[lecture - 6]

[Exploring the world] Date.....

- **Monolith Architecture →**
  Traditionally when the apps were developed they were all developed using monolith architecture.
  Earlier we use to have a huge big project this project it self has code of APIs (we have developed APIs inside this project) we also have UI code inside the same project we have authetication code inside it, DB connectivity code, SMS code

- **Microservice Architecture →** we have different services for different jobs we have Backend service, UI service, Auth service, DB service, SMS sending service

- All these service combined together form a big App.

- For each and every thing we have different project

- This is known as separation of concerns.

- It follows a Single responsibility principle where each and every service has it's own job.

Spiral

Scanned with CamScanner

Date.....

Q.7 How do these service interact with another

A.7 - - - -

- Our Namaste React project which we are making is a UI microservice
- This UI service written in react
- Advantage of microservice is we can have different tag stag for different things.
- In monolith, we had one big project in Java, then we have to do everything in Java.
- But in Microservice Architecture we have UI written in react, backend written in Java, DB in python, SMS service in golang.
- How these services are connected just like our UI service was deployed on port :1234, similarly all the services run on their own specific ports.

:1234 → UI
:1000 → BE
:3000 → SMS

Spiral

Scanned with CamScanner

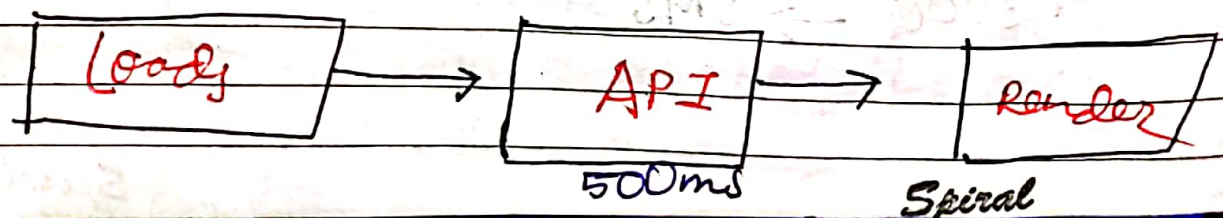
Date.....

- And At the end of the day all these ports can be mapped to domain name.
- ~~Similarly~~ Suppose this backend is mapped to "/api"
- Suppose we have domain name of nanastadel.com/api
- all these APIs are deployed onto the same URL and similarly we can have SMS on /SMS, to send the SMS we just have to call the service /SMS
- we can deploy the UI on "/", no URL ahead of that
- How do these service interact, they make all to different URLs.
- Suppose if UI wants to connect with backend, they will make call to /api

---

2 approach →

(1) Suppose our App loads we can just make an API call to fetch the data and when we get the data then we can render it on the UI.



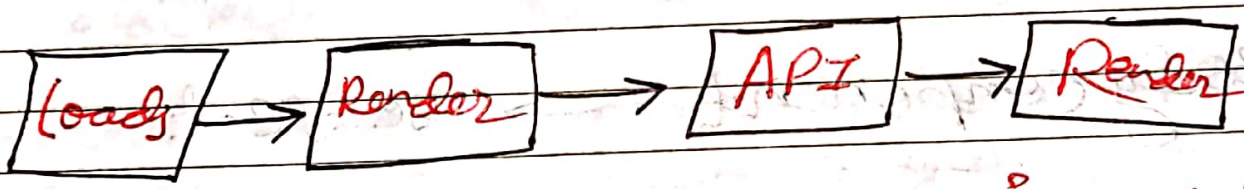
Scanned with CamScanner

18:00

Date.....

- First, as soon as our page load we can make an API call wait for data to come then we render the UI, suppose this call take 500ms, so our page will wait 500 ms and after 500ms it will render the UI.

(2) As soon as the page loads we will render the UI, now we will make an API call and as soon as we get the result back from API we will now populate, re-render our app with new data.



- In react we will be using 2nd approach

- React's render cycles are very fast, React has one of the best render mechanism

- Two renders are OK, it gives better user experience. In react we will follow the second approach.

