspense allows React components to “wait” for a specific condition to be met before rendering.

This can improve the performance of applications that rely on asynchronous data, such as fetching data from a server.

**How does Fiber work?**



**An Introduction to React Fiber - The Algorithm Behind React (**<https://www.velotio.com/engineering-blog/react-fiber-algorithm> )

* Fiber works by maintaining a linked list of fibers called the “fiber tree“.
* Each Fiber in the tree represents a single element in the VDOM tree and contains information about the element, such as its type, props, and state.
* When a component’s state changes, React begins the reconciliation process by creating a new fiber tree based on the current state of the component. It then compares the new fiber tree to the previous fiber tree to determine the minimum number of changes required to update the actual DOM.
* During the reconciliation process, React traverses the fiber tree and updates each Fiber individually. It does this by starting at the root fiber and working its way down the tree, updating each Fiber in a depth-first order.
* As it traverses the fiber tree, React can pause the reconciliation process at any point and schedule the remaining work for the next frame. This allows React to prioritize the reconciliation of certain fibers over others and ensure that the UI remains responsive.

**Benefits of Fiber**

Fiber brings several benefits to React applications, including:

**Improved Performance**

As mentioned earlier, Fiber allows the reconciliation process to be broken down into smaller chunks and scheduled over multiple frames, which helps improve React applications’ performance, especially when a large number of elements need to be updated.

**Better user experience**

By prioritizing the reconciliation of certain fibers over others, Fiber helps ensure that the UI remains responsive, even when there are a large number of updates. This leads to a better user experience.

**Asynchronous rendering**

The introduction of Suspense in Fiber allows React components to “wait” for a specific condition to be met before rendering, which can be used to improve the performance of applications that rely on asynchronous data.

*If you’d like to know more about Suspense in React, which is related to the Asynchronous rendering mentioned earlier, I’d recommend checking the article we’ve written*[*here*](https://upmostly.com/tutorials/how-to-add-lazy-loading-to-react-components)*.*

**Better concurrency**

Fiber allows React to interrupt the reconciliation process at any point and schedule the remaining work for the next frame. This helps improve the concurrency of React applications, as it allows React to perform updates in parallel with other tasks.

**Summary**

In summary, Fiber is a reconciliation algorithm used in React to improve the performance of web applications.

It does this by allowing the reconciliation process to be broken down into smaller chunks and scheduled over multiple frames and by introducing the concept of suspense, which allows React components to “wait” for a specific condition to be met before rendering.

Fiber brings several benefits to React applications, including improved performance, a better user experience, asynchronous rendering, and better concurrency.

**Chapter 05 - Let's get Hooked!**

Assignment

● What is the difference between Named Export, Default export and \* as export?

● What is the importance of config.js file

● What are React Hooks?

● Why do we need a useState Hook?

Coding Assignment:

● Clean up your code

● Create a Folder Structure for your app

● Make different files for each Components

● Create a config file

● Use all types of import and export

● Create a Search Box in your App

● Use useState to create a variable and bind it to the input box

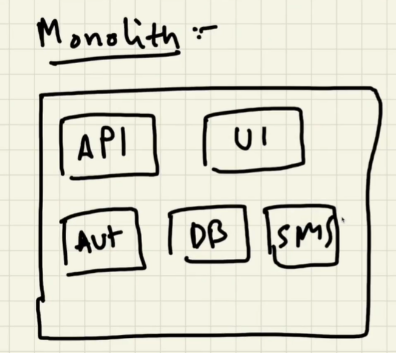
● Try to make your search bar work

References

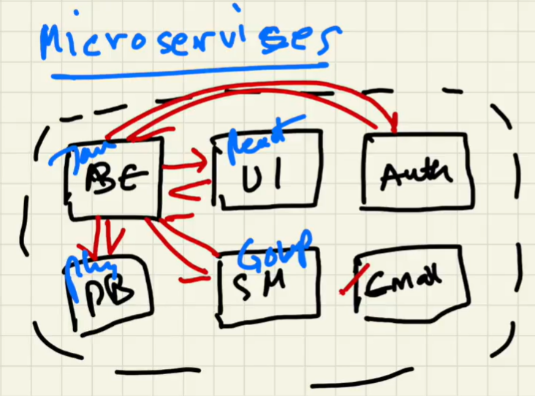
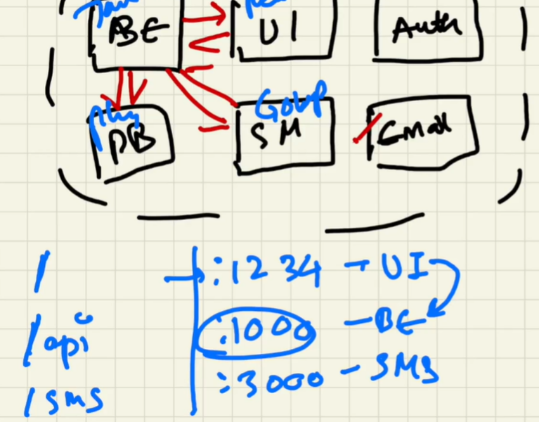
● Code Link - <https://bitbucket.org/namastedev/namaste-react-live/src/master/>

**EPISODE 6 :- Explore the World**

**#Monolith Architecture**

* When the web apps were developed they were all developed using a monolith architecture. What does it mean?
* So earlier we used to have a huge big project and suppose if we are building an application in this huge big projedt we used to have small pieces
* This project itself has code where there are api’s written we have developed **API’s** inside this project. We also have **UI** code inside the same project, we have **Authenticate** code inside the same project we have **Database** connectivity code inside this project we can have **notification/Sending** SMS code inside this project.
* Exmaple : Even if have to add a button the whole code need to be compile once again

