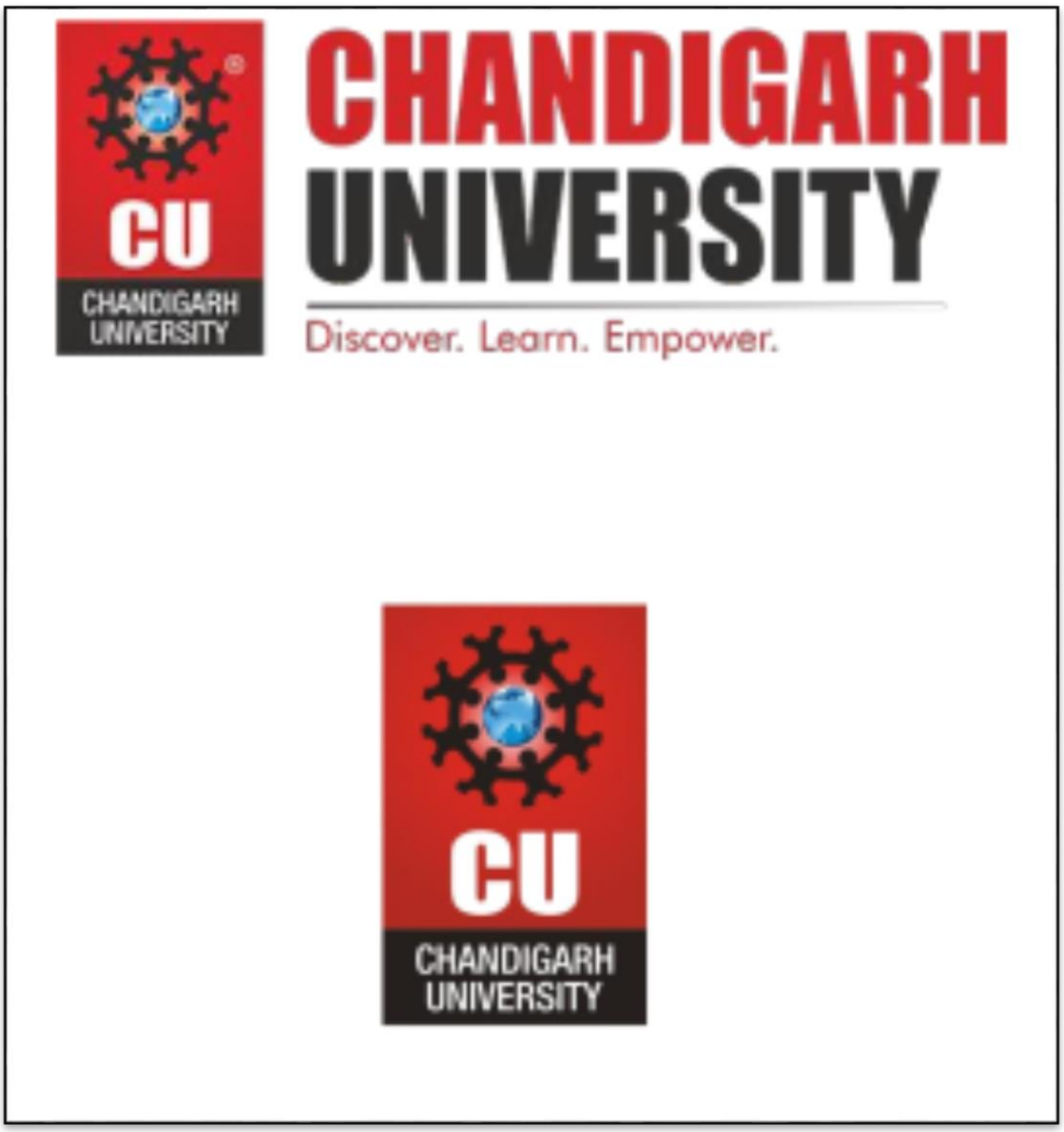


Subject : Full Stack
Subject Code : 23CSH-382

Student Name: Lakshay Goel

UID : 23BAI70088

Branch : BE CSE AIML

Section/Group : 23AML-2A

ASSIGNMENT 1

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

Ans 1) Design patterns are proven approaches that developers use to solve problems that appear again and again in UI development. In frontend projects, using design patterns gives multiple advantages:

- Easy Maintenance: Patterns such as Atomic Design or separating components into different roles make the UI easier to debug and update without affecting unrelated parts.
- Better Scalability: When the application becomes large, patterns keep the structure clean and prevent the code from becoming messy and difficult to manage.
- High Reusability: Design patterns encourage modular and reusable components. This reduces duplicate code and saves development time.
- Improved Team Collaboration: Since patterns are widely understood, they create a common language. If someone says "Provider pattern is used," the team instantly understands the approach without long explanations.

