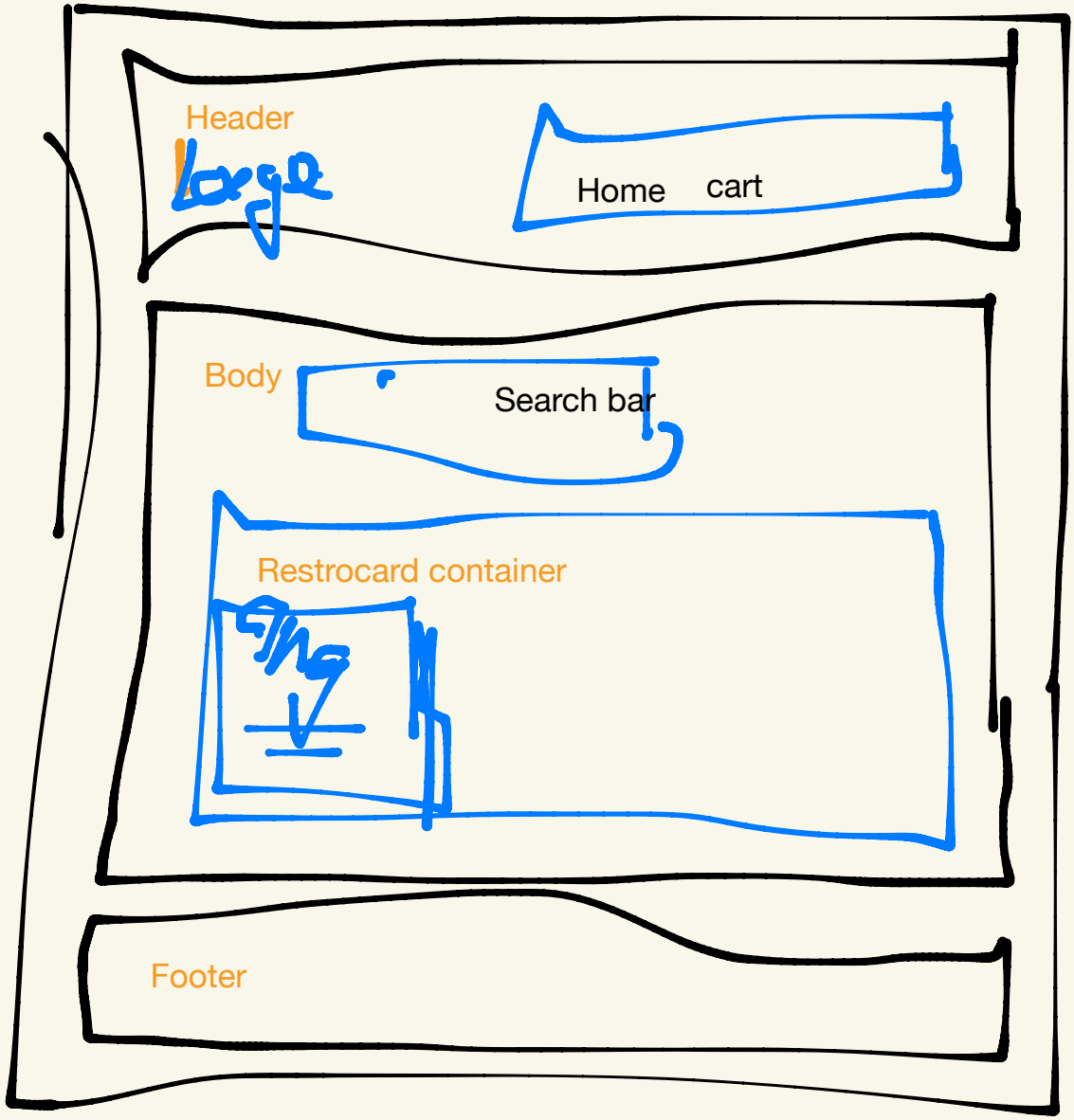
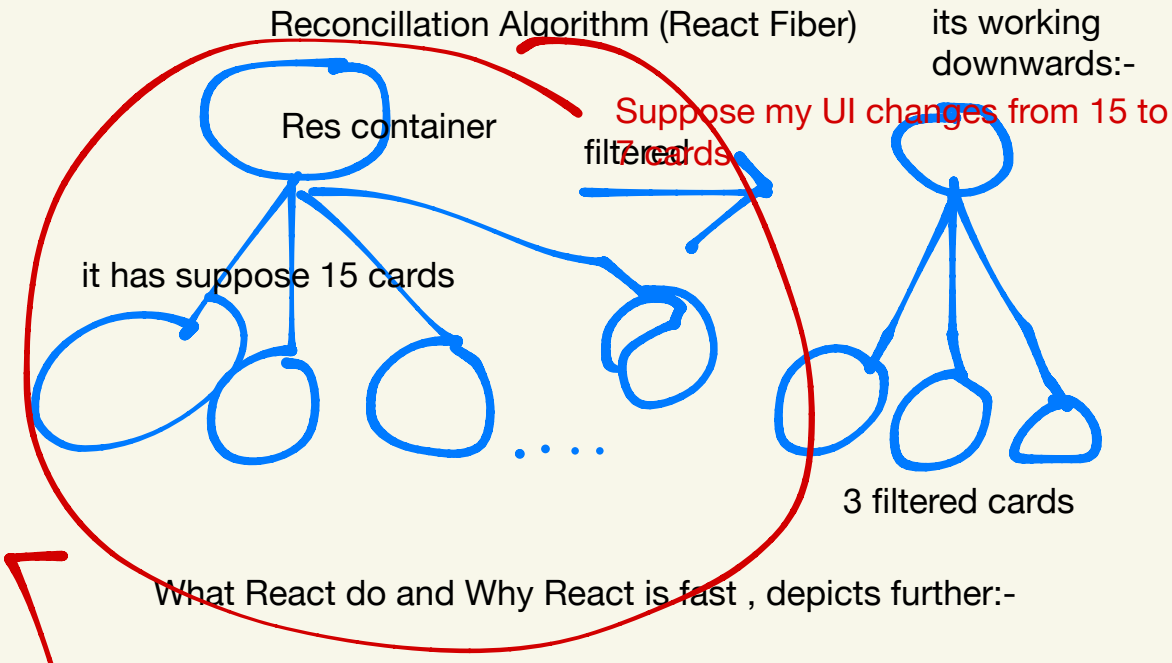


