1. Summarize the benefits of using design patterns in frontend development.

Design patterns act as "blueprints" for solving recurring problems. In the frontend, their benefits include:

- **Maintainability:** Decoupling logic (like API calls) from the UI makes it easier to update one without breaking the other.
- **Reusability:** Patterns like "Hooks" or "HOCs" allow you to share logic across multiple components, reducing code duplication (DRY principle).
- **Predictability:** Standardized patterns make the codebase easier for new team members to navigate.
- **Testability:** By isolating concerns (e.g., separating business logic from view), you can write more effective unit tests.

2. Classify the difference between global state and local state in React.

Feature	**Local State**	**Global State**
Scope	Restricted to a single component or its children.	Accessible by any component in the application.
Tooling	useState, useReducer.	Redux, Context API, Zustand, MobX
Use Case	Form inputs, toggle switches, hover states.	User authentication, theme settings, shopping carts.
Complexity	Low; easy to manage.	Higher; requires boilerplate and "store" management.

3. Compare different routing strategies in Single Page Applications (client-side, server-side, and hybrid) and analyze the trade-offs and suitable use cases for each.

Client-Side Routing (CSR)

The browser manages URL changes via the History API without reloading the page.

- **Trade-offs:** Fast transitions, but slow initial load (large JS bundles).
- **Use Case:** Highly interactive dashboards.

Server-Side Routing (SSR)

Each route change triggers a full page request to the server, which returns rendered HTML.

- **Trade-offs:** Great for SEO and initial load speed, but feels "clunky" during navigation.
- **Use Case:** Content-heavy sites (blogs, e-commerce).

Hybrid Routing (Universal/Isomorphic)

Initial load is handled by the server, but subsequent navigations are handled by the client (e.g., Next.js).

- **Trade-offs:** Best of both worlds, but increases architectural complexity.
- **Use Case:** Modern SaaS platforms.

4. Examine common component design patterns such as Container–Presentational, Higher-Order Components, and Render Props, and identify appropriate use cases for each pattern.

- **Container/Presentational:** Separates *how things work* (Container) from *how things look* (Presentational).
  - *Use Case:* When you want to reuse a UI layout with different data sources.
- **Higher-Order Components (HOC):** A function that takes a component and returns a new component with injected props.
  - *Use Case:* Cross-cutting concerns like withAuth or withLogging.
- **Render Props:** A technique for sharing code between components using a prop whose value is a function.
  - *Use Case:* Logic that needs to be highly dynamic, like a mouse-tracker or data-fetcher.

5. Demonstrate and develop a responsive navigation bar using Material UI components while applying appropriate styling and breakpoint configurations.

Using MUI's AppBar and Toolbar with Box for breakpoint-based display logic.

```
import { AppBar, Toolbar, Typography, Button, IconButton, Box } from
'@mui/material';
import MenuIcon from '@mui/icons-material/Menu';

const Navbar = () => {
  return (
    <AppBar position="static">
      <Toolbar>
        <Typography variant="h6" sx={{ flexGrow: 1 }}>CollabTool</Typography>

        <Box sx={{ display: { xs: 'none', md: 'block' } }}>
          <Button color="inherit">Dashboard</Button>
          <Button color="inherit">Projects</Button>
        </Box>

        <Box sx={{ display: { xs: 'block', md: 'none' } }}>
          <IconButton color="inherit"><MenuIcon /></IconButton>
        </Box>
      </Toolbar>
    </AppBar>
  );
};
```

6. Evaluate and design a complete frontend architecture for a collaborative project management tool with real-time updates. Include:

a) SPA structure with nested routing and protected routes

- **Nested Routing:** Use react-router-dom to nest project-specific views (e.g., /project/:id/board, /project/:id/settings).
- **Protected Routes:** A wrapper component that checks for an auth token in the state; if null, redirects to /login.

b) Global state management using Redux Toolkit with middleware

- **Slices:** Create slices for auth, projects, and tasks.
- **Middleware:** Use **Redux Thunk** (default in RTK) for async API calls.
- **Real-time Integration:** Use a middleware to handle **WebSockets (Socket.io)**. When a server message arrives, the middleware dispatches a Redux action to update the UI instantly.

c) Responsive UI design using Material UI with custom theming

- Use ThemeProvider to define a custom palette (e.g., deep purples for "brand" identity).
- Apply Grid and Stack for layout consistency.

d) Performance optimization techniques for large datasets

- **Virtualization:** Use react-window for large task lists to render only visible items.
- **Memoization:** Wrap heavy components in React.memo and use useMemo for filtering large datasets.
- **Lazy Loading:** Code-split routes using React.lazy.

e) Analyze scalability and recommend improvements for multi-user concurrent access.

- **Optimistic Updates:** Update the UI immediately when a user moves a task, then sync with the server. If the server fails, roll back the state.
- **Conflict Resolution:** Implement "Last Write Wins" or CRDTs (Conflict-free Replicated Data Types) for text editing.
- **Recommendation:** Move to a micro-frontend architecture if the team grows, allowing different squads to own "Boards," "Chat," and "Analytics" independently.