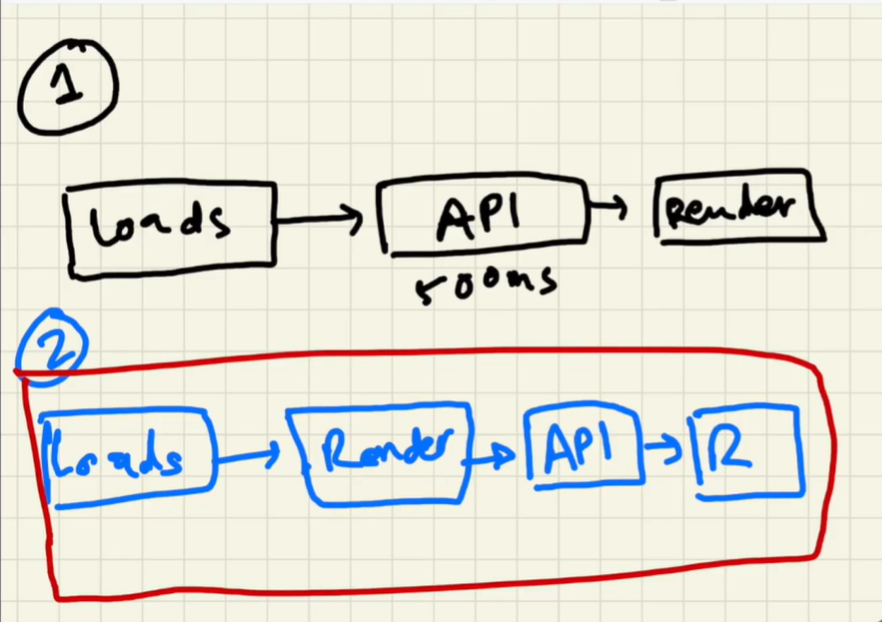
**#MicroService Architecture**

* In the Microservice Architecture we have different services for different jobs. So we have a service which is a backend service, we have a UI service, we have a Authentication service we have a service which connects through DB which maintains a database, we have service for SMS Sending we have a service email notification so there are different services for different jobs.
* This is known as Seperation of concerns and it follows single responsibility principle, each and every service has it’s own job
* These all microservices come combine together forms a big app
* Suppose assume that our react project that we are making is UI microservice and this microservice written in React so one more advantage of microservice we can have different text app for different things.
* Suppose in monolith architecture we had a one big project a java application then we have to do everything in java.
* But in Microservice Architecture we can have UI written in React, Backend written in JAVA, Database written in Python, we can have SMS service written in Golan.
* you can write your microservice in any archticture you want to.
* Q :- How do these service interact with each other?
* => Just like our UI service was deployed on port localhost:1234, similarly all these services run on their own specific port.
* - Example:-
  + UI service : 1234
  + Backend : 1000
  + SMS service : 3000
* So, on different ports we can deploy different services. At the end of the day all these different port can be mapped to Domain Name.
* Suppose the Backend is mapped to Namaste.com/API and all these api’s are deployed onto the same URL, and similary for SMS on namaste.com/sms we just have to call this service with slash / SMS
* They interact with each other by make a call to different URL’s. suppose UI want’s to connect to Backend they will make a call to slash /api or will call this port.
* That’s how they interact with each other.

**Monolith and Microservices Architecture =>** <https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith#:~:text=A%20monolithic%20application%20is%20built,on%20a%20number%20of%20factors>.

**CORS Policy =>** <https://portswigger.net/web-security/cors>

Q : - How web api and UI fetch the data from backend?

* There are two ways to show data on Display
* 1. So suppose our app loads one thing, what we can do is we just make an API call to fetch the data as soon as our page load, when we get the data then we can render it onto the UI [ as soon as our page loads we can make an API call wait for data to come and then we render the UI ]
* Suppose if this API call takes 500ms, the page will load and then after 500ms it will render the UI.
* So what is will happen if you must have seen when you open the web page so you suddenly don’t see anything and then suddenly as soon as the API responds it quickly shows a lot of stuff on the screen, so this is approach one.
* 2. The second approach is as soon as the page loads we will quilckly render it(our UI).
* Now we have quickly rendered application, we will make an API call as soon as we get the result back from the API we will now re-render our application with the new data from the API one again
* [we will just render the skeleton and as soon as the skeleton is rendered now we will make an API call, and as soon as the API responds we will just rendere the data onto the web page]
* So this approach is much more bettern than first one in react we always do this approach,
* Q:- Now you would say that we are rendering twice, is does not matter?
* => This is most important part of React. Why react is popular it is because that it’s render cycles are very fast. React has one of the best render mechanism. React Render our UI very fast so we don’t have to bother much about how many times we are rendering

**#Search Button**

*const* [searchText, setSearchText] = useState("");

<input *type*="text" *className*="search-box" *value*={searchText} *onChange*={(*e*) => {setSearchText(e.target.value)}}/>

          <button *onClick*={() => {console.log("Searched")}}>Search</button>

* In this input box ***value*={searchText}** it is tied with searchText State Variable, so whenever we will type something wherever something changes inside this input box, we want to update our searchText. And this is how it will update the searchText and it will show the value inside the input box over their.
* Whenever this input search text is getting updated on every key press my Component is getting rendered this whole body component is getting rendered.
* React Exactly knows, it is Checking the Difference every time when we hitting that key press, React is finding out the difference between two VirtualDOM’s and it is updating only the input box Because React precisely know that only the input has changed from the older version of VirtualDOM to the Newer version of VirtualDOM and once it has find the difference now it is updating that inside the DOM only the input box
* React is re-rendering the whole body component but it only updating input box value inside the DOM. DOM Manipulation is very expensive and React is very efficient in doing this
* Example when we type Café, C a f e. the whole body is getting re-rendered and refreshed 4 times on each key. That is speed of React that how much is fast even we don’t know how quickly is doing behind the scene. React has such amazing Reconciliation process such an amazing rendering process that it does not matter how many times, 😅 it does matter if we will do like 10000 times it will matter, but the best part is react triggering the Reconciliation cycle, It is finding the difference between the older VirtualDOM and the Newer VirtualDOM
* **EPISODE 7 :- Finding The Path**

**#useEffect**

* Q:- How is useEffect Hook Called?
* => we import useEffect from react library
* import { useState, useEffect } from "react";
* useEffect is called using 2 arguments, 1st is Callback Function and 2nd is it’s Dependency Array
* useEffect(() => {
* console.log("useEffect is Called from Header-Component");
* }, []);
* Q:- When is this useEffect called? [i,ii,iii]
* => i) So basically useEffect is called after every render of that component, But but but we have put in dependency array over here so this dependency array changes the behavior of it’s render. What does it mean?
* If we don’t put any dependency array, by the way it’s not mandatory, only callback function is mandatory inside useEffect.
* So when we call useEffect without a dependency array it will render every time when our component render [if no dependecy array => useEffect is called on every component render]
* useEffect(() => {
* console.log("useEffect is Called from Header-Component");
* });
* Q:- What if there is Empty dependency array?
* => ii) So then useEffect is called on only initial render and just once when the component is rendered for the first time. In the initial render when the page loads the component loads when the component loads useEffect is called after component loads.
* useEffect(() => {
* console.log("useEffect is Called from Header-Component");
* }, []);
* But the basic nature or the default behavior of the useEffect is to be called after each render but if we give it a dependency array then it is just called after once. And
* iii) what if we put something inside the dependency array? Then it will only be called when the dependency changes.
* useEffect(() => {
* console.log("useEffect is Called from Header-Component");
* }, [logBtn]);
* For example out Login button is dependency in useEffect then useEffect is called Every time when login Button is updated.