DESIGN PLANNING BEFORE MAKING ANY PROJECT

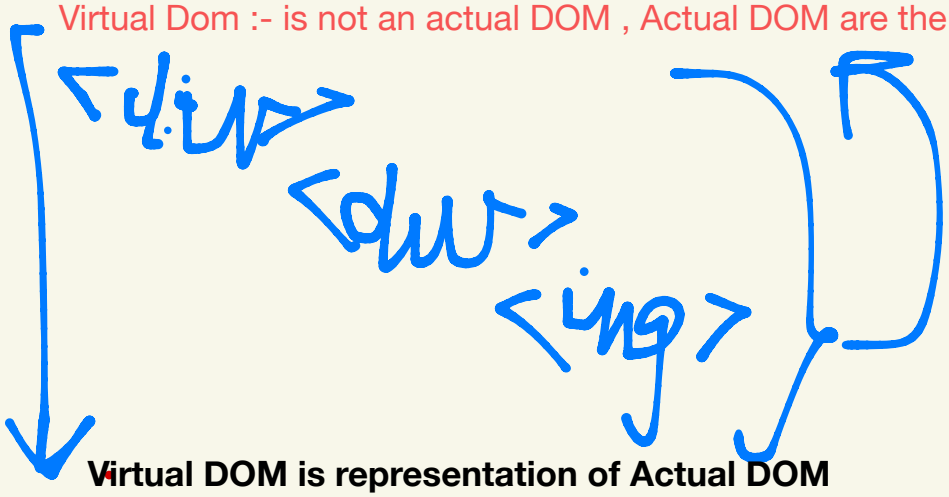
How React works behind the scenes , React uses the following algo



What React do and Why React is fast , depicts further:-

Suppose when ever u have this above UI of 15 cards , react creates Virtual DOM of it.