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

Local State

Local state is the data that belongs to a single component. It is managed inside that component using hooks like useState or useReducer.
This type of state is limited in scope and can only be passed to child components through props.

Global State

Global state is the data that needs to be shared across multiple components, even if they are not directly related.
To manage global state, tools like Redux, Context API, Zustand, Recoil, etc. are used.
This state can be accessed from different parts of the component tree without passing props repeatedly.

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.

Routing in SPAs can be handled in different ways depending on the type of application. The main routing strategies are client-side, server-side, and hybrid routing.

Routing Type	**How it Works**	**Advantages**	**Best Use Cases**
Client-Side Routing	Navigation is managed inside the browser using JavaScript	Very fast transitions, smooth user experience	Web apps with heavy interaction (dashboards, social apps)
Server-Side Routing	Each route loads a new page from the server	Strong SEO support, easier for content indexing	Blogs, news sites, content-based platforms
Hybrid Routing	Mix of server-rendered pages + client-side navigation	Good balance of performance and SEO	Large modern apps (e-commerce, SaaS platforms)

Trade-offs (Quick Analysis)

- Client-side is fastest, but SEO can be weaker.

- Server-side is best for SEO, but slower navigation.
- Hybrid gives both benefits but requires more setup and planning

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

Modern React applications use component design patterns to improve reusability, separation of concerns, and maintainability. Some commonly used patterns are:

a) Container–Presentational Pattern
This pattern separates logic from UI. Container components handle data fetching, state management, and business logic. Presentational components focus only on rendering UI using props.

Use cases:
- Large applications where UI and logic need to be clearly separated
- Improves readability, testing, and reuse of UI components

Example:
A container fetches user data, while a presentational component displays the user profile.

b) Higher-Order Components (HOC)
A Higher-Order Component is a function that takes a component and returns an enhanced component. Used to reuse logic across multiple components. Commonly used for authentication, logging, or data fetching

Use cases:
- Cross-cutting concerns like authorization or analytics
- When multiple components need the same behavior

Example:
Wrapping components with an withAuth() HOC to restrict access.

c) Render Props Pattern
This pattern involves passing a function as a prop that determines what to render. Allows sharing logic while keeping rendering flexible. Avoids deep inheritance or wrapper components

Use cases:
- When components need dynamic UI rendering
- Useful for handling mouse tracking, animations, or conditional layouts

Example:
A component that provides mouse position and lets the parent decide how to display it.

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

The navigation bar is developed with Material UI components to ensure a seamless user experience across all devices.

Key Components & Logic

Structure: Built using AppBar, Toolbar, and Box for layout.

Responsive Breakpoints: Utilizes theme.breakpoints to toggle visibility. Elements use the sx prop (e.g., display: { xs: 'none', md: 'flex' }) to hide/show desktop links versus a mobile hamburger menu.

Mobile Navigation: Implements a Drawer (sidebar) triggered by an IconButton (MenuIcon) for touch-friendly targets (minimum 48px).

