**Day-21 Morning Assessment**

**Code Splitting by Route**

1. Code splitting is breaking a large bundle into smaller chunks that are loaded on demand.  
Benefit: reduces initial bundle size, improves first load performance, and loads only what’s needed.

2. import React, { Suspense, lazy } from "react";  
import { BrowserRouter, Route, Routes } from "react-router-dom";  
  
const Home = lazy(() => import("./Home"));  
const About = lazy(() => import("./About"));  
  
function App() {  
 return (  
<BrowserRouter>  
<Suspense fallback={<div>Loading...</div>}>  
<Routes>  
<Route path="/" element={<Home />} />  
<Route path="/about" element={<About />} />  
</Routes>  
</Suspense>  
</BrowserRouter>  
 );  
}

3. It defines what to render while the lazy-loaded component is being fetched (e.g., a loader/spinner).

4. Only the required chunk for the current route loads initially → smaller JS payload → faster first render.

5. By route: Loads each page lazily (best for SPAs with distinct pages).  
By component: Loads heavy/rarely used components lazily (e.g., charts, modals).

6. React will throw an error.  
Handle with ErrorBoundary or custom fallback to show error/retry UI.

7. By default, Webpack generates numeric IDs (0.js, 1.js) unless you specify a webpackChunkName.

8. Anonymous/numeric IDs → 0.js, 1.js, etc.

9. Use Promise.all inside lazy:  
  
const [Comp1, Comp2] = lazy(() =>   
 Promise.all([import("./Comp1"), import("./Comp2")]).then(  
   ([m1, m2]) => ({ default: () => (<><m1.default /><m2.default /></>) })  
 )  
);

10. Yes. Wrap nested <Route> elements with React.lazy() and <Suspense>. Each nested route loads separately.  
  
**Webpack Bundle Analyzer**

11. A tool to visualize bundle size, dependencies, and module composition.

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

13. -Bundle size breakdown  
-Which libraries/components are largest  
-Duplicate modules  
-Unused dependencies

14. Check the visualization chart — large colored blocks represent bigger dependencies.

15. -Code splitting  
-Lazy loading  
-Tree shaking  
-Replacing heavy libraries with lighter alternatives

16.Development: run analyzer with npm run start --analyze (if configured).  
Production: analyze minified, optimized build using npm run build --analyze.

17. Static: Generates an HTML file report.  
Server: Starts a server to view analysis interactively in browser.

18. Use excludeAssets or analyzerMode: 'disabled' with filters in config.

19. Removes unused exports from dependencies (e.g., using only specific lodash functions instead of whole lib).

20. It helps group common code into shared chunks → reduces duplication → analyzer shows smaller, optimized bundles.  
  
**State Lifting (State Up)**

21. Moving state from a child component up to a common parent so multiple components can share it.

22. When they need to share data.

23. Child calls a callback function passed as a prop by parent.  
<Child onChange={(val) => setParentState(val)} />

24. Causes unnecessary re-renders and makes parent components complex.

25. State is stored higher, so siblings can directly access it instead of passing props deeply.

26. Parent keeps form state (name, email, password) → inputs are controlled via props.

27. -Use React.memo on child components  
-Use useCallback for functions passed as props

28. Lift state to a common parent → wrap children with Context.Provider → access via useContext.

29. Prevents re-creation of callback functions → avoids triggering child re-renders.

30. Keep inputs controlled by parent state and pass value + onChange from parent to child.