Name: Uday Kapila

UID: 23BAI70456

Sec: 23AML-2-A

Subject: Full Stack Assignment 1

Q1) Discuss the primary advantages of implementing design patterns in modern frontend engineering.

Answer: Design patterns serve as architectural blueprints for solving common software problems. In the context of frontend engineering, their advantages are significant:

- **Standardized Vocabulary:** They provide a universal language. When developers talk about a "Singleton" or "Factory," everyone understands the underlying structure immediately, speeding up collaboration.
- **Code Decoupling:** Patterns help in loosely coupling modules. For instance, using the *Observer pattern* allows different parts of the UI to react to data changes without being directly connected, making the system more flexible.
- **Ease of Maintenance:** By organizing code into predictable structures (like the *Module pattern*), it becomes much easier for new developers to onboard and for existing teams to debug issues without side effects.
- **Proven Reliability:** Since these patterns are industry-tested solutions, using them reduces the risk of architectural failure compared to ad-hoc, experimental solutions.

Q2) Differentiate between Local State and Global State in the context of a React application.

Answer:

1. Local State:

- **Scope:** Confined to a single component or passed down to immediate children.
- **Management:** Handled natively using hooks like useState or useReducer.
- **Lifespan:** The data exists only as long as the component is mounted on the screen.
- **Example:** Capturing text in an input field, managing the open/close status of a dropdown menu, or tracking a simple counter.

2. Global State:

- **Scope:** Accessible by many components across different parts of the application tree, regardless of hierarchy.
- **Management:** Often requires external libraries like Redux, Zustand, or the built-in Context API.
- **Lifespan:** Persists across route changes and component unmounts; often synchronized with local storage or server data.
- **Example:** Storing the current user's authentication token, application-wide theme preferences (Dark Mode), or a shopping cart object.

Q3) Analyze the various routing techniques available for Single Page Applications (SPA) and discuss their trade-offs.

Answer:

- **Client-Side Routing (CSR):**
    - *How it works:* The browser downloads a minimal HTML page and a large JavaScript bundle. The JS handles rendering views as the URL changes.
    - *Pros/Cons:* Results in a very "app-like" fluid feel after the initial load. However, the initial load time is slower, and search engine crawling can be challenging without configuration.
    - *Best for:* Gated applications (dashboards, admin panels) where SEO matters less.
- **Server-Side Routing (SSR):**
    - *How it works:* The server renders the complete HTML for every unique URL request and sends it to the client.
    - *Pros/Cons:* excellent for SEO and faster initial content visibility. However, it puts more stress on the server and page transitions are not as instant as CSR.
    - *Best for:* Public-facing websites, news aggregators, and e-commerce stores.
- **Hybrid (SSG & ISR):**
    - *How it works:* Pages are pre-built at build time (Static Site Generation) and updated periodically (Incremental Static Regeneration).
    - *Pros/Cons:* Offers the best of both worlds—speed of static files with the freshness of dynamic content. Can be complex to set up.
    - *Best for:* Blogs, marketing sites, and documentation.

Q4) Explain the usage of Container–Presentational, HOCs, and Render Props patterns with examples.

Answer:

1. **Container vs. Presentational:**
    - *Logic:* Separation of concerns. "Smart" components (Containers) handle logic and fetching, while "Dumb" components (Presentational) handle the look and feel.
    - *Example:* A <StockTickerContainer /> fetches the price from an API, while <StockView /> just receives the price as a prop and renders it green or red.
2. **Higher-Order Components (HOC):**
    - *Logic:* A wrapper function that takes a component and returns a new, enhanced component. It allows for logic reuse.
    - *Example:* withLoading(Component)—a wrapper that displays a spinner while data is fetching, and only renders the actual component once data is ready.
3. **Render Props:**
    - *Logic:* Sharing code between components using a prop whose value is a function.
    - *Example:* An <InputValidator render={(isValid) => <SubmitButton disabled={!isValid} />} />. The logic of validation is inside the parent, but the UI is determined by the render function.

