25/2/24

Monolith Architecture (old way)

→ Whole application is made in single framework or language
↳ If you made a small change you need to re-build the whole application.

Micro Service Architecture (new way)

All small services combined together form a big app

→ Seperation of concerns
→ Single responsibility principle
→ Different languages can be used in different microservices
→ They all are connected ~~and~~ How?
they all are runned ~~&~~ or deployed on their particular ~~domain and~~ ports.

At the end All these ports are deployed to domain Name

Now we will use api calls in our project for dynamic data

→ Two type of calling

① Get all the data from API → Then render whole page (Poor UX)

② Fisrst render all the thiggs we have → Make API call for Data → Rerender the page (Good UX)

React uses 2nd approach as React is so good and fast in Rendering Pages

---

Use Effect Hook → ① import { useEffect } from "react"

↳ Now to make api calls we use Use Effect Hook

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

callback function — dependancy array

* This function will be called after the page is rendered

fetch()

↳ This is a feature given us by the browser, by the JS Engine

↳ Used to fetch data

Example

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

Can we directly fetch data from live api?

Extension → Allow : CORS

→ Yes we can but it gives us a CORS Error to resolve chrome this use extension

→ Because are making a Cross Origin Call.
Resource sharing

Making a Loading Button

↳ As fetching data takes time

→ Just add or return a `<h1>Loading</h1>` tag. before the fetch data

```
if (listOfRestaurants.length === 0)
    return <h1>Loading...</h1>;
}
```

optional chaining

ex → data.name ✗
data?.name ✓

→ Even if that object is not available it will not throw an error, it will give undefined.

Shimmer UI / Conditional Rendering ← Rendering according to condition.

↳ load a fake frame work before the actual UI so that the user experience become smooth.

→ we created a shimmer.js

Creating a login button using Use State

→ when we use a normal function the code is changed but the UI is not rendered again to make it happen we use local state variable that is Use-State

```
const [btnNameReact, setBtnNameReact] = useState("Login");

<button className="login"
    onClick={() => { btnNameReact === "Login" ? setBtnNameReact("Logout") :
                     setBtnNameReact("Login"); }}>
    {btnNameReact}</button>
```

ternary operator

Making a Search Box → Now for the input we are not taking a normal variable we are using a useState. Why? Because You known when you type one-by-one letter in the search box the page shows us the result like that. setting first the box empty.

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

this rerenders page on every small change in text box

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

Whenever state variable update, React triggers a reconciliation cycle (re-renders the component)

```
const filteredRestaurant = listOfRestaurants.filter((res) =>
    res.data.name.toLowerCase().includes(searchText.toLowerCase())
);
```

toLowerCase() is to make both string same

here we are comparing that is there any card who's name string includes the string that we searched.

And that whole list is stored in filteredRestaurant.

```
setFilteredRestaurant(filteredRestaurant);
}}
    Search
</button>
</div>
```