Akshay Sainis React

Day 3:

Parcel is a bundler

This below all are transitive dependiecies for such below operatrions parcel is depended on some other packages

React cant make performance app alone,it requires diff thngs like parcel,

➔ Dev dependencies and nrml depeendienecies

What we have learned:
-> npm init it gave package.json

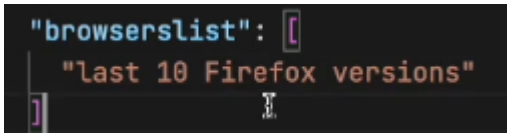
➔ For installing parcel,
➔ Npm I –D (for dev dependencies) parcel
➔ Cmd to executr our project: npx parcel index.html
➔ It created parcel.cache and a server for us, it enables many things like in the above img
➔ Installing react: npm i react

- ➔ Then, import, for that, we have to give type=”module” in script tag in html file,
- ➔ Diff b/w packagelock and pck.json
- ➔ Should I push parcel cahce on gitignor ? YES
- ➔ You should everyting in gitigonre that u can regenerate on server

HomeWork:

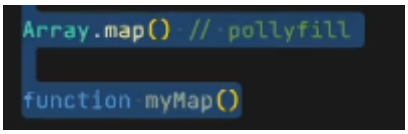
Should have the curiosity

DAY 4: React-Laying the foundation



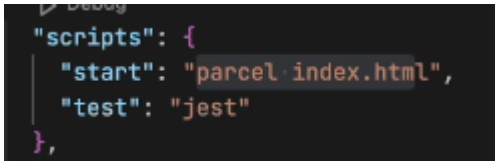
broweserlist: it means it definetly work on this, but other browser also it does, but few features may not be supported

Polyfil: a code which is replication for a newer version of code



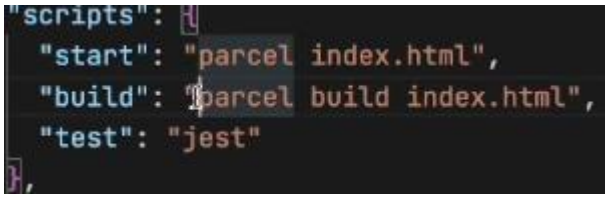
we do not do this, babel does for us(new version to older version).

For building our app, we type the cmd **npx parcel index.html** Rather we can add these cmds in script and jst have **npm run start**



Today we are going to talk **about jsx, babel**, **Npm init configuration for managing dependiencies,** Babil is a node pckg, a lib.

Same can be happened with build also, we can write build script, now we can type **npm run build**



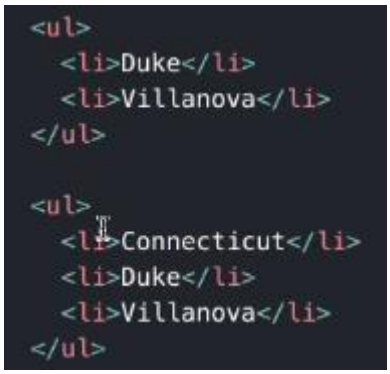
Npm start and npm run start are same

➔ In dist folder, we can see, no **conslole logs been removed, this can be done using pckg called**, **babel plugin transform** remove console, for this we are using a plugin we are configuring it

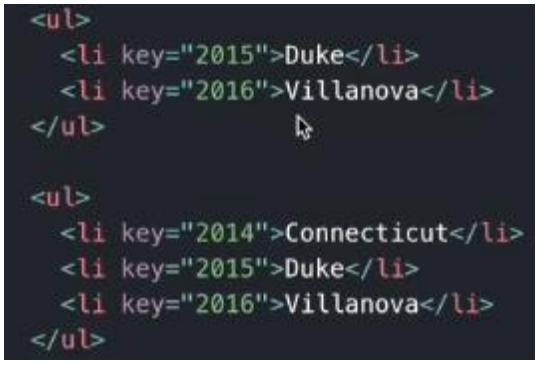
➔ **npm i babel-plugin-transform-remove-console --save-dev**

➔ now after installing it wont work we need to configure it, will be creating .babelrc file, writing this in file

➔ Then npm run build which will create files in dist fldr, it will not contains console.log but it contains .error



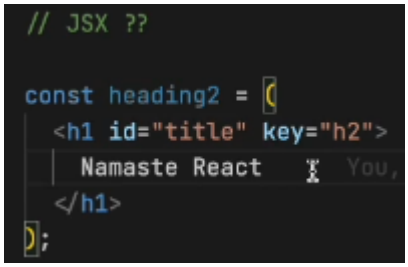
➔ **Check react reconsilation key, Read about** Diffing algo, i.e., we uses key, consider the li tag, if we want to add one more li tag on top of prev li tags **then it is time consuming, as need to rerender the whole dom**, which **effects changes in dom tree, Therefore** it makes easy to **use key in such scenario**



➔ **How does creatElement woks? .**

➔ Well it is creating an object, <u>React.CreateElement is creating an object, which then conv into html code, then put upon the DOM</u>.

➔ **Creating huge html structures with createElement will mess things up, instead of this we use jsx**



➔ **Is jsx html inside** JavaScript ? T/F -> it false

➔ It is a html like syntax but not html

➔ Diff b/w html and jsx

➔ In jsx we use camelCasing i.e, tabIndex, not tab-index **and className not class**

➔ How does jsx executes the code, if u type in browser it doesn't supports it, **babel understands it**,

➔ Img tag in jsx ?

➔ Babel comes along with parcel , html in () Is known as jsx expression

➔ Now let's learn **React Components**

➔ Everything is a component in react, **we got 2 types of component, functional(NEW) and class(OLD)** in this course will woking with functional components

➔
```
// React Component
// - Functional - NEW - I'll use this most of the time
// - Class Based Component - OLD - - We will learn this tool
```

➔ **Functional Component** is nothing but a function, A function returing a react ele is known as react functional comp

➔ For any Compoent **the name starts from a Capital letter**, for good practices

➔ No need of () for single line code, for multiple lines need () can also be written as this, without any return or arrow function thing.

```
const HeaderComponent = () => (
  <div>
    <h1>Namaste React functional component</h1>
    <h2>This is a h2 tage</h2>
  </div>
);      You, last week • chapter-01 …
```

➔ It is returning jsx

➔ When we have to render the functional component then will render this way **<function1/>,** and react ele, nrmlly

➔ We can also use react element inside function using **{heading}**

➔ Any peace of js code can be written in {}

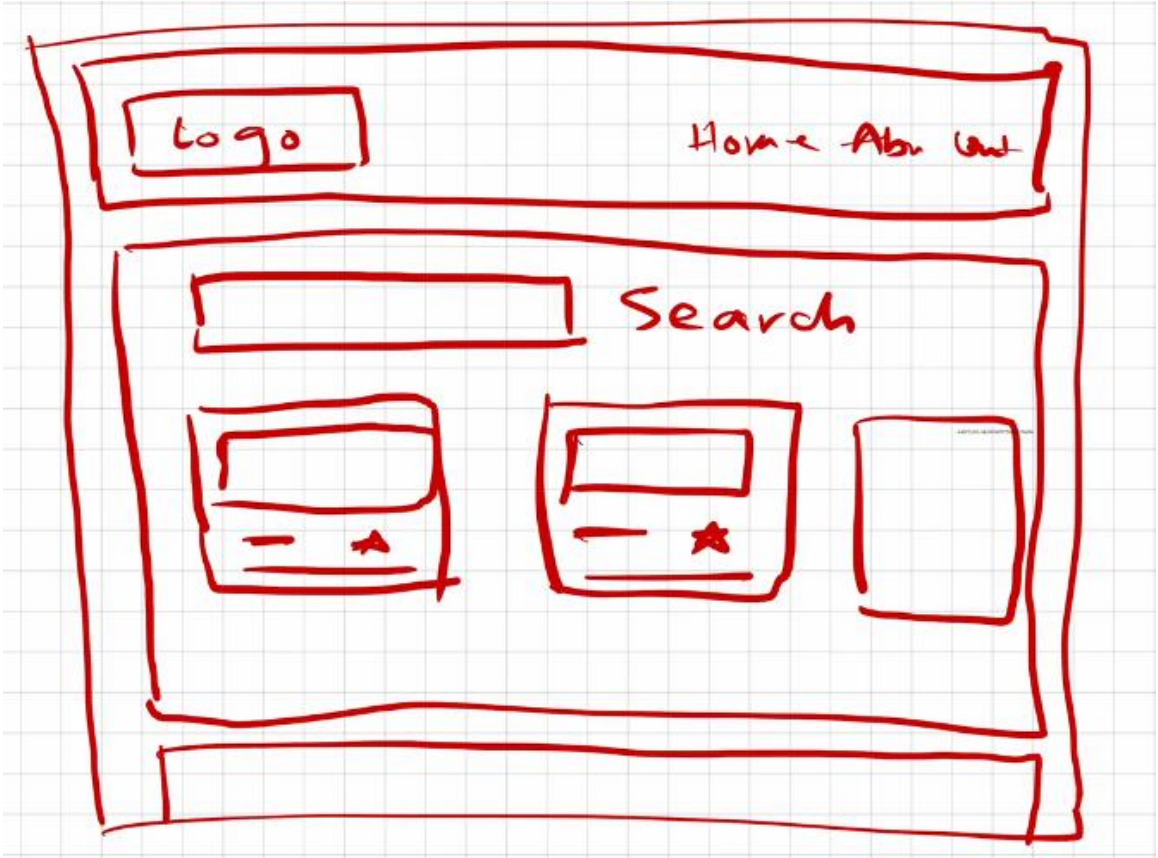
➔ Now lets say the api, is returning some malicious code, such attacks are known as cross site scripting attack (xss), by injecting some js code, if it is able to run in our lap then it takes data, **BUT JSX takes care of such attack, i.e, it sanitizes the code and protects from such attack**

➔ **Component Composition:** such that, we've to use a component inside a component

Revision:

- ➔ Browserlist
- ➔ Babel, and its config
- ➔ Why do we need keys and diff algo
- ➔ What is jsx? It is using React.CreateElement behind the scenes
- ➔ Functional component, and class its using, {} </>

Day 5: Talk is cheap, show me the code

➔ We can be able to write javascript code inside these {}
➔ We can call our functional component like {Title()} or <Title/> or <Title> </Title>
➔ React is a proper library
➔ Is jsx / es6 is mandatory for react-> NO
➔ **Now Let build our APP**
➔ Will be building **Food Villa App**
➔ Before writing the code, do plan about our App
➔ 

➔ 1st let's build header
➔ 1st will be building our layout->structure for that build a component name APPLayout

```
const AppLayout = () =>{
  return (
    {/**
     Header
        - Logo
        - Nav Items(Right Side)
        - Cart
     Body
        - Search bar
        - RestrauntList
          - RestaurantCard
              - Image
              - Name
              - Rating
              - Cusines
     Footer
      - links
      - Copyrights

     */
     }
```

➔

➔ Start coding for header component, **add home, about, contact**, cart put **some CSS to align it**

```
.header {
    display: flex;
    justify-content: space-between
    margin: 10px;      You, 32 seco
}

.nav-items > ul{
    list-style-type: none;
    display: flex;
}

.nav-items > ul > li {
    padding: 10px;
}
```

➔

➔ Add an image/ logo in title component Give a class and resize it also try to add anchor tag

➔ **NOTE: Any piece of jsx component, there can only be 1 parent**

➔ i.e., const jsx = <h1>hii</h1> is Correct but

➔ const jsx = <h1> hii </h1> <h1> hello </h1> is Wrong

➔ Therefore, we can wrap this thing inside div

➔

➔ Therefore, we can use React.Fragment

Now will be building our resturent card

We've to work on our body part

Whenever I'm having this restaurant cards then I need to pull data from some where, So remember this thing, Whenever I'm using UI I'will all way's be conserned about where will my data come from ?

➔ For now we will be using hard coded data, then will also learn how to use it through API's as well

➔ Will be constructing a functional component with name restaurant card, which will be **returning a jsx element**

➔ After creating the card, put this in Body, make the returning stmt as div and add **the <Restraurcard/>**

➔ **Now, give some styling in css to card**

➔ **It will appear on browser.**

➔ Now, as we've hard coded our restaurant card, but the img, name, lable, rating and all is diff for all, Therefore, we need to make it dynamic

➔ For that, let's utilize the javascript, will be creating a js obj, to do dynamic access of data we can do this way.

```
const burgerKing = {
  name: "Burger King",
  image: "https://res.cloudinary.com/swigg
  cusines: ["Burger", "American"],
  rating: "4.8"
}

const RestrauntCard = () => {
  return (
    <div className="card">
      <img src={burgerKing.image} />
      <h2>{burgerKing.name}</h2>
      <h3>{burgerKing.cusines}</h3>
      <h4>{burgerKing.rating} stars</h4>
    </div>
  );
};
```

➔ To have a (,) in cusines as it a array, we can use join, i.e., **burkerking.cusines.join(",")** this how you join items in an array.