Virtual Dom :- is not an actual DOM , Actual DOM are the tags

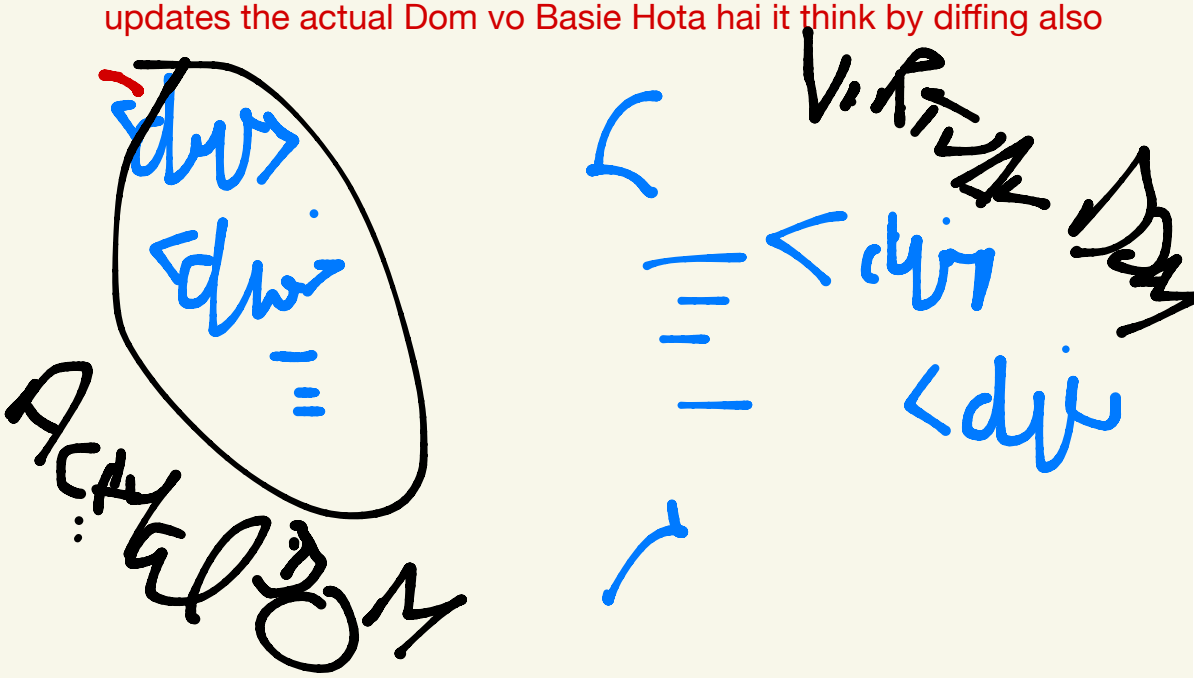


Virtual DOM is representation of Actual DOM

What is representation of Actua lDOM :- Virtua lDOM that object {}

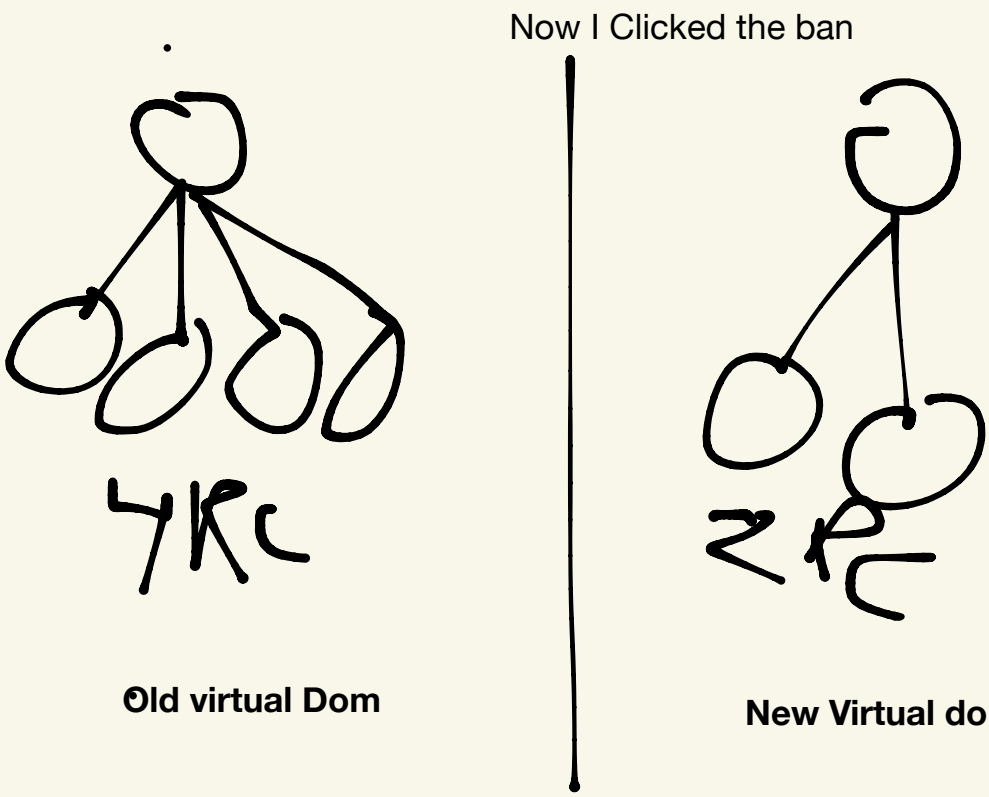
That React.createEleemnt returns React element (which is an object) <Body /> is console.log gives an object (react element) ,how react treats the jsx of Body component :- it creates an obj out of it , this object is basically react virtual loom

Basically what I understood is jo actual Dom hai vo bs like vo jo dom Mei nconsole pr html dikhti hai vo hai , and virtual Dom vo hot o Har eke comp aha ek line ko in react app ko as a React element execute kati hai jo ki ek <body /> comp ko bhi React element ban kr as an obj execute kati hai, too react ne inn ska virtual loom bandy hotah ai and jaime hi koi Dom manipulation hot hai it creates new virtual Dom and updates the actual Dom vo Basie Hota hai it think by diffing also



So virtual Dom is not an actual Dom , its rep of actual Dom as ca n be nested obj (as earlier explained react event)

Diff Algorithm :- it finds out the difference btw tw virtual does (the updated virtual lady and previous virtual dom



Old virtual Dom

New Virtual dom

It ll try to find out the diff btw old and new virtual dom , diff ll be of 2 nodes , it ll calculate the diff and then actually updates the dom on every render cycle.

Revision :-

This whole also is called React fibre , it comes in react 16 to update the dom , so this is. known as reconciliation whenever sth changes o nun its called reconciliation.
after react 16 this also is known as React fibre (is a new way of riding the diff and updating the dom)
Diff :- finds the diff btw the ow virtual does (ie diff btw two objects), when it finds the diff btw these objects then it updates the actual loom and that's how react becomes fast.
for eg , suppose u have these html (dom nodes) , so find out the diff btw two html codes, so finding out the diff btw two html codes is difficult , finding out the diff btw tw o objects is fast : js is fast, so it finds out the diff btw tw o objects .

What ever u see the ui here I console eleentsd :- react keeps track of all this ui , all this dom node , all this html as a virtual loom, : is a kind of object representation of this whole dom , it ll have react ll have the jsx html as react relent (obj) over these in virtual dom.
As son as I like on this bun of top rated res :- a new obj is formed, react fins out he diff btw these two objects earlier 15 res now 3, the nit finds out theidff then it actually updates the dom.
it does not find the diff btw html, it does not touch the html (ie it think will be actual dom) React doesn't to touch the dom a lot, ie why react is fast.

