

**Assignment
of
Full stack-II {23CSH-382}**

Submitted in the partial fulfilment for the award of the degree of

**BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE WITH SPECIALIZATION IN
ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**

Submitted by:

23BAI70368 Vansh Soin

Under the Supervision of:

Er. Akash Mahadev Patil



**CHANDIGARH
UNIVERSITY**
Discover. Learn. Empower.

**CHANDIGARH UNIVERSITY,
GHARUAN, MOHALI - 140413,
PUNJAB**

4 February , 2026

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

Design patterns in frontend development provide developers with standardized methods to solve common challenges which arise during development of user interface components and their corresponding state management and application performance and maintainability. Modern frontend frameworks, such as React, require developers to use design patterns because they help manage the increasing complexity of their applications.

i) Improved Code Organization

- Design patterns create a structured system which requires:
- The UI must exist separately from its underlying business logic
- The system should have a defined file structure together with an expected file naming system
- The system should enable components to operate independently without any direct contact between them
- The system prevents developers from creating tightly connected code which leads to system breakdowns when developers modify a single component.

ii) Reusability and DRY Principle

Patterns enable:

- The reuse of components
- The development of shared logic components
- The prevention of writing duplicate code
- The authentication logic which developers build through a Higher-Order Component can be reused across various routes.

iii) Maintainability and Readability

- The system achieves code readability through pattern implementation which enables developers to create self-documenting code.
- The system allows new developers to study its entire framework while becoming familiar with all system elements.
- The system enables faster error detection and solution.
- All team members gain advantages because they share an identical understanding of the project.

iv) Scalability

- Design patterns enable application scalability through:
- The system enables developers to build applications using multiple separate modules
- The system prevents developers from creating large monolithic software systems
- The system enables developers to create application features through different architectural elements

- The system requires developers to create patterns its unmanageable technical debt will grow uncontrollably.

v) Testability

- Patterns define specific task areas:
- The UI system allows for independent testing
- Business logic requires unit testing
- Testing for side effects produces results which developers can foresee.
- This serves as a crucial requirement for applications which need to function at an enterprise level.

vi) Performance Optimization

- The implementation of memoization together with lazy loading and virtualization patterns becomes simpler when developers use proper code organization.

Hence, Design patterns serve as mandatory best practices which establish essential guidelines for creating systems which deliver reliable performance and capacity to grow and production-ready software.

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

The state of an application creates changing information which determines how the user interface will display content. React applications use two types of application state which developers define through their respective usage patterns.

i) Local State: A single component controls all functions of local state.

Characteristics

- The system uses useState together with useReducer for state management
- The system only exists for the time that the component operates
- The system achieves its updates at a faster speed
- The system needs only a small amount of resources

Examples

- Input field values
- Modal visibility
- Toggle buttons
- Dropdown selections

Advantages

- The system provides an easy implementation
- The system creates performance enhancements
- The system creates simpler design solutions

Limitations

- Sibling components cannot reach this state
- The state requires prop drilling for shared access between components

ii) Global State : Global state exists throughout an entire application which multiple components and routes can access.

Characteristics

- The system uses Context API together with Redux Toolkit for state management
- The system operates as a unified storage facility for all data
- The system provides predictable state behavior through defined state transition mechanisms

Examples

- Authentication status
- User profile data
- Theme preferences
- Notifications
- Project data

Advantages

- The system eliminates the need for prop drilling
- Data access remains consistent throughout the system
- The system enables easier debugging through developer tools

Limitations

- The system requires additional code for its basic functions
- The system produces a small performance decrease
- The system needs strict organizational methods to function properly
- The system requires local state as the first choice except for cases when data needs to be shared. Global state creates additional difficulties when developers use it excessively.

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 defines the path which users take to move through different screens of a web application.

i) Client-Side Routing : The browser performs all operations through its JavaScript capabilities.

How It Works

- The browser requests a single HTML document.
- JavaScript handles the process of resolving routes.
- The system operates without requiring any page refreshes.

Advantages

- The system enables users to move between different sections of the application without delay.
- The system provides users with a continuous experience without interruptions.
- The system decreases the amount of work that servers must perform.

