What is an effect in react ?

Effects let you specific side effects(changes) that are caused by rendering/ or re-rendering of a component.
Example of an effect can be:

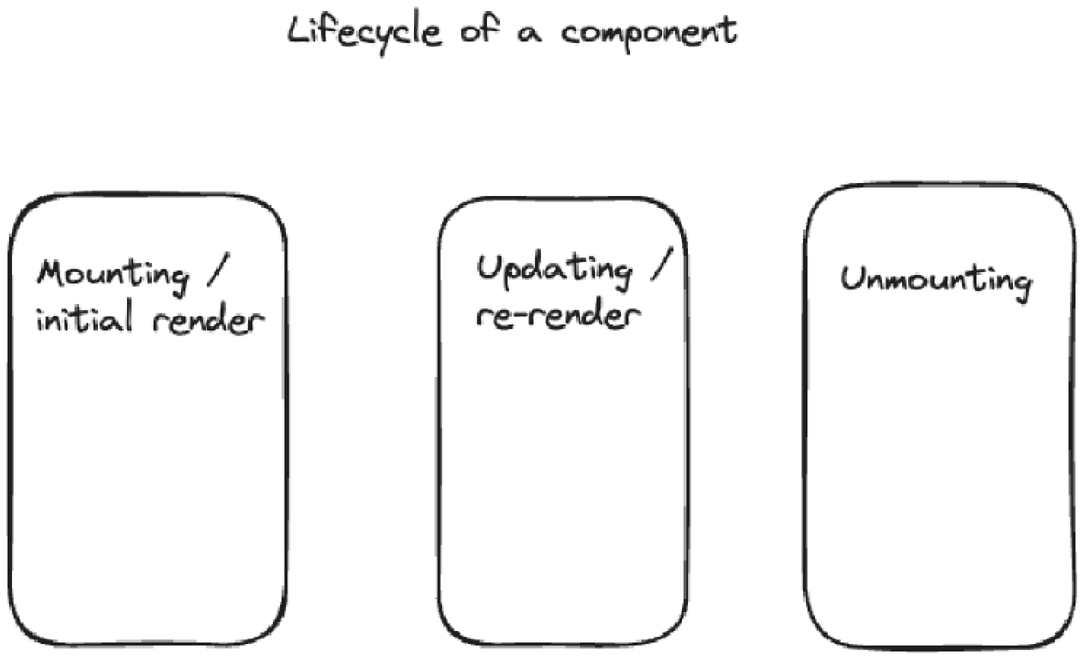
- Downloading data
- Reading data from local storage
  when your component is being rendered, we want to download some data.
  To implement effects in react, we have a hook called as `useEffect`.

According to the official docs:
`Effects can help you to synchronize your frontend with external systems.`

Lifecycle events in a component:

Whenever a component is brought into the picture, there are multiple lifecycle events that it goes through

1. Mounting / initial render - This is the phase of the first time loading of the component, i.e. when the component is added to the dom for the first time.
2. Re-render / Updates - When due to a state update or parent re-render, a component re-renders then this is the phase of updating / re-rendering a component.
3. Unmounting - This is the phase of removing component from dom.



If we want to control, What should happen or what logic needs to be executed while mounting a component, unmounting or re-render a component, we can control this by `useEffect`.

State vs Effects

States can cause a component to re-render, but effects can helps us to control when a re-render happens , what to do.
In a nutshell, states are one of the causes of re-render and effects are consequences of re-render.

Protocols :

Protocol is a set of rules that govern how data is transmitted over a network.

there are so many protocols like

- for email sending there is `SMTP` protocol
- if you want to transfer file we need `FTP` protocol.
- `HTTP (Hypertext Transfer Protocol)` is a client-server protocol that enables data exchange between web browsers and web servers. There are lot more protocols......
- `Servers` are those machines which can take requests process the request and give you the response
- `Clinets` are those machines which are capable of raising the request. Browser is kind of like a client

- In order to make sure that browser communicate the server browser used to expose a function called `XMLHttpRequest`.
- we can use that `XMLHttpRequest` in order to download some content from internet.

XMLHttpRequest:

For a very long time to make a network request from the browser, we used to use an object of HttpRequest. This is capable of making a network call and downloading content.

```
const xhr = new XMLHttpRequest(); // we need to create an object of
XMLHttpRequest

// This open function initiates a connection request when we call send
function
// This open function takes the method type and url for the network call
xhr.open("GET", "https://jsonplaceholder.typicode.com/todos/2");

// Once we have the response from server, then the call back of onload is
```

```
executed
xhr.onload = function () {
    if(xhr.status >= 200 && xhr.status < 300) {
        console.log("Response", xhr.responseText);
    } else {
        console.log("seomthing went wrong");
    }
}

xhr.send(); // This triggers the final call.
```

XMLHttpRequest : open() method:

https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/open

The XMLHttpRequest method open() intializes a newly created request, or re-initializes an exixting one.

```
open(method, url)
open(method, url, async)
open(method, url, async, user)
open(method, url, async, user, password)
```

In method we have "GET" , "POST", "PUT", "DELETE" etc;

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form

In this html form we have action and method are there in method we mention the HTTP method . HTML form only support "GET " or "POST" methods only. If you try to put here another method HTML form cannot process them.

