Assignment 9 - Optimizing Our App

Q1 When and why do we need Lazy()?

⇒ React.lazy() allows developers to easily create components with dynamic imports and renders them as normal components. When the component is rendered, it will automatically load the bundle that contains the rendered component.

You need to provide a single parameter to call React.lazy(). It accepts a function as an input parameter, and that function should return a promise after loading the component using imports(). Finally, the returned promise from React.lazy() will give you a module with a default export containing the React component.

// Code

```
const AboutComponent = React.lazy(() => import('./AboutComponent'));
```

Q2 What is Suspense?

⇒ When we use lazy loading, components are rendered as we navigate. So, we need to have a placeholder for those components until they are loaded. As a solution, React.suspense was introduced and it acts as a wrapper for the lazy component.

You can wrap a single lazy component, multiple lazy components, or multiple lazy component with different hierarchy level with React.Suspense. In addition, it accepts a prop named fallback as the placeholder, and you can pass a component or an element for that

Q3 Why we got this error: A component suspended while responding to synchronous input. This will cause the UI to be replaced with a loading indication. To fix, updates that suspend should be wrapped with startTransition? How does suspense fix this error?

⇒ This error is typically encountered in asynchronous contexts where components are fetching encountered in asynchronous contexts where components are fetching data or handling code splitting

Suspense is used to manage async data and code-splitting, allowing you to display a loading indicator while the data or code is being fetching.

This error is telling you that a component that was responding to synchronous input (meaning its not supposed to be waiting for anything) encountered a suspension. This should not happen because Suspense is primarily designed to handle asynchronous operations, and you generally don't want to introduce

delays in the rendering of synchronous user interactions.

How to fix this issue:

1) You should identify which part of your code is causing the synchronous component to suspend. This could be due to a network request, a dynamic import of a component, or another asynchronous operation

2) Ensure that the asynchronous code which might suspend, is wrapped within a suspense boundary and that you provide a fallback UI to display while waiting for the operation to complete

Q4 Advantages and disadvantages of using this code splitting pattern?

⇒ Code splitting is a technique used to break down a large monolithic JavaScript bundle into smaller, more manageable pieces, which can be loaded on demand

Advantages:

1) Smaller initial bundles result in faster load times for your web app.

2) Code Splitting can lead to better performance, as smaller bundles can be passed and executed more quickly. This can reduce the time it takes to render the initial page and improve the overall responsiveness of the application

3) Components or features that are rarely used may never be loaded unless needed. This conserves bandwidth and memory, making your application more efficient

4) Simpler Maintenance. You can make updates to a specific parts of your application, you can be more confident that you won't introduce unexpected issue in unrelated components.

Disadvantages:

1) Setting up code splitting and configuring it correctly can be complex, especially in large applications. You may need to make adjustments to your build tools and bundler settings

2) When a component is loaded on-demand, there may be a slight delay the first time it is needed, which can impact user perception of your application speed. However, this delay is usually minimal

3) Handling asynchronous loading and rendering of components requires careful design to ensure a seamless user experience. You need to consider scenarios such as loading indicators and error handling

4) Not all frameworks and libraries have built-in support for code-splitting

Q5 When do we and why do we need suspense?

⇒ There will be a slight delay when the code split component is fetched, so it is important to display a loading state otherwise it will not render the page & show error

1) Suspense helps to display fallback UI components like shimmer component while data is being fetched or code is being loaded

2) Increases performance as it allows you to load code and data only when its needed reducing the initial bundles size and making your app faster to load

3) Use suspense for data fetching when you want to make your application more responsive and provide a smooth loading experience for data-driven components. It simplifies the management of loading states and error handling

4) Use suspense for code splitting when you want to improve your application's initial load time and performance. It allows you to load parts of your application on-demand which can lead to faster rendering times and better resource usage