**Code Splitting by Route (10)**

**1. What is code splitting in React and why is it beneficial for large applications?**  
Code splitting is the process of breaking your JavaScript bundle into smaller chunks so the browser only loads the code needed for the current view.  
**Benefits:**

* Reduces initial bundle size that is, faster first load
* Delays loading of unused code until it’s needed
* Improves perceived performance

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

import React, { Suspense, lazy } from 'react';

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

const Home = lazy(()

const About = lazy(()

function App() {

return (

<Router>

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

<Routes>

<Route path="/" element={<Home />} />

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

</Routes>

</Suspense>

</Router>

);

}

**3. What is the role of the fallback prop in Suspense?**  
The fallback prop specifies what to render while the lazy-loaded component is still loading. It can be text, a spinner, or a skeleton UI.

**4. How does code splitting improve initial page load time?**  
By loading only the essential code for the first view, the browser has fewer bytes to download, parse, and execute before showing content. Non-critical code is deferred until needed.

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

* By Route: Splits chunks for entire pages/views. They are ideal for large, distinct sections.
* By Component: Splits chunks for specific components . They are useful for rarely used or heavy UI elements inside a page.

**6. What happens if a dynamically imported route component fails to load? How would you handle that?**  
If it fails (e.g., network error), React will throw an error inside Suspense. You can wrap routes in an Error Boundary to show a fallback UI and retry.

**7. How does Webpack handle chunk naming in dynamic imports?**  
You can use a magic comment:

lazy(() => import(/\* webpackChunkName: "about" \*/ './pages/About'));

**8. What is the default chunk naming strategy if you don’t specify webpackChunkName?**  
Webpack assigns automatically generated IDs like 0.js, 1.js, or [hash].js based on build order.

**9. How would you lazy-load multiple components for the same route?**  
You can load them in parallel using Promise.all inside lazy:

const [CompA, CompB] = await Promise.all([

import('./CompA'),

import('./CompB')

]);

**10. Can you apply code splitting in nested routes? If yes, how?**  
Yes — lazy-load each nested route component just like top-level routes. Each nested <Route> can have its own lazy() import.

**Webpack Bundle Analyzer (10)**

**11. What is Webpack Bundle Analyzer used for in React development?**  
It visualizes your Webpack output as an interactive treemap to identify large dependencies and optimize bundle size.

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

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

In webpack.config.js:

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

module.exports = {

plugins: [new BundleAnalyzerPlugin()]

};

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

* Bundle size per file/dependency
* Unused code
* Duplicate libraries
* Which files contribute most to bundle size

**14. How do you identify large dependencies in your Webpack bundle?**  
Open the analyzer report → look for large rectangles in the treemap → hover to see their size.

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

* Code splitting
* Tree shaking
* Replacing heavy libraries with smaller alternatives
* Lazy loading
* Removing unused dependencies

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

* Dev: Run Webpack with mode: 'development' and the plugin enabled.
* Prod: Run webpack --mode production for optimized results.

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

* Static: Generates an HTML file report
* Server: Starts a web server to view the report live

**18. How do you exclude certain packages from Webpack’s analysis?**  
Use the excludeAssets option in plugin config:

new BundleAnalyzerPlugin({ excludeAssets: /vendor.\*\.js/ })

**19. How can tree shaking help after analyzing your Webpack bundle?**  
It removes unused exports from ES modules, reducing bundle size without affecting functionality.

**20. How does Webpack’s splitChunks configuration interact with bundle analysis results?**  
splitChunks can group shared dependencies into separate files, which the analyzer will display, helping visualize optimization effects.

**State Lifting (State Up) (10)**

**21. What does “lifting state up” mean in React?**  
Moving state from a child component to a common ancestor so multiple components can share and sync that state.

**22. Why might two sibling components need state to be lifted?**  
When they need to share data or coordinate behavior.

**23. How do you pass data from a child component to a parent when lifting state?**  
By passing a callback from the parent as a prop to the child, and calling it with the new value.

**24. What is the main drawback of lifting too much state?**  
It can cause unnecessary re-renders of unrelated components, hurting performance.

**25. How can lifting state up help prevent prop drilling?**  
By centralizing state in a common ancestor, fewer intermediate components need to pass it down.

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

function ParentForm() {

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

return <>

<NameInput value={formData.name} onChange={v => setFormData(p => ({ ...p, name: v }))} />

<EmailInput value={formData.email} onChange={v => setFormData(p => ({ ...p, email: v }))} />

</>;

}

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

* Use React.memo
* Use useCallback for event handlers
* Split state into smaller pieces

**28. How do you combine state lifting with context to avoid deep prop passing?**  
Lift state to a top-level provider and pass it through React.createContext, so consumers can access it anywhere.

**29. When lifting state, why might you use useCallback in the parent?**  
To memoize event handler functions so children don’t re-render unnecessarily when props change.

**30. How can you lift state without breaking controlled form elements?**  
Always pass both value and onChange from the parent to keep form fields controlled and synced with state.