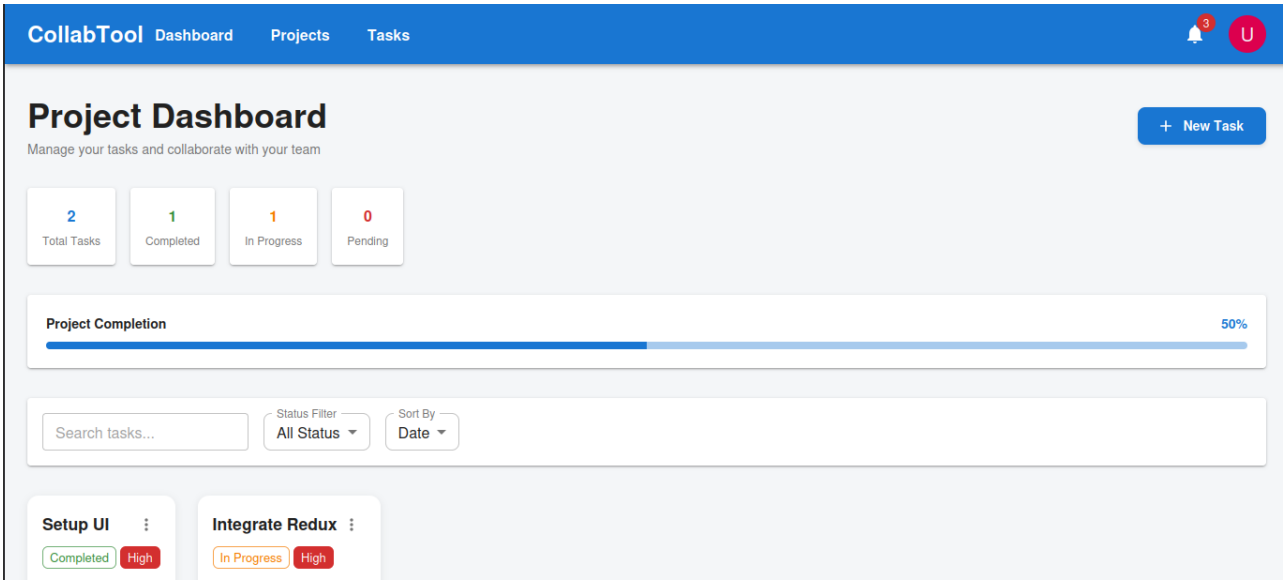
Interactive Elements: Includes a Badge for real-time notifications and an Avatar with a dropdown Menu for profile settings.

Visual Design & Accessibility

Styling: Professional blue primary palette (#1976d2) with white contrasting text.

Accessibility: Semantic HTML buttons, ARIA labels for icons, and WCAG AA color contrast compliance.

Screenshots :

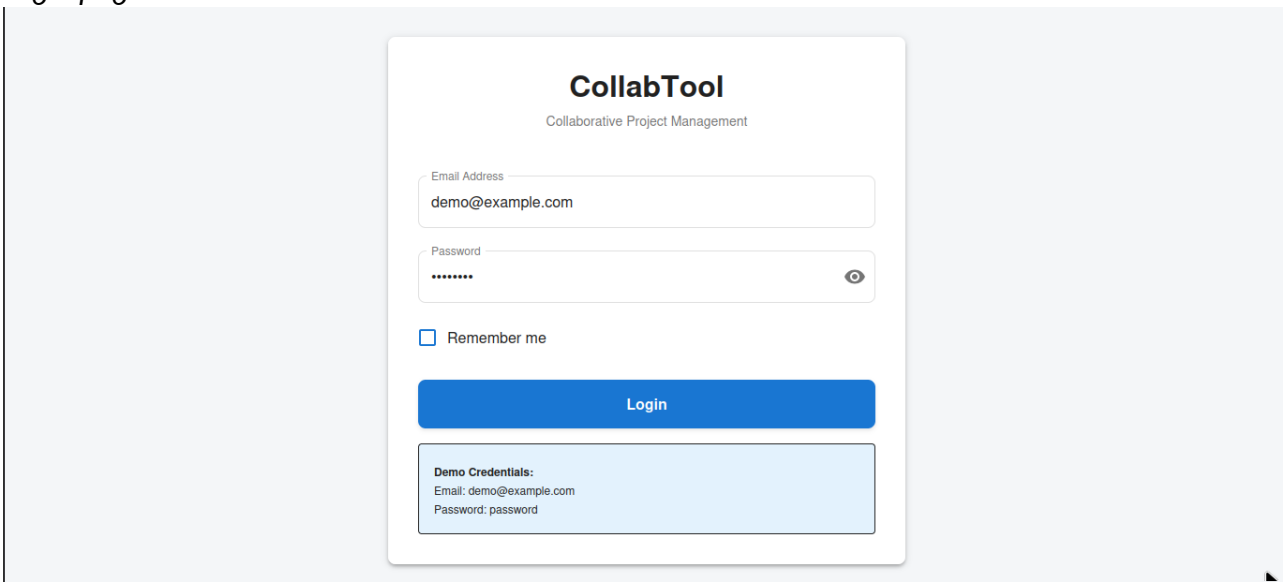


6) Evaluate and design a complete frontend architecture for a collaborative project management tool with real-time updates. Include: a) SPA structure with nested routing and protected routes b)

