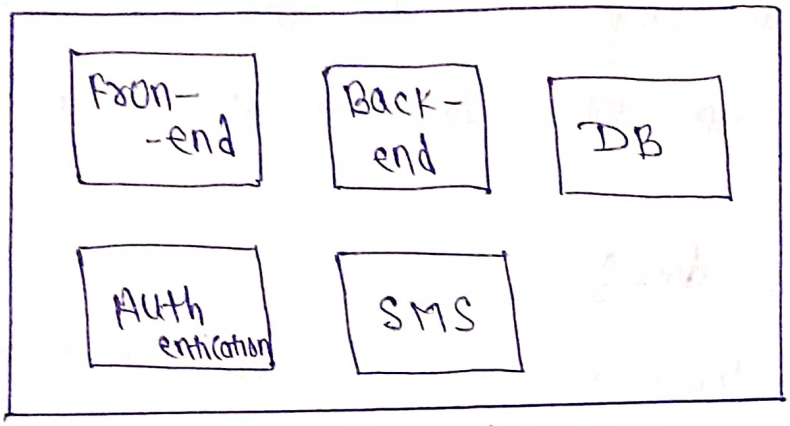
18/02/23

Episode-06 Exploring the world

→ **Monolith Architecture** : In this architecture we have a big Single Project which have all the Services or codes into it



Monolith Architecture.

Here you have to write everything in same language, if your project is in Java then your frontend, backend all need to be in Java

→ In monolith architecture every one (front end eng., backend eng, database eng.) all works on a same project (on same github repo) on a single code file, which have different sections into it.

→ For ex :- If you just need to update the color of a button you need to compile all this big code and deploy this whole project.

→ **Micro Services architecture** :- In this architecture we have seperate code files for seperate work. and they combinly make a big project.

Micro Services architecture

→ Here frontend eng. is working on a seperate code and backend eng on a different seperate code file.

→ Seperation of concerns: for every small work / task we have a different project a different folder.

→ Single responsibility Principle: Every project / Folder have its own single responsibility.

In micro-services architecture, both Seperation of concerns and Single responsibility principle is followed.

→ In micro services every team have their own seperate project, their own deployment cycle

→ Here in micro-services architecture you can have different tech stacks for different project, like for frontend you can use react for backend you can have Java etc.

Que: How this services which have different tech-stacks are connected to each other? How they talk / interact with each other?

Ans: All different services run on different ports. or on different domains

For example:-

Port No.		
1234	→ UI	All these ports can be mapped to domain name. ↓
1000	→ Backend	
3000	→ SMS	