Q5) Build a responsive header component using Material UI that adapts to mobile and desktop screens.

Answer: Below is a React component named AppHeader. It utilizes the sx prop for styling and renders a Drawer for mobile viewports.

JavaScript

```
import React, { useState } from 'react';
import {
  AppBar, Toolbar, Typography, Box, Button, IconButton,
  Drawer, List, ListItem, ListItemText, useTheme, useMediaQuery
} from '@mui/material';
import MenuIcon from '@mui/icons-material/Menu';

const AppHeader = () => {
  const [mobileOpen, setMobileOpen] = useState(false);
```

```
const theme = useTheme();
const isMobile = useMediaQuery(theme.breakpoints.down('md'));

const navLinks = ['Home', 'Features', 'Pricing', 'Contact'];

const toggleDrawer = (open) => () => {
  setMobileOpen(open);
};

return (
  <Box sx={{ flexGrow: 1 }}>
    <AppBar position="static" sx={{ bgcolor: '#1976d2' }}>
      <Toolbar>
        <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>
          TaskMaster
        </Typography>

        {/* Desktop View */}
        {!isMobile && (
          <Box>
            {navLinks.map((text) => (
              <Button key={text} color="inherit" sx={{ mx: 1 }}>
                {text}
              </Button>
            ))}
          </Box>
        )}

        {/* Mobile Hamburger */}
        {isMobile && (
          <IconButton edge="start" color="inherit" onClick={toggleDrawer(true)}>
```

```
            <MenuIcon />
          </IconButton>
        )}
      </Toolbar>
    </AppBar>

    {/* Mobile Sidebar */}
    <Drawer anchor="right" open={mobileOpen} onClose={toggleDrawer(false)}>
      <Box sx={{ width: 250 }} role="presentation" onClick={toggleDrawer(false)}>
        <List>
          {navLinks.map((text) => (
            <ListItem button key={text}>
              <ListItemText primary={text} />
            </ListItem>
          ))}
        </List>
      </Box>
    </Drawer>
  </Box>
 );
};

export default AppHeader;
```

Q6) Propose a comprehensive frontend architecture for a real-time collaborative project management application.

Answer:

A) Routing & SPA Structure:

- **Framework:** React with react-router-dom.
- **Routing Logic:** Implement "Nested Routes" to manage complex UI states (e.g., /workspace/:id/board).
- **Security:** Use a ProtectedRoute wrapper that intercepts navigation. If the Redux auth.isAuthenticated state is false, redirect users to the Login page immediately.

B) State Management (Redux Toolkit):

- **Structure:**
    - tasksSlice: Stores the heavy task data.
    - usersSlice: Stores collaborator details.
    - presenceSlice: Tracks who is currently viewing the board (for real-time avatars).
- **Middleware:** Utilize **RTK Query** for caching API responses. This reduces network requests by serving cached data while re-fetching in the background (stale-while-revalidate).

C) UI & Theming:

- **Library:** Material UI (MUI).
- **Customization:** Define a central theme.js file. This allows us to change the primary brand color or font globally in one line.
- **Responsiveness:** Use MUI's Grid system to switch from a 4-column Kanban board on desktop to a single-column vertical list on mobile.

D) Performance Strategies:

- **Code Splitting:** Use React.lazy to load the "Reports" and "Settings" modules only when the user clicks them.
- **Windowing:** Use react-window for the task lists. If a user has 500 tasks, we only render the 10 visible on screen to save memory.
- **Debouncing:** Debounce search inputs and auto-save functions to prevent spamming the server with requests.

E) Real-Time Scalability:

- **WebSockets:** Use Socket.io for bi-directional communication.
- **Conflict Resolution:** Implement "Last-Write-Wins" or Operational Transformation (OT) logic to handle two users editing the same text.
- **Optimistic Updates:** When a user drags a card, move it instantly in the UI. Do not wait for the server response. If the server returns an error later, snap the card back. This ensures the app feels "native" and fast.