Disadvantages

- The system presents challenges for search engine optimization.
- The system requires a substantial amount of JavaScript code to operate.

Use Cases

- Administrators use the system to monitor and control their networks from their central offices.
- The system enables users to access their internal company resources.
- Web applications require user authentication to access their protected areas.

ii) Server-Side Routing : The server processes each route request which it receives from users.

Advantages

- The system provides better search engine optimization results.
- The system enables faster display of initial content.
- The system provides easier operation for users through its simplified design.

Disadvantages

- The system requires more time to move between different sections of the website.
- The system requires additional resources to operate at maximum capacity.
- The system requires complete browser document refreshes to present new content.

Use Cases

- Blogs
- News portals
- Marketing websites

iii) Hybrid Routing : The system uses two routing methods which include SSR and CSR.

Advantages

- The system enables search engines to index its content.
- The system delivers outstanding performance results.
- The system supports user experience improvements through its basic system functions.

Disadvantages

- The system requires complete knowledge of its components.
- The system requires users to spend more time learning its features.

Use Cases

- E-commerce platforms
- SaaS products
- High-traffic websites

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 Components : The system needs three components to handle state management and API processing and business functions.

Presentational Components : The components become dedicated to displaying user interface elements.

- The system needs only basic functions to operate because it provides users with multiple ways to reuse its components.
- The system delivers two main advantages through its implementation which establishes clear boundaries between different system components.

- The system produces better testing results because it enables developers to create tests for each separate system component.

The system generates two main advantages through its implementation which enables developers to create reusable code components that handle multiple system functions.

- Data-driven dashboards
- Forms with complex logic

ii) Higher-Order Components (HOC) : A function that enhances a component.

Advantages

- Code reuse
- Cross-cutting concerns handling
- Cleaner main components

Common Use Cases

- Authentication
- Authorization
- Analytics tracking

Limitations

- The system requires multiple wrapper components to function properly which increases the amount of nested components.

iii) Render Props : A component uses a function as a prop which it receives through its component interface.

Advantages

- High flexibility
- Logic sharing
- Dynamic rendering

Limitations

- JSX requires more code to display
- JSX code becomes more difficult to understand because of its increased volume.

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

The responsive navigation bar serves as a vital UI component for contemporary web applications. The system enables users to move between application sections while it provides consistent access and responsive design across different display dimensions. The frontend applications using Material UI framework achieve their responsive design through the combination of built-in components and theming and breakpoint utilities.

The navigation bar serves to display navigation links in a horizontal format on desktop and laptop screens while it changes to a compact hamburger menu with a slide-out drawer on mobile devices and tablets. The design guarantees users an excellent experience which works across all display dimensions.

Design Approach

The navigation bar follows a mobile-first responsive design approach. The layout dynamically adapts based on screen width using Material UI's breakpoint system. The navigation bar displays navigation options as buttons for users on medium and larger screens. The screen buttons become invisible to users on smaller screens, who can access the same navigation options through a menu icon that opens a vertical layout drawer.

The approach provides uniform navigation choices throughout the system while enhancing the experience for users who navigate using touch screens.

Breakpoint Handling and Responsiveness

Material UI provides built-in breakpoint definitions such as xs, sm, md, and lg. The application uses the useMediaQuery hook together with the current theme to track real-time changes in screen dimensions. The navigation bar changes to its mobile layout when the screen width falls below the medium breakpoint. The method enables automatic CSS media query handling while maintaining centralized styling control.

Implementation Code

The following React component demonstrates a fully functional responsive navigation bar implemented using Material UI. The code is intentionally kept clean and free of inline comments for easy copying and submission:

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

function Navbar() {
```

```

const theme = useTheme();

const isMobile = useMediaQuery(theme.breakpoints.down("md"));

const [open, setOpen] = useState(false);

const navItems = ["Dashboard", "Projects", "Teams", "Profile"];

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

        {isMobile ? (
          <IconButton color="inherit" onClick={() => setOpen(true)}>
            <MenuIcon />
          </IconButton>
        ) : (
          navItems.map(item => (
            <Button key={item} color="inherit">
              {item}
            </Button>
          ))
        )}
      </Toolbar>
    </AppBar>

