EXPERIMENT

Aim- Manage Complex State with Redux or Context API

The aim of this project/module is to demonstrate effective management of complex application state in a React-based application using **Redux** or **Context API**. By implementing these state management tools, we seek to create scalable, maintainable, and efficient front-end applications that handle state changes predictably and improve developer productivity.

Introduction

In modern web applications, especially those built with React, managing the state (data that changes over time) is crucial but can become challenging as applications grow in size and complexity. React's built-in state management using useState and useReducer hooks works well for local component states but can become cumbersome when the state needs to be shared or updated across multiple components.

To address these challenges, state management libraries like **Redux** and React's **Context API** have been developed:

- **Redux** is a predictable state container for JavaScript applications, enforcing a unidirectional data flow and a strict structure to manage state globally.

- **Context API** is a React built-in feature that allows you to pass data through the component tree without having to pass props down manually at every level.

Both tools help in managing complex, shared states by centralizing state logic and improving code maintainability.

Theory

- **State Management:** The process of handling the data that controls the UI, user input, and server responses. Efficient state management ensures that components render correctly and stay in sync.

- **Redux Concepts:**

    - **Store:** The centralized place that holds the application state.

    - **Actions:** Plain objects that describe "what happened" and carry data.

    - **Reducers:** Pure functions that take the previous state and an action, then return the new state.
    - **Dispatch:** The method used to send actions to the store.
    - **Middleware:** Functions that intercept actions for logging, asynchronous operations, etc.
- **Context API Concepts:**
    - **Provider:** A component that provides the state to its descendants.
    - **Consumer:** Components that consume or use the shared state.
    - **useContext Hook:** Allows functional components to subscribe to React context changes easily.

Both Redux and Context API reduce prop drilling (passing props through many nested components) and make state management more structured and manageable.

What We Have Done

- Created a React application that requires sharing and updating complex state across multiple components.
- Implemented **Redux** for global state management:
    - Set up the Redux store with reducers and actions.
    - Connected React components to the Redux store using react-redux library (Provider, useSelector, useDispatch).
    - Handled asynchronous data fetching using middleware (e.g., Redux Thunk).
- Alternatively, implemented **Context API** in some modules:
    - Created context providers and consumers.
    - Used the useContext hook for easier access to the shared state.
- Demonstrated the pros and cons of both approaches in terms of scalability, complexity, and ease of use.

- Managed actions like updating user data, toggling UI elements, and handling form inputs via the chosen state management method.

- Ensured components re-render only when necessary by optimizing selectors or context value updates.

Technology Used

- **React.js:** Frontend JavaScript library for building user interfaces.

- **Redux:** State management library for managing complex global state.

- **React-Redux:** Official React bindings for Redux.

- **Redux Thunk:** Middleware to handle asynchronous actions in Redux.

- **Context API:** React's built-in state sharing feature.

- **JavaScript (ES6+):** Core programming language.

- **Node.js & npm:** For package management and running development servers.

- **Webpack / Create React App:** For project bundling and development environment.

- **VS Code / Any Code Editor:** For coding and debugging.

Tools Used

- **Redux DevTools:** Browser extension to debug and inspect Redux state changes.

- **React Developer Tools:** Browser extension to inspect React component tree and state.

- **Postman:** For testing backend APIs (if applicable).

- **Git:** Version control system to manage project code.

- **Chrome / Firefox:** Browsers used for testing and debugging.

Code:

App.css

```css
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
}

.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
}
.logo:hover {
  filter: drop-shadow(0 0 2em #646cffaa);
}
.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafbaa);
}

@keyframes logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

Index.html

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>flux-loom-magic</title>
    <meta name="description" content="Lovable Generated Project" />
    <meta name="author" content="Lovable" />

    <meta property="og:title" content="flux-loom-magic" />
    <meta property="og:description" content="Lovable Generated Project" />
    <meta property="og:type" content="website" />
    <meta property="og:image" content="https://lovable.dev/opengraph-image-p98pqg.png" />

    <meta name="twitter:card" content="summary_large_image" />
    <meta name="twitter:site" content="@lovable_dev" />
    <meta name="twitter:image" content="https://lovable.dev/opengraph-image-p98pqg.png" />
  </head>

  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

Main.tsx

```tsx
import { createRoot } from "react-dom/client";
import App from "./App.tsx";
import "./index.css";

