Session 6 ⇒14/01/2023

Chapter 06 - Exploring the world

- What is a Microservice?
- What is Monolith architecture?
- What is the difference between Monolith and Microservice?
- Why do we need a useEffect Hook?
- What is Optional Chaining?
- What is Shimmer UI?
- What is the difference between JS expression and JS statement
- What is Conditional Rendering, explain with a code example
- What is CORS?
- What is async and await?
- What is the use of `const json = await data.json();` in getRestaurants()

←——————— Monolith & Microservices ———————————————————————>

"The world is moving towards Microservices…."

Independent App for different functionalities/services

There are around 100s of microservices being used inside a large scale application.

In Monolith architecture :

For example a java app containing the code for all sorts of functionalities like SMS, NOTIFICATIONS, UI, APIs, JSP inside one project/one repository.

So, the con of such an architecture is like if i want to just change one button then i would have to deploy my entire app again. ⇒ This is not good.

Although there exists a lot of benefits of monolith architecture.

Whereas in Microservices Architecture :

Different independent repositories/projects/apps for different functionalities like LOGS, BACKEND APIs, UI, AUTHENTICATION, NOTIFICATIONS, etc. exist at different ports.

Not just this, there could be different servers and different replicas of DBs as well.

1. Easier to Test.
2. Easier to Maintain. To change just one button, ew can just deploy the UI project.
   `Separation Of Concern OR Single Responsibility`
3. Different functionalities can use different tech stacks(based on the use-case) as they all are independent projects.

We are building the UI Microservice deployed on Swiggy.com.

How all these apps are connected ?🤔

Ports.

swiggy.com : 3000 → /

:4000 → /dapi

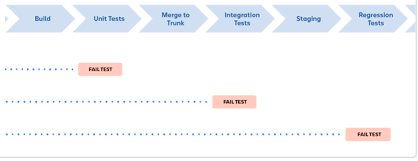
:5000 →/notifications

Reference :

Microservices vs. monolithic architecture | Atlassian

A monolithic architecture is a traditional model of a software program, which is built as a unified unit that is self-contained and independent from other applications. The word "monolith" is often attributed to something large and glacial, which isn't far from the truth of a monolith architecture for software design.

https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith



←———————————————— useEffect Hook ————————————————————————————>

→ useEffect() Hook ⇒ just a normal js function at the end of the day ⇒ what's the need ?

→ to execute some piece of code, whenever some specific things change.

→ means we want to execute code at some particular events only.

→ as the name suggests, use the effect of some thing to execute this

→ has a dependency array

→ no dependency array, executes this code at every re-rendering of the component

→ [] (empty dependency array), executes this code after initial render

→ [searchText] ⇒ whenever searchText changes

←———————————————— Best Place To Use API ————————————————————————>

```
useEffect(() => {
    // API Call
    getRestaurants();
  }, []);
```

←———————————————— Optional Chaining ————————————————————————>

Optional chaining (?.)

The **optional chaining (`?.`)** operator accesses an object's property or calls a function. If the object accessed or function called is `undefined` or `null`, it returns `undefined` instead of throwing an error.

Try it

JavaScript Demo: Expressions - Optional chaining operator

```
1  const adventurer = {
2    name: 'Alice',
3    cat: {
4      name: 'Dinah'
5    }
6  };
7
8  const dogName = adventurer.dog?.name;
9  console.log(dogName);
10 // Expected output: undefined
11
12 console.log(adventurer.someNonExistentMethod?.());
13 // Expected output: undefined
```

←———————————— Shimmer UI ————————————————————>

Shimmer

A shimmer screen is a version of the UI that doesn’t contain actual content. Instead, it mimics the page’s layout by showing its elements in a shape similar to the actual content as it is loading and becoming available (i.e. when network latency allows).

A shimmer screen is essentially a wireframe of the page, with placeholder boxes for text and images.

Usage

A shimmer UI resembles the page’s actual UI, so users will understand how quickly the web or mobile app will load even before the content has shown up. It gives people an idea of what’s about to come and what’s happening (it's currently loading) when a page full of content/data takes more than 3 - 5 seconds to load.

Shimmer effect loading for cards => https://codepen.io/andru255/pen/wvBdxbb

Although, we can ourselves create a basic one!

←——————————————— JS Expression vs Statement ———————————————————————>

```
{
        // any js code
        // Expression => Accepted
        // ((a = 10), console.log(a))
        // Statements => Not Accepted
        // if() {

        // }

        // similarly, ternary ? operator is an expression whereas if else is a statement
        (isLoggedIn ? <button onClick={() => {setIsLoggedIn(false)}}>Logout</button> : <button onClick={() => {setIsLoggedIn(true)}}>Login<
      }
```

←——————————————— Conditional Rendering ———————————————————————>

```
// Conditional Rendering
  // if restraunt is empty -› shimmer Ui
  // if restraunt has data =› actual data UI

  // Early Return
  if (!allRestaurants) return null;

  return allRestaurants.length === 0 ? (
```

```
    <Shimmer />
  ) : (
    <>
      <div className="searchContainer">
        <input
          type="text"
          className="search-input"
          placeholder="Search"
          value={searchText}
          onChange={(e) => {
            // e.target.value => whatever you write in it
            setSearchText(e.target.value);
          }}
        />
        <button
          className="search-btn"
          onClick={() => {
            // need to filter the data
            const data = filterData(searchText, allRestaurants);
            // update the state -> restaurants variable
            setFilteredRestaurants(data);
          }}
        >
          Search
        </button>
      </div>
      <div className="restaurantList">
        {/* map in jsx : - */}
        {(filteredRestaurants?.length === 0) ? <h1>No Restaurant Found!</h1> : filteredRestaurants.map((restaurant) => {
          return (
            <RestaurantCard {...restaurant.data} key={restaurant.data.id} />
          );
        })}
      </div>
    </>
  );
```

←———————————————— CORS ————————————————————>

→ Cross Origin Request Sharing

→ Browser prevents one application at origin1 to access any resources(for e.g. api) of any other app present at origin2.

→ In order to be able to share resources, a preflight options are sent to the origin2 by the browser and in response some additional headers are sent back.

→ if the Accept_Control_Allow_Origin contains url of app1 at origin1 then it means now it can access those resources.

→ `Accept_Control_Allow_Origin : *` ⇒ any domain outside of this(it's own) domain can access it.

→ Additional headers also mention the capabilities such as what all GET, POST, etc. are allowed.

←———————————————— Async & Await ————————————————————>

→

Async/await

There's a special syntax to work with promises in a more comfortable fashion, called "async/await". It's surprisingly easy to understand and use. Let's start with the async keyword. It can be placed before a function, like this: The word "async" before a function means one simple thing: a function always returns a promise.

https://javascript.info/async-await