➔ Now, we made our data dynamic, but in the real world there will be so many restaurant, if function call them again and again not good,

➔ **Display flex, is useful to view the div comp, in horizontal format, also flex-wrap : wrap, can also use grid do ur css**

➔ **Now lets try to make our all cards dynamic**,

- In real world the data comes in list formate, such that there will be so many of it.
- **Config driven UI:** dynamic ui is called as config driven UI, such that, the whole ui is driven by a config which is sent by a backend.
- **It is Very much useful if you built your Project, using config driven UI**
- How, will be able to control our UI with backend.
- As shown in img, if at some place there are no offers and

```
//Config Driven UI

const config = [
  {
    type: "carousel",
    cards: [
      {
        offerName: "50% off",
      },
      {
        offerName: "No Delivery Charge",
      },
    ],
  },
  {
    type: "resutaurants",
    cards: [
      {
        name: "Burger King",
        image:
          "https://res.cloudinary.com/swigg
```

there is no need of any carousel, then backend wont send us such kind of cards, will be directly showing the resutaruants.

- **In your system desingn round you have to tell this to your, interviewer.**
- **Now,** Will be building such kind of dynamic UI
- **Let's use real data, of swiggy not** the mog data
- For this, go to inspect, then FetchXR then click on preview it shows the data, or also try reloading the data. Now copy the restaurants data into your code, now how to use that data? Use it normally as shown in img.
-
- After getting the data from swiggy, now how to use it, will be using as restuarentList[0].data.name suppose if the name doesnot exist then u've to use this **restuarentList[0].data?.name this called optional**

```
const RestrauntCard = () => {
  return (
    <div className="card">
      <img
        src={
          "https://res.cloudinary.com/swiggy/image/upload/fl_lossy,f
          restrautList[1].data?.cloudinaryImageId
        }
      />
      <h2>{restrautList[0].data?.name}</h2>
      <h3>{restrautList[0].data?.cuisines.join(", ")}</h3>
      <h4>{restrautList[0].data?.lastMileTravelString} minutes</h4>
    </div>
  );
};
```

chaining (JS thing), also the image can be used as shown

- Now let's try **to make this dynamic**, such **that the 1st, 2nd and 3rd card should** be showing the info as of list ordering
- **1 way is to use props**

```
<RestrauntCard restaurant={restrautList[0]} />
<RestrauntCard restaurant={restrautList[1]} />
```

- **Props->properties,** i.e, I'm passing my data into my component, functional comp is just a function like in js
- In functions we've concept of arguments and parameter, we pass in arguments we receive parameters
- In react to use, it as we are passing the props, to use in function we've take in as parameter, and can name it anything (props).
- **Props is like jst a nrml function call as we do in our codes**
- We can pass any no of args

```
<RestrauntCard restaurant={restrautList[0]} hello="world"/>
```

```
const RestrauntCard = (props) => {
  console.log(props);
  return (
    <div className="card">
      <img
        src={
          "https://res.cloudinary.com/s
          props.restaurant.data?.cloudi
        }
      />
      <h2>{props.restaurant.data?.name}
```

  - It is received as a para here
  - Now, here's the thing come up by cool developers, what they do is **destructuring** the object, instead of props use,
- Now we don't need props.

```
const RestrauntCard = ({ restaurant }) => {
```

```
const { name, cuisines, cloudinaryImageId, lastMileTravelString } =
  restaurant.data;
```

  I can use my restaurant
- Again you can destructure it like this, with this we can directly use the tags.
-

```
const RestrauntCard = ({ name, cuisines, cloudinaryImageId, lastMileTravelString }) => {
  return (
```

- For this will call like:

```
<RestrauntCard name={restrautList[0].data.name} cuisines={restrautList[0].data.cuisines} />
```

- **If lets say you have destructured the whol at params u got name, cusines, ratings and all so can you pass from u args**

```
<RestrauntCard name={restrautList[0].data.
```

this way ? well the answer is NO.

- **What u do is, u cann pass each with separate tags from the args, or u can use (…) this operator to jst travers all**
- `const RestrauntCard = ({name, cuisines, cloudinaryImageId, avgRating}) =>{`
- It will work, It `<RestrauntCard {...restrautList[0].data} />` is your **spread operator, this is JS es6, in {} can write any** js
- So**, what if there are 50 cards, then** we cant go on writing this way, therefore, we **uses for loop, but** in the industry we don't use for loops instead e use **MAPS,** but also we can have for loop,
- **For maps, as the restaurentList is an array, restaurentList.map(give a callback fun), the fun takes each obj, and for each obj I want my fun to return jsx, which is my <restuarentCard/>**

- **Everything we've builded is like a config driven UI**
- **Today, we've made many things, we made up our header, nav bar, logo, cards (coming from swiggy),**
- **Now lets dive into virtual DOM**
- Virtual dom is not jst the concept of react it is a software engineering concept.
- **We keep a representation of the dom, with us this is known as virtual DOM,**
- **Why do we need it ?**
- **A: we need it for reconsilation, reconsilation is an algorithm that react uses, to diff one tree from other and it determines what need to be changed and what not in the ui.**
- Here the key concept comes into the picture, i.e., lets say we got multiple divs and one dive is been added, as we

know the virtual dom only changes the updated div, Therefore to identify the divs we use **keys.**

- **VirutalDOM is the representation of the dom, and react uses something known as reconsilation, it will find out the different between the tress and only the portion that is required,**
- Hotmodule reconsilation, this is different, this is on file, which is been down by parcel.
- Also read about **React Fibre**, which came in react16 its new reconsilation engine and it is responsible for div

reconciliation

The algorithm React uses to diff one tree with another to determine which parts need to be changed.

```
return <RestrauntCard {...restaurant.data} key={restaurant.data.id} />;
```

- **Now, you should never forget to give key**
- **Why don't we use index as the key ?** `key={index}`
- It will not give error, but u should never have to use index as your key.

```
// no key (not acceptable)<<<<<<<<<<< index key(use ONLY if you don't have anythinge) <<<<< unquie key (best practi
```

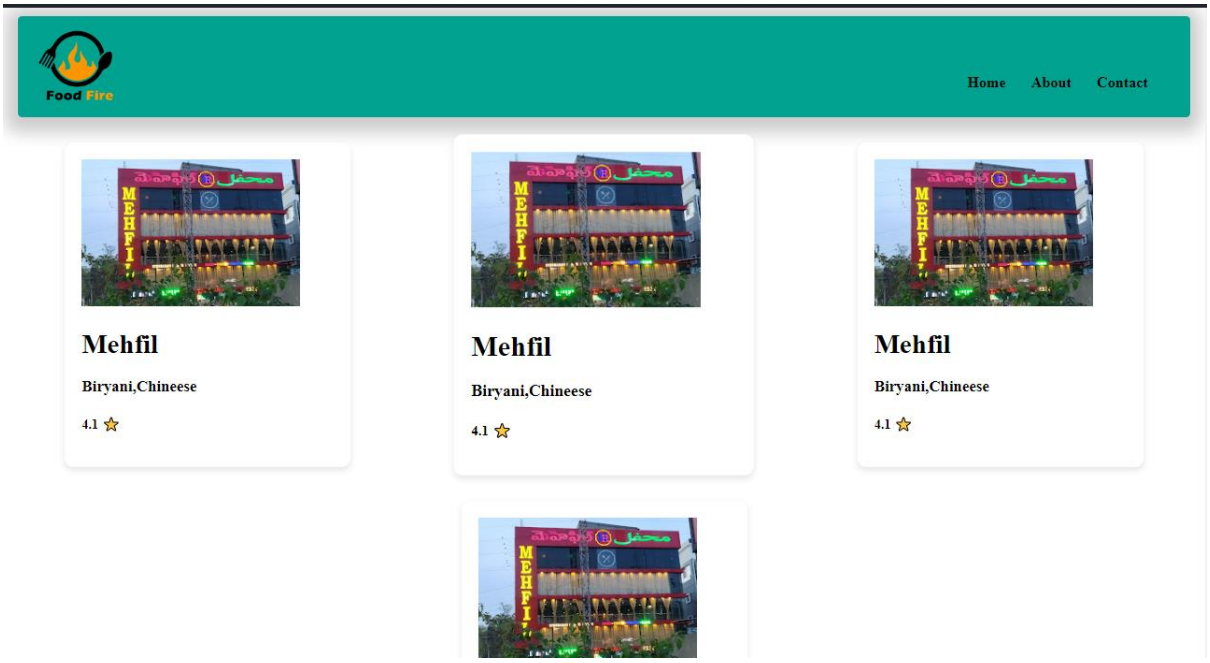
Now lets revise:

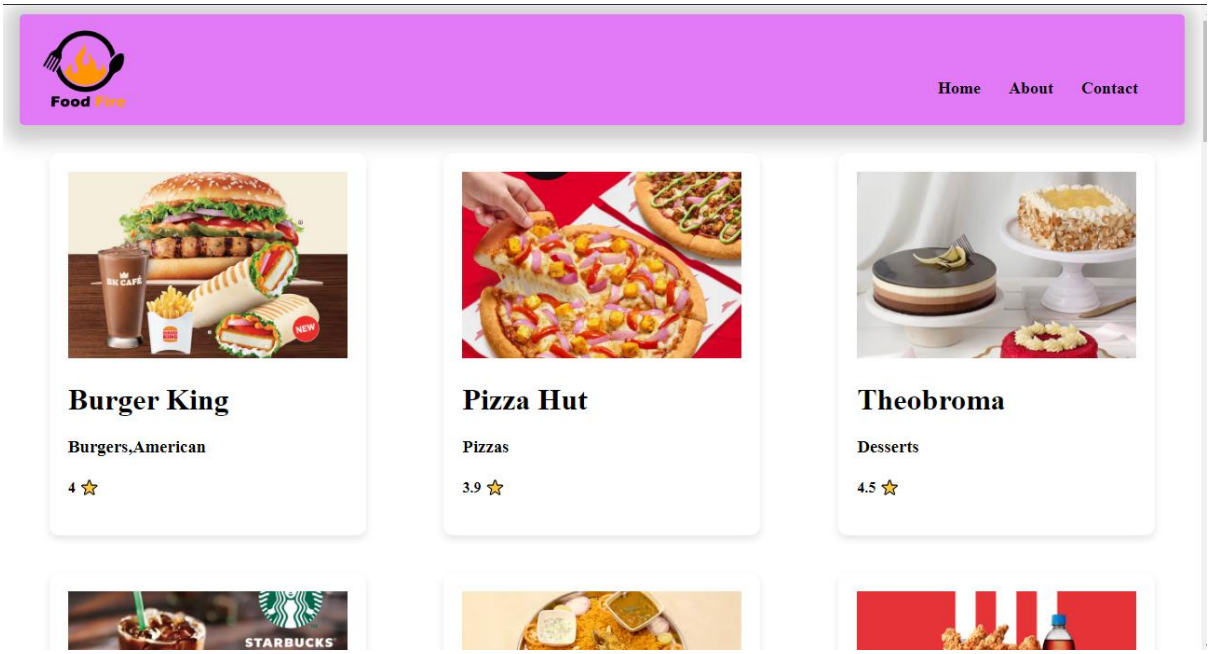
➔ **1st we** did planning, then learn about <> </> this
➔ **Then** we also learn about functional component
➔ Then we started building our card.
➔ Then we builded our UI
➔ Then we had our hard coded data
➔ Then we made our card dynamic, Then we wanted multiple cards to be dynamic
➔ Then went to swiggy api, then also learn about carousal of swiggy, i.e., wants to hide and all, such thing are known as **config driven UI, such that u give me a config, my ui will render automatically**
➔ **Then** we started using props.
➔ **Such that, pass**ing args
➔ **Then we** studied about map function and also spread operator

➔**Then** we learned about Virtual dom, reaconsilation algo

➔**Then** we learned can I have index as a key

Practice session:





Day 6: Let's get hooked! 01-02-2024

Created a folder named src, do we need to create folder well not, but maintaining proper folder structures this gives us a better modular structure and findable, src is a comman convention been using used in the industry, Now will keep all our files related to our app inside src folder.

- ➔ React folder structure, read about it. We can distribute our files depending on the feature of the app .
- ➔ Create a src folder then a comp folder, then a file name Title.js add the title code in it, then export the Title component from the file, then import it in app.js and use it
- ➔ There are 2 ways of exporting it 1. Export default Title, it will export Title as default i.e., Learn about it, also when u export by using default u use it like `import Header from "./components/Header";`
- ➔ Now have header and title components in same file, now can I export both of them using default, U can't I can export only one thing, so if I export this `export const Title = () => (` way such that**, exporting by name** then u've to use like

```
// Named Import
import { Title } from "./components/Header";
```

In default import u don't put {} for named import u have to u {}, when to u use what??

- ➔ If you named exported both the comp then you can import this way `import { Title, Header } from "./components/Header";`
- ➔ Now if my Header is default export then we can import this way `import Header, { Title } from "./components/Header";` , does this name have to match the export one, well NO for the default export there is no need, you can change the name, but also try to keep the same name it is good practice.
- ➔ And I can also import as ./components/Header.js it will also works fine. And in the react dev community some people are

try to name there file as .jsx cause it contains react code in it. And you can also import as .jsx

➔ Sometimes this can break when you use external library.

➔ **Now if I'm exporting all components** and named comp, then I can import like this `import * as XYZ from "./components/Header";` and I can use like `<XYZ.Header />`, Now you must be getting why we used `<React.Fragment>` in the code.

➔ **Now,** What should I be following, Well I try to **export default Header,** Cause I do not export what is not needed, cause title in not needed in App.js right we need it only in our Header component, so I don't export it, So I jst import things without {}, I only use it when there is real need.

