**Theory Chapter 06**

**Q: What is Expression in JS?**

**A:** At a higher level expression is something which evaluates to value.

2 types of express

1. Those which has side effects (eg assigning a value) x= 7
2. Those who simply evaluates a value. Eg: 3+4. Complex Expression are joined by operators

**Q: What are statements?**

**A:** Statements perform specific actions. Like declaring a variable, looping through an array, function statement.

**Q: What are microservices?**

A: they are architectural approach . In this software is composed of small independent services which communicates through api. These services are owned by small , self contained team.

Features

1. Easy to develop
2. Faster in scaling
3. New features (New idea) can be easily added according to new market.
4. All process are loosely coupled.

**Monolithic vs. Microservices Architecture**

With monolithic architectures, all processes are tightly coupled and run as a single service. This means that if one process of the application experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application’s features becomes more complex as the code base grows. This complexity limits experimentation and makes it difficult to implement new ideas. Monolithic architectures add risk for application availability because many dependent and tightly coupled processes increase the impact of a single process failure.

With a microservices architecture, an application is built as independent components that run each application process as a service. These services communicate via a well-defined interface using lightweight APIs. Services are built for business capabilities and each service performs a single function. Because they are independently run, each service can be updated, deployed, and scaled to meet demand for specific functions of an application.

**Q**:  **why do we need useEffect hook?**

**A:**  helps you to perform any side effect in function based component

It accepts one callback and array dependency , if one is not passing any dependency then after each render useEffect ll be called. If one passed a blank array then use effect will be called only once after initial render . When one add dependency then useEffect will be called after initial render plus when ever the dependency is changing.One can use this hook for cleaning purpose also , it call when the component is unmounting from the DOM

useEffect(()=>{

return (()=>{

console.log(‘called when the component is unmount’)

})

},[])

**Q: Optional Chaining (?.)**

**A: it**  is an operator which access a object property or calls a function. And if the function call or function does not exist or the object property is undefined or null it does not throw an error instead it returns undefined

It’s a shorthand for accessing the chained property and there is a possibility that one property can be missing

Eg obj?.name , funcatuonName?.(), arr?.[42]

**Q: Shimmer ui**

**A:** Simmer is mimics the actual ui but does not contain any data , it shows the same elements of the page layout without having any data into it. It tells the user how the actual data ll look (layout part) like before rendering any data on the screen

**Q: Conditional Rendering**

**A:**  conditional rendering in react work in the same way conditions work in java script. One can use **if else**  or **conditional (ternary) operator. Like**

1. [condition ? true : false](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Conditional_Operator).
2. Array.length >0 && <h1>Greetings</h1>
3. const E1={

return(<h1> Greeting</h1>)

}

const E2={

retun(<h2>Heading 2</h2>)}

const App =()=>{

const [isLogin,setIsLogin] = useState(true)

if(isLogin){

return <E1/>

}

Else{

Return <E2/>

}

}

**Q: What is CORS?**

**A:** Cross origin resource sharing. it’s a mechanism which uses extra http headers and tell the browser if one app can share data/resources with other app and both the app should have different origin

Different origin like having different port no , different origin or sub origin of same domain , different protocols like http and https

Before making the actually api call a cors prefight also called as option call is made first when one app is accessing the resources from other app . Other app takes the responsibility for verifying this call and if this call is valid then the other app set addition header and then these header ll tell the client/ browser that this is safe and then the actually call is made.

Additional Header – access -control-allow-origin :\* or the domain name

access -control-allow-method :PUT

for some request this prefight call is not made as the browser has marked that request as simple so directly request call is made.

**Q: Async and await**

**A:** async and await helps us to write promises in a better way

Async make a function as promise returning function and return promise a fulfilled or a rejected . The resolved value is treated as the return value of the function. If the return value of async function is not promise then it ll implicitly wrapped into a promise

Eg async function foo() {

return 1;

}

function foo() {

return Promise.resolve(1);

}

Both the above function are same.

Code after each await expression can be thought of as existing in a .then callback. In this way a promise chain is progressively constructed.

The await inside the async function suspend its execution until till is a fulfilled or rejected value we get . but the normal execution of js code continues

**Async function t(){**

Console.log(1)

Const val = await new Promise((res,rej)=>res(2))

Console.log(2)

**}**

**Console.log(‘A’)**

**t()**

**Console.log(‘B’)**

**OUTPUT** : A 1 B 2

**Q: Need of await before data.json()**

**A:**  data.json returns a promise . So one needs to await for that promise to resolve.

response.json() is a method on the Response object that lets you extract a JSON object from the response

After the initial fetch() call, only the headers have been read. So, to parse the body as JSON, first the body data has to be read from the incoming stream. And, since reading from the TCP stream is asynchronous, the .json() operation ends up asynchronous.

Note: the actual parsing of the JSON itself is not asynchronous. It's just the retrieving of the data from the incoming stream that is asynchronous.