So this was the diff also
whenver change in any state variable , react ll find out the diff race btw virtual dom and it ll re Redner our comp, it ll update the dom

React fiber :- https://github.com/acdlite/react-fiber-architecture

WHY REACT IS FAST ?
REACT IS DOING EFFICENT DOM MANIPUALTION : HOW ? COZ IT HAS VIRTAUL LDOM
virtual dom is not reacts concept, it executed earlier also , virtual dom is the dom u see the tags in element section : its kind js rep of it(the html code, the object representation of it.)

React took that concept (virtual dom) and build its core also over that

React can efficelty find out the diff btw virtual looms and update the ui , this is the core of Reacts Algo (react fibre)

react is fast : has virtual loom, has diff algo which s very efficent , can do effects dom manipulation , it can find out the diff and update the ui : this is the core of react.

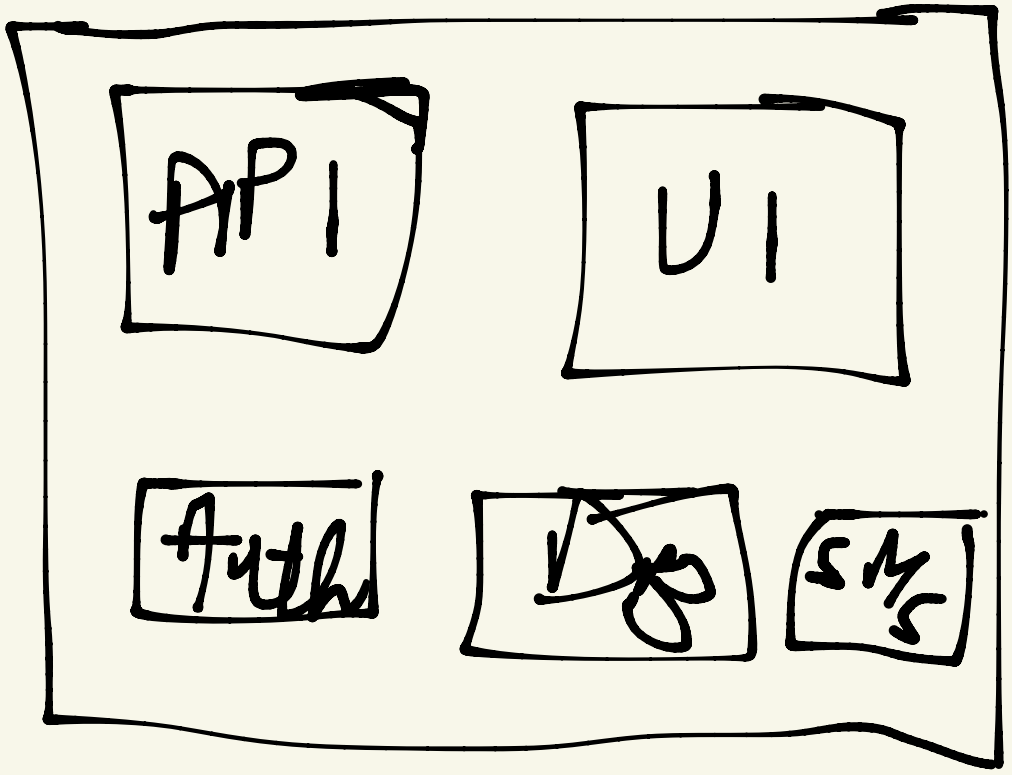
READ EVERYTHING OUT , BE CURIOUS

React is constantly checkignthids state variable as it s a special var fro misstate , as soon as it updates ,find out the diff and update the ui , it ll basically flush out and render what is required.
and top rated rest : functionality : react dci undo it effiectly , this is core of react.
as soon as this setlist is called , react starts its reconcilaltio nalco, react starts sreredenring ur page.
So , this s why there is another set list here, coz as soon as u called setlist, react ll find out the diff and update the ui : this is the core of react.

why it returns an array of two things the usestate, why can't we just modify directly :- coz there needs to be a trigger to start the diff algo and update the ui ie why they Makde thesecodn fxn : whence u call the second fin , it ll automatically reredner the comp. This is reacts algo.

