**When passing props do not use**

**const EstimatePaymentDetails = ({ invoice }) => {**

**not ( invoice )**

**2. For table**

To change the width, must change the all the cells in 1 column to change the table width here

**Use light house to do the load testing for frontend**

**When is frontend component unmounted?**

A React component is unmounted when it is removed from the DOM. This can happen for various reasons:

* Parent component unmounts:

If the parent component of a component unmounts, all its child components will also unmount.

* Conditional rendering:

If a component is conditionally rendered based on some logic, and the condition becomes false, the component unmounts.

* Routing:

In a single-page application (SPA), when navigating to a different route, components associated with the previous route unmount.

* Explicit removal:

You can explicitly unmount a component using ReactDOM.unmountComponentAtNode().

1. And then here we have some code

Hwo to return a map in 1 go here

 map(() => ())

**Conditionally assign fields**

const a = {

...(someCondition && {b: 5})

}

Where is the error showing up specifically?

Using frontend code here

1. Using frontend code

Create an array of 5 0s here

const stars = Array(5).fill(0)

**Using the onclick**

onClick = {welcomeUser} and that’s it here

**Map in 1 go**

const promises = asyncThingsToDo.map(asyncFetch)

asyncFetch() is a function here

Promise.All() will run in parallel here

What’s the best way to create a map?

4 mistakes when using useEffect in react here

How do we inherit type in typescript

export type MedicalHistoryCardConfig = EmailEntityConfig & {

recipients: EmailEntityConfigRecipient[] | Nil

}

**Common react mistakes**

**1) Mistake 4: Using State Variables Inside useEffect Without Dependency**

Another common mistake is using state variables inside useEffect without including them in the dependency array when they are needed.

Consider the following example:

import React, { useEffect, useState } from 'react';  
  
function MyComponent() {  
 const [count, setCount] = useState(0);  
  
 useEffect(() => {  
 if (count > 5) {  
 alert('Count is greater than 5');  
 }  
 }, []);  
  
 return (  
 <div>  
 <p>Count: {count}</p>  
 <button onClick={() => setCount(count + 1)}>Increment</button>  
 </div>  
 );  
}

In this code, we want to show an alert when count becomes greater than 5. However, since count is not included in the dependency array, the effect will capture the initial value of count, which is 0, and never update when count changes.

To fix this, include count in the dependency array:

useEffect(() => {  
 if (count > 5) {  
 alert('Count is greater than 5');  
 }  
}, [count]);

Now the effect will run whenever count changes, as expected.

2) Do not use async directly inside the async function here (otherwise you have to do the clean up here)

3) Need to use the function version to set the state here:

export function Counter() {

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

function adjustCount(amount) {

setCount(prevCount => prevCount + amount);

setCount(prevCount => prevCount + amount);

}

What are some frontend mistakes one can make?

Frontend technique for loading the images faster?

Lazy Load:

Only load images visible on sceren (using window event listener)

*1. Lazy loading is a strategy that delays the loading of some assets (e.g., images) until they are needed by the user based on the user's activity and navigation pattern. Typically, these assets are only loaded when they are scrolled into view.*[*(Source: MDN Docs)*](https://developer.mozilla.org/en-US/docs/Glossary/Lazy_load)

2. minify the css and javascript files here

**3. UseMemo[]**

- In React, there are three techniques four memoization: React.memo(), useMemo(), and useCallback().

The useMemo() hook optimizes performance by memoizing the result of a function call or an expensive computation.   
  
It caches the result and recalculates it only **when the input values change**. Below is an example on how to use the useMemo hook in functional component:

**What are some diff between javascript object and map?**

1. Key for object can only be string

2. **Object**, on the other hand is not iterable. To loop over every property in the Object, we need to get hold of either entries, keys or values which are returned as arrays and then possibly iterate over them.

3. With object you can get a direct support to convert them into a js function here

**Expalin how promise works in Javascript?**

Promise helps you avoid callabck hell?

2. In JavaScript, a Promise is an [object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_objects) that will produce a single value some time in the future. If the promise is successful, it will produce a resolved value, but if something goes wrong then it will produce a reason why the promise failed.

3. You use .then() to create a

promise.then(handleResolve, handleReject);

Can also use promise.all() and all that here

<https://www.freecodecamp.org/news/guide-to-javascript-promises/#why-should-you-care-about-promises>

The promise here would help

handleResolve and handleReject are the 2 functions needed for this

A Promise is in one of these states:

* *pending*: initial state, neither fulfilled nor rejected.
* *fulfilled*: meaning that the operation was completed successfully.
* *rejected*: meaning that the operation failed.

We can use spread operator to change 2 objects with the same name value properties

**52. What is event loop in JavaScript?**

JavaScript is a synchronous and single threaded language, that means JavaScript executes the code line by line in one go and do not wait for any code to get execute. The event loop is a mechanism built in JavaScript engine to handle the asynchronous tasks efficiently. The event loop continuously check the **call stack**and the **callback queue.**If it finds that the call stack is empty, it immediately pushes the asynchronous task from the callback queue to call stack to get executed.

1. **Call Stack:**
   * JavaScript uses a call stack to keep track of the currently executing function (where the program is in its execution).
2. **Callback Queue:**
   * Asynchronous operations, such as I/O operations or timers, are handled by the browser or Node.js runtime. When these operations are complete, corresponding functions (callbacks) are placed in the callback queue.
3. **Event Loop:**
   * The event loop continuously checks the call stack and the callback queue. If the call stack is empty, it takes

53.

CSS

**. Differentiate between the Inline and the Block elements in HTML.**

The [**Inline elements**](https://www.geeksforgeeks.org/html-block-and-inline-elements) in HTML are the elements that do not start from a new line every time and take up the same space and width as acquired by the content. The margin and padding properties applied at the top and the bottom of these elements may not have the expected effect. Examples: <a>, <strong>, <img>, <input> etc.

The Block elements automatically starts from a new line and takes up the whole view-port width irrespective of the contained content. The padding and margin properties have the same effect on all the four sides. Examples: <div>, <h1> to <h6>, <p>, <table> etc.

And then we have the code here