Scanned with CamScanner

- fetch function is not given by JS, it is given by Browsers.

Date.....

- UseEffect Hook → {useEffect} from "react"
- useEffect will take two arguements
  (i) arrow function → It is a callback fxn
  (ii) Dependency array
- When will this useEffect callback fxn be called
  A:- This callback fxn will be called after our Component renders.
- When the body fxn will render it will render it and as soon as the render cycle finished it will just quickly call this callback function.
- Body Component → useEfect
- First the body will be render then the useEffect will be called.
- We will fetch the data inside useEffect.

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

Spiral

Scanned with CamScanner

Date.....

```
const fetchData = async () => {
const data = await fetch(" ");

const json = await data.json();
};
```

Error → Access to fetch at 'www.swiggy.... has been blocked by CORS policy.

- calling swiggy.com from localhost has been blocked due to CORS policy.
- Our browser blocks us to call API from one origin to different origin, if there is origin mismatch the browser blocks the API call that is a CORS policy.
- There is a way that we can get rid of the CORS. → CORS Extension
- we have loaded the page, the page will be rendered, we made the API call now after we have received the data now want to render our component once again with this loaded data.

Scanned with CamScanner

Date.....

- This page is rendered using listOfRestaurant so what will happen is once we have got the data, we will update of listOf Restaurant with new data.
  React will re-render the component and a new data will be populated onto the page.
  we will use setListOfRestaurant, this is a state variable, not a local variable of JS

```
const Body = () => {

const [listOfRestaurants, setListOfRestaurant] = useState(
                                                  resList)

  useEffect(() => {
    fetchData();
  }, []);

const fetchData = async () => {
  const data = await fetch("  ");
  const json = await data.json();
setListOfRestaurant(json.data.cards[2].data.
                    data.cards)
                    ↓
                    here we can do optional chaining
                    json?.data?.cards[2]?.data?.
};
```

Scanned with CamScanner

Date.....

```
data?.cards);
};
if (listOfRestaurant.length === 0) {
  return <Loading />;
}
```

- Does showing a spinner a good way?
- No it is not, acc. to the latest standard practices that are followed in the industry
- Today in the UI world, there is a concept of Shimmer UI.
- It is kind of like we load fake page until we get actual data from the API.
- Instead of showing Loading we can show a skeleton or fake page or fake cards over here till the data is loaded.
- A good user experience says that the end user can anticipate that OK there are cards which will be loading over here
- eX → Youtube is following shimmer UI.

Spiral

Scanned with CamScanner

Date.....

• We have made a fake impression here cards will be coming.
It is a psychologically better.

→ conditional Rendering –
– A rendering on the basis of condition

ex→
```
if (listOfRestaurants.length === 0) {
    return <Shimmer />;
}
```

So, we can club all the code – conditional if ---- and

```
return (
  <div className="Body">
    <div className="filter">
      —
      —
```

we can club all this code together, we can directly write a single return and we can use a ternary operator

ex → `return listOfRestaurants.length === 0 ? <Shimmer /> : (`

```
  <div className="body">
    —  —
    —  —  —
```

Scanned with CamScanner

Date.....

Q.7 Why useState hook came into frame

ex- const Header = () => {

let btnName = "Login";

return (

<button className = "Login"
onClick = {() => {

btnName = "Logout";

}} {btnName} </button>

→ btnName - a normal JS variable, but as soon as we click on the button we want our "btnName" to be updated - on click this will update and we can see it in console window but our UI didn't render.

- This Header component didn't refresh, then would be some way that we refresh our Header component so that it can take the latest value of "btnName".

This is why JS variable doesn't work in such cases; if want to make our component dynamic, something should change in component. We use local state

Spiral

Scanned with CamScanner

• we just follow a nomenclature of putting "set" in front of these variable names but we can write whatever we wish to.

Date.....

variable, here is where useState & state variables comes into picture & this is why we use state variables.

