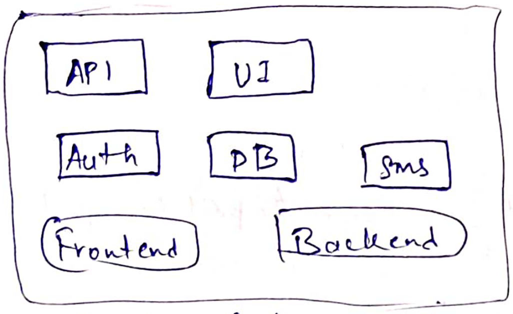
Ep 06 - Exploring the World (Monolith/Microservice Architecture)

Monolithic Architecture: Traditionally All Web App's were developed using Monolith Architecture.



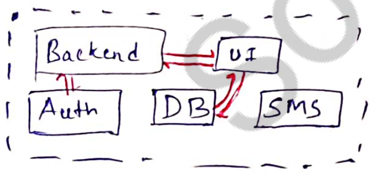
- Earlier, we used to have huge project all codes are writing into these one project.
  eg:- API Code In same project
  - UI Code in same project
  - Auth Code in Same project
  - DB Connectivity Code in Same Project etc

- All the Code written in Same Service, Same Project.
  - Frontend Code in Same Project, Backend Code in Same Project

- Suppose, Now even if we want to change a Small Piece Code. (Small Change) eg:- Change the Color of button
  - We need to build whole project / compile whole project / deploy whole project.

MicroService Architecture: In today's time, all the big Companies are preferring MicroService Architecture

We have different Services for different Jobs



- Here we have different MicroServices / Small Services
  eg - different Service for Backend proj
  - different Service for UI proj
  - different Service for Auth
  - different Service for DB

And all these Micro Service, Combines together forms a big App

As we have Separate project for each Service, This is called **Separation of Concern**

It follows Single Responsibility Principle where each Service has its own job

- Main adv: you can use diff TECH STACK, for diff Services eg:- UI in React
  Backend in Java
  DB in py
  etc

Two approaches How Web Apps | UI Application Fetch data From Backend

1)
$$\begin{tikzcd}\boxed{\text{App Loads}} \arrow[r] & \boxed{\text{API call}} \arrow[r] & \boxed{\text{Render}}\end{tikzcd}$$

As soon as the App loads we will make an API call & when we get the data, then we can Render it onto UI

~~Suppose API call takes 500ms,~~ then page will load & wait for 500ms to get the data. and after that it will Render the UI

Skeleton Loading (Shimmer UI)

②
$$\begin{tikzcd}\boxed{\text{App Loads}} \arrow[r] & \boxed{\text{Render}} \arrow[r] & \boxed{\text{API call}} \arrow[r] & \boxed{\text{Re-render}}\end{tikzcd}$$

~~Always~~ Always use these approach

Better approach gives better UI

As soon as the page loads, we will just Render our UI (quickly Render UI) after we have Quickly Rendered, now will make API call and as soon as the data is Fetched from API, we will Re-Render our UI and Populate the data

useEffect() hook: import {useEffect} using "react";

Syntax:

$$\text{useEffect}(\overbrace{\underbrace{() \Rightarrow \{\}}_{\text{callback Fn}}, \underbrace{[\,]}_{\text{dependency array}}}^{\text{Two arguement}});$$

This Callback Fn will called after your Component Renders

How useEffect Hook works? When the Component get Rendered & as soon as ~~the Component Rendering is finished, useEffect~~ Callback fn is Called.

If you have to do something after rendering the Component, you have write i ~~inside the useEffect.~~

Whenever you try to Fetch a live api From a Website to your local Server it will give you error. i.e. live api will be blocked if you try to fetch to your local Server. becoz its AGAINST CORS POLICY.

Hence, to bypass these Error use Chrome Extensions.

Allow CORS: Access-Control-Allow-Orig

Shimmer UI :- In order to Increase User Experience, instead of using loader, loading Circle, Now-a-days Industries are Using Shimmer UI.

- It generates a fake loading Screen as per your Componen dimension. As soon as App loads. Shimmer UI is show
- It shows Skeleton of Component.

Mostly used in API calls before showing data on UI

UseEffect : → Manage Side Effects → Something unexpected occurs and not occurs as per our expection.

↓ [change]

A side Effect is a change that affects Something outside the Component being Rendered.

⇒ Use Effect allows a Component to handle SideEffect.

Syntax :

```
useEffect (Setup, dependencies?)
```

↓ callback function

↓ dependenices list (~~empty~~ Array)

Jis bhi Component ke andhar aap useEffect hook likhte hai, wo use Component ke render hone ke baad → useEffect ke andar jo bhi Code (fn) likha hoga, usko run karega.
(execute hona shuru ho jayega)

Eg. DOM update / API call / Update doc title

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

There are 4 Variation of using UseEffect in React.

1)

```
useEffect ( () => {
    //___
});
```

- updates everytime even after Small change
- No dependencies passed as 2nd parameter
- Every Render

2)

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

- updates only on first render.
- Empty passed which leads to render only one time
- ✗✗✗ mostly used

3)

```
useEffect ( () => {
    //___
}, [text]);
```

- first render +
- Dependency name is passed in array
- It will only render, when dependency is changed
  i.e, callback fn executed Only when dependency passed is changed

4)

```
useEffect ( () => {
    //___
    return () => {
        //___
    }
}, [text]);
```

- used Inorder remove listener or clean listener
- to handle unmount of component
- ~~first return~~ State executed & then call back fn execute