SO, here if you want to make a "DELETE" ,"PUT", request also along with "GET " and "POST " in that case we need to use JavaScript

JS can actually make any requests but in HTML forms can only make "GET " and "POST" requests.

- HOW DO WE MAKE JAVASCRIPT REQUEST ? BY USING `XMLHttpRequest.`
  The first parrameter of XMLHttpRequest is method

```
open(method, url)
open(method, url, async)
open(method, url, async, user)
open(method, url, async, user, password)
```

Fetch API:

The fetch API provides an interface for fetching resources. It is a more powerful and flexible replacement for XMLHttpRequest.

Interview question

What is the browser property that technically helps you to make network request? XMLHttpRequest now we have FetchAPI its like a replacement of XMLHttpRequest which is now a much more standard way to make API calls or network calls.

Fetch - Alternative to XMLHttpRequest :

So, XMLHttpRequest is an old way to make network calls. It is more callback based and syntax is also not so simple. That's why modern browser support `fetch.`

```
async function download() {

    const response = await
fetch("https://jsonplaceholder.typicode.com/todos/2");

    console.log(response);

    const result =  await response.json();

    console.log(result);

}

downlod();
```

Here, fetch returns a response, which is a promise, once that promise is resolve from the resolved we need to call `.json()` which is again a promise based call which gives us the final resultant json.

NOTE :

Fetch doesn't internally use HTMLHttpequest instead it is just an alternative. Fetch is promise based whereas HTMLHttpequest is callback based.

useEffect Hook:

useEffect is one of the most important hooks in react and is a way to handle life cycle of the component in which it is present .
useEffect takes two arguments

- First argument is the callback that callback is the function which you want to execute on a particular life cycle of an event.
- Second argument is a dependency array if you do not put this dependency array what will happen is every time due to any reason if the component re-render this callback will re-render again and again.
- But if you put a empty dependency this callback will be only executed on the first render that is mounting after that if there is any subsequent re-render callback won't execute.

```
useEffect(callback,[]);
```

```
import React, { useState, useEffect } from "react";

function CoinTable() {

  // Use useState to declare state

  const [count, setCount] = useState(0);

  const[flag, setFlag] = useState(false);



  async function download() {

    const response = await
fetch("https://jsonplaceholder.typicode.com/todos/2");

    console.log(response); // This logs the full response object

    const result = await response.json(); // Fetch the JSON data from the
response

    console.log(result); // Log the parsed JSON data

  }


  useEffect(() => {

    // Because the dependency array is empty, this effect only runs once when
the component mounts

    download();

  }, [count]); //whenever count changes effect will be execute whenever flag
changes effect won't be executed


  useEffect(() => {

    // here dependency we have flag so  whenever re-render happens this will
execute again
```

```
  console.log("Flag changed");

  }, [flag]);



  useEffect(() => {

    // Because the dependency array is not empty, everytime it will excute
there is no dependency

  console.log("every Time changed");

  });



  useEffect(() => {

    // everytime re-render happens it will execute for both falg and count
  console.log("Flag or count changed");

  },[count, flag]);



  return (

    <>

      CoinTable

      {count}

      <br />

      <button onClick={() => setCount(count + 1)}>Increment</button>

      <br />

      {flag && <div>Falg is true</div>}

      <br />

      <button onClick={() => setFlag(!flag)}>Toggle</button>
    </>

  );

}



export default CoinTable;
```

JSON:

Json stands for `JavaScript Object Notation`. It is a lightweight format for storing and transporting data and it is often used when data is sent form a server to a web page and also it is easy to understand.

```
{
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]
}
```

Difference between query and mutations :

Queries and mutations are two classifications of our requests :

1. Queries
2. Mutations

queries :

Queries generally have read or fetch requests are used to read data or retrieve data.

- queries we can cacheable the queries (**if you request the same data again, React Query will return the cached results instead of making another API call**. The cache is automatically invalidated)

Mutations:

while mutations are used to change data like add, change, delete data. Mutations are similar to POST, PUT, DELETE requests.

- no cache require here.
- here we can use like Optimestic Update
  immediately updating the UI with expected changes, assuming that the corresponding server request will succeed.

useQuery

useQuery is a hook, or function, from the React Query library. It's used to fetch and manage data,, it can handle API requests and state management for React applications.