➔ **HomeWork:** named import, diff ways to export, export default, how do u export a default exp, a named export, what happens when u write *, also what happens when u name ur files.

➔ **Now** let's try to create components for other files as well,

➔ Let's create for body comp, create a body.js also do it for footer, so u can destructure you files as much as u can

➔ **Now Listen very important thing we also've to create a config** file in our project i.e., **config.js, Well** I put all the **hard codded things into my config file, n**ow I'm using a url for image, suppose I want it in other file as well, so I need to again copy paste it there, SO I just wan to import this from one place kind a global var, **in some company it named as constant.js** files as well.

➔ Here's how you can've your url, better to export as named

```
export const IMG_CDN_URL =
  "https://res.cloudinary.com/swiggy/image/upload/fl_lossy,f_auto,q_auto,w_508,h_320,c_fill/";
```

also try to have the restrauent list in the constant.js file and export it, and import it in body file. Now also create a new file for the RestaurantCard comp as well and use it in body file and we need that img url in Card comp file.

➔ Now our code looks cleaner.

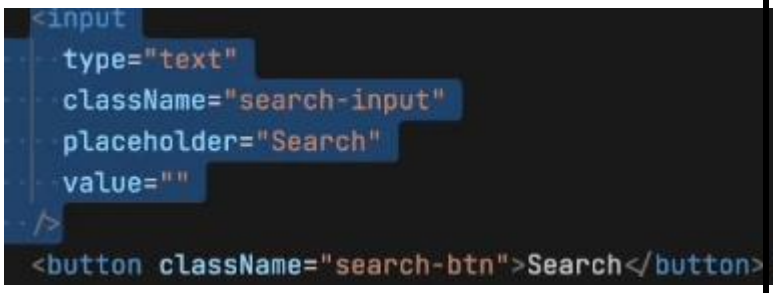
➔ So, you've to always keep your code cleaner and maintainable this was 1st part of this session.

➔ Where to keep our css file, generally will be using tailwind css later onn, for that u don't need css file, for now keep the css in one file only not in sub folder.

Now will be building the Search bar functionality

➔ Now, where do I need search I need it in my body, jst above my res List , give a className "search-container"

➔ Search container should I've input tag in it, also the input should have the value, let it be empty for now, also build a search button. Now in browser when try to write in the input, you wont be able to see **any thing WHY?**



➔ It will work fine in our html, Cause that input is not same as this, Cause this is controlled by react.

➔ **How to make this WORK?**

➔ Even if I hard code the value, and assign it to the value tag in input but, you again you wont be able to edit it.

➔ **React uses One way DATA BINDING-> i.e.,** I may give the data in the code, but when I try to change from the app, it wont effect the value in the code.

➔ Now somehow I want to change it thing, How?

➔ **onChange tag can be used**, will write onChangeInput which takes a function, this function is basically a callback function which e event here `onChange={(e) => onChangeInput}` so whenever input is changed this function get called, you can see it `onChange={(e) => console.log(e.target.value)}`, u can see in console the value that is being written.

➔ But why is it not getting printed over the here, in the box, very Imp question.

➔ SO, the thing is, whenever you write it the react rerender this component but the value is hardcoded right

➔ Now, will this thing works ? It wont work like this as well, So how to make this work, So I'm trying to say is Local variable like (searchTxt) is not preferred in react.

```
value={searchTxt}
onChange={(e) => {
  searchTxt = e.target.value;
}}
```

➔ **But if I need to maintain a variable that changes itself, then you need to maintain a variable that is a react kind of a variable.**

➔ **Now what is a react var? It is a kind of STATE variable, So every component in react maintains a STATE, and you can put your all the variable into the STATE, and everytime you've to create a local variable you create a STATE inside react.**

➔ **Now, Let's Know what is STATE in react?**

➔ **Now,** Suppose I've to create a local variable like searchTxt, then we will create a with something known as **useState hook, Where** does it comes from ? `const [searchText] = useState();`, so such varaible are been created using useState hook, Here hook are nothing but kind of like normal funtions, at the end of the day hook is a nrml function. **Where do I get this function from?** Well I get it **from react** `import { useState } from "react";` **, who wrote this, FACEBOOK developers. So, useState is use to create state variables.**

➔ **Now, How do I use my state variable ? SO,**

`useState(); // To create state variable` this function returns a array, and the 1st variable of you array is the variable name `[searchText]` and the searchTExt is a local state variable.

➔ **These hooks gives** some functionalites, there are different types of hookes that we use across our live course, one of the

such important hook is useState hook, this is used to create a local state variable.

➔ **How** to use this var, we can use it like a normal variable.

➔ **How** to give a default value for our var, you give it like `useState("KFC");` this. Also if you want to modify your varaibel then directly you can do that, how to modify it then? Well useState gives us that function to modify the var, good practice to write setVarname `const [searchInput, setSearchInput] = useState("KFC");`, after this u don't do searchINput = value; u do `setSearchInput("")` or i.e., `setSearchInput(e.target.value);`. Now go and see it will work fine. What ever I write it is getting updated in my local variable

➔ In angular js we got 2 way data binding

➔ So, in react it isn't why it is a good thing, cause your app **becomes unpredictable**. i.e., you may use this var in so many place, so for what we are been changing it its unpredictable. And it is not good for optimization also

➔
```
const searchvar = useState(); // returns = [variable nae, function to update the variable]

const [searchText, setSearchText] = searchvar;
```
It is normal JS destructureing

➔ **React primarily supports one-way data binding, meaning data flows in one direction from parent components to child components. However, you can achieve a form of two-way binding in React using state and hooks**

➔ **Very Important Interview Question -> Why do we need state variable?** Generaly if you've created a var, and presented in the ui, and with some click you want to update the var, but such thing doesn't happen in react, react wont rerender it will just print the code saved value, To do that, react says every time **you want your variable in sync with the UI, you need to use your state variable.**

Crucial Interview Question: Why Use State Variables in React?

In React, the concept of state variables is fundamental for ensuring that your UI stays in sync with your data. Unlike traditional variables, where a change doesn't automatically trigger a re-render, React introduces the idea of state.

Consider this: if you create a variable and display it in the UI, clicking or interacting with it doesn't automatically update the UI in React. React won't refresh; it simply reflects the saved value from the initial render.

To address this, React emphasizes the use of state variables. These variables are specifically designed to keep your UI in harmony with the underlying data. Whenever you want a variable to dynamically influence your UI and trigger re-renders on updates, React insists on employing state variables. They act as a bridge, ensuring that changes to the data are reflected seamlessly in the user interface.

So, in essence, when you're working in React and want a responsive, dynamic UI that updates as your data changes, state variables become the linchpin for achieving this synchronization. They're the key to making your React components truly interactive and reactive to user interactions.

➔ **Try** creating a variable, i.e., when clicked it updated to true value,

```
const [searchClicked, setSearchClicked] = useState("false");
```

```
<button
  className="search-btn"
  onClick={() => {
    setSearchClicked("true");
  }}
>
```

With this the UI gets changed, and get updated. Now let's understand how it is being done. Now when u create the variable state, the react is keeping track of it, now when ever my var gets updated my whole component get rerendered, react is jst destroying the body component and creating the body once again, and it is happening once again. **Reconsilation is happening here. React is very smart it will just rerender the h1 tag,** React just rerender that portion of your dom that is why react is fast, it using diff alaogrithm.

➔ **Let the search work,** when type it, let the rest filter out, the thing is when **clicked on the search buttion need to filter the data, the restaurant list**

➔ Let's create a function and filter data, after filtering we've to update the list for that, we require STATE's. create a state variable by default the data should be assigned to dumy data right, yes

```
const [restaurants, setRestaurants] = useState(restaurantList);
```

, now use your restaurants it will work, with map as well

```
{restaurants.map((restaurant) => {
```

, it worked. Now if I pass with filtered data, will the list changes? YES So what I will do is modify the local state variable.

- ➔ I've to filter the restaurant list using the input I get, and send it to the function

```
<button
  className="search-btn"
  onClick={() => {
    //need to filter the data
    const data = filterData(searchText, restaurants);
    // update the state - restaurants
    setRestaurants(data);
  }}
>
```

then I should be implementing the function which does such work, which give me filtered data.

- ➔

```
function filterData(searchText, restaurants) {
  return restaurants.filter((restaurant) => restaurant.data.name.includes(searchText))
}
```

- ➔ Now try to run it, it will work.
- ➔ **HomeWork:** Do, the toggle thing that true/false.and answer this

```
// What is state
// what is React Hooks? - functions,
// What is useState
```

then also restructure your folder then play with export and import, also clear your understanding in onChange(e) on 'e' . Also find out empty search why it did not work make it work.

HomeWOrk ReadOUTS:

What is the Virtual DOM?

The virtual DOM (VDOM) is a programming concept where an ideal, or "virtual", representation of a UI is kept in memory and synced with the "real" DOM by a library such as ReactDOM. This **process is called reconciliation.** This approach enables **the declarative API of React**: You tell React what state you want the UI to be in, and it makes sure the DOM matches that state. This abstracts out the attribute manipulation, event handling, and manual DOM updating that you would otherwise have to use to build your app.

React, however, also uses internal **objects called "fibers"** to hold additional information about the component tree. They may also be considered a part of "virtual DOM" implementation in React.

Is the Shadow DOM the same as the Virtual DOM?

No, they are different. The Shadow DOM is a browser technology designed primarily for scoping variables and CSS in web components. The virtual DOM is a concept implemented by libraries in JavaScript on top of browser APIs

What is "React Fiber"?

Fiber is the new reconciliation engine in React 16. Its main goal is to enable incremental rendering of the virtual DOM. Read more.

React Fiber is a fundamental rewrite of the core algorithm used by React to update the user interface (UI). It was introduced in React version 16.0 as a new reconciliation engine. The term "Fiber" refers to the internal data structure used by React to represent the components in the virtual DOM.

The main goal of React Fiber is to enable incremental rendering of the virtual DOM. In the context of React, reconciliation is the process of determining what changes need to be made to the DOM to reflect the updated state or props of a component. Incremental rendering means breaking down the rendering work into smaller chunks and spreading it over multiple frames. This allows React to better prioritize and manage the rendering process, making UI updates more efficient and responsive.

Key points about React Fiber:

1. **Incremental Rendering:** Fiber allows React to work on rendering and updating the UI in smaller, prioritized units. This means that the rendering work can be interrupted and resumed, allowing for better responsiveness and perceived performance.
2. **Prioritization:** Fiber introduces a priority-based scheduling system that enables React to prioritize different types of updates. This is crucial for ensuring that high-priority updates, such as user interactions, are processed quickly while less critical updates may be deferred or canceled if necessary.
3. **Concurrency:** React Fiber introduces the concept of concurrent rendering, which means that React can work on multiple tasks concurrently without blocking the main thread. This is particularly beneficial for applications with complex UIs and interactions.
4. **Better User Experience:** The improvements brought by React Fiber lead to a more responsive user interface, especially in applications with dynamic and frequently changing content.

In summary, React Fiber is the internal engine of React that powers the reconciliation process. Its incremental rendering approach, along with prioritization and concurrency, enhances the efficiency and performance of React applications, providing a smoother user experience.

Guide on Reconsilialtion

https://legacy.reactjs.org/docs/reconciliation.html

process of reconciliation in React, which is the mechanism by which React updates the user interface efficiently in response to changes in state or props. The generic solutions for transforming one tree into another have a complexity of O(n^3), making them too expensive for practical use.

React relies on a heuristic O(n) algorithm based on two key assumptions:**Elements of different types will produce different trees. Developers can use the key prop to hint at stable child elements across renders.**

Diffing Algorithm:

- When diffing two trees, React compares the root elements.
- Elements of different types lead to a full rebuild of the tree.
- DOM elements of the same type are updated by modifying only the changed attributes.

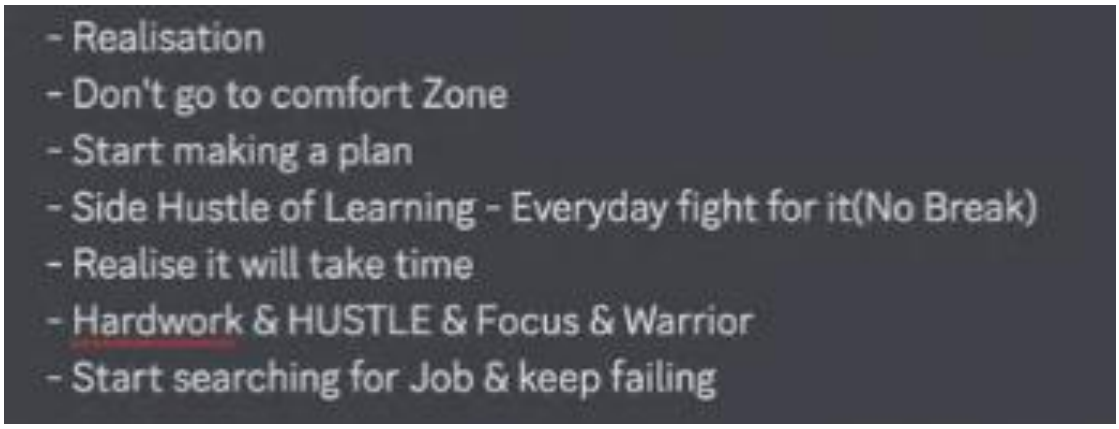
- Component elements of the same type update the props of the underlying component instance

Stability, predictability, and uniqueness are crucial for keys. Unstable keys, like those from Math.random(), can lead to unnecessary recreations and performance issues.

