Code Splitting by Route (10)

1. What is code splitting in React and why is it beneficial for large applications?

Code splitting means to break the app bundle into smaller pieces that load on demand and it also reduces initial load time and improves performance as well.

2. How do you implement route-based code splitting using React.lazy() and Suspense?

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

<Suspense fallback={<div>Loading...</div>}>

<Route path="/about" element={<About />} />

</Suspense>

3. What is the role of the fallback prop in Suspense?

Fallback prop is used to display a temporary UI (loader) when the lazy-loaded component,using React.lazy() is being fetched.

4. How does code splitting improve initial page load time?

code splitting improves initial page load time by loading essential code first

And then other route/component code is fetched later which reducs bundle size at startup.

5. Compare code splitting by route vs code splitting by component.

Route-based means it splits entire pages,it is simpler

Component-based means it splits individual components,it is finer.

6. What happens if a dynamically imported route component fails to load? How would you handle that?

The app might show a blank screen or error.

We have to use ErrorBoundary or try/catch in Suspense to handle failures.

7. How does Webpack handle chunk naming in dynamic imports?

Using magic comment

import(/\* webpackChunkName: "about" \*/ './About');

8. What is the default chunk naming strategy if you don’t specify webpackChunkName?

Numeric or hashed chunk names are default names genarted automatically.

9. How would you lazy-load multiple components for the same route?

We have to use multiple React.lazy() imports in the same route and wrap them in a single <Suspense>.

10. Can you apply code splitting in nested routes? If yes, how?

Webpack Bundle Analyzer (10)

Yes

lazy-load components for nested routes individually using React.lazy() inside each nested <Route>.

11. What is Webpack Bundle Analyzer used for in React development?

webpack bundle analyzer visualizes the size of webpack output files and dependencies and optimizes bundle size.

12. How do you install and configure Webpack Bundle Analyzer?

npm install --save-dev webpack-bundle-analyzer

In web-config:

const { BundleAnalyzerPlugin } = require('webpack-bundle-analyzer');

plugins: [ new BundleAnalyzerPlugin() ]

13. What kind of insights can Webpack Bundle Analyzer provide about your build?

It shows bundle size breakdown, unused code and largest dependencies.

14. How do you identify large dependencies in your Webpack bundle?

To identify large dependencies the analyzer visual map is used, which highlights oversized modules and libraries.

15. What strategies can you apply after identifying large bundles in Webpack?

Strategies:

1. Code splitting
2. Tree shaking
3. Replacing heavy libraries with lighter alternatives
4. Lazy-loading

16. How do you run Webpack Bundle Analyzer in development mode vs production mode?

Development mode: it runs via dev server for real-time updates

Production mode: it runs after it is build ,in order to analyze optimized bundles

17. What is the difference between static and server modes in Webpack Bundle Analyzer?

Staticvmode is used to generate an HTML report file.

Server mode is used to runs a local server to explore interactively.

18. How do you exclude certain packages from Webpack’s analysis?

We can exclude certain packages from webpack;s analysis by using the plugin’s excludeAssets -option to filter specific modules.

19. How can tree shaking help after analyzing your Webpack bundle?

It removes unused code like dead imports which reducs bundle size automatically.

20. How does Webpack’s splitChunks configuration interact with bundle analysis results?

It controls how common code is extracted into separate chunks.

We can fine-tune it after seeing analyzer output.

State Lifting (State Up) (10)

21. What does “lifting state up” mean in React?

lifting state up means moving the shared state from child components to their common parent to control data centrally.

22. Why might two sibling components need state to be lifted?

Instead of being in isolated state , they need state to be lifted

So that they can share data or stay in sync with the parent

23. How do you pass data from a child component to a parent when lifting state?

When lifting state we can pass a callback prop from the parent to the child.then later the child calls it with updated data.

24. What is the main drawback of lifting too much state?

The drawback is that it can cause unnecessary re-renders and make components less modular.

25. How can lifting state up help prevent prop drilling?

It prevents prop drilling by managing data higher in the tree and passing only what’s needed to children.

26. Give an example of a form with multiple inputs where state lifting is used.

import React, { useState } from "react";

function NameInput({ value, onChange })

{

return (

<div>

<label>

Name:

<input type="text" value={value} onChange={e => onChange(e.target.value)} />

</label>

</div>

);

}

function EmailInput({ value, onChange })

{

return (

<div>

<label>

Email:

<input type="email" value={value} onChange={e => onChange(e.target.value)} />

</label>

</div>

);

}

export default function Form()

{

const [formData, setFormData] = useState({ name: "", email: "" });

const updateField = (field, value) =>

{

setFormData(prev => ({ ...prev, [field]: value })

);

};

const handleSubmit = e => {

e.preventDefault();

console.log("Submitted data:", formData);

};

return (

<form onSubmit={handleSubmit}>

<NameInput

value={formData.name}

onChange={val => updateField("name", val)}

/>

<EmailInput

value={formData.email}

onChange={val => updateField("email", val)}

/>

<button type="submit">Submit</button>

</form>

);

}

27. How do you prevent unnecessary re-renders when lifting state up?

We can use React.memo, useCallback, or break components into smaller pieces.

28. How do you combine state lifting with context to avoid deep prop passing?

To avoid deep prop passing we can use store lifted state in a Context provider so any nested component can access it directly.

29. When lifting state, why might you use useCallback in the parent?

In order to memorize callbacks and avoid creating new function instances on every render.

30. How can you lift a state without breaking controlled form elements?

We have to make sure parent always provide value and onChange props consistently so that inputs remain controlled.