    <Drawer anchor="right" open={open} onClose={() => setOpen(false)}>
      <Box sx={{ width: 250, display: "flex", flexDirection: "column", p: 2 }}>
        {navItems.map(item => (
          <Button key={item} sx={{ justifyContent: "flex-start" }}>
            {item}

```

```

        </Button>

    )))
  </Box>

</Drawer>

</>

);
}

export default Navbar;

```

Explanation of the Implementation

The AppBar component functions as the primary navigation element container which builds the navigation bar structure. The Toolbar component inside the AppBar system maintains uniform content alignment through its design which establishes fixed distance and alignment rules. The application title uses the Typography component for display which creates space to position navigation items at the right side of the interface.

The system uses useTheme together with useMediaQuery to create its responsive functionality. The component uses these two hooks to check whether the current screen width has reached the medium breakpoint threshold. The component displays desktop navigation buttons when the screen width exceeds the medium breakpoint threshold while it shows a mobile menu icon for smaller screen devices.

The menu icon on small screens activates a Drawer component which emerges from the right side of the display. The drawer displays the same navigation options which appear as vertically arranged buttons that users can easily touch on their devices.

The single array which contains all navigation items enables the component to function across both desktop and mobile formats while providing simple methods to update or enhance the system.

Advantages of This Implementation

The system delivers a clean navigation system which grows easily to handle greater user needs. The system achieves responsive design through automatic CSS media query generation which protects visual elements across multiple display sizes while meeting Material Design accessibility requirements. The component structure enables developers to create modular systems which connect routing systems with role-based access control together with dark mode and other user interface components.

Conclusion

The responsive navigation bar showcases Material UI elements together with theming and breakpoint functions to develop a professional user interface which adapts to different

situations. The solution is suitable for real-world applications such as dashboards, project management tools, and enterprise web platforms, and it can be extended easily as application requirements grow.

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 collaborative project management tool is a complex, data-intensive, multi-user application that must support real-time updates, role-based access, scalability, performance optimization, and a responsive user interface. Designing the frontend architecture for such a system requires careful planning of routing, state management, UI design, data flow, and concurrency handling.

This solution presents a complete frontend architectural design using React as the core framework, Redux Toolkit for state management, and Material UI for responsive UI design.

6(a) SPA Structure with Nested Routing and Protected Routes

Single Page Application (SPA) Architecture

A Single Page Application loads a single HTML page and dynamically updates the content without reloading the browser. This approach is ideal for project management tools because it provides fast navigation, smooth user experience, and efficient real-time updates.

The SPA is divided into **public routes** and **protected routes**.

Public routes include:

- Login
- Register
- Forgot password

Protected routes include:

- Dashboard
- Projects
- Tasks

- Team collaboration
- Activity logs
- Settings

Nested Routing Structure

Nested routing is used to reflect logical hierarchy within the application.

Example logical structure:

- /dashboard
 - /dashboard/overview
 - /dashboard/activity
- /projects
 - /projects/:projectId
 - /tasks
 - /members
 - /files
 - /chat

This structure improves:

- URL readability
- Component reusability
- Logical separation of concerns

Protected Routes and Authentication Control

Protected routes ensure that only authenticated users can access sensitive pages.

The protection mechanism works as follows:

- Authentication status is stored in global state
- A route guard checks user authentication
- Unauthorized users are redirected to the login page

This approach prevents:

- Unauthorized data access
- Direct URL manipulation
- Security loopholes in SPA navigation

6(b) Global State Management Using Redux Toolkit with Middleware

In a collaborative project management tool, many components depend on shared data such as:

- User authentication details
- Project lists
- Tasks and statuses
- Team members
- Notifications
- Real-time updates

Managing this data using local state would lead to excessive prop drilling and inconsistent application behavior.

Redux Toolkit Architecture

Redux Toolkit provides a standardized and efficient way to manage global state with minimal boilerplate.

The global store is divided into feature-based slices, such as:

- Authentication slice
- User slice
- Projects slice
- Tasks slice
- Notifications slice
- Real-time updates slice