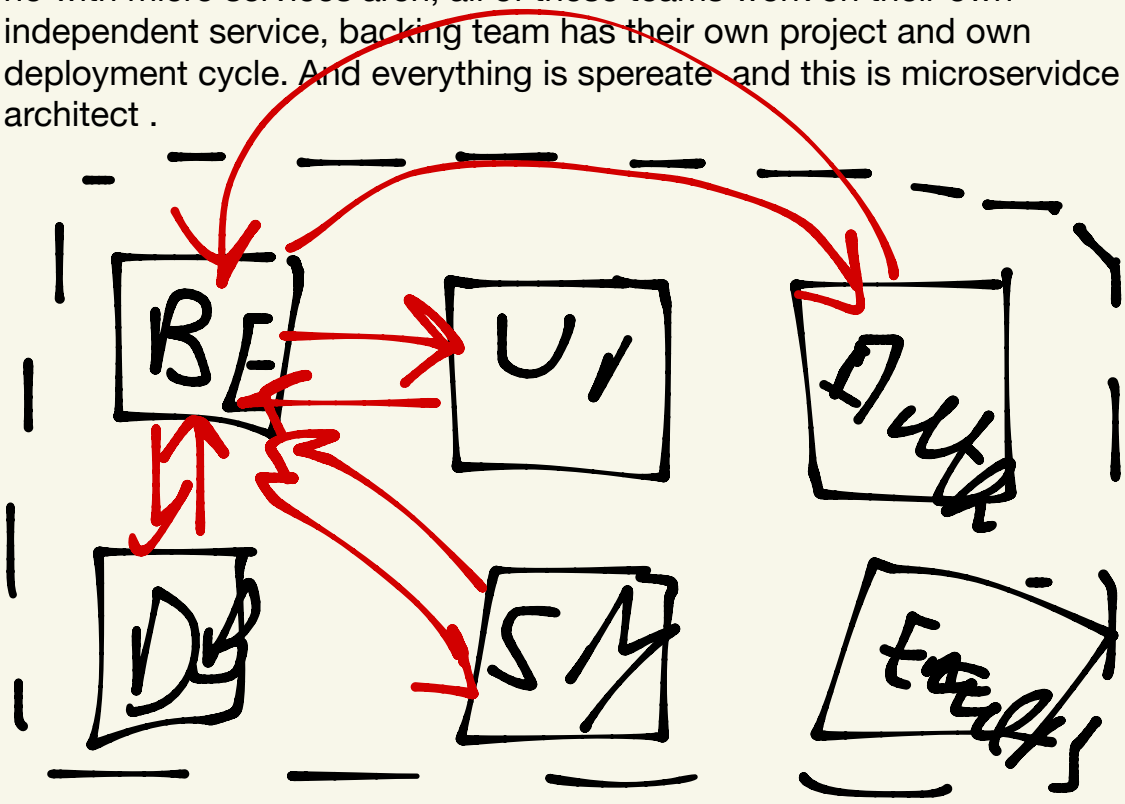
Hooks : are nothing but norma l js utility fan , (usestate) , utitly fan gives access to super power site variables I react , why they are super pwerful , coz react is keeping an eye on it , track on it and whenever these var's updates ,react ll trigger its diff algo , it l find out the diff virtual dose and automagically updates the ui , it keeps the ui layer and data layer synced and ie what the core algo of react is.