The goal of React Fiber is to increase its suitability for areas like animation, layout, and gestures. Its headline feature is **incremental rendering**: the ability to split rendering work into chunks and spread it out over multiple frames.

Other key features include the ability to pause, abort, or reuse work as new updates come in; the ability to assign priority to different types of updates; and new concurrency primitives.

https://robinpokorny.com/blog/index-as-a-key-is-an-anti-pattern/



Think about how much value you give to the company, by haiving that stack

➔ **Learn to negotiate for the salary**

8. Exploring the world 02-02-2024

- ➔ Last class we tried to build a food ordering app like swiggy, also learned about config driven UI, it is a big configuration or a json object is been send from backend most of the times, or u can keep it hardcoded. Any config can power the UI.
- ➔ Revision on hooks, states (for syncing).
- ➔ **Why is react fast? : virtual Dom , reconciliation, diffing algo, fibre architecture.**
- ➔ How **does reconcilitation process works?**
- ➔ With diffing algorithm, two tress been compared the old and updated one, the diff is updated in the virtual dom a
- ➔ **React Fibre is the new reconciliation algorithm**.
- ➔ Well what is virtual DOM, we know it got a representation of tree, but at the end it a javascript object.
- ➔ The react **is FAST because of it's fast DOM manipulation, it is the most expensive operation in the UI state. IT is done because of the super powerful diff and fibre.**
- ➔ **We use useState variable to make our variable in sync cause rect does not maintain a sync with normal variable, so useState hook provide us the local react variable**
- ➔ **Diff Algorithm is the core of react**
- ➔ **We** consider {} while importing useState cause it is named export
- ➔ **React** will keep track of such variable only

```
const [title, setTitle] = useState("Food Villa");
```

these state var

- ➔ On button click we can update the title, using setTitle

```
<h1>{title}</h1>
<button onClick={() => setTitle("New Food App")}>Change title</button>
```

➔ While doing this, consider your inspect window, and observe the code, will be able to see that only the title h1 tag get's changed. Here's where the reconciliation process ticking in .

➔ Write a console.log and observe you see that, after click in the button the console again prints, **you see that it is rerendering the whole component.**

➔ In last class in the search box if we type text, then when each key is being pressed the react is rerendering every single time in such fast speed.

➔ Every UI change requires State, suppose if I don't want to change any thing in my react then I can use a local variable

Now let us try to make his thing with actual data, such that not limited to local data, rather let's explore the world

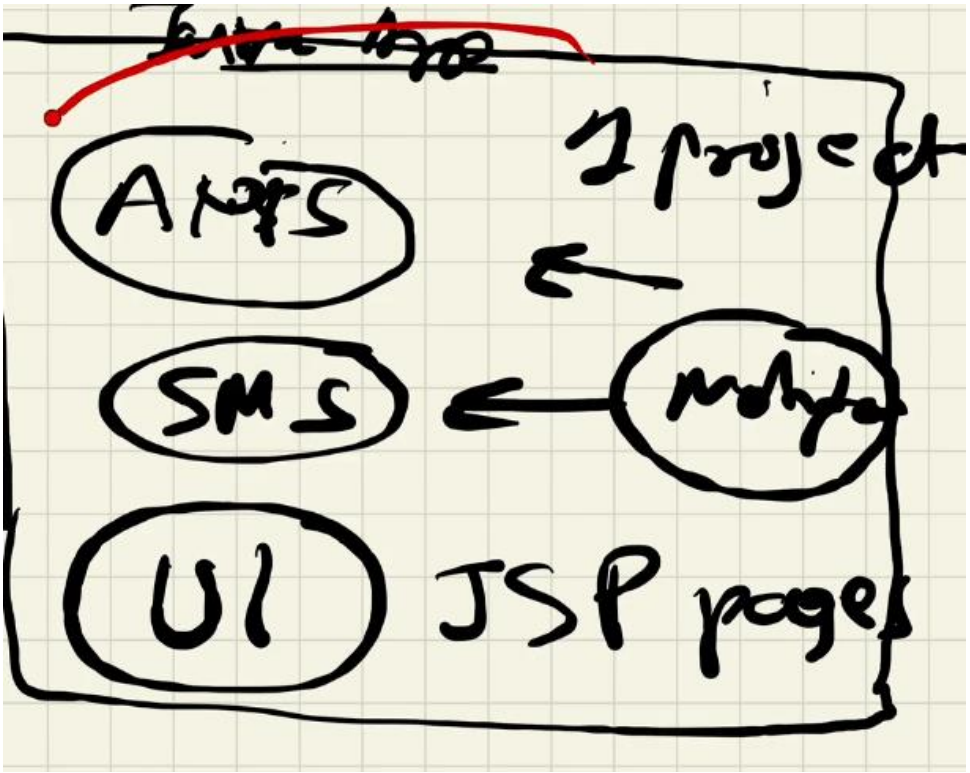
➔ Such that, will be taking to some other API the other world, which our application does not know.

➔ **Let us talk about important concept which we use in our industries : MICROSERVICES**

➔ **When** you've to build large application like swiggy/uber/amz do you think there is only one react project which is working behid the scenes, It is not possible. Guess how many microservices are their inside uber there are 100s of them.

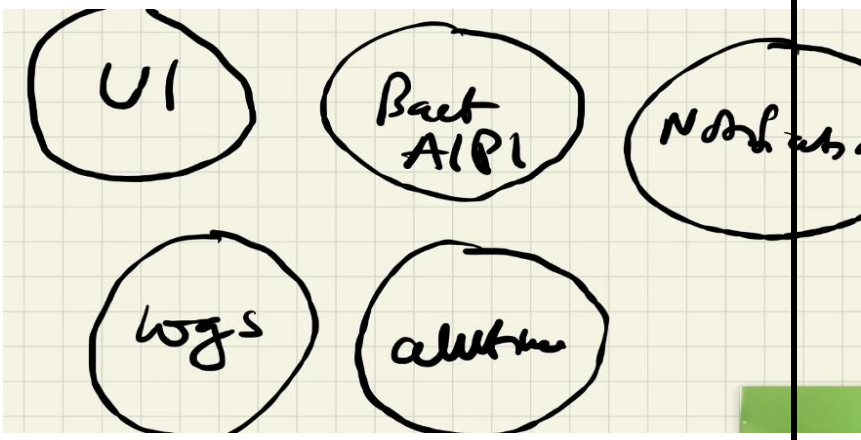
➔ **What is microservices:**

➔ Way back, there used to be a single big application



➔ Every thing used to be in the same project a java project, how we used to deploy **we used to build the whole application to change one button, jst for 1 but change used to deploy the whole java application**

➔ **This architecture is known as monolith architecture, it got its advg but world is moving towards microservices**



➔ Now, in microservice, instead of having just one project will be having small small diff projects, **also known as separation of concerns**

➔ **Major adv of microsercies:** Easier to test. The best part is you can use different techstack here, java->bcnkd, notification->pythonscript, log->pyth, UI->react, auth->go lang etc., That is why big companies use everyting.

➔ **In uber most of the bcknd was written in GO LANG.**

➔ **You know how swiggy have been build it is** using that dapi

```
https://www.swiggy.com/dapi/restaurants/list/
```

➔ **Now tell me where this foodApp lies in?**

➔ **It is the UI microservice that we are building**

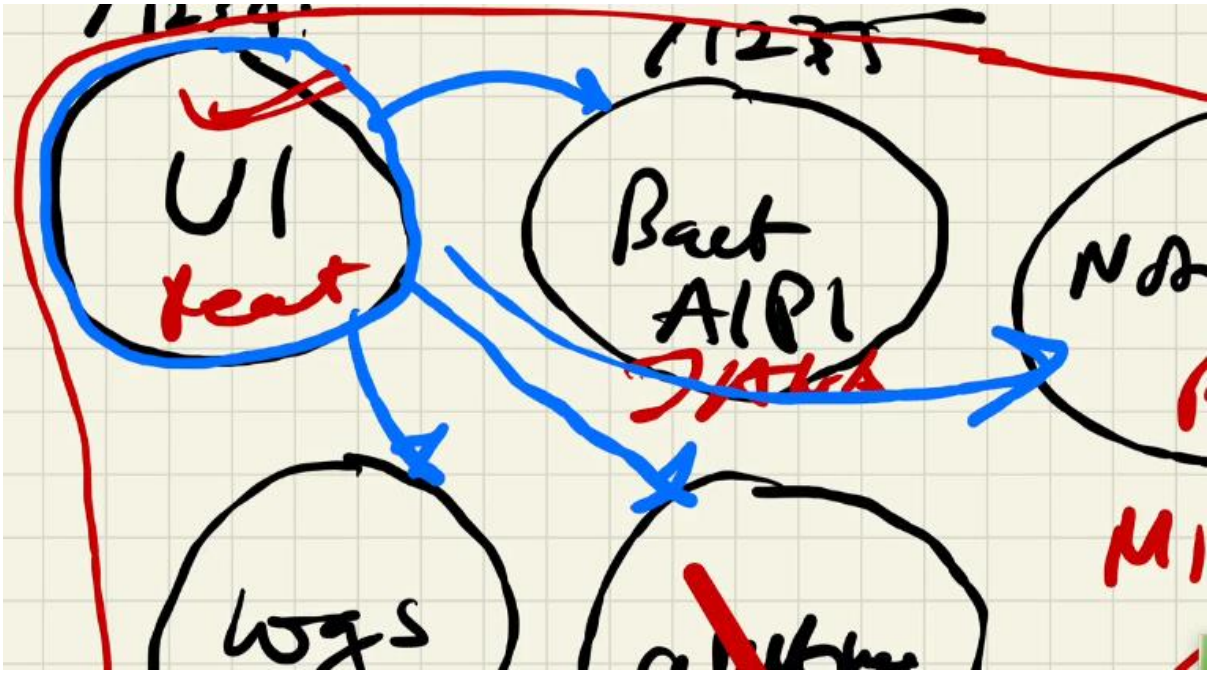
➔ **It is one UI project, basically we are building that is deployed in swiggy.com**

➔ **Now how this applications are all connected?**

➔ Different port is mapped to

➔ All these are deployed to diff ports but the same domain name

➔ Now will be exploring the world, the world for UI projects is exploring other services



➔ **Now tell how do we explore the world in JS?**

➔ **It is fetch, promises, ajax etc.,**

➔ *Fetch is available to us through js window object, it's a browser API, super power given to us*.

➔

➔ Let me tell you one **thing where should I make my API call ?**

➔ **Basic concept even experienced react dev don't know this**, if I make it insider the body comp, as we know on any state change react rerender the component. Therefore placing the API call there is not good, cause **it will fetch every single time**.

```
const Body = () =
  const [restaurar
  const [searchTex

  fetch();
```

➔ **this is not a good place**

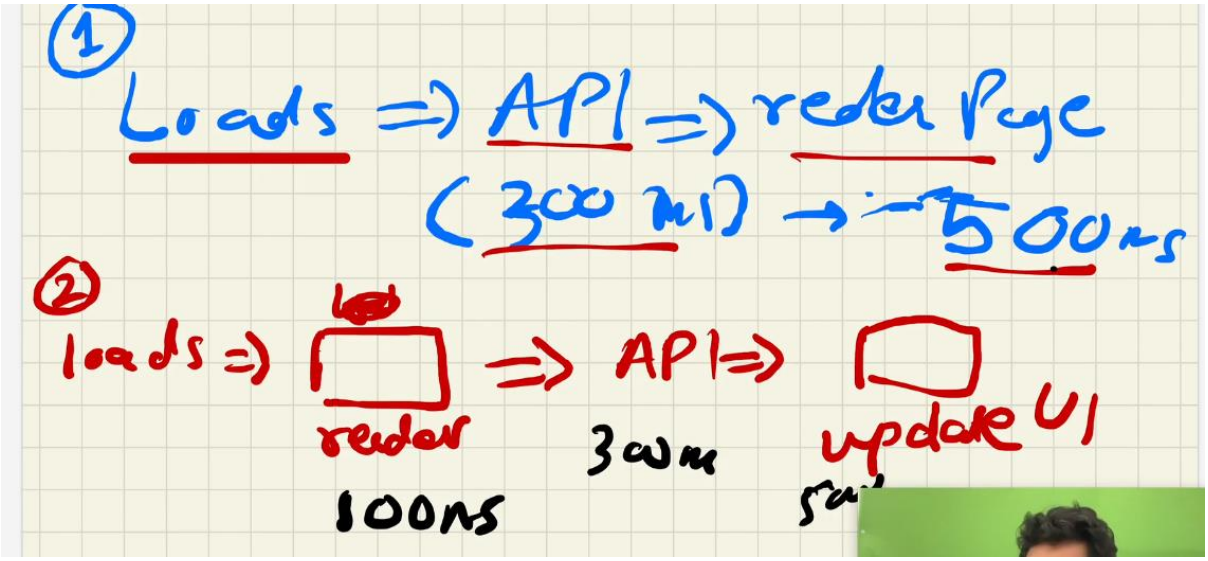
➔ **The very good way to call an API inside our body.**

➔ **Let know the feature we are going to build**

➔ **The feature Is as when our page loads it used to call an API and fill the data.**

➔ **So, lets check the good way to handle this in react**

➔ **There are 2 ways,**



➔ Which **one is the best way? Why 2nd is the good way**

➔ Because of **the user experiencec, the page** is **availbe fast, than that of 500ms**

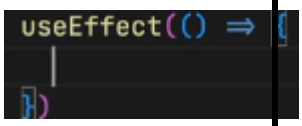
➔ In React we generally we always do like this.