**NOTE:-** whenever we are using useState Never use useState never create this useState variable ***const* [logBtn, setLogBtn] = useState("Login");** outside of your component. It is used for creating local state variables inside our functional component, So always call it inside this functional component. And try to call this hooks on the top means when the function starts always try to use call this useState on the top, so that you don’t have a lot of inconsistency in your code. Because JavaScript is a synchronous single threaded language the code will run line by line so when you start from top the first thing you should do is to create your state variable. It’s good practice and a habit.

One more thing never use your useState Hook or create your state variable insid **if else /for loop / function,** this is perfectly valid code I mean you can do that javascript allows you to do this but don’t do this. This can create inconsistencies in your program.

In Documentation it is written that never use your useState Hook inside a condition so never do this.

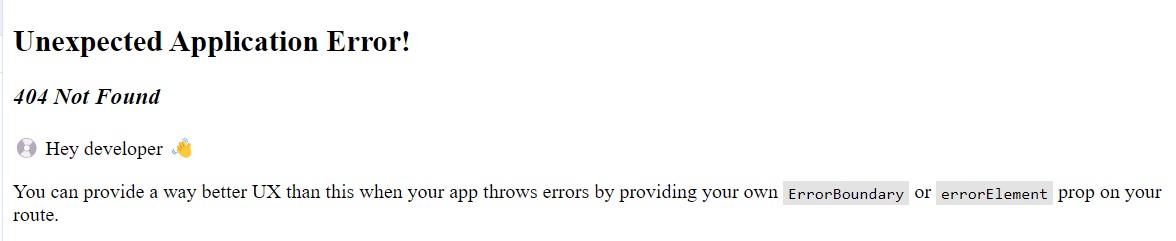
If you have function don’t create your state variables inside a function.

So basically is the state variable are meant to be created inside the functional components on the higher level right! On the first level

**#react-router-dom**

* what is react-router-dom(<https://www.geeksforgeeks.org/what-is-react-router-dom/>)
* npm install/i react-router-dom
* "react-router-dom": "^6.22.0" // version
* if we see in the code there were two changes in package.json and package-lock.json added this version.
* So how can we use this library? We need to create Routing Configuration in our App(root level component), wherever we have to develop routes we have to create routing configuration for that we need to import ***createBrowserRouter*** from *react-router-dom.*
* import { createBrowserRouter } from "react-router-dom";
* So we are goint to use createBrowserRouter and this will create a routing configuration for us
* *const* appRouter = createBrowserRouter([
* { path: "/", element: <AppLayout />, },
* { path: "/about", element: <About /> },
* { path: "/contact", element: <Contact /> },
* ]);
* Creating routing configuration inside our appRouter and we are passing configuration inside our appRouter, so what is configuration? Configuration means that some information that will define what will happen on a specific route [an information that will tell BrowserRouter that what will happen on a specific path.
* So this createBrowserRouter takes a array list of(object) paths, path is nothing but an object. An object contains 2 things i)path and ii)element.
* if my path is  **path: "/",** slashthen load the element: <AppLayout />, element(Home Page) Component. if my path is  **path: "/about",** slash *about*then load the element: <About />, element(about Page) Component.
* {Revsion :- we imported createBrowserRouter form react-router-dom, this takes in some configuration that configuration is an array of objects, this is list of objects, each and every object defines a different path and what should happen on that path }
* This is how we give confiuration but just creating configuration won’t work we will have to provide this configuration to render it onto the page. So for that we have one more important component ***RouterProvider*** that can import from react-router-dom.
* import { createBrowserRouter, RouterProvider } from "react-router-dom";
* This RouterProvider will actually provide this routing configuration to our App. So earlier what were doing just rendering the AppLayout directly. root.render(<AppLayout/>); Now instead of this we will provide it our router configuration. How? So basically this RouterProvide is component right! How we call a component <RouterProvider /> so we will provide this appRouter configuration over here like this *router*={appRouter}
* root.render(<RouterProvider *router*={appRouter} />);
* Now our URL is working fine <http://localhost:1234/About>

**#Error Handling by react-router-dom**

* what if we type random URL after Homepage like this [<http://localhost:1234/xyz>] we will not get any ugly Red color marked error, we will get an Error generated by react-router-dom like this **Unexpected Application Error!**
* 
* Let’s create our Own Error page then add this errorElement: <Error /> to the path
* like this **{** path: "/", element: <AppLayout />, errorElement: <Error /> }
* if my path is path: "/", then show the element element: <AppLayout /> if there is an Error then load this Component errorElement errorElement: <Error />
* if we type random URL after Homepage like this [<http://localhost:1234/xyz>] we will get Error that we created(Component)
* ****
* There is one more cool feature give to us by react-router-dom in term of Error. React-router-dom gives us access to an important Hook { useRouteError }. Hook is a function at the end of the day but it has a specific purpose.
* import { useRouteError } from "react-router-dom";
* useRouteError Hook gives more information about Error, so instead of just displaying Oops!!! Something went Wrong!!, we should also tell some more details, a better message on our page so it can be a route fail error, or it can be a network error or there can be multiple types of error right! So we can read the message using this useRouteError and we can show a specific details to the user. That’s why we use this useRouteError.
* How to use it? Basicalluy we have Error right!
* import { useRouteError } from "react-router-dom";
* *const* Error = () => {
* *const* err = useRouteError();
* return (
* <div>
* <h1>{err.error.message}</h1>
* <h1>{err.status} : {err.statusText}
* </h1>
* </div>
* );
* };
* export default Error;
* if we type random URL after Homepage like this [<http://localhost:1234/xyz>] we will get Error that we created(Error Component) using useRouteError
* ****
* So this is a better way of handling Error’s and we can have our own customized error page now right! We can show whatever we want over here and using this hook we can get more information about the error.