Monolith Architecture :-
traditionally, when the apps were developed using monolith architecture, earlier we used t have a huge big project, and suppose we are building huge big project , this project (in it we used t have small pieces) , the project itself has code of apis (where apis are written , we have developed apis in this project) , we also have ui code in the same project , auth etc … as below, notifications, sending sms In same proejct.
for eg :0 in a project , we had apis , frontend writer, auth written, connection to db written , notifications ,written , all the code was written in same service, suppose even have to make a single change ,to change a colour of btn , we have to build this whole predict ,compile it whole project : Monolith architecture.

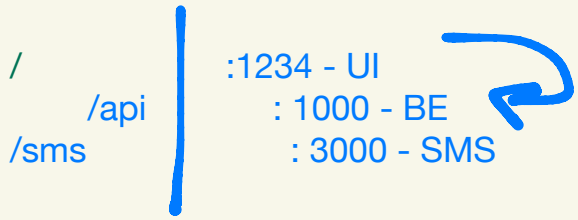


But now all the compaiesare preferring to move towards **micro service architecture. :- we have diff services for diff jobs , we have service which is backend service, we have ui project , auth service, service which connects to DB , SMS service, maybe another email notification service , and these all micro services (small services) combine together forms a big app.**
for eg : in company which uses this approach., having lot of micro services , and all these micros services talk to reach other depending on the usecarses, wee have sep ui project (can be fro eg separate git) , separate backed nprkejct etc, for each and every small thing, we have diff project, and this is known as Seperation of Concerns and it follows Single Responsibility principle. When each and every service has its own job, no one is interfering in it.

earlier in monolithic ,a big predict and all the dev's , backends , frontend , everybody used to work on same project/same repository, no with micro services arch, all of these teams work on their own independent service, backing team has their own project and own deployment cycle. And everything is spereate and this is microservidce architect .