➔ In react it always tells us to do somethig like this, and it has given us a functionality to make this happen, It gives us the access of 2nd most important hook it **is useEffect**

➔ Jst like useState, the useEffect comes from react library

➔ **W**ell useEffect is a function and you call this function by passing another function to it, it the callback function, i.e., it is not **called immediately it is called whenever the useEffect wants it to call**, and **react make sures that it call it at a specific time the time is**

➔ Whenever our component renders and rerenders, whenever it rerender what happens is 1st the code of this function is called and after every **render,it will the function we passed inside the useEffect. And when will my component render, there are 2 times**



➔ **1 is when state changes or my props changes.**

➔ But we don't want to call it after each rerender, that's the bad wa**y right, SO, pass in a empty dependency array to it.**

➔ **Now,** Suppose I want to call this useEffect only when searchText changes! Then will pass it in dependency array

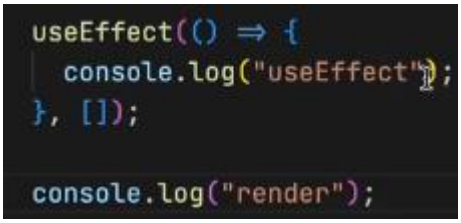
➔ **Try** this it will call on every searchText update.

➔ **If** it is not dependent then it will call only 1s, because it is not depended on anything. Hope this get cleared about useEffect.

➔ **Now lets play with it for all ans cleareance**

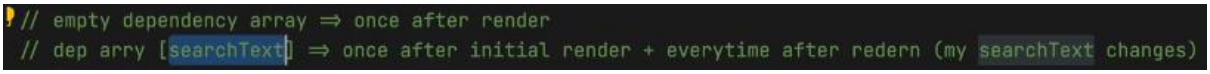
➔ Let it be dependent on restaurant change, check it out

➔ Now if no dependency then when it will get called once, before render or after render ? also try this what will be called 1st render or useEffect ?



➔ So, it will call render 1st then after initial render useEffect.

➔



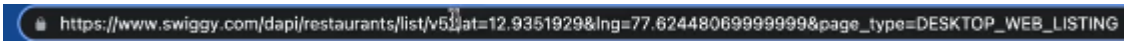
➔ **Now we got our answer, when should we be calling our API, it is using empty dependency array.**

```
useEffect(() => {
  // API call

}, []);
```

➔ **How do u make a API call?** FETCH

➔ Let's try to fetch swiggy's API



➔ So, the call happens asynchronously right, so should we use promises or asyn await.

➔ Let us use asyn await, it is a good way. One and same thing, but asyn await is the most preffered way.

```
useEffect(() => {
  // API call
  getRestaurants();
}, []);

async function getRestaurants() {
  const data = await fetch("https://ww
  const json = await data.json();
  console.log(json);
}
```

➔ It is giving error, cant access who is giving it is it swiggy ? the browser is giving these error, it says from localhost to some api,you can make it.

➔ To modify this, there is a plugin named Allow cors. If you don't know watch the cors video https://youtu.be/tcLW5d0KAYE?si=aBPQAc6CGt7n2vE-

➔ It worked, and we see the json data in the console.

➔ Even after updating the state, we can see that the api is not being called once again.

➔ **Now, we** can use this data, now we don't need the restaurant data we got in our constant file. Will update the restaurant state variable

➔ `setRestaurants(json.data.cards[2].data.data.cards);` **this is how we call in from getRestaurants fun itself.** But this practice is not good such that, what if my data was not here it wil break.

➔ **We've** to do optional chaining

```
const json = await data.json();
console.log(json);
// Optional Chaining        You, 1 second ago • Uncommitted
setRestaurants(json?.data?.cards[2]?.data?.data?.cards);
}
```

➔ It will work, you can see new restaurents in the UI, this is amazing about react, the sequence is, 1st the state var contains the old data, it get rendered then the api call get executed then the data gets fetched then the data of state var get updated then again the component gets renderd.

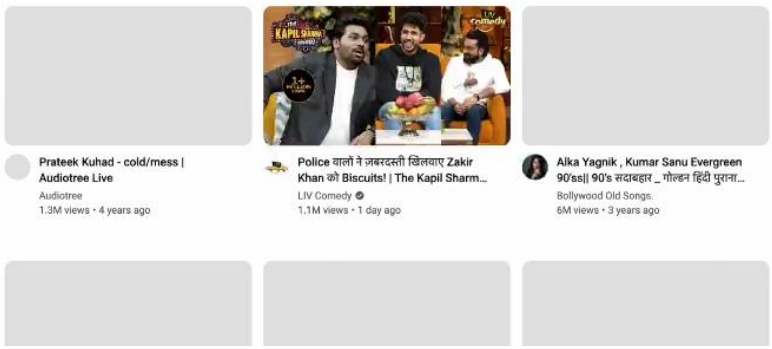
➔ When you reload you will be able to see the change.

➔ Let's try to watch it let's debug it from inspect

➔ But debuggers and run and let us see what order do they happen.

➔ Now when you are trying to lead the page, it is not good, as we see it. So work on it, also consider watching what is swiggy is doing, when u reload it's page. It's amazing the way it is

➔ Earlier there used to be loaders, now psychologist figured out let us show them empty boxes, cause our eyes don't hurt, there is no sudden change, **this is some UI design principlal it is called shimmer effect. Every cmpny is following this effect.**



➔ This is shimmer effect.

➔ Take this as a homework you should be showing a shimmer UI, in you app.

➔ Now, where do you will be fitting the shimmer effect ?

➔ How do you render your shimmer effect when data is not called

➔ **One more important concept, the last concept today I want to teach in the class which is known as Conditional Rendering.**

➔ i.e., you've to render eather the shimmer UI or the nrml UI.

➔ Psueido code

```
//Conditional Rendering
//if restraunt is empty ⇒ shimmer Ui
// if restraunt has data ⇒ actual data UI
```

➔ Let's make a shimmer UI component, As we what is everything in react -> it's compnents

src > components > JS Shimmer.js > default

```
1  const Shimmer = () => {
2    return <h1>Shimmer UI Loading.....</h1>;
3  };
4
5  export default Shimmer;
6
```

➔ **will be using ternary operator**

```
return restaurants.length === 0 ? (
  <Shimmer />
) : (
  <>
    <div className="search-container">···
    </div>
```

➔ **Have your curiosity in you and consider debugging and exploring the inspect**

➔ **You** see, the search is not working here, cause of no data, also because we have changed the layout of the data. Our restaurents get filterd out.

➔ **Now let's** keep a copy of restaurents and change the other one to filtered restrautents

```
const [allRestaurants, setAllRestaurants] = useState([]);
const [filteredRestaurants, setFilteredRestaurants] = useState([]);
const [searchText, setSearchText] = useState("");
```

➔ **Now, when I call the API, I set the allrest var,** when you do this, it will give error , Therfore, for the 1st time let us make both.

➔

```
setAllRestaurants(json?.data?.cards[2]?.data?.data?.cards);
setFilteredRestaurants(json?.data?.cards[2]?.data?.data?.cards);
```

➔ **Also** change here `return filteredRestaurants.length === 0 ? (`

basically the shimmer is only shown when there is no data, why

filterres, why not all res, cause you are mapping your cards from filtereRes.

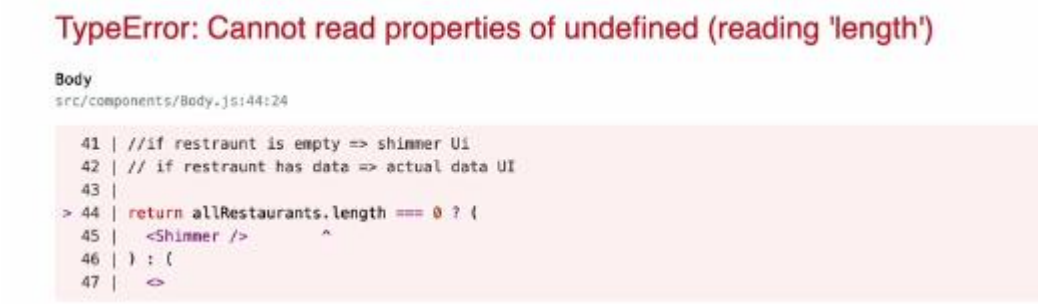
➔ **Now** let us try to use search.

➔ **After** searching one, and getting the res, if you again search something, wil it get poped up ? YES,

➔ **When my page loads it should be having all restaurens,**

```
return allRestaurants.length === 0 ? (
  <Shimmer />
) : (
```

, When we rendering It, it's givng error.



➔ At that span of time, there is nothing like all restrauent

➔ **So, how to avoid rendering a component?**

➔ You, can using optional chaing as well as, you can say that when you are rendering `ants?.length === 0 ? (` or

```
// not render component (Early)
if (!allRestaurants) return null;
```

Will also be doing for filter restrauent

```
if (filteredRestaurants?.length === 0)
  return <h1>No Restraunt match your Filter!!</h1>;
```

➔ **Now, you know about it make your UI smoot**

➔ **Do this homework:**

```
<div className="restaurant-list">
  {/* You have to write logic for NO restraunt fount here */}
  {filteredRestaurants.map((restaurant) => {
    return (
      <RestaurantCard {...restaurant.data} key={restaurant.dat
    );
```

➔ **Considering doing this:**

```
"Roti" === "roti"
false
"Roti".toLowerCase() === "roti".toLowerCase()
true
```

➔ **Always** remember to put optional chaining

➔ **Now** let's try to build header containing login and long out

➔ `<button>Login</button>` `<button>Logout</button>` should I be showing two login buttons, No right so let's do it.

```
const authenticateUser = () => {
  // API call to check authentication
  return true;
};
```

➔ Suppose we got a authentication api, now let's build function for this.

➔ One person asked this question can we write this way, can we include js this way in our react code. The answer is NOOOO

```
</div>
{
  a=10;
  console.log();
}
```

➔ SO, the crct answer, is any JS works over here but only a JS expression not statement.

➔ Same, way writing if is not valid it is a stmt, where as ternary operator is an expression, so it's ok.

➔
```
{loggedInUser() ? <button>Logout</button> : <button>Login</button>}
```

➔ **Now,** what if, when we click it, then it should change the button itself in the UI and show me the other button.

➔ **For** that we've use state variable.

`const [isLoggedIn, setIsLoggedIn] = useState(true);`, will the passing of true works here, or is that is taken as string, well in the expression we can have it.

➔
```
{isLoggedIn ? (
  <button onClick={() => setIsLoggedIn(false)}>Logout</button>
) : (
  <button onClick={() => setIsLoggedIn(true)}>Login</button>
)}
```

➔ **This** can make a toggle effect.

➔ And here the concept **of reconciliation comes into picture.**

➔ **So,** when someone ask why your app is fast, then you can say, there are a lot of things making it fast, there are bundlers, which does minification , trans.., removing console.log, doing image optimization , and parcel is the beast, and we also have

in our application REACT, and virtual Dom , which has reconsilation , diff algo, Dom updates are don very fast. **This is how you've to explain your interviewer like this.**

➔ Where do you think that diff algo is written in react or reactDOm library ?

➔ The answer is, Diff algorithm is core of react it is written in react's core. And the DOM updates happen via reactDom, also the div algorithm works the same way in react native.

Now, Let's wrap things UP.

Homework:

➔ built the login logout button

➔ make your search work,

➔ explore the world

➔ , handle shimmer,

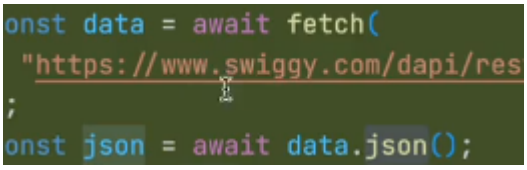
➔ handle edge cases

➔ , make a loading screen,

➔ you've to do early return,

➔ DO useEffect

➔ Also learn this why do we do this twice, data.json after our fetc



➔ Read about microservice and monolithic arch

Recap:

➔ We read about useEffect, a diff hook. Be perfect In this 2 hooks.

➔ We make our search work for upper and lower case.

➔ Early we had 1 res, now we've 2 list of res state variable

➔ But we are rendering-> we're rendering filterdRestr in the resd function.

➔ Then why allRestrauent req? TO SEARCH to filter

➔ We learned about dependency array.
➔ Then we made a API call also made async await fun
➔ **Then** we set our setallRest and setFilterres, then did conditional rendering
➔ **We** also studied microservices, monolithic services.
➔ **We've** build small mircorservice UI.
➔ **ALos,** studied how these are connected
➔ **Also, studied** good way of rendering when using API

Chp 9: Finding the Path 05-02-24

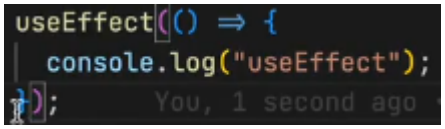
We learned about useEffect hook in the last class, what is its use?

Using API is jst what we've done in it, well basically useEffect is a hook, which react gives us which is called after the component is rendered.

It Takes, 2 parameters, 1 is callback function and the other is dependency array.

➔ The callback function will get called after the rendering.

➔ The useEffect can even get called without passing 2nd para, If

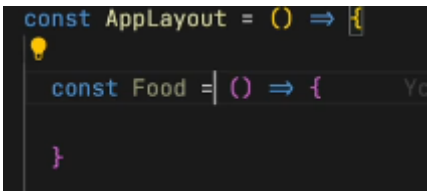


we don't pass anything the useEffect get called after every render.

➔ If **[], then onlt be called once after the 1st render it mean** it **got no dependency**. And if I put searchText, then it will be called everytime my searchText changes.

➔ 1 more important thing react doc says that**, never create a component inside a component.**

➔ Never do this:



➔ Such that keep it on the top, level. Cause of rendeing, if our apps get renrded then ur comp get created how many times?

➔ **One more important thing about hooks, also never ever write a useState inside if else**

u don't've to do this.

➔ **React does not like inconsistency, cause** react don't know it will there are not, i.e., it will be available for the app or not.

➔ It is **not a optimized way, react** likes concreate things.

➔ Never do this, and also u never get a case to do this, hardly .

➔ Also, never write inside a for loop.

➔ **useState is a hook that react gives to create local state variable inside your functional components. SO never use useState outside your functional component.**

➔ Can **I use more than one useState inside my functional comp, YES. According its usecases.**

➔ If you want to keep local images then you can do this, by creating a folder named assests. Then keep your img in it.

➔ Also, you can consider the images from cdn, when u r building an app**, cdn images are great place to host your images your homework is to learn about this.**

➔ When we put images into cdn we can put optimized imgs, cause from there no optimized images comes.

➔ Did you implemented the Shimmer effect? NO.

➔ Lot of u implemented using react shimmer, using npm package.

➔ Don't use it, it is such an easy task to create. No need of external package for this.

➔ For everything you should not be importing packages, be causious about what you are importing.

➔ Npm is-odd package. Such an easy fun, helps to say num is odd or not.

➔ **Now, let's try to create a shimmer. We''ve to create a similar kind of view with dummy cards.**

Simply we're creating an empty array of 10 and mappig it to div

➔ Consider adding the key to it, for nw I added index.

➔ The thing is when you've to do tideous work then consider going for external libraries.

➔ Like **let's say you've to build forms which contains** lot of constraints. Then you can consider using npm pckg, like formik

➔ **Formik is a great library.**

➔ In this lesson, will be learning and **creating lots of routs.**

➔ Such that, let us t**ry to make diff pages for Home, contact, About etc.,**

➔ For this will be **using a npm package a library, it is a very popula libr, and a standard lib now and ever one uses it.**

➔ **It is REACT ROUTER.**

➔ Will be installing through our terminal : npm I react-router-dom, we can see in package.json

➔ Let us start using it.

➔ Will build about page, contacat page and see how routing works.

➔ Shortcur for comp, rafce for a component.

➔ Always try to make your component from scratch, make a habit

components > JS About.js > About

```
const About = () => {
  return (
    <div>
      <h1>About Us Page</h1>
      <p>
        {" "}
        This is the Namaste React
      </p>
    </div>
  );
};

export default About;
```

➔ Now, I want to create a route, i.e., when I write /about it should display about page.

➔ For this I've to create a routing configuration.

➔ Will be importing a function: createBrowserRouter from our new pckg. It will help us create routing.

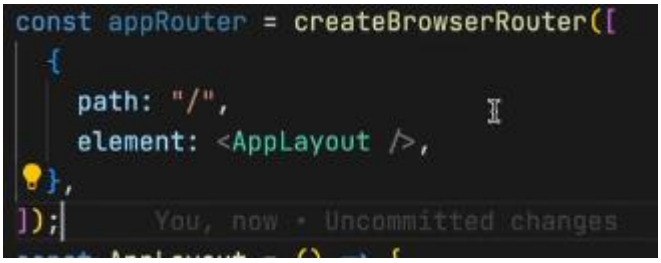
➔ How do I create routing now?

`const appRouter = createBrowserRouter();`, Now when we use this function, we're creating a router. It takes in some configuration. This is the place where I will define what will

happen if I load my /path. Now I will create routing configuration

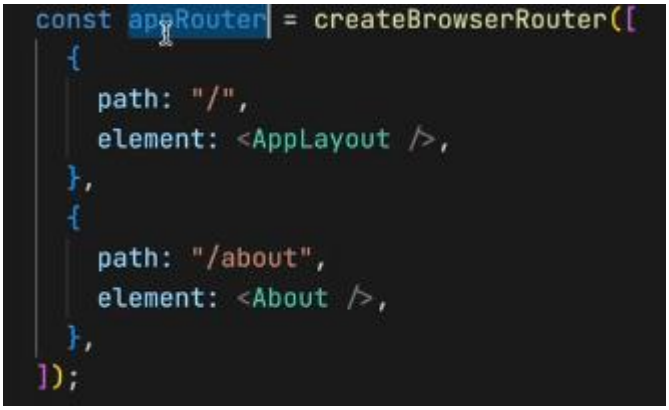
➔ Some are saying why r u not using diff types of router, there are multiple router that we can use, that has there own way's of creating routes. Got to the documentation and read about all routeres. For now we are using CreateBRowser Router it is the recommended router for all react router web projects.

➔ Now for the configuration inside the function, will be giving path and in the element will will mentioning the component.



➔ Also, you should be keeping this fun, below your applayout component, i.e., it should be in sequence.

➔ Now, import about comp, and try to do same as you did in above.



➔ Now, do you think that this will work Nooo

➔ Why? Because we need provide this approuter to our app.

➔ How do we provide this?

➔ There is a component, RouterProvider, coming from react-router-dom.

➔ Now, in render we are rendering our APplayout do we always want to render APplayout ? No, we want to render according to our path. SO what we say is that render my RouterProvider by passing some config i.e., router={appRouter}/>

➔ Now what ever the rout will rnder it will render according to this configraion.

➔ `root.render(<RouterProvider router={appRouter} />);` this is known as props, passing.

➔ Everything is available in documentation.

➔ Run it you see the working.

➔ If you try giving wrong path, then the library is giving us a good UI for the error, it is not showing any error in the console.

➔ This is something a great user experience.

➔ Now, let us create **our error page.**

➔ E**rror page is also a component.**

```
src > components > JS Error.js > ...
1   const Error = () => {
2     return (
3       <div>
4         <h1>Oops!!</h1>
5         <h2>Something went wrong!!</h2>
6       </div>
7     );
8   };
```

➔

➔ Pass this error, component in the router configuration.

➔ Now, will be giving errorElement.

```
const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
    errorElement: <Error />,
  },
  {
    path: "/about",
    element: <About />,
  },
]);
```

➔ **now if lets say we want to display some more information about this error, HOW?**

➔ **React-router gives us, useRouteError.**

➔ It is a hook, which react-router-dom gives us. `const err = useRouteError();` `console.log(err);` have a look,

➔ Consider **display what you want to**.

➔ This was the routing session, take break will be **learning, nested component, nested routing, and dynamic routing.**

➔ Now, if want to go to about page by clicking it, what should I use ? some saying onClick and some anchor tag.

➔ The problem **with the anchor tag is it will refresh the whole page.**

➔ **Building a single page application.**

➔ **This concept- SPA: with single page application our app does not reload it is not making network call when we are changing pages.**

➔ **There are two types of routing.**

➔ **1. Client side routing and 2. Server side routing.**

➔ In **server side routing it is the way where all our pages comes from server.**

➔ But in our applications, **in our React application will be building client side routing. i.e**., when I click my about page, it will not make any network call for it**. Don't've to go to server and fetch Cause all our compoenets are already there in our code**.

```
<a href="/about">
  <li>About</li>
</a>
```

➔ When we do this it will reloade the page.

➔ Now react-router-dom helps us here, without reloading the page, it give **us something known as Link**, import it from **router-dom**, will use our link exactly as we did for our a tag.

```
<Link to="/about">
  <li>About</li>
</Link>
```

➔ Same **can be done with Home**.

➔ **When you go on checking in the console, you see an anchor tag. Why? At the end of the day, Link uses anchor tag.**

➔ **Rect-router-dom is keeping track of all these links.**

➔ **Now lets work on Nested Routing.**

➔ So, the problem with my app is, when I open my about page, **where is my header and footer?**

➔ Now, the thing is I **want to keep the header and footer intact.**

➔ Will **have to change our routing config.**

➔ What I want is something like this, I will have to change my confi some thing like, 1st I will have to make **about as children of AppLayout. For**

```
<Header />
<About /> // if path is /about
<Body /> // if path is /
<Footer />
```

this, will be using, children: in our config, It will take the same configuration as the path do.

```
const AppLayout = () => {
  return (
    <>
      <Header />
      <About />
      <Body />      You, 7 days ago • chapt
      <Contact />
      <Footer />
    </>
  );
};
const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
    errorElement: <Error />,
    children: [
      {
        path: "/about",
        element: <About />,
      },
      {
        path: "/contact",
        element: <Contact />,
      },
    ],
  },
]);
```

➔ This way

➔ Now let's create a contact component, and also will be adding one more children as contact.

➔ Will this thing works, well we've put all of them together, every thing comes in our page.

➔ This is not the right way to do it. If the page is /about I want about page if it is /Contact I want contact page.

```
<Header />
{/* { Outlet } */}
<Footer />
```

➔ **We've want such layout, we want the header and footer to be intact, and the changes happens in outlet.**

➔ **React-router-dom, gives us access to something known as Outlet.** It is a component

```
<Header />
{/* { Outlet } */}
<Outlet />      You
<Footer />
```

➔ **This outlet will be filled by my children configuration.**

➔ **SO the thing, is all our children will go into our OUTLET according to the route.**

➔ **/about /contact it gives but the body is not available, for that I will need my body compoenet as children in the config.**

➔ **Now you can see that working, header and footer does not go anywhere while you switch between apps.**

➔ **Now consider cheking the inspect, you will be noticing that header and footer are intact.**

➔ Our reconislaiton algorithm exactly knowns what need to be reloaded.

➔ **Html doesn't recognize outlet.**

➔ Consider playing with this

➔ Will now learn a important co**ncept known as Dynamic Segment.**

➔ You can observe in the swiggy website, when you click the restaurant you see it is changing dynamically from restaurant to restaurant.

➔ `path: "/restaurant/:id"` **This id is dynamic.**

➔ Now let us try to build this cool thing.

➔ Let's build a component, **named RestaurantMenue.jsx**

➔ When I clikck some hotel, it should take us to that page.

```jsx
const RestaurantMenu = () => {
  return (
    <div>
      <h1>Restraunt id: 123</h1>
      <h2>Namaste</h2>
    </div>
  );
};

export default RestaurantMenu;
```

```jsx
{
  path: "/restaurant/:id",
  element: <RestaurantMenu />,
},
```

➔ When you add that path, it will work.

➔ **Now let us try to read the id from the url.**

➔ **Import useParams from react-router**-dom, and use. For reading id, I can destructure our params

```jsx
const params = useParams();
console.log(params);
```

```jsx
const params = useParams();
const { id } = params;
```

```jsx
const { id } = useParams();
```

➔ both ways are same.

➔ **Try going to some hotel in swiggy and see the API call,**

➔ You will be **seeing all the restaurant details**

➔ **Now let us try to take those details here, using useEffect**

➔ **In that we pass a callback function and dependency array, also will call a asyc function, async is to handle our promises from that function.**

```
useEffect(() => {
  getRestaurantInfo();
}, []);

async function getRestaurantInfo() {
  const data = await fetch(
    "https://www.swiggy.com/dapi/menu/
  );
  const json = await data.json();
  console.log(json);
}
```

➔ Will be having our api data in console.

➔ **Let us try to render this in our page.**

➔ **Now how to render this.**

➔ **Will be using useState for this.**

➔ Don't relay on autoImport, you should be knowing from where it is coming.

➔ `const [restaurant, setRestauraunt] = useState({});` and call setRes from the getRes function .

➔ Now will display the resInfo, will be using same url for img, import from costants.

➔ **Will've to iterate on this.**

➔ `<div>{console.log(Object.values(restaurant.menu.items))}</div>` We are doing this, cause It is converting our objects to array? Read about it and understand.

➔ Always **remember when you r using map, use key.**

➔ We'r getting an error, cause at initial render, our restaurant are not available so in useState don't **pas {} object, pass null,** cause as it is an empty obj and still I'm reading a lot of things form that empty object, also **write optional chaing there**

➔ **then we can also do a early return.**

➔ **will be showing shimmer**,

```
if (!restaurant) {
  return <Shimmer />;
}
```

➔ Or **can also write as ternay**

```
return (!restaurant)? <Shimmer /> :(
```

```
  return (
    <div class="menu">
      <div>
        <h1>Restraunt id: {resId}</h1>
        <h2>{restaurant.name}</h2>
        <img src={IMG_CDN_URL + restaurant.cloudinaryImageId} />
        <h3>{restaurant.area}</h3>
        <h3>{restaurant.city}</h3>
        <h3>{restaurant.avgRating} stars</h3>
        <h3>{restaurant.costForTwoMsg}</h3>
      </div>
      <div>
        <h1>Menu</h1>
        <ul>
          {Object.values(restaurant?.menu?.items).map((item) => (
            <li key={item.id}>{item.name}</li>
          ))}
        </ul>
      </div>
    </div>
  );
};

export default RestaurantMenu;
```

➔ Now, **when I try another id, we are seeing same restaurant, it is not loading another res, cause we've hard coded our API**.

➔ **I've to pass dynamic id and add this to the url.**

➔ Listen it carefully, when we write our code, in modular fashion then we can reuse it a lot.

➔ Now, this is the power of react-router-dom.

➔ Now, let's make this happen i.e., **when I click on some res, I should be able to land on it.**

➔ We've **to link the resMenue comp**

➔ **In our body, we're mapping the resCards, let us wrap it inside the Link,**

➔

```
{filteredRestaurants.map((restaurant) => {
  return (
    <Link to={"/restaurant/" + restaurant.data.id}>
      <RestaurantCard {...restaurant.data} key={restaurant.data.id} />
    </Link>
  );
```

➔ Let me tell you on thing, the key should be in Link, as we are mapping our **link component, the key should be in our link comp.**

➔ 

This way

➔ **Now it will work, when you click on it , it will redirect to our resMenu comp.**

➔ This is very important thing, you homework for today is you've to make everyting dynamic in you app, routing should be proper, you **should be able to exactly know about nested routes, dynamic routes, how we handle error in our route, how do we create a error page, lots of things we did today**.

➔ Let's recap:

➔ I can create multiple useEffect.

➔ I should not write a useState inside our if else.

➔ Also should not in forloop.

➔ Showed how to import image in our app , assets/img

➔ Also told about how to import shimmer, and don't use external lib if not needed.

➔ For form building the lib was, formic. It is very helpful

➔ ExtraHomeowrk: Build a login Page, when u click on it it should take us to the Body page.

➔ Try to use formic, are read it's whole documentation.

➔ Then we learn about routing, we used react-router-dom lib.

➔ We've used createBrowserRouter, it takes and obj or array?

➔ It takes, array of configuration.

➔ Each andy every obj had a path and element, also learned about having errorElement.

➔ After making that configuration How do you provide it, we use routerProvided.
➔ Tell me a component where you fill in ? It's OUTLET it comes from react-router-dom
➔ Whatever children that we make they go insider our outlet
➔ Does he seq of children matters**, No**
➔ **We** went on creating a Error Component.
➔ **To get** the error, we used a hook named, useRouteError
➔ **How to** put this error page into the config, we used, errorElement
➔ Now tell me how do we read our dynamic id, from the url
➔ useParams hook
➔ We are writing our code, in a proper way, that is why we're able to reuse it in a proper way. The shimer, the cdn url
➔ **Make a comp for every logical thing, help your code modular, readable, maintainable, reusable, testable**
➔ We again fetched data from swiggy API
➔ Try using optional chaining. Be careful.
➔ Link component is used, behind the scenes this link component is using Anchor tag.

10. Let's go Classy 11-02-2024

➔ Today we're going to study class based components
➔ What we will be learning today is goining to help you a lot in your interviews.
➔ React initially started with class based component. There was no concept of functional comp, and no hooks, and it was very tough, but we were coming from jQuery.
➔ The concept was same but the code, we used to write was diff.
➔ The code was not clean thing were messy, with class comp, and it used to be very big, more code, less maintainable.
➔ Now we've enough knowledge, we can understand class based component.
➔ Class based component are no longer used, If you are creating project, u wont be using class component.
➔ But why to learn, cause companys got older project which may be written in class based comp
➔ Also, in the interviews many questions come from class based component only so have your full attention on this lecture.
➔ Last class we studied how we can do routing, nested routes, also try to read about other routers whos doc is available, now lets go into code of class based component.

➔ Suppose in our about page, I want to make a profile section, how will you mak a route /about/profile ?
➔ Will be using children of children.
➔ In children don't write /about/profile, you should be writing relative path. The other way might work didn't tried it.
➔ Consider writing element, and create profile.jsx compoenet.

➔ `path: "/profile", /`, if I write this way then it means localhost/profie.

```jsx
import { Outlet } from "react-router-dom";

const About = () => {
  return (
    <div>
      <h1>About Us Page</h1>
      <p>
        This is the Namaste React Live Course Cha
      </p>
      <Outlet />      You, now • Uncommitted chan
    </div>
  );
};

export default About;
```

➔ After doing this, when you go to /about/profile you will seing nothing. Why? Did you forget the basics, cause The basics are the childrens are always renderd inside the an OUTLET. Now where should I be creating my outlet, Outlet should always be created in parent, the parent is ABOUT, so create there. This will work.

➔ If I want my profile component whithout using an outlet here what will I be doing, We can jst directly import our profile component.

➔ Now let us move to writing Class based component.

➔ Let try to build this component as profile component ProfileClass.jsx

➔ Tell me what Is functional component at the end of the day?

```jsx
import React from "react";

class Profile extends React.Component {
  render() {
    return <h1> Profile Class Component </h1>;
  }
}

export default Profile;
```

➔ It is a normal javascript function.

➔ Same, the class based component is normal js class.

➔ You've to say js that it it's not a nrml class Therefore you extends, React.Component. It comes from react.

➔ **Note: The most imp part of a class based component is your render method.**

➔ You cannot create without render, it is the only mandatory method in you class based component.

➔ In nrml fun comp, it is function that return some jsx. And here now the render method return some jsx.

➔ And we will export our class as we did in fun

➔ If I use it in my about, then it will work.

➔ Also but functionalComp, and will try to send props, and receive using parameters of comp. props.name

➔ Now How do I pass these props inside our class Component.

➔ So in class you get with this.props, when you've class based comp u will be having lot of use of this.

➔ this.props.name

➔ Let know what did react did?

➔ React basically is tracking you class, now as it is a class based compoenet, when ever there is a props change or state change in our classs, I need to rerender it.

➔ It takes the props, and attach it to the this keyword of the class

➔ This happen during our reconsilaiton process, react automatically get this props,

```
<h2>Name: {this.props.name}</h2>
```

➔ Now, suppose I want to create a state inside a functional component, how do I do it. With import of a hook, useState.

➔ **Class Compoenet** also have its state here.

➔ **We** know class has a constructor.

➔ **Will've to do 2 research question ?**

```
constructor(props){
    super(props);
}
```

➔ **1. Always when you r creating a class based component you've to get your props inside your constructor and do a super(props). SO do research Why do we use super, what is a constructor why react is passing props there.**

➔ Why I'm creating a constructor, basically constructor is a place, That is used for intialization.

➔ When ever this class component is rendered, the constructor is called And this is the best place to create your states.

➔ You will create state var in constructor.

```
constructor(props) {
  super(props);
  // Create State
  this.state = {
    count: 0,
  };
}
```

➔ This is how we do it.

```
const [count] = useState(0);
```

➔ In function we used to do this.

➔ Now how will I use this count?

➔ `<h2>Count: {this.state.count}</h2>`

➔ `<h3>Count: {count}</h3>` **we used this in functional Component.**

➔ If I want to create 2 state var, then in func compom, we do the same way .

```
this.state = {
  count: 0,
  count2: 0,
};
```

➔ But here we don't do this way, So all the state variable are created as a part of one object.

➔ **React uses one big object to maintain the whole state.** Even in our functional component, react still uses one big object behind the scenes.

➔ **Now,** let us understand how do we do our setCount ?

➔ **In funComp, we** `const [count, setCount] = useState(0` and we can use it with button on onClick.

```
<button onClick={()=>{
    this.state.count = 1
}}>SetCount</button>
```

➔ **This** is not good

➔ **In class based comp, will be using setState,** will be passing a modiffed obj.

```
<button
  onClick={() => {
    this.setState({
      count: 1,
    });
  }}
>
```

➔ We do not mutate, state directly never do this.state = something.

➔ When you do setState, react keeps track of it, and automatically reconsilaion process sync our UI.

```
onClick={() => {
  setCount(1);
  setCount2(2);
}}
```

➔ For two state variable in funcComp we do this.

```
this.setState({
  count: 1,
  count2: 2,
});
}}
```

➔ In classComp we do this.

➔ We can update it together.

➔ In functional compoenet, will update for each and ever variable.

➔ New way is bad or good?

➔ Well new way is v v v good , in past if my state var or 2, and I change only 1 var, then thinking of it how it doing this all?, it is jst modifiying the partial object that I'm passing.

➔ Now, fun is very specific I exactly know the thing.

➔ Try to research how the setState is working behind the scenes.

➔ When I write my consolelog in functionalComp, above return when will it prints, will it print always I render it ? YES

➔ Now if I write consoleLog in my constructor and also, we've consoleLog in render Now what will be printed 1st.

➔ So, how the react lifecycle works, every class is a lifecycle, every **time in the lifecycle, 1st the constructor is called, then the render.**

➔ One more thing, where do I do my API call in funcComp ? In useEffect. When is my useEffect is called. After every render. Why it is a great place to do API call? To make our application fast, 1st of all we render whatever we can by the default state, later on we jst updated the state. Its like render update, after updated will it rerender ? YES.

➔ **Now** let us see in classBased component, here also same patter, 1st we render, then update something. How we do this in classComp So in classComponent we've our constructor, render and componentDidMount method.

➔ **This** is the function which will be called after my render.

```
componentDidMount() {
  console.log("componentDidMount");
}
```

➔ So, when ever class based component is loaded in the page, it has some lifecycles methods that is called. The constructor, Mount, render.

➔ The sequence is 1. COnsturctor, 2. Render 3. DidMount

➔ **Now with you logic tell me what is the best place to make an API call? compoentDidMount**

➔ **Why we r doing, that many don't know this, but we know cause we know how react works, 1st of all will render, then update it later.**

➔ So it is the best place to have API calls'

➔ We've got many such lifecycle methods.

➔ Now keep your mind inTack, what we're going to learn is very important thing.

➔ Pay attention to each and every word I speaks.

➔ **Let us revise, By creating an About component, as class based component.**

➔ Rather than extending from React.Component we can also do this. `import { Component } from "react";` and directly use Component.

➔ Next thing Is constructor. And write super(props) this is our research homework.

➔ Will also be implementing ComponentDidMount.

➔ And the order will be 1st, **the constructor will be called**, **then rende**r, then **componentDidMount**.

➔ Best place to make an API call is c**omponentDIdMount**.

➔ Where do you initialize your state variable, Constructor ?

➔ Why do we do that in our constructor?

➔ Cause, when class is been initialzed you constructor is by default called.

➔ Under stand what is the 1st thing that will be called?

```
You, 29 seconds ago | 1 author (You)
class About extends Component {
  constructor(props) {
    super(props);
    console.log("Parent - constructor");
  }
  componentDidMount() {
    // Best place to make an Api call
    console.log("Parent - componentDidMount");
  }
  render() {
    console.log("Parent - render");      You, 28 second
    return (
      <div>
        <h1>About Us Page</h1>
        <p>
          This is the Namaste React Live Course Chapter
        </p>
        <ProfileFunctionalComponet name={"Akshay"} />
        <Profile name={"AkshayClass"} xyz="abc" />
      </div>
    );
  }
}
```

```
class Profile extends React.Component {
  constructor(props) {
    super(props);
    // Create State
    this.state = {
      count: 0,
      count2: 0,
    };
    console.log("Child - Constructor");
  }

  componentDidMount() {
    // API Calls
    console.log("Child - componentDidMount");
  }

  render() {
    const { count } = this.state;
    console.log("Child - render");
    return (
      <div>
```

➔ 1st will be parent constructor, then 2nd thing will be parent render, then 3rd thing will be child constructor in profile comp, cause when it calls render, then it follows the sequence as we've Profile comp, then it goes there and follows the sequence.

➔ i.e., parent will trigger the lifecycle of children.

➔ **Here,** 3rd the child-coonstructor get's called.

➔ **After this?** 4th thing is, child render.

➔ **5th thing?** It will be child component did mount.

➔ **i.e.,** It will complete the lifecycle of children.

➔ **Next 6th** will be parentCompDidMount.

➔ **Mount means to Load**

```
Parent - constructor
Parent - render
Child - Constructor
Child - render
Child - componentDidMount
Parent - componentDidMount
>
```

jst assumeig

➔ **Now, What if I've two children?**

```
    </p>
    <Profile name={" First Child"} xyz="abc" />
    <Profile name={" Second Child"} xyz="abc" />
  </div>
);

* Parent Constructor
* Parent Render
*   First Child - Constructor
*   First Child - Render
* Second Child - Constructor
```

➔ **Till this point ok, now what's next?** Fist Child Compoenet did mount? I said Yes.

➔ **But you are wrong over here.**

➔ The answer, is nxt will be the second child constructor.

➔ Nxt, will be secnd child render.

➔ Nxt? I said 2nd childCompDidMount.

➔ But answer is 1st componentDidMount., nxt will be secondCompoenet did Mount.

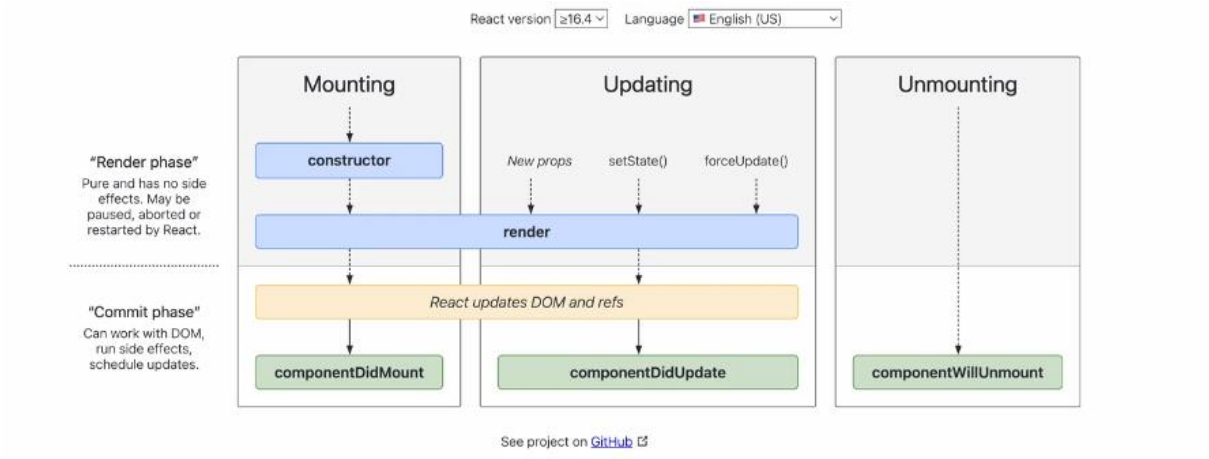
```
Parent - constructor
Parent - render
Child - Constructor First Child
Child - render First Child
Child - Constructor Second Child
Child - render Second Child
Child - componentDidMount First Child
Child - componentDidMount Second Child
Parent - componentDidMount
```

➔ THEN it will be ParentComponentDidMOunt.

➔ So, this is why class based component were very confusing.

➔ **Now,** react has become very simplified but this questions are still there.

➔ **Search React Life Cycle method.**



➔ **If** you understand this, you will never get confusing in you life.

➔ **Focus on** the mounting for nw.

➔ **This** is very important thing now,
➔ **When** react is rendering things up, it does it in two phase.
➔ **Then 1st** phase is the render phase.
➔ **The** 2nd phase is the commit phase.
➔ **What** Happens Is, 1st of all react finishes it's render phase.
➔ **What we got** in the render phase is: it includes your constructor, includes ur render method.
➔ **Commit phase,** is the phase where react is actually modifiying your DOM.
➔ **ComponentDidMount, is** done after you've updated the DOM.
➔ **i.e.,** after the initial render has finished. After my component is on the browser.
➔ **That** is why we see Shimmer effect.
➔ **Let understand in the code,** 1st react will call constructor, then render and it will generate the html, that needs to be put into the DOM, Who helps us converting this jsx to html, That's BABEL. 🎉
➔ Now, after this render, we r ready to put things into our dom.
➔ In the commitPhase react will update the dom.
➔ After updateing the DOM, react will call componentDidMount.
➔ This is how the lifecycle works.
➔ Now, with this knowledge why there were discripences.
➔ When there are two childrens, 1st of all react tryies to patch up the render phase.
➔ Ok, tell me render phase is fast or Commit phase is fast.
➔ Render phase is fast, cause commit phase to update the DOM, which is touffffff
➔ **React** batchs up things inside render phase,
➔ **Suppose** we've 1 parent & 1 child, parent will call its constructor, parent render, 1st child constructor, 1st child render, Now at this point if we got more childrens then react say's that there are more children let us complete the render for everyOne, I will commit it again.

➔ **Cause,** Suppose one of the children making api call, then it started the commit phase, then it will delay the render phase for the second child.

➔ **SO, it batches up the render phase for the 1st and 2nd child.**

➔ After completion of render phase, now is the place react will update the DOM, Dom is updated for children.

➔ Now, our commit phase starts, in commit phase, 1st child componentDidMount, then 2nd child ComponentDidMount.

➔ And then parentComponentDidMount.

Now moving to the new Concept.

➔ Let us make an api call, let's make some api call to fetch some user information, lete's call github API

➔ Google, github api user api, get user.

➔ `https://api.github.com/users/USERNAME` How, do I make an api call?

➔ I will use fetch, this is an async operation so I should make my function async.

➔ Now comes the important part, your 2nd research homework

➔ **We can make the componentDIdMount asyn, we've written there** `async componentDidMount() {` **, but you cannot make a useEffect async. SO homework is why can I make compoentDidMount async but not the useEffect in functional component.**

➔ **This is asked to seniors advanced developers.**

➔ **If you understand this, u understand the internals of react.**

➔ IT's a diff way how hooks are build.

➔ After fetching, I will convert it into json.

```
constructor(props) {
  super(props);
  this.state = {
    userInfo: {
      name: "Dummy Name",
      location: "Dummy Location",
    },
  };
```

➔ How will I put this data into ui, yes I need to create a state, will be creating that inside the constructor.

➔ Now, how do I push this data, into the state.

➔ this.setState({userInfo: json})This all is created in profile class.

```
this.setState({
  userInfo: json
})
```

➔ Such that, I had made my API call inside my child.

➔ Will be accessing it this way.

```
<img src={this.state.userInfo.avatar_url} />
<h2>Name: {this.state.userInfo.name}</h2>
<h2>Location: {this.state.userInfo.location}</h2>
```

➔ **SEE, in** the browser we made an API call.

➔ **Now** let us see what are the sequence method is been caleed.

```
Parent constructor
Parent render
child constructor
child render

DOM is updated
json is logged in console
child componentDidMount

Parent componentDid Mount
```

➔ **This is how it goes on doing, 1st the render phase, the update phase, then commit phase.**

➔ What If **he says that this is wrong.**

```
Parent - constructor
Parent - render
Child - Constructor First Child
Child - render First Child
Parent - componentDidMount

▸ {login: 'akshaymarch7', id: 12824231, nod
  tps://avatars.githubusercontent.com/u/128
Child - componentDidMount First Child
Child - render First Child
```

➔ **So, what is being happened is:**

➔ **Parent componenDidMount is called before making an API call.**

➔ **Cause, as we said react finish it's render cycle 1st then it goes to commit cycles.**

➔ **But nw, as the CompoenetDidMount has an API call, this will be called later, as we are using async and it will take some data to load.**

➔ **The way we've written it, it will follow that way, but as we are making an API call in childCompDidMOunt, as it is asyc it**

will wait for the data to fetched, till that it will call the parenCompoenDidMounbt.

➔ **After parentComp, it wount stops over here,** as we got the data we did a setstate, what will setState trigger, YES, it will trigger next render, It will triger the recocilation process.

➔ **This rerender cycleis known as UPDATING.**

```
*  child constructor
*  child render
*  child componentDidMount
*
*  API call
*  Set State
```

➔ **Here's how the children follow's,**

➔ After the setState, then this cycle is update cycle, What will happen when you trigge the update cycle.

➔ It will render, update the dom then it will trigger another method componentDIdUpdate,

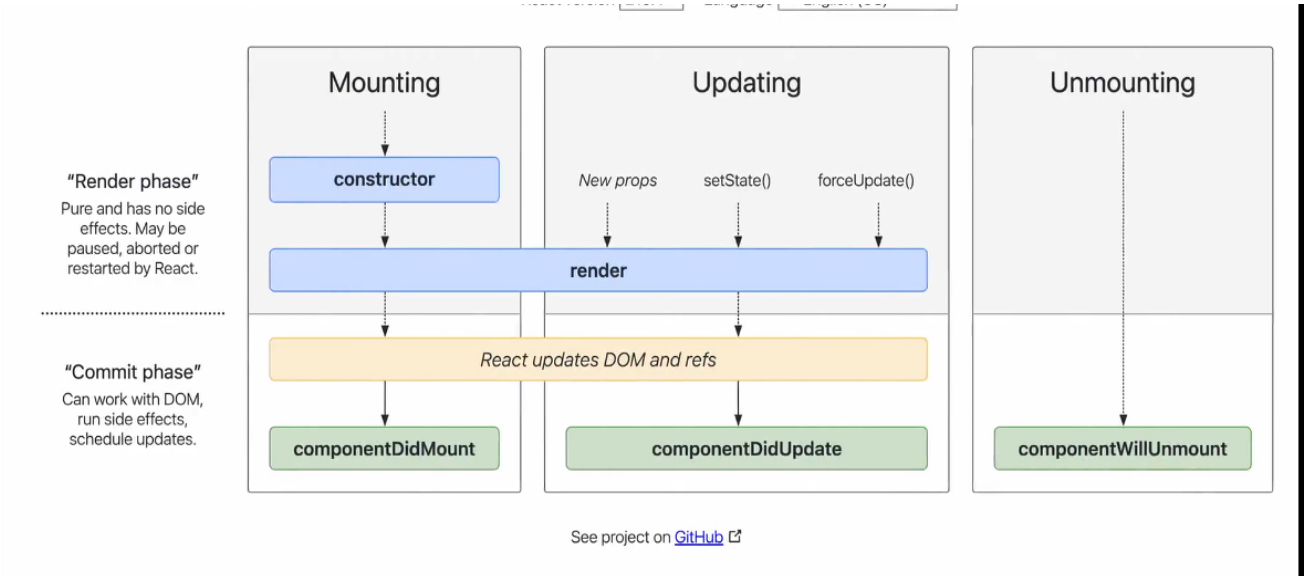
➔ I can've this as well.

➔ Mount is called, when? After 1st render.

➔ And update will be called after every next render.

➔ Now, think of it's use in past, it is hectic, to write Mount and update.

➔ Actual react was this. And 90% of the developers don't understand this cycle,



➔ Most of them will be confusing.

➔ There is something known as unmounting also.

➔ A component mounts and a component unmount also.

```
componentDidUpdate() {
  console.log("Component Did Update");
}
componentWillUnmount() {
  console.log("ComponentWillUnmount");
}
```

➔ When it is destroying it will call this.

➔ The method called is componentWillUnMount.

➔ **Now in browser, keep an EYE in the console, when I go to someother page, the compoennWillUnMount get called.**

➔ DISCLAMER: never ever compare react's lifecycle method, to functional component.

➔ Bad habbit they say, useEffect is equivalent to ComponentDidMount. Don't' say this.

➔ **Behind the scenes, useEffect is not using componentDidMount. It's new way all together.**

➔ **After each and every render, my useEffect will be called, thing are different with classBasedComponent.**

➔ **Here one 1st render componentDidMount, and after** subsequent render componentUpdate will called.

➔ **Also,** in useEffct we do empty dep array, for once calling in intial render.

➔ **And if** give some varaiable then every time my count changes it will rerender.

```
useEffect(() => {
  // API Call
  //console.log("useEffect");
}, [count]);
```

➔ **If you've to do this, then u've to write like this,**

```
componentDidUpdate(prevProps, prevState) {
  if(this.state.count !== prevState.count) {

  }
  console.log("Component Did Update");
}
```

➔

```
[count, count2]);
```

➔ **If more than 1 var, then can write this way.**

```
if (
  this.state.count !== prevState.count ||
  this.state.count !== prevState.count
) {
```

➔ **Had to do this way,**

Let’s do this, I want to do something when my count changes and some other thing when my count2 change. How will I be writing this.

➔ We uses two useEffect.
➔ Where as in class we used to write two if’s stmt.
➔ Now also let’s understand the useCase of CompoenetWIllUnMount.
➔ It will be called when we r leaving the page.
➔ Guy’s how many pages, do we’ve -> from day 1 I’m saying it is single page application.
➔ IT is single page we r just changing the component.
➔ There are lots of things we should be clearing when we leave the page.

```
componentDidMount() {
  setInterval(() => {
    console.log("NAMASTE REACT OP ");
  }, 1000);
```

➔ Let us say I got this, setInterval function, now when I’m on page, it will print this each second.
➔ But guess what if I movie to other component, then also it is printing, which is not expected.
➔ **And** also if I again go to some other page, then it will strt twice.
➔ **This** is the problem of the single page application.
➔ **Cause** When you are changing your page, it is not reloading your page, it is jst reconsilation, jst rendering.
➔ **So this is the answer, why we need to unmount things.**
➔ **I will be using clear Interval in componentWillUnMount.**
➔ **But** how will I reference that interval here, Yes mayuri is write, will be using this.setInterval. this is shared between all functions of this class.
➔ `this.timer = setInterval(() => {` **will be using this way.**
➔ This what you should know, when u r creating a mess u should clear it also.
➔

```
componentWillUnmount() {
  clearInterval(this.timer);
  console.log("ComponentWillUnmo
}
```

➔ Now, friends what will happen if I create setInterval inside useEffect?
➔ It wont stop.
➔ Cause we'r not cleaning things.
➔ SO, now how will you destroy this in your functional component.
➔ So, we can write return function in useEffect,
➔ This return function is called when you are unmounting it.

```
render
useEffect
useEffect Return
```

➔ This is how it works.
➔ Your home works is to play everything we did today.

```
useEffect(() => {
  // API Call
  const timer = setInterval(() => {
    console.log("NAMASTE REACT OP ");
  }, 1000);
  console.log("useEffect");

  return () => {
    clearInterval(timer);
    console.log("useEffect Return");
  };
}, []);
console.log("render");
```

➔ And 2 research topics.
➔ 1. Superprops
➔ 2. Async compoDIdMount
➔ And in useEffect callback I cant make it async.
➔ React will complaint, find out the reason.
➔ Chapter 8 DONE>
➔ Now we are diving deep into topics.
➔