Backend is mapped to /api mrcanzova/api

SMS is mapped to /sms mrcanzova/sms

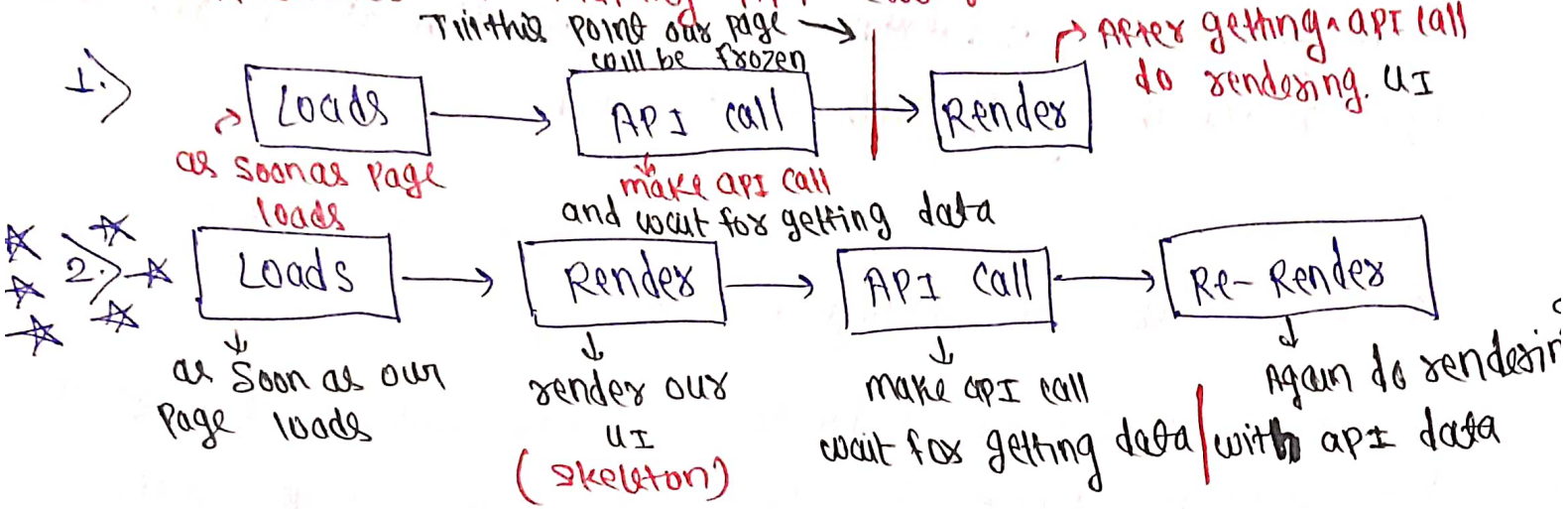
UI is mapped to / mrcanzova/

→ If UI wants to connect to backend it can call to /api url or will call their port.

→ Now we will see how our react app. will make connection or talk to different micro services outside its world. (outside react)

→ How our react application will make a backend API call and how it will fetch data from their?

2 Approaches of making API call :-



☆ In React we will always be using 2nd approach, It is a better approach

☆ useEffect HOOK → Normal JS function which have its own special usecase

→ You have to import useEffect from "react" using named ~~export~~ import.

~~function~~ add (4, 5) → Arguments

function add (int a, int b) return a+b;

→ parameters.

21/02/24

→ useEffect is a normal JS function which have a special usecase in react.

→ You can use useEffect() if you want Something to do after the rendering of any component.

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

callback function

onClick = {doSomething} ✓

onClick = {() => {doSomething()}} ✓

onClick = {doSomething()} ✗ (immediately)

do-Something will be called while rendering. Before even clicking.

Similarly useEffect(doSomething(), []); ✗

This will also immediately called while rendering. So don't do this

→ You can also write

useEffect(doSomething, []);

you can also pass callback func. like this, but remeber that it should be written above useEffect line.

→ If you do rendering on basis of some condition then it is called conditional rendering. like we did for shimmer effect

★ Note:-

const [key, setKey] = useState("Hie");

→ Have you ever thought that if it is a const, they how with the help of setKey it will be changed ?

→ Because const can not be changed, it's value cannot be changed

→ When React basically keeps an eye on setKey function, when-ever it will be called, React will come to render the whole component again and at this time it will create a new key variable with new default value.

→ After ~~clicking~~ calling ~~on~~ setState() : React will come to render the whole component but this time, it will compare new virtual dom with the previous one, and it will find out which part has been changed, or what new part has been added and it will re-renders only that part.

→ The whole component will be re-render, But it will update the updated element.

→ Here update means making changes to the DOM to reflect the new changes in component

→ Rendering again means / it is equivalent to calling that react functional component again.

22/02/24

→ when ever setState function is called, React is rendering that component, but with the help of reconciliation process it is only updating the required new changes into the dom.

★ onChange : Event listner — You can use this event listen when you are working on with an input field & you want to do Something if its value / input get's changed.

Why we can not write onClick={update()}

In JavaScript and React, when you use `onClick={update()}`, it means that the `update` function is immediately invoked when the component renders. This is not what you typically want for event handlers in React.

When you use `onClick={update}`, you are passing a reference to the `update` function, and it will be called only when the actual click event occurs. This is the correct way to set up event handlers in React.

If you use `onClick={update()}`, it results in the function being called immediately when the component renders, which is not the intended behavior for handling a click event. This is why it's important to omit the parentheses when passing a function as a handler to events in React.

Note : It is similar to the reason why we cannot write onClick(setState());

Certainly! Let's break down event.target.value:

1. event: This represents the event that occurred, in this case, the onChange event. The event object contains information about the event, such as the type of event, the target element (where the event originated), and other relevant details.

2. event.target: This refers to the element that triggered the event. For example, if the onChange event occurred on an input field, event.target would point to that input element.

3. event.target.value: On an input element, this property represents the current value entered by the user. In the context of an input field, it provides the text that the user has typed or the value they have selected (in the case of a dropdown, for instance).

Putting it all together, event.target.value in the onChange handler of an input field gives you access to the text that the user is typing or has typed in that specific input field. This is useful for capturing and updating the state with the current input, allowing you to respond to changes in real-time as the user interacts with the input element.