

FULL STACK ASSIGNMENT - 01

Name- Rakshit Jaryal

UID - 23BAI70103

Class – 23AML-3A

Date- 5TH FEB 2026

1) Benefits of using design patterns in frontend development

Design patterns are standard and reusable solutions to common problems in software design. In frontend development, they help developers create well-structured, scalable, and maintainable applications. One major benefit of design patterns is improved code readability. When developers follow well-known patterns, other developers can easily understand the code structure. This is especially helpful in team-based projects.

Another benefit is maintainability. Design patterns create a consistent structure, making it easier to debug, modify, or extend the application without affecting other parts of the system.

Design patterns also promote reusability. Components and logic can be reused across different parts of the application, reducing development time and effort.

They also support scalability. As the application grows, patterns help manage complexity and keep the architecture organized.

Overall, design patterns encourage best practices, reduce errors, and improve the overall quality of frontend applications.

2) Difference between global state and local state in React

In React, state refers to data that controls the behaviour and appearance of components. There are two main types of state: local state and global state.

Local state is managed within a single component using hooks like useState. It is only accessible within that component. It is commonly used for UI-related data such as form inputs, toggles, dropdowns, and modal visibility. Local state keeps components independent and simple to manage.

Global state, on the other hand, is shared across multiple components. It is used when different parts of the application need access to the same data. Examples include user authentication, theme settings, notifications, or shopping cart data.

Global state is usually managed using tools like the Context API, Redux, or Zustand. These tools provide centralized state management and make it easier to handle complex applications. In summary, local state is best for component-specific data, while global state is used for application-wide data shared across multiple components.

3) Comparison of routing strategies in Single Page Applications (SPAs)

Routing is the process of navigating between different pages or views in an application. There are three main routing strategies: client-side routing, server-side routing, and hybrid routing. Client-side routing is handled in the browser using JavaScript libraries like React Router. When a user clicks a link, the page does not reload. Instead, JavaScript updates the content dynamically. This results in faster navigation and a smoother user experience. However, it may have SEO limitations and slower initial load times.

Server-side routing is handled by the server. Every time a user navigates to a new page, the browser sends a request to the server, which returns a new HTML page. This approach is better for search engine optimization (SEO) and initial page loading, but navigation can feel slower because the page reloads each time. Hybrid routing combines both approaches. Frameworks like Next.js use server-side rendering for the first page load and client-side routing for later navigation. This provides better SEO, faster performance, and a smoother user

experience. Each routing strategy has its own advantages and trade-offs, and the choice depends on the requirements of the project.

4) Common component design patterns

Frontend development uses several component design patterns to improve code organization and reusability.

The Container–Presentational pattern separates logic from the user interface. The container component handles data fetching, state management, and business logic. The presentational component focuses only on displaying the UI. This separation improves readability and reusability.

Higher-Order Components (HOCs) are functions that take a component and return a new component with additional functionality. They are commonly used for cross-cutting concerns such as authentication, logging, or permissions.

Render Props is another pattern where a component shares logic by passing a function as a prop. The child component uses that function to decide how to render the UI. This approach provides more flexibility compared to HOCs.

These patterns help developers create modular, reusable, and maintainable components in large applications.

5) Responsive navigation bar using Material UI

The following React component demonstrates a responsive navigation bar using Material UI with breakpoints and a mobile drawer.

```
import React, { useState } from "react";
import {
  AppBar,
  Toolbar,
  IconButton,
  Typography,
```

```
Drawer,
List,
ListItem,
ListItemText,
Button,
Box,
useTheme,
useMediaQuery
} from "@mui/material";
import MenuItem from "@mui/icons-material/Menu";

function Navbar() {
  const [open, setOpen] = useState(false);
  const theme = useTheme();
  const isMobile = useMediaQuery(theme.breakpoints.down("md"));

  const menuItems = ["Home", "Projects", "Tasks", "Contact"];

  return (
    <>
    <AppBar position="static">
      <Toolbar>
        {isMobile && (
          <IconButton
            color="inherit"
            edge="start"
            onClick={() => setOpen(!open)}
          >
            {open ? <CloseIcon /> : <MenuIcon />}
          </IconButton>
        )}
        <Typography variant="h6" style={{ margin: 0 }}>Material UI</Typography>
        <List style={{ listStyleType: "none", padding: 0 }}>
          {menuItems.map((item) => (
            <MenuItem key={item}>{item}</MenuItem>
          ))}
        </List>
      </Toolbar>
    </AppBar>
  );
}

export default Navbar;
```

```
    onClick={() => setOpen(true)}  
    >  
    <Menulcon />  
  </IconButton>  
})}  
  
<Typography variant="h6" sx={{ flexGrow: 1 }}>  
  My App  
</Typography>  
  
{!isMobile &&  
  menuItems.map((item) => (  
    <Button key={item} color="inherit">  
      {item}  
    </Button>  
  ))}  
</Toolbar>  
</AppBar>  
  
<Drawer open={open} onClose={() => setOpen(false)}>  
  <Box sx={{ width: 250 }}>  
    <List>  
      {menuItems.map((item) => (  
        <ListItem button key={item}>  
          <ListItemText primary={item} />  
        </ListItem>  
      ))}  
    </List>  
  </Box>  
</Drawer>
```

```
    ))}
  </List>
</Box>
<Drawer>
</>
);
}
```

```
export default Navbar;
```

6) Frontend architecture for a collaborative project management tool

Designing a frontend architecture for a collaborative project management tool requires a scalable, maintainable, and responsive structure.

a) SPA structure:

The application can be built using React as a Single Page Application. It should have nested routes for sections like Dashboard, Projects, Tasks, Teams, and Settings. Protected routes ensure only authenticated users can access private areas.

b) Global state management:

Redux Toolkit can be used to manage shared data such as user profiles, projects, tasks, and notifications. Middleware like Redux Thunk or Redux Saga can handle asynchronous operations and API calls.

c) Responsive UI:

Material UI can be used to build a responsive interface. Components like Grid, Box, and Stack, along with breakpoints, allow layouts to adjust for mobile, tablet, and desktop screens. A custom theme ensures consistent styling.

d) Performance optimization:

Techniques like code splitting, lazy loading, memorization, and virtualization for large lists can improve performance. These techniques reduce load time and improve the user experience.

e) Scalability:

Real-time updates can be implemented using Web Sockets or services like Firebase. Backend services can scale horizontally to support more users.

Caching and rate limiting can improve performance. A modular architecture ensures the system can grow smoothly as the number of users increases.