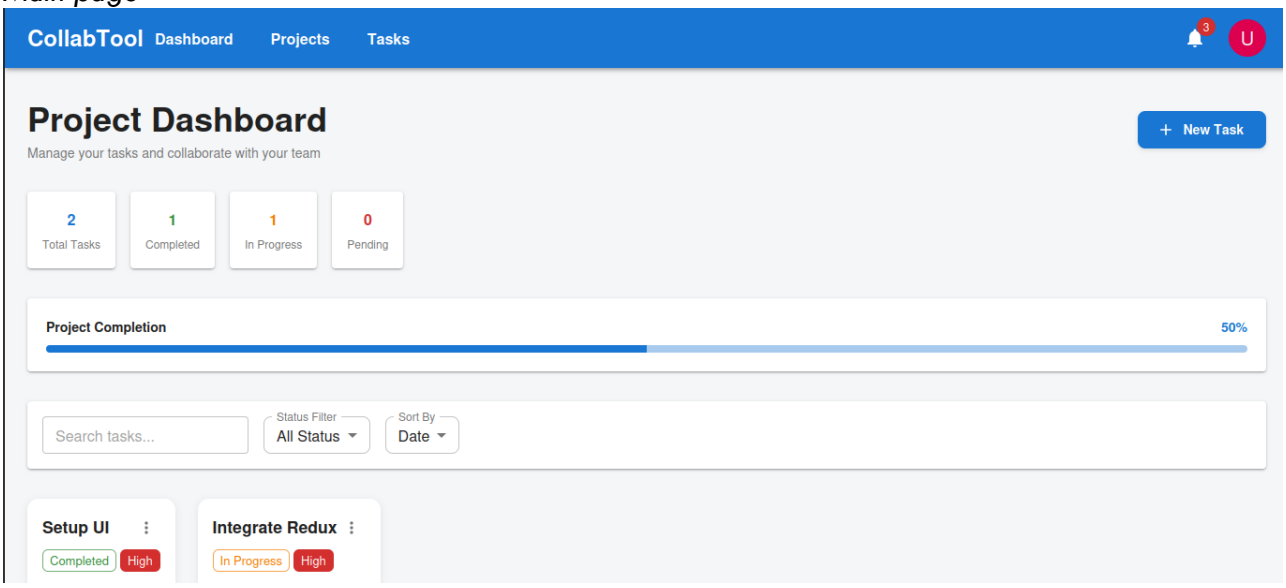
Global state management using Redux Toolkit with middleware c) Responsive UI design using Material UI with custom theming d) Performance optimization techniques for large datasets e) Analyze scalability and recommend improvements for multi-user concurrent access.

Screenshots:

Login page



Main page



a) SPA Structure, Nested Routing, and Protected Routes

Routing: Utilizes React Router v6 for client-side navigation without page reloads.

Hierarchy: Implements nested routes for scalability (e.g., /dashboard, /projects/:id/tasks).

Security: A ProtectedRoute wrapper checks for an authToken in localStorage. Unauthorized users are redirected to /login using the <Navigate /> component.

b) Global State Management (Redux Toolkit)

Centralized Store: Configured with configureStore to manage tasks, auth, and notifications.

Logic: Uses createSlice for synchronous reducers (e.g., addTask, deleteTask) and createAsyncThunk for handling asynchronous API lifecycles (pending, fulfilled, rejected).

Middleware: Custom middleware is integrated to intercept actions for logging and future WebSocket integration.

c) Responsive UI Design & Custom Theming

Theming: A global createTheme configuration defines the color palette, typography, and component overrides (e.g., 8px border-radius for buttons and cards).

Fluidity: Employs responsiveFontSizes to automatically scale text from desktop (2.5rem) to mobile (1.8rem).

Grid Layout: Uses MUI Grid with xs={12} (mobile) and md={4} (desktop) for a flexible task board.

d) Performance Optimization for Large Datasets

Memoization: Implements useMemo for heavy filtering/sorting and React.memo for the TaskCard component to prevent unnecessary re-renders.

Virtualization: For datasets exceeding 1,000+ items, react-window is recommended to render only visible DOM nodes, reducing memory usage by up to 95%.

User Experience: Employs Skeleton loaders during data fetching to improve perceived performance.

e) Scalability Analysis & Multi-User Recommendations

Real-time Synchronization: Recommends moving from HTTP polling to WebSockets (Socket.io) for sub-100ms latency updates.

Concurrency Control: Update the UI immediately and revert only if the server fails.

Conflict Resolution: Use Last-Modified Timestamps or Operational Transformation (OT) for collaborative text editing.

Infrastructure: Implement horizontal scaling with a Load Balancer and Redis caching to handle up to 10,000+ concurrent users.