**Q:- How to fix Header at the top and all changes only done belove this?**

* to do to make this type of functionality we have to create children routes, so children routes for our AppLayout.
* { path: "/", element: <AppLayout />, errorElement: <Error /> }

**Q:- How to create children routes inside our App?**

* *const* appRouter = createBrowserRouter([
* {
* path: "/",
* element: <AppLayout />,
* children: [
* { path: "/", element: <Body /> },
* { path: "/about", element: <About /> },
* { path: "/contact", element: <Contact /> },
* ],
* errorElement: <Error />,
* },
* ]);
* We will give this configuration some children and then these children is again a list and children is a list of paths, so now we will copy the same configuration of this about and contact and putting it inside a list, inside children of parent route on the route.
* Now /about and /contact are the children of AppLayout
* one more thing to do
* *const* AppLayout = () => {
* return (
* <div *className*="app">
* <Header />
* Over here…
* </div>
* );
* };
* if my path is slash “/” then over here we should have this body, if my path is slash about “/about” then over here we should have this about component over here and so on…
* again react-router-dom comes into the picture it gives us something know as ***Outlet*** component
* import { createBrowserRouter, RouterProvider, Outlet } from "react-router-dom";
* (we have created Outlet over here) we just need to putOutlet over there, Now whenever where is this <Outlet /> this Outlet is over here in the AppLayout so whenever there is a change in the path so this outlet will be filled with the children according to the path
* *const* AppLayout = () => {
* return (
* <div *className*="app">
* <Header />
* <Outlet />
* </div>
* );
* };

**Q :- why our React application are known as single page applications?**

* [**https://stackoverflow.com/questions/62529631/why-is-react-js-called-as-single-page-application**](https://stackoverflow.com/questions/62529631/why-is-react-js-called-as-single-page-application)
* Example when we are on Homepage the header component and the body component is shown when we jumped to about page only the body component change to about(page) component, the page will not be reloaded only the body will be replaced by about code, the Header will remain intact.
* Isn’t wonderful if we use anchor tag(a) it reload the whole page and when we use Link it does not reload the whole page. It just refreshes the components. There is a different reload the page and page refresh the components, it just changes the component.
* That is why our React Application are known as Single Page Application (SPA)
* It’s a whole single component, our AppLayout and all the routing, all the new pages are just components interchanging themselves, Everything is component in React. This is why we call this as a single page application. Only one page, if we goto a new route it’s just changing the components it’s not reloading a new page. If we have to navigate to different pages we don’t have to refresh the page, our website will not reload or browser will not reload our website, browser will not make a external call or something, so this in known as SPA

**#There is 2 types of routing that we can have in our web Application**

1. **Client side routing (we are building client side routing)**
2. **Server side routing (traditional way that html is coming from server)**

* [**https://www.telerik.com/blogs/what-is-react-router-what-is-react-location#:~:text=Unlike%20server%2Dside%20routing%2C%20client,even%20when%20the%20URL%20changes**](https://www.telerik.com/blogs/what-is-react-router-what-is-react-location#:~:text=Unlike%20server%2Dside%20routing%2C%20client,even%20when%20the%20URL%20changes)
* in Client side routing we are not making any network call all the components are already loaded into our app when we load the app for the first time, it already has the code for all the pages(components). So it just load the component when we click on any page and it does not make any network call but only network call is made when we make a network call in the API.
* we are not fetching a new page so that’s why this is known as client-Side Routing.

**#RestauratMenu Component**

* added the path in root component(App)
* { path: "/restaurants/:resId", element: <RestaurantMenu /> },
* Here /:resId this means that resId is Dynamic, resId can be changed according to the ID of the Restaurant.
* So every restaurant has its own ID that uniquely identifies it, so our routes will be unique Whenever we have new restaurant.
* So whenever we have path something like this Dynamic Path /restaurants/:resId then load <RestaurantMenu />

**#useParams() Hook**

* **Practical Steps on How to Apply the useParams() Hook of React Router (**[**https://blog.stackademic.com/practical-steps-on-how-to-apply-the-useparams-hook-of-react-router-5cd43a2106b2**](https://blog.stackademic.com/practical-steps-on-how-to-apply-the-useparams-hook-of-react-router-5cd43a2106b2) **)**

**Episode 08: Let’s Get Classy**

**#Class Based Component**

* All about Class Component: <https://www.w3schools.com/react/react_class.asp>
* In interview Class based component will be asked a lot. Because a lot of companies are working on older projects which are already using class-based components
* What is class-based component, its normal javascript class.
* import React from "react";
* *class* UserClass *extends* React.*Component* {
* *constructor*(*props*) {
* *super*(props);
* console.log(props);
* *this*.state = {
* count: 0,
* count2: 2,
* };
* }
* render() {
* *const* { name, location } = *this*.props; *// Extracting name and location*
* *const* { count, count2 } = *this*.state;
* return (
* <div *className*="user-card">
* <h1>Class Component</h1>
* <h1>Count = {count}</h1>
* <h1>Count2 = {count2}</h1>
* <h2>Name: {name}</h2>
* <h3>Location: {location}</h3>
* <h4>Contact: riyazpathan193.rp@gmail.com</h4>
* </div>
* );
* }
* }
* export default UserClass;
* So basically this *extends* React.*Component* will make React know that is a class-based component, so react will start tracking it. So what is React.Component? it is class which is given by React and UserClass inheriting some properties from it. So React.Component is given from React so that means we have to import it.
* React.Component is a class which is inside react package.
* So we have render method in the class-based component that will return a piece of JSX which will be displayed on to the UI. Similarly if we compare it with functional component that return some piece of JSX, in class-based component a class has render method which return some piece of JSX.
* This class will have a Constructor and this Constructor will now receive props and we will also write super(props) over here, you cannot avoid writing it
* Once we receive props inside Constructor, How to use props inside render function? Ex: **Name: *{this.props.name}*. we can** Extract it by destructing because it looks lengthy and ugly like this Ex: ***const {name, location, Email-ID} = this.props***
* So always have to use a ***this*** keyword inside your class, so this props can be accessed anywhere inside Class.
* #**What is the purpose of using super constructor with props argument in React?**
* <https://dev.to/codeofrelevancy/what-is-the-purpose-of-using-super-constructor-with-props-argument-in-react-2ea3>
* <https://www.dhiwise.com/post/the-complete-guide-to-using-react-super-keyword-effectively>

