Assignment 1

Name: Shreeyansh Kumar

UID: 23BAI70152

Section: 23AML-2(A)

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

A 1. Design patterns provide reusable solutions to common design problems.

Benefits:

1. Reusability

- Patterns allow reuse of proven solutions.
- Reduces duplicate code.

2. Maintainability

- Code becomes modular and structured.
- Easier to debug and refactor.

3. Scalability

- Applications grow without becoming messy.
- Clear separation of responsibilities.

4. Readability

- Developers understand structure quickly.
- Easier onboarding in teams.

5. Reduced Development Time

- Avoid reinventing solutions.
- Faster feature implementation.

6. Better Collaboration

- Standardized architecture improves teamwork.

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

A 2.

Feature	**Local State**	**Global State**
Scope	Specific to one component	Shared across multiple components
Managed By	useState, useReducer	Redux, Context API, Zustand
Access	Inside component only	Accessible anywhere in app
Use Case	Form inputs, UI toggles	Authentication, user data, cart
Performance	Faster updates (limited scope	Can cause re-renders if not optimize

Q 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.

A 3. **1. Client-Side Routing**

Handled entirely in browser (React Router).

Pros:

- Fast navigation
- No full page reload
- Smooth UX

Cons:

- SEO issues (without SSR)
- Initial bundle size larger

Best For:

- Dashboards
- Admin panels
- SaaS apps

2. Server-Side Routing (Traditional)

Each request loads new page from server.

Pros:

- Better SEO
- Smaller JS bundle

Cons:

- Slower navigation
- Full reload each time

Best For:

- Blogs
- Marketing websites

3. Hybrid Routing (Next.js style)

Mix of CSR + SSR + SSG.

Pros:

- SEO optimized
- Fast UX
- Flexible rendering

Cons:

- More complex architecture

Best For:

- Large-scale SaaS
- E-commerce
- Public + Private dashboards

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

A 4. **1. Container–Presentational Pattern**

Container:

- Handles logic & state
- Fetches data

Presentational:

- Handles UI only
- Receives props

Use Case:

- Dashboard layout
- Data-heavy UI components

2. Higher-Order Components (HOC)

Function that takes a component and returns enhanced component.

```
withAuth(Component)
```

Use Case:

- Authentication wrappers
- Role-based access
- Logging

3. Render Props Pattern

Component shares logic using a function as child.

```
<DataFetcher render={(data) => <UI data={data} />} />
```

Use Case:

- Sharing reusable logic
- Animations
- Data fetching abstraction

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

A 5. **Features:**

- AppBar
- Toolbar
- Drawer (for mobile)
- Breakpoints

```
import React from "react";
import {
  AppBar,
  Toolbar,
  Typography,
  IconButton,
  Button,
  Drawer,
  Box,
  useMediaQuery
```

```
} from "@mui/material";
import MenuIcon from "@mui/icons-material/Menu";
import { useTheme } from "@mui/material/styles";

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

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

        {isMobile ? (
          <>
            <IconButton color="inherit" onClick={() =>
setOpen(true)}>
              <MenuIcon />
            </IconButton>

            <Drawer open={open} onClose={() =>
setOpen(false)}>
              <Box sx={{ width: 250, p: 2 }}>
```

```
                <Button fullWidth>Dashboard</Button>
                <Button fullWidth>Projects</Button>
                <Button fullWidth>Profile</Button>
              </Box>
            </Drawer>
          </>
        ) : (
          <>
            <Button color="inherit">Dashboard</Button>
            <Button color="inherit">Projects</Button>
            <Button color="inherit">Profile</Button>
          </>
        )}
      </Toolbar>
    </AppBar>
  );
}
```

Breakpoint Used:

```
theme.breakpoints.down("md")
```

Q 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 b) Global state management using Redux Toolkit with middleware c) Responsive UI design using Material UI with custom theming d) Performance optimization techniques for large datasets e) Analyze scalability and recommend improvements for multi-user concurrent access.

A 6. **a) SPA Structure**

```
src/
 ├── app/ (Redux store)
 ├── features/
 │     ├── auth/
 │     ├── projects/
 │     ├── tasks/
 ├── routes/
 ├── components/
 ├── layouts/
```

Routing Example:

```
<Route path="/dashboard" element={<ProtectedRoute />}>
   <Route path="projects" element={<Projects />} />
   <Route path="tasks/:id" element={<TaskDetails />} />
</Route>
```

ProtectedRoute checks authentication.

b) Global State (Redux Toolkit)

Store configuration:

```
import { configureStore } from "@reduxjs/toolkit";
import authReducer from "./features/authSlice";

export const store = configureStore({
  reducer: {
    auth: authReducer,
  },
});
```

Middleware:

- Thunk (default)
- Logger
- WebSocket middleware (for real-time updates)

c) Responsive UI + Custom Theme

```
const theme = createTheme({
  palette: {
    primary: { main: "#1976d2" },
    secondary: { main: "#ff4081" },
  },
});
```

Use:

```
<ThemeProvider theme={theme}>
```

d) Performance Optimization (Large Data)

Techniques:

- React.memo
- useMemo
- useCallback
- Virtualization (react-window)
- Lazy loading
- Code splitting
- Pagination

e) Scalability for Multi-User Access

Real-time:

- WebSockets / Socket.io
- Optimistic UI updates

Improve Scalability:

- Normalize Redux state
- Use RTK Query for caching
- Debounce updates
- Role-based rendering
- Sharding projects
- Load data on demand