React (Day-6)

Monolithic vs Microservices architecture

Monolithic :-

→ All the services are written in a single codebase.
→ Since it is a single codebase, same tech stack / language should be used throughout the project.
→ Even for a small change in project, whole project needs to go through build and deployment cycle.

Microservices :-

→ Different services are written separately.
→ These projects follow separation of concern and Single Responsibility Principle
→ Diff tech stacks can be used throughout different services.
→ Each service is deployed on separate port and is mapped to an endpoint.
→ These services interact with each other through APIs and make up a one big project.

Two ways to fetch data from APIs

1) As soon as our page loads → API call → Render UI with data

Bad UX, as user have to wait (see blank screen) until data is fetched through API

2) As soon as page loads → Render UI → Make API call → Re-render page with new data

↳ Shimmer UI

The second way of fetching API data gives the user a better User Experience (UX) by rendering a UI initially (skeleton) which can later be re-rendered after data is fetched through API.

→ We will use the 2nd approach to fetch data.

→ Isn't it costly to ~~re~~ render twice?

↳ React is fast AF in render cycle!

useEffect Hook

↳ Comes from react library

Can be imported like this

↳ import { useEffect } from 'react';

It takes two args

↳ call back function

↳ dependency array

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

Q) When is the call back function called in useEffect?

⇒ It'll be called as soon as the component (in which useEffect is defined) renders.

Now, we can fetch live data from SWIGGY API.

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

const fetchData = async() => {
  const data = await fetch('API_URL');
  const json = await data.json();
  log(json);
}

return (

);
```

But this will give an error. A CORS error.
We can't access swiggy data through API in our code.
Who blocks us from doing this?
Is it swiggy? No lol! It is the browser that is blocking us from accessing data from other origin.
=> We can tackle this by installing Allow CORS extension on chrome. Enable it while fetching data.

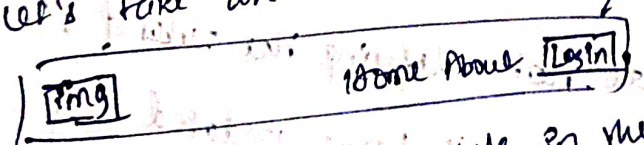
=> [Read about "optional chaining"] (in JS)

Conditional Rendering

Before we discussed about 2nd way of fetching API, where we talked about shimmer UI displaying until data is fetched through API.
So this is done using conditional rendering.
We will display shimmer UI when data is not yet fetched.
Once the data is fetched, we will render the UI filled with data.

Specially about State ***

Let's take an example:-



Let's say we defined a state in the <Header /> component which is login state with default value "Login" So we will toggle between "Login" & "Logout" when the button is clicked.

=> Some things to note about what happens when button is clicked :-

* State gets updated
* For every state update, whole component is re-rendered. (re-render = Header function is called again).
* But here, interesting thing is, reconciliation takes place while every render cycle.
  - Whole header component virtual DOM is compared with prev virtual DOM (diff algo) and only the button is updated.
    { Image, Home & Navbar is same in both cases so they don't get updated }
  - This is the best thing in react.

Though whole component is re-rendered, but only the differences from prev state gets updated in the UI. (This makes our app very fast)

* Another doubt is, when we declare state with const : const [state, setState] = useState(default);

(Q) How is state being updated despite being a const variable?

Ans: For every time the state is changed, a whole new state is created with the updated value of state (in the new render cycle)

- So, we are not defying the rules of JS.
  - → const variable isn't being updated
  - → but a new state with same name is created in new render cycle with the updated state value.

" DOM MANIPULATIONS IN WEB APPS ARE ONE OF THE MOST EXPENSIVE OPERATIONS.

AND REACT IS SUPER GOOD AT "HANDLING DOM MANIPULATIONS EFFICIENTLY"

↳ using reacts' new (React Fiber - reconciliation cycle) algorithm

Input & Input event handling

let's say we do this,

```
const [inputText, setInputText] = useState("");

return (
    <input value={inputText} />
);
```

Now interesting thing to note here is, we can't enter anything in our input box of our web page.

but why?

Because we have binded our local state inputText with the input tag. so our input tag value will be locked / tied to the inputText state variable.

→ So to fix this, we need to make our input tag the ability to change it's value ⇒

```
return (
    <input value={inputText}
        onChange={(e) => {setInputText(e.target.value)}}
    />
);
```

Now we are changing the state 'onchange' of input value.

state's value is 'flexible' now which was then locked to its default value (empty string)

BUT

FUN FACT: Every character we enter in our input box will trigger the onChange event which will further trigger state change and re-render of whole component.

NOW: No matter how many re-renders react has to do because of the unnecessary amount of characters we enter in the input field, react smartly renders only the input field as an updated field and everything else on the component stays the same even though re-render is happening.

NOW: Another interesting thing I have noticed is, whenever we use a button inside a form, to handle onClick event, we should first do a e.preventDefault() to stop the button from reloading the page.

→ It's not the button, but it is the forms default nature to reload the page whenever we submit the form by clicking the button which is "inside" the form.

→ Other buttons outside form also not refresh the page as default behavior and e.preventDefault() is not needed for them.

→ But form has its advantage. Even though we have to use preventDefault() for the button inside the form. But form gives better semantics (HTML) and also better UX (clicking ENTER is enough to make the button click which is inside form).