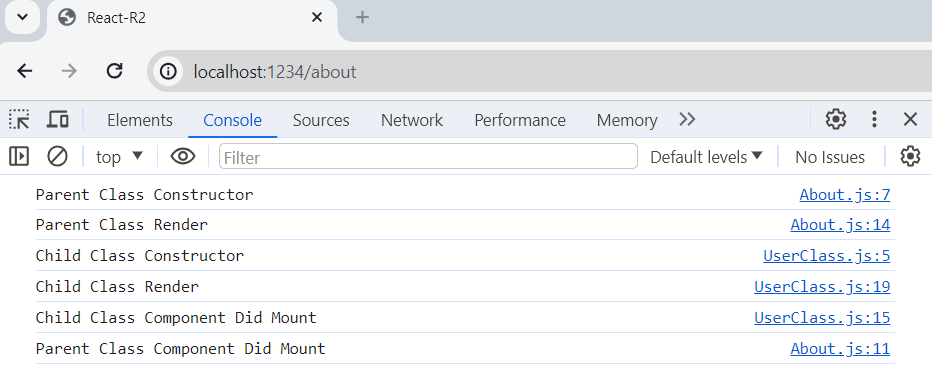
**#State in Class**

* Search More on internet…
* Never Update State Variable Directly. Then how to update state variable inside class?
* So React gives you access to important function which is known as ***this.setState()*** and you can use this.setState() function anywhere inside Class
* We will pass in an object and this object contain the updated value of your state variable
* *constructor*(*props*) {
* *super*(props);
* *this*.state = {
* count: 0,
* count2: 2,
* };
* }
* render() {
* *const* { count, count2 } = *this*.state;
* return (
* <div *className*="user-card">
* <h1>Count = {count}</h1>
* <h1>Count2 = {count2}</h1>
* <button *onClick* = {() => { *this*.setState({
* count: *this*.state.count + 1,
* count2: *this*.state.count2 + 2,
* })}}> Count Increase </button>
* Whenever we click on Button React is re-rendering the component, React is quickly rendering things up. So basically react is updating, re-rendering the component and it is just changing this button portion of the HTML
* How it works behind the scenes whenever react sees the button was clicked and ***this.state*** was called So react will take this Object and we will find out what over here in (***this.state*** variable) suppose if there are so many counts like 7,8,9 or 10 different state variables. But if you are sending just two 2 variable inside this setState object it will only update the two without touching all of them.
* Important thing to note, the state variable is a big object right! Suppose if you have just sent count1 and count2 so react will not touch the value of others it will just update the value whatever is passed from ***this.setState()***
* Behind the scenes this is the object and it will only update the portion of this object, it will kind of find the diff between the two object, now you can understand whenever this onClick button is clicked React will trigger that reconciliation process. It will find diff of the object it will update the setState variable it will re-render the component.

**#How Class Based Component is Mounted(loaded)?**

* **Now we have About is Parent Component and UserClass is Child Component**

|  |  |
| --- | --- |
| * import UserClass from "./UserClass"; * import { Component } from "react"; * *class* About *extends* *Component* { * *constructor*() { * *super*(); * console.log("Parent Class Constructor"); * } * componentDidMount() { * console.log("Parent Class Component Did Mount"); * } * render() { * console.log("Parent Class Render"); * return ( * <div> * <h1>About Us</h1> * <h2>This is Namaste React Web Series</h2> * <br /> * <UserClass *name*="Riyaz" *location*="Solapur" /> * </div> * ); * } * } * export default About; | import React from "react";  *class* UserClass *extends* React.*Component* {  *constructor*(*props*) {  *super*(props);  console.log("Child Class Constructor");  *this*.state = {        count: 0      };    }    componentDidMount() {      console.log("Child Class Component Did Mount");    }    render() {      console.log("Child Class Render");  *const* { name, location } = *this*.props;  *const* { count } = *this*.state;      return (        <div *className*="user-card">          <h1>Class Component</h1>          <h1>Count = {count}</h1>          <button  *onClick*={() => {  *this*.setState({                count: *this*.state.count + 1,              });            }}          >            Count Increase          </button>          <h2>Name: {name}</h2>          <h3>Location: {location}</h3>          <h4>Contact: riyazpathan193.rp@gmail.com</h4>        </div>      );    }  }  export default UserClass; |

* Parent is mounting and as soon as it sees that there is a **UserClass** Component there[UserClass inside About] so it start to **Load/Mount** UserClass Component now.
* Just like we have **Constructor** and **render** method in class based component we also have one more important method **componentDidMount**
* Once the class based component is Mounted/Loaded onto the DOM then this componentDidMount is called
* So when you load this about component into the web page or maybe you render it. When you go line by line then you sees there is Class-Based Component (UserClass) there so it start loading the class based component right! Now a new instance of this class is created and what happens when this class is called or maybe this class is instantiated? So what happen, is the constructor is called.
* The first thing when the class load a constructor is called [ when I say class is load that means I am creating a new instance of that class].
* First Parent Class Constructor is called then Parent Class Render is called, Child Class Constructor is called then Child Class Render then Child Class ComponentDidMount called in the end Parent Class ComponentDidMount will be called.
* Once the children has been mounted properly then the parent componentDidMount will be called
* 
* This how life cycle of Class components works

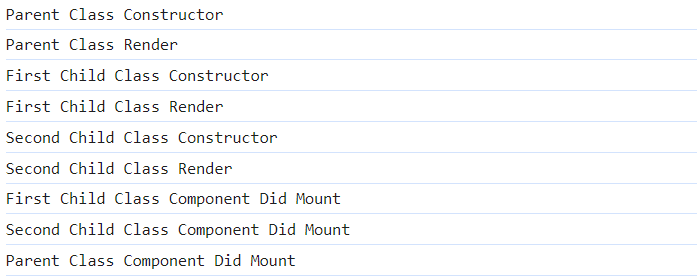
**#Why this componentDidMount is used?**

* There are some things that we do once the component is mounted successfully, componentDidMount is used to make API calls. But why it is made in inside componentDidMount?
* In useEffect() method from functional component we know first of all we load our component Once the component is loaded with basic details then we make an API call and fill the details. Why? So that React Component loads fast it does not wait for API call to return the result then load right!
* We want quickly render our component as fast as possible then make an API call get data fill the data. So what we will do over here is
* Constructor will be called render will be called, now our component has been rendered now we will make an API call over here in **componentDidMount** and then fill the data of the API call inside our component right! It will re-render the component.
* This is how we do inside class-based component, very very important a lot of people don’t understand why we make an API call inside componentDidMount they know that we make but they don’t know why. Right!

