Chapter 06 Assignment Theory

1. **What is a Microservice?**
   Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs. These services are owned by small, self-contained teams.

   Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features.

2. **What is Monolith architecture?**
   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. A monolithic architecture is a singular, large computing network with one code base that couples all of the business concerns together. To make a change to this sort of application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface. This makes updates restrictive and time-consuming.

3. **Difference between Monolith and Microservice?**

Monolith	**Microservice**
If all the functionalities of a project exist in a single codebase, then that application is known as a monolithic application.	It is an architectural development style in which the application is made up of smaller services that handle a small portion of the functionality and data by communicating with each other directly using lightweight protocols like HTTP
Pros: **Easy development and deployment:** Developers need to perform a single chunk of deployable code instead of making updates in separate entities. **Performance**: A microservice-based application may have to make 100 different API calls to 100 other microservices to load one UI screen. Whereas in the monolithic, one API call can serve the same purpose because it has a centralized code and memory. Cons: **Tight Coupling**: Service modules in monolithic applications are tightly coupled. Business logic is tightly entangled and makes it difficult to	Pros: **Better Organization:** The microservices have a particular job and are not dependent on other components. Increased Agility: With microservices, individuals of a team can work on individual modules.

isolate the application, and hence scalability becomes a challenge. **Slow Build and Release cycle:** Since the code base is enormous, this retards the velocity of the development and testing cycle of the application.	

4. **Why do we need a useEffect Hook?**
   The useEffect Hook allows you to perform side effects in your components.
   Some examples of side effects are: fetching data, directly updating the DOM, and timers.
   Syntax:
   useEffect(<function>, <dependency>)

5. **What is 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.

6. **What is Shimmer UI?**
   Shimmer is a temporary animation placeholder for when data from the service call takes time to get back and we don't want to block rendering the rest of the UI.

7. **What is the difference between JS expression and JS statement?**
   JS statement: A statement performs an action. Loops and if statements are examples of statements.
   JS expression: An expression produces a value and can be written wherever a value is expected, for example as an argument in a function call.

8. **What is conditional rendering?**
   conditional rendering refers to the process of delivering elements and components based on certain conditions.
   Code:

```
import React, {useState} from "react"
import {restaurantData} from '../config';

export default RestaurantList = () => {
    const [restaurant, setRestaurant] = useState(true);

    const searchHandler = ()=>{
            setRestaurant(!restaurant);
    }

    return restaurant ? (
        <>
            {/* code */}
        </>
    ) : (<NotFound />);
}
```

9. **What is CORS?**
   Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources. CORS also relies on a mechanism by which browsers make a "preflight" request to the server hosting the cross-origin resource, in order to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.

10. **What is async and await ?**
    The keyword **async** before a function makes the function return a promise.
    The await keyword makes the function pause the execution and wait for a resolved promise before it continues
    The **await** keyword can only be used inside an async function.

```
async function f() {
  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("done!"), 1000)
  });

  let result = await promise; //wait until the promise resolves

  alert(result); // "done!"
}

f();
```

    The function execution “pauses” at the line with *await* and resumes when the promise settles, with result becoming its result. So the code above shows “done!” in one second.

11. **What is the use of `*const json = await data.json();*` in getRestaurants()**
    The json() method of the Response interface takes a Response stream and reads it to completion. It returns a promise which resolves with the result of parsing the body text as JSON.