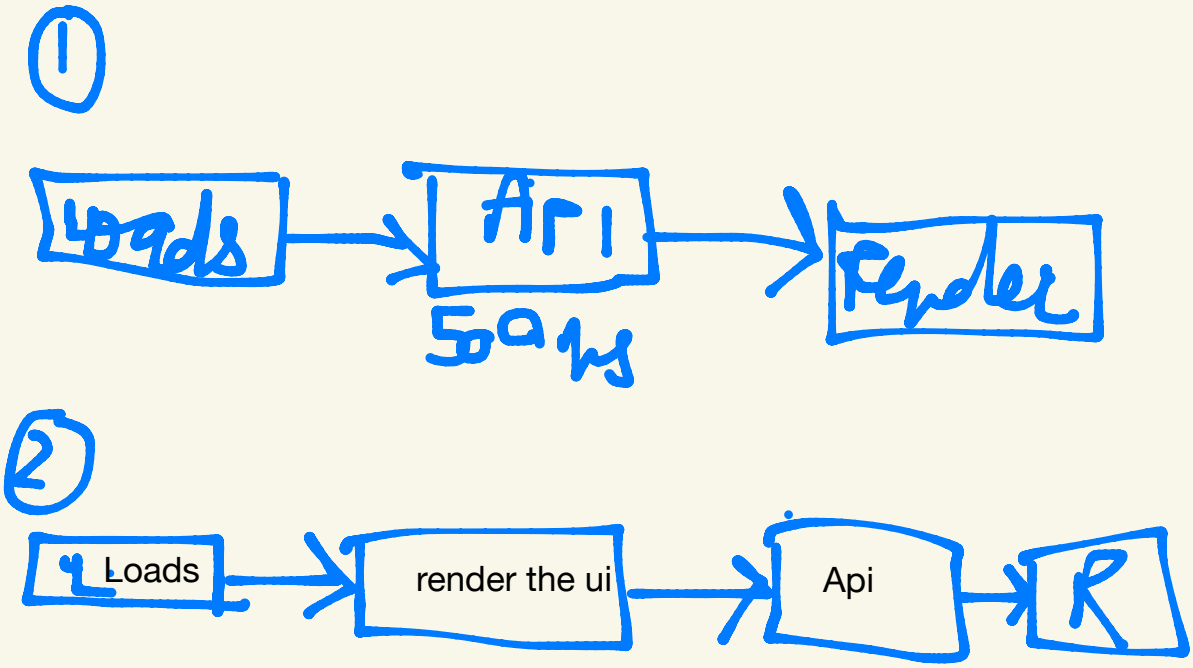


One more thing is **how do these services interact with each other.**
So this is that these services have to interact with each other , to make the app work , the UI will talk to backend ,fetch the data and show on ui , the be service need to talk to db , auth , be to notification serve , so all these services talk to each other.
One question :- when we have all these services , how are all these services deployed, and how can we accesss these servcies
for eg , where does our project comes in , supposed our project namastey react main is a ui micro service, its on localhost :1234.
One more adv of micro service ,u can have diff tech stacks for diff things , suppose in monolith u have one big prjcct , a java project, then u have to do everything in java , but in micro service, u can have ui written in react, be written in java, db written in python sms in Solang, U can written ur microservie in any architecture u want.So how these service are connected , they run on their own specific ports, suppose on port 124 ui etc so on diff ports we can deploy def services, at the ned of day ,all these ports ca nee mapped to domain name.
Suppoe backed is mapped to /api, whatever ur domain name is eg:- nanmasteydev.com/api and all these apis are deployed on /api and for sms can be /sms , and ui just slash as soon as it hits to domain name the slash , it redirects to port 1234. So that's how it works .
How these services interact , : they made call to diff urls, suppose ui wants to connect to BE, they ll call to /api or we ll cal this /api port.
Thats how these services are connected and interact with each other.



Now we ll see how this ui react app will make connection or explore the world, talk to diff micro services outside of the world ,how does react app make be api called fetch the data.

How webApps / UI apps fetch the data from backend :-
Two Approaches:-



1. when our app loads , we can make the api call, when we get the data , then we can render it on to ui, suppose api takes k500ms ,then out page will wait for 500ms and after that it ll render it. U have seen when u got t osoem website it doesn't load , and the nwhe data comes it shows the page.
2. as soon as page loads (jsx) , we ll just render the ui (REDNER THE SKEELTON) U can latest see sth on page and slowly the website loads, bette user expericen and use r not see the lag , after what we have quickly rendered, we ll make the api call, and as soon as get the data , we ll now redredner the app with the data from the api once again.

(IN REACT WE WILL ALWAYS USE 2ND APPROACH , IT SAME AS USEEFFECT ,FIRST RETUR NFX NJSX WILL BE RETURNED THEN AFTER THAT USEEFFECT IS CALLED AND API DATA IS FETCHED AND SHOWN

THIS GIVES U A BETTER UX. COZ IN FIRS T APPROACH FOR 500 MS WE DONT SEE ANYTHIGGN AND AFTER HAT WE SUNDDELTY SEE

1. when our app loads , we can make the api call, when we get the data , then we can render it on to ui, suppose api takes k500ms ,then out page will wait for 500ms and after that it ll render it. U have seen when u got t osoem website it doesn't load , and the nwhe data comes it shows the page.
2. as soon as page loads (jsx) , we ll just render the ui (REDNER THE SKEELTON) U can latest see sth on page and slowly the website loads, bette user expericen and use r not see the lag , after what we have quickly rendered, we ll make the api call, and as soon as get the data , we ll now redredner the app with the data from the api once again.

   (IN REACT WE WILL ALWAYS USE 2ND APPROACH , IT SAME AS USEEFFECT ,FIRST RETUR NFX NJSX WILL BE RETURNED THEN AFTER THAT USEEFFECT IS CALLED AND API DATA IS FETCHED AND SHOWN

   THIS GIVES U A BETTER UX. COZ IN FIRS T APPROACH FOR 500 MS WE DONT SEE ANYTHIGNN AND AFTER THAT WE SUDDELTY SEE EVERYTHING, SO THATS A POOR UX.

Now u ll say we are rendering twice , doesn't matter React is mostly part cos its redress cycle are very fast , react has best render mechanism. So we ll Bothe abt tow much rendering but I think here , two renders are ok.