Each slice contains:

- Initial state
- Reducers for synchronous updates
- Async actions for API communication

Middleware Usage

Middleware plays a critical role in handling side effects.

The architecture includes:

- Async middleware for API requests

- WebSocket middleware for real-time updates
- Logging middleware for debugging
- Error handling middleware for graceful failure recovery

Middleware ensures that:

- Business logic stays out of UI components
- Side effects are predictable
- Debugging is easier using Redux DevTools

Real-Time State Synchronization

Real-time updates such as task status changes, comments, or collaborator activity are handled via WebSocket connections. Incoming updates are dispatched as Redux actions, ensuring that all connected users see consistent data instantly.

6(c) Responsive UI Design Using Material UI with Custom Theming

Material UI for Design Consistency

Material UI is used to build a consistent, accessible, and responsive interface. It provides prebuilt components that follow Material Design principles, reducing development time while maintaining UI quality.

Custom Theme Configuration

A custom theme is defined to ensure:

- Brand consistency
- Centralized color management
- Typography standardization
- Easy implementation of dark and light modes

The theme controls:

- Primary and secondary colors
- Font families and sizes
- Spacing and layout density
- Component-level style overrides

This ensures visual uniformity across all pages and components.

Responsive Layout Strategy

Responsiveness is achieved using:

- Grid system

- Flexbox layouts
- Built-in breakpoints (xs, sm, md, lg)

Key UI elements such as navigation bars, side drawers, project boards, and task lists adapt dynamically based on screen size. This ensures usability across desktops, tablets, and mobile devices.

Accessibility Considerations

Material UI components follow accessibility standards by default, ensuring:

- Keyboard navigation support
- Proper ARIA attributes
- High contrast text
- Touch-friendly controls

6(d) Performance Optimization Techniques for Large Datasets

A project management tool often handles large datasets such as thousands of tasks, logs, or comments. Without optimization, this can lead to performance degradation.

Efficient Rendering Techniques

The following techniques are applied:

- Component memoization to prevent unnecessary re-renders
- Callback memoization for event handlers
- Conditional rendering for heavy UI components

List Virtualization

Only visible items are rendered on screen using virtualization techniques. This dramatically reduces DOM size and improves scrolling performance for large task lists and activity feeds.

Pagination and Lazy Loading

Data is fetched in chunks instead of loading everything at once. This:

- Reduces initial load time
- Improves perceived performance
- Minimizes memory usage

Code Splitting

Routes and heavy components are lazy-loaded. This ensures that users download only the code required for the current view, improving startup performance.

Debouncing and Throttling

User interactions such as search, filtering, and drag-and-drop updates are debounced or throttled to reduce unnecessary API calls and state updates.

6(e) Scalability Analysis and Improvements for Multi-User Concurrent Access

Scalability Challenges

In a multi-user collaborative system, several challenges arise:

- Simultaneous updates from multiple users
- Data consistency across clients
- Network latency
- Conflict resolution
- Performance bottlenecks

Frontend Scalability Strategies

The frontend architecture addresses these challenges using:

Optimistic UI Updates

User actions update the UI immediately before server confirmation, creating a fast and responsive experience.

Normalized State Structure

Data is stored in a normalized format to avoid duplication and reduce update complexity.

Role-Based Rendering

UI elements are rendered based on user roles such as admin, manager, or contributor, reducing unnecessary UI complexity.

Concurrency Handling

Real-time collaboration is handled through:

- WebSocket connections
- Event-based state updates
- Timestamp or version-based conflict detection

This ensures that:

- All users see updates in near real time
- Conflicts are detected and resolved gracefully
- Data integrity is preserved

Future Scalability Improvements

Recommended improvements include:

- Micro-frontend architecture for large teams
- Offline-first support using local caching
- Background synchronization
- Progressive Web App features
- Advanced real-time conflict resolution strategies

Final Architecture Flow

User Interface

- React Components
- Redux Toolkit Store
- Middleware (API + WebSocket)
- Backend Services
- Real-Time Updates
- Redux Store
- UI Re-render

Conclusion

This frontend architecture provides a robust, scalable, and high-performance foundation for a collaborative project management tool. By combining structured SPA routing, centralized state management, responsive UI design, performance optimization techniques, and real-time synchronization strategies, the system ensures a seamless experience even under heavy multi-user load. The architecture is future-proof, maintainable, and suitable for enterprise-grade applications.