createRoot(document.getElementById("root")!).render(<App />);

```

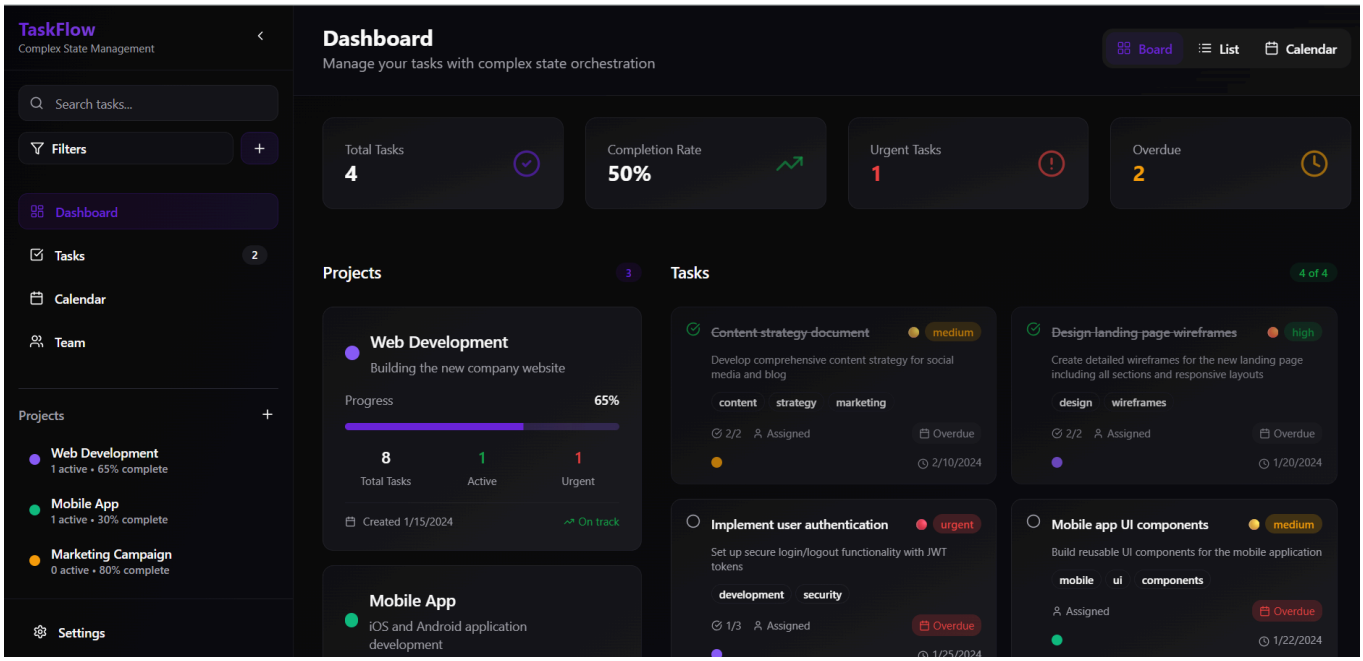
App.tsx

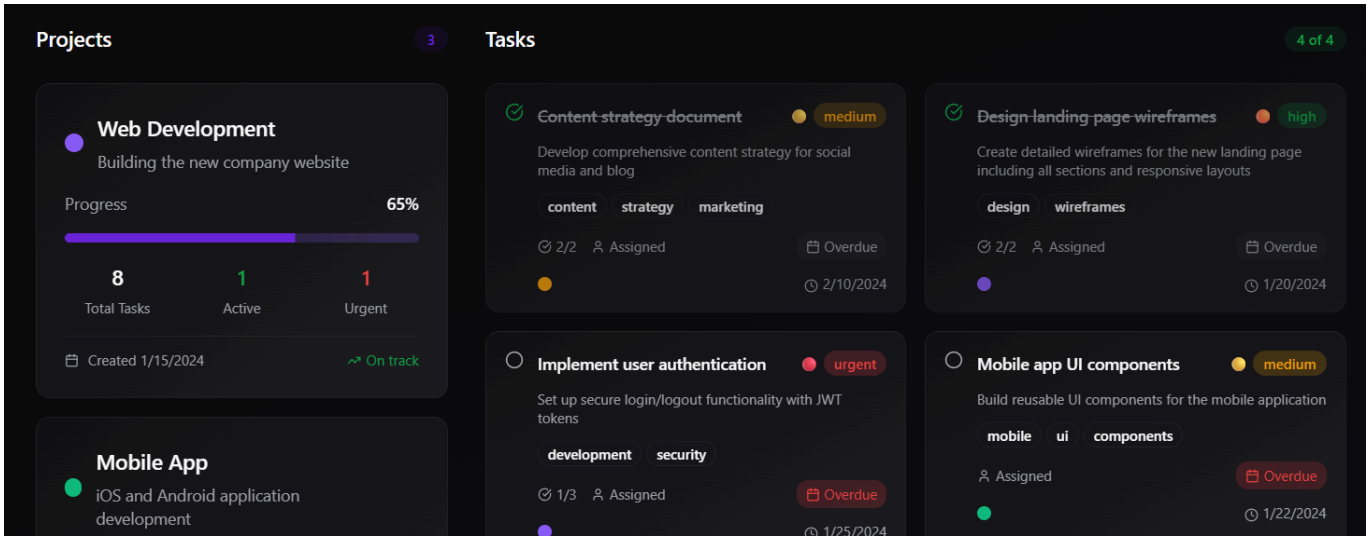
```tsx
import { Toaster } from "@/components/ui/toaster";
import { Toaster as Sonner } from "@/components/ui/sonner";
import { TooltipProvider } from "@/components/ui/tooltip";
import { QueryClient, QueryClientProvider } from "@tanstack/react-query";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Index from "./pages/Index";
import NotFound from "./pages/NotFound";

const queryClient = new QueryClient();

const App = () => (
  <QueryClientProvider client={queryClient}>
    <TooltipProvider>
      <Toaster />
      <Sonner />
      <BrowserRouter>
        <Routes>
          <Route path="/" element={<Index />} />
          {/* ADD ALL CUSTOM ROUTES ABOVE THE CATCH-ALL "*" ROUTE */}
          <Route path="*" element={<NotFound />} />
        </Routes>
      </BrowserRouter>
    </TooltipProvider>
  </QueryClientProvider>
);

export default App;
```