→ const [btnName] = useState("Login")

return (

<button onClick={() => {

btnName = "Logout"

}}>

→ No, this is not the right way, we can't directly modify but we will have to use the 2nd argument which we get from useState, a function which will update this variable.

const [btnName, setBtnName] = useState("Login")

return (

<button onClick={() => {

setBtnName("Logout");

}}>

→ This btnName will be updated with this new value → Logout.

Scanned with CamScanner

1:27:08

Date.....

→ Whenever this state variable → btnName will changed using this setBtnName, react will re-render the component, react will refresh the header component and all the updated value will be there. It triggers the reconcilliation cycle

→ Is the react is refreshing the whole Header or it is just refreshing the button?

A:- It will re-render the whole Header component.

→ What is mean by rendering a component?
A:- Rendering a component means it will just call the header fxn/component once again.

Q:- const [btnName, setBtnName] = useState("login")

Q How a constant variable gets updated with the raw value?

A:- Rendering the component once again but this time when we invoke this function this btnName is a new variable than it was before.

On initial time this set variable was Login, but as soon as we do

Spiral

Scanned with CamScanner

Date.....

setBtnName it will call the Header fxn once again & it will create a new instance of this btnName is different than the older btnName which was being printed.

And now when this new btnName is created it is not created with the default value "Login" but it is created with the updated value "Logout" from setBtnName.

→ Every we click on the button Header is getting rendered but only that button changes.

<input type="text" className="search-box"

<button onClick={() => {

Date.....

• To get the data from input box we need to take the value of input box and we have to bind this input box to a local state variable.

To track the value of this input box whatever the user is typing in to get that value we have to bind this value with a local state variable of react.

```
const [searchText, setSearchText] = useState("")

return (
<input type="text" className="search-box"
value={searchText} />
```

→ But we are not able to write text in search box

→ Because the "value" is bind to the searchText (local variable) and the searchText is empty so this input box will not change unless we change the searchText

→ To fix this we will write onChange event handler over here, as soon as input changes our onChange function should also change our searchText & it will update with a new value.

Spiral

Scanned with CamScanner

→ Whenever state variables update, react triggers a reconciliation cycle (re-renders the component) Date.....

```
<input type="text" ClassName="search-box"
value={searchText} onChange={(e) => {
setSearchText(e.target.value);
}}>
```

→ Whenever we change a local state variable react re-renders the component.

→ Whenever we are writing in input box ~~o~~ letter by letter, this searchText is getting updated on every key press our component is getting re-render, the whole body component is getting re-render

→ But the best part is react is triggering the reconciliation cycle, it finding the difference between the older virtual DOM & newer Virtual DOM.

→ React is re-rendering the whole body component but it is only updating the input box value inside the DOM.

→ DOM manipulation is very expensive and react is very efficient in doing this.

Spiral

Scanned with CamScanner

2:00:00

Date.....

→ Why React is faster?

A→ Virtual DOM is not making it fast, the react fibre (new reconciliation algorithm) which find out the difference between the two virtual DOM and updates the DOM only when required in portion of DOM which only that portion is required that's why react is fast.

→ <button
onClick={() => {

const filteredRestaurant = listOfRestaurant.
filter((res) => res.data.name === searchText
);

setListOfRestaurant(filteredRestaurant)

}}

>

→ When we click on the button, it will filter out the restaurant and it will update the variable → listOfRestaurant, and when we update this variable and when we update this variable — it will trigger the

Spiral

Scanned with CamScanner

Date.....

reconciliation process our body component will re-render and it will re-render with the updated value.

- filter ((res) => res.data.name === searchText

↓

→ This will not work until we type the exact name,

→ so we will use this one

↓

- filter ((res) => res.data.name.includes (searchText)

↓

— we can do it more precisely, because there may be difference in letter upper & lower case so it's better to make both of them in lower case. Case sensitive

listOfRestaurants.filter ((res) => res.data.name.toLowerCase()
.includes (searchText.toLowerCase()))

→ suppose if we are searching for coffee over here, we got the ~~zoom~~ 2 restaurant that have "coffee" in their name

Spiral

Scanned with CamScanner

Date.....

→ we updated our listOfRestaurant and now when we are searching inside it once again then it means we are searching inside only these 2 restaurant not the whole 15 restaurant that we got.