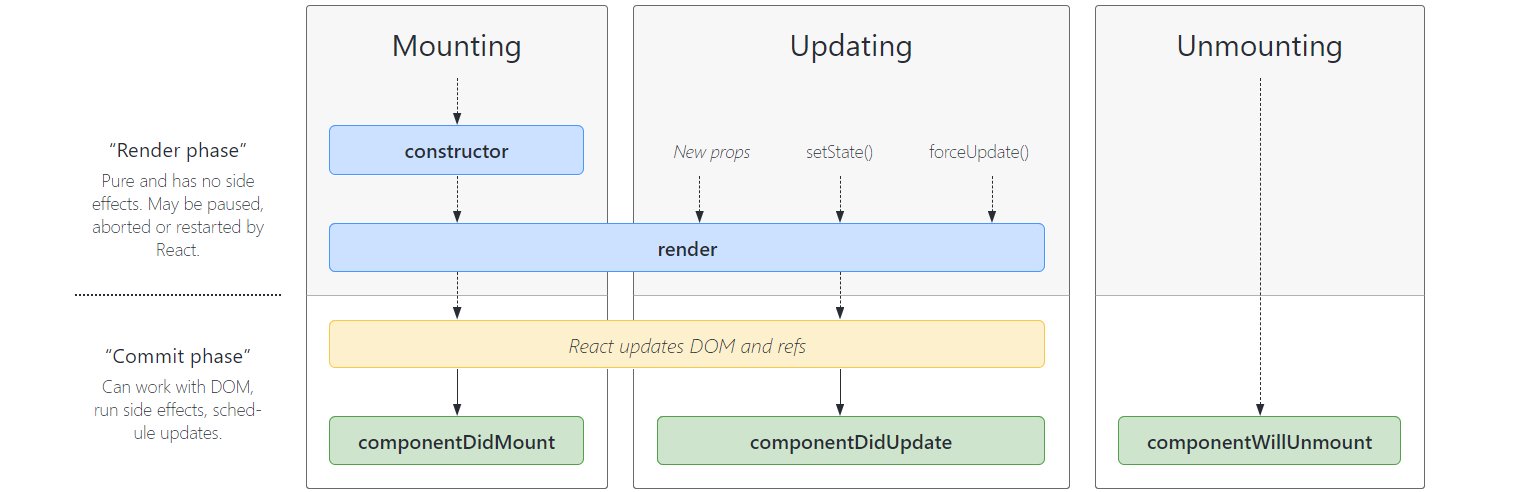
**#what if have multiple children**

* **Now we have About is Parent Component and 2 UserClass Child Components**

|  |  |
| --- | --- |
| * import UserClass from "./UserClass"; * import { Component } from "react"; * *class* About *extends* *Component* { * *constructor*(*props*) { * *super*(props); * console.log("Parent Class Constructor"); * } * componentDidMount() { * console.log("Parent Class Component Did Mount"); * } * render() { * console.log("Parent Class Render"); * return ( * <div> * <h1>About Us</h1> * <h2>This is Namaste React Web Series</h2> * <br /> * <UserClass *name*="First" *location*="Solapur" /> * <UserClass *name*="Second" *location*="Solapur" /> * </div> * ); * } * } * export default About; | import React from "react";  *class* UserClass *extends* React.*Component* {  *constructor*(*props*) {  *super*(props);  *this*.state = {        count: 0,        count2: 2,      };      console.log(*this*.props.name + "Child Class Constructor");    }    componentDidMount() {      console.log(*this*.props.name + "Child Class Component Did Mount");    }    render() {      console.log(*this*.props.name + "Child Class Render");  *const* { name, location } = *this*.props;  *const* { count, count2 } = *this*.state;      return (        <div *className*="user-card">          <h1>Class Component</h1>          <h1>Count = {count}</h1>          <h1>Count2 = {count2}</h1>          <button  *onClick*={() => {  *this*.setState({                count: *this*.state.count + 1,                count2: *this*.state.count2 + 2,              });            }}          >            Count Increase          </button>          <h2>Name: {name}</h2>          <h3>Location: {location}</h3>          <h4>Contact: riyazpathan193.rp@gmail.com</h4>        </div>      );    }  }  export default UserClass; |

* How the life cycle works now? In what order this will be printed?
* **This is not the right order**
* *Parent Constructor*
* *Parent Render*
  + *First child Constructor*
  + *First child Render*
  + *First child componentDidMount*
  + *Second child Constructor*
  + *Second child Render*
  + *Second child componentDidMount*
* *Parent ComponentDidMount*
* **Let’s see how it will work in right order**
* **
* But why did the life cycle method works like this?
* => There is something known as react life cycle method diagram and this diagram will make you clear everything.

**#React Life Cycle method**

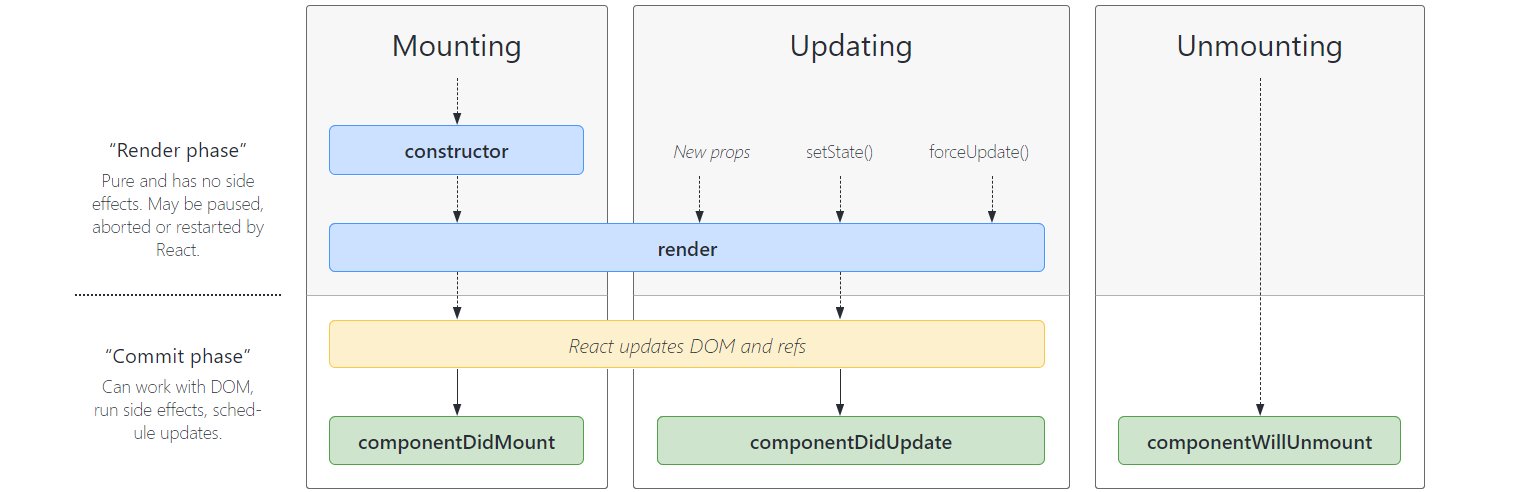


* <https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/>
* So in React When Component is mounted, it is mounted in two 2 Phases, React is fast because React has 2 phases. 1st phase is ***Render Phase*** the 2nd phase is ***Commit Phase***.
* Let me tell you this how the life cycle works when the component is mounting first of all our **Constructor** is called and then the **Render** is called, this constructor and render is the ***Render Phase.***
* Then the React updates the actual DOM and Once DOM is updated then **componentDidMount** is called.
* In our Example: we have UserClass, so first of all **Constructor** will be called then the **render** will be called then the react will updated the DOM once the DOM is updated/Loaded then the **componentDidMount** is called. That is why this is the best place to make an API call.
* *This life cycle happens for every child for every parent for every component in the React goes through this life cycle method*
* Remember it in the **Render Phase** has Constructor, Render and in the **Commit Phase** React actually update DOM and then **componentDidMount** is called.
* Our Code Explanation: our Parent component **About** start mounting phase right! When the About mounting happens, Constructor will be called, so Parent Constructor will be printed, what will happens now? ComponentDidMount is ***not*** printed now, a render will happen in the Render Phase and now Parent Render will be printed.
* <UserClass *name*="First" *location*="Solapur" />
* <UserClass *name*="Second" *location*="Solapur" />
* Now there is turn for our First Child UserClass, it goes to the first child and it starts the life cycle method of these child. So it will call the constructor of the first child then it will call the render of the first child. Now Because there are two children over here React optimizes, this React will not call the **componentDidMount** of the first child UserClass. What it will do is **it will just batch the *Render Phase* for these 2 child**
* So what will happen is for these 2 children the **Render Phase** will happen then the **Commit Phase** will happen together. This is an optimization of React.
* So the First Child **Constructor** is called then the First Child **render** is called and then **Render Phase** finished now. Same for 2nd child. Now the **Commit Phase** will be batch together for 1st and 2nd Child. Okay so first **Render Phase** is Batched then the **Commit Phase** is Batched. That is why the output will be in this order.
* Once the Commit Phase starts React tries to Update the DOM and the DOM Manipulation is the most expensive thing when we are updating/loading a component.
* Let’s see how life cycle works / how our code works after adding Profile (name, location, avatar) Fetching through API

|  |
| --- |
| * import React from "react"; * *class* UserClass *extends* React.*Component* { * *constructor*(*props*) { * *super*(props); * *this*.state = { * userInfo: { * name: "Dummy", * location: "Dummy", * }, * }; * } * *async* componentDidMount() { * *const* data = await fetch("https://api.github.com/users/riyazr2"); * *const* json = await data.json(); * *this*.setState({ * userInfo: json, * })} * render() { * *const* { name, location, avatar\_url } = *this*.state.userInfo; *// Extracting name and location* * return ( * <div *className*="user-card"> * <h1>Class Component</h1> * <img *className*="avatar-logo" *src*={avatar\_url} *alt*="" /> * <h2>Name: {name}</h2> * <h3>Location: {location}</h3> * <h4>Contact: riyazpathan193.rp@gmail.com</h4> * </div> * ); * } * } * export default UserClass; |

* As soon as UserClass was loaded the Constructor was called and when the Constructor is called this.state variable was created with default value okay! Now the render happens with a default values, our component renders with a default value. That means react will update the DOM with the dummy data For some milliseconds we see react updated the DOM with some dummy data. Now our **componentDidMount** was called with an API call was made when this API called was made. componentDidMount was called it called **this.setState()** When the setState will call now there is **Updating Cycle** begins okay!

***#componentDidMount***

* 
* Mounting cycle finished when the component was rendered once. When we do this.setState the **Updating Phase** start. setState updates the state variable, this state variable is updated now when the state variable is updated React triggers the **render function** once again, Constructor is called just once.
* When render happens once again but this time the state variable has been changed it with the updated value ***const* { name, location, avatar\_url } = *this*.state.userInfo;** the state has been updated with the new value so react will render it now. In the update cycle react will now updates DOM. See the will calculate the diff and it will update the DOM with the new value. And now this profile’s image, name, location is updated with the new value.
* And then the update cycle says that now it will call ***componentDidUpdate*** when React has updated our DOM there is something known as **componentDidUpdate** also, just like we had **componentDidMount** we also had **componentDidUpdate** and this cDU is called later on at the end. That’s how our update cycle is worked.

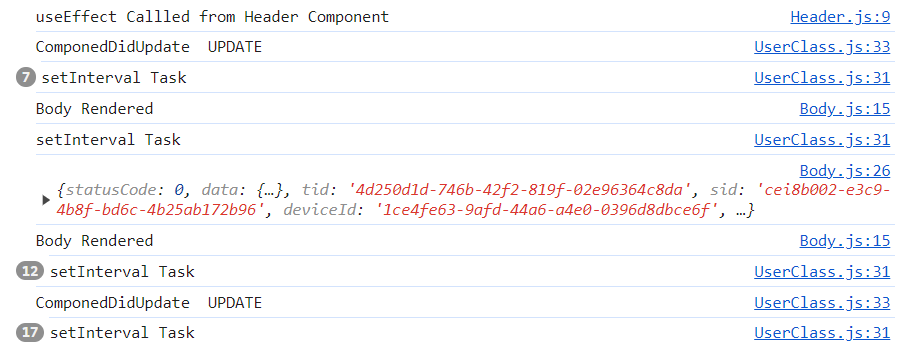
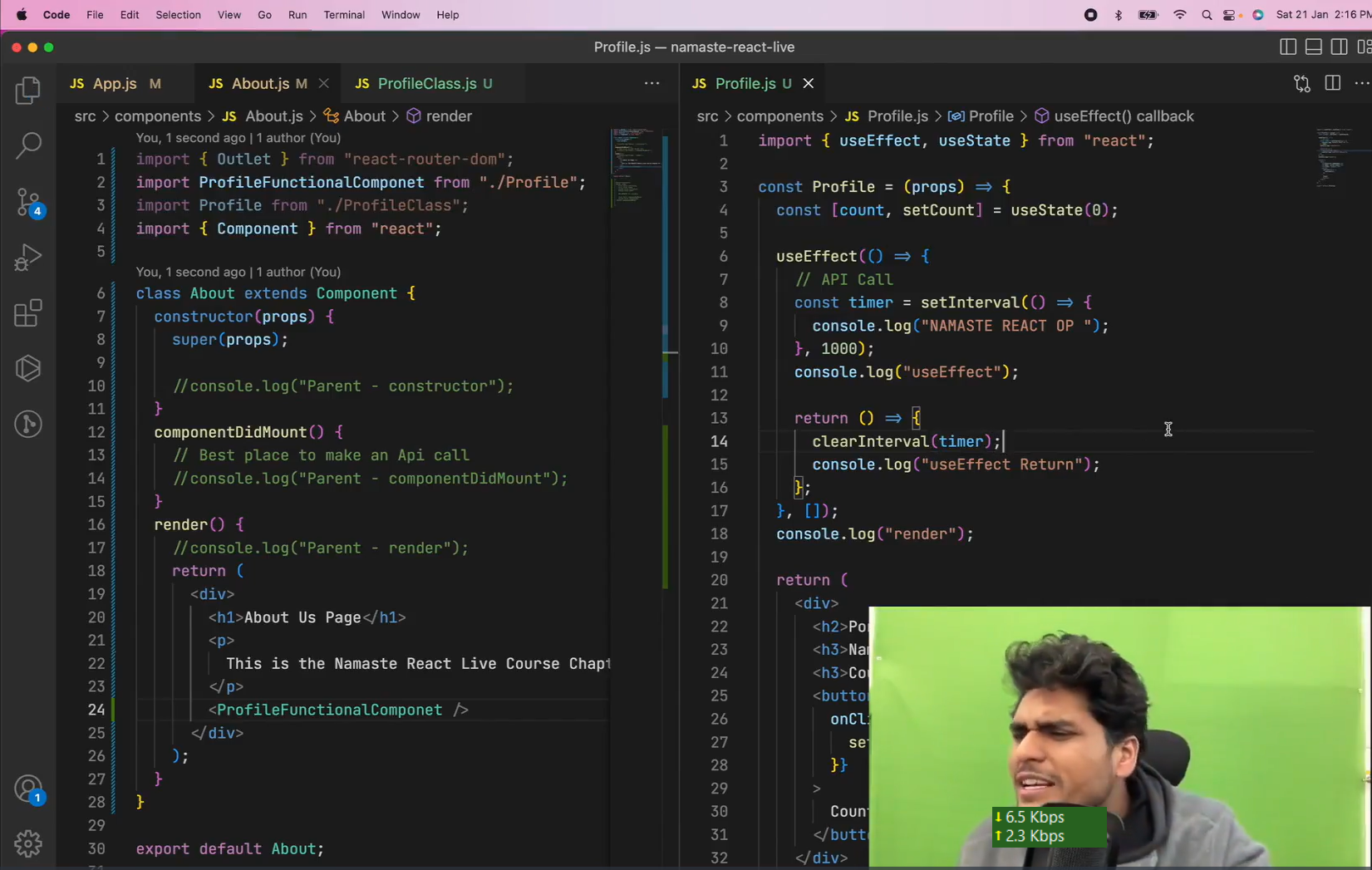
**--- Mounting Cycle---**

* Constructor (Dummy Data)
* Render (Dummy Data)
  + <HTML Dummy Data> : webpage loaded with dummy data for some milliseconds
* componentDidMount()
  + <API call> make an api call
  + <this.setState> 🡪 state variable is updated
* *Here Finishes Mounting Cycle*
* *When Mounting cycle finish now setState is called, whenever there is setState it triggers the Reconciliation process and then update cycle is called*

**--- Updating Cycle---**

* Render(API data)
* <HTML (new API data) 🡪 webpage loaded with new API data : image, name , location etc.
* componentDidUpdate()

***#componentWillUnmount***

* so basically this function is called just before our component is unmounting. What do you mean by unmount? 🡪 Unmount means when this component will disappear from this HTML form our webpage.
* Mounting means Showing onto the UI, Unmounting means like Removing from the UI.
* But when we will use this **componentWillUnmount**()? A lot of people say clean up, what type of clean up? What clean should we do?
* 🡪 componentWillUnmount is called when we are leaving the page [So we have just one single page in our app(SPA), so what is this? It is just a single page we are just changing the component].
* See when we go from Home-Page to About-Page or About-Page to Home-Page we are changing the page. When we are leaving the page lot of things we need to clear when we are leaving the page
* componentDidUpdate() {
* setInterval(() => {
* console.log("setInterval Task ");
* }, 1000);
* console.log("ComponedDidUpdate  UPDATE");
* }
* So suppose if we have setInterval() with 1 second so what will happen?
* => it started printing setInterval task After Every second
* The **Issue** with **Single Page Application(SPA)** is that if we move to a new page it is still calling the setInterval task
* If we move About-Page once again now this setInterval is calling twice. It is started 2 intervals without our permission this is the problem of single page application. Or every time when we come to this page(about) it is calling thrice and goes on.. it is creating a new setInterval
* ****
* When you are changing your pages it is not reloading your page! Did you understand this? it is not reloading your page it is just changing the components it is just rendering, the react is reconciling, it is the power of SPA. But it is a bad thing about single page application also.
* What if there was a setTimeout? It will again keep setTimeout and this setTimeout will be just hanging in the browser after a while this will blow up your app. Suppose if left your code something like this without unmounting, a user often goes from one to another page think of it how much performance loss you are doing.
* How do we clear this interval? By using **clearInterval()** and this clearInterval will be used in **componentWillUnmount** But how will we reference that(task) interval over here(componentWillUnmount)
* componentDidUpdate() {
* *this*.timer = setInterval(() => {
* console.log("setInterval Task ");
* }, 1000);
* console.log("ComponedDidUpdate  UPDATE");
* }
* 🡺 we can reference it by ***this***, ***this*** keyword is shared with all the functions all the methods of this class. So whenever you write *this*.timer
* componentWillUnmount() {
* console.log("ComponentWillUnmount  WILL");
* clearInterval(*this*.timer);
* }
* Now if we jump to new page it will stopped setInterval task
* So if you are senior developer you should know that when you are creating a mess you have to clean your mess also.
* **What if we create setInterval in useEffect() method in functional Component?**
* **🡺 if we jump from about page to some other page it won’t stop it continuously call after mentioned seconds. Why will it stop? It won’t stop because we are not cleaning up things over here.**
* **So how we will Destroy it in our functional component?**
* **🡺 there is something known as return inside. You can return a function from useEffect, this function basically called when you are unmounting this component. When we have *render(), useEffect()* and *useEffect-return* in useEffect. So render is first called then useEffect after we jump/leave from this page the *return* is called (this is unmounting phase in Functional Component)**
* ****

|  |  |
| --- | --- |
| **1 : about page** | **2 : jump to Contact page** |



|  |
| --- |
| **3 : then again back jump to About page** |

**#Important thing to remember**

* **Never Ever Compare React Life cycle method to functional component.**
* **In class based component what will be the called at the first time render? After first render *componentDidMount* is called**
* **And after every subsequent render it is updated, it is not mounted it is updated. There is a difference between Mount, Update and Unmount**
* **componentDidUpdate is called after every Updates**
* **Search on Net:** 
  + **Why do we write super props and constructor props**
  + **Async componentDidMount()**
  + **Why we cannot use async in useEffect() callback function**