→ we have modified our listOfRestaurant with this 2 data and we have lost all those 15 restaurant which were there, the data we got from API. we have lost the data. we have updated it with the new data.

→ Actually search should work on all those 15 restaurants which were there from the API call, this is the bug.

Bug solve →

- Bug over here because we are just updating the list of restaurant with filtered list. we are just modifying the original list itself, instead what we should do is this we don't want to modify this, we will keep another copy of filtered list.

we will keep all the list of Restaurant in our → listOfRestaurant → variable

Spiral

Scanned with CamScanner

Date.....

And we will create another state variable only ~~for~~ the filtered restaurant

```
const [filteredRestaurant, setFilteredRestaurant]
    = useState([]);
```

And whenever we are doing filter so instead of updating our ~~listOfRestaurant~~ we will just update ~~setFilteredRestaurant~~

~~and now when we are rendering we will render it~~

```
<button
onClick={() => {
~~const filteredList = listOfRestaurants.filter~~
~~((res)~~
```

and now when we are rendering we will render it with filteredRestaurant

→ filteredRestaurant.map ((- -

→ And while calculating we would need this listOfRestaurant, this listOfRestaurant is intact we will never modify listOfRestaurant; we are modifying it once when we are getting data from API

Spiral

Scanned with CamScanner

Date.....

→ Initially our filtered restaurant is empty

const [filteredRestaurant, setFilteredRestaurant] =

useState([]);

empty

→ If we refresh the page, we will see nothing on the page.

→ So when we are fetching data we are updating a copy of listOfRestaurant as well as filteredRestaurant with whatever data we have got

setListOfRestaurant (json?.data?.cards[2]?.data?.data?.cards);

setFilteredRestaurant (json?.data?.cards[2]?.data?.data?.cards);

we are updating ~~data~~ that data both of variable with this.

Spiral

Scanned with CamScanner

Date.....

→ we will use listOfRestaurant intact, whenever we have to filter something we will just use FilteredRestaurant and then we will update our FilteredRestaurant and to display it on the UI we are just using the FilteredRestaurant.

→ when we will search for "coffee", filteredRestaurant got updated.

→ listOfRestaurant will still have the data which was coming from the API, so next time we will filter it will filter from the listOfRestaurant.

```
const filteredRestaurant = listOfRestaurant.
filter((res) => res.data.name.toLowerCase
().includes(searchText.toLowerCase())),
setFilteredRestaurant(filteredRestaurant)
}}
```

Spiral

Scanned with CamScanner

- import RestaurantCard from "./RestaurantCard";
- import {useState, useEffect} from "react"
- import Shimmer from "./Shimmer";

```
const Body = () => {
const [ListOfRestaurants, setListOfRestaurant] = useState([]);
const [filteredRestaurants, setFilteredRestaurant] = useState([]);

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

useEffect(()=> {
fetchData();
}, []);

const fetchData = async () => {
    const data = await fetch("                    ");
```

Date.....

Spiral

Scanned with CamScanner

~~const data = await~~

```
const json = await data.json();
// optional chaining
setListOfRestaurant(json?.data?.cards[2]?.data?.data?.cards);
setFilteredRestaurant(json?.data?.cards[2]?.data?.data?.cards);
});

return listOfRestaurants.length === 0 ? (
  <Shimmer />
) : (
  <div className="body">
    <div className="filter">
      <div className="search">
```

Date.....

Scanned with CamScanner

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

<button
onClick={() => {
const filteredRestaurant = listOfRestaurants.filter((res) =>
res.data.name.toLowerCase().includes(searchText.toLowerCase())
);
setFilteredRestaurant(filteredRestaurant);
}}
>
  Search
</button>
</div>
```

Date.....

Spiral

Scanned with CamScanner

<button
className = "filter-btn"
onClick = {() => {
const filteredList = listOfRestaurants.filter((res) => res.data.avgRating > 4);
setListOfRestaurant(filteredList);
}}
>
Top Rated Restaurants
</button>

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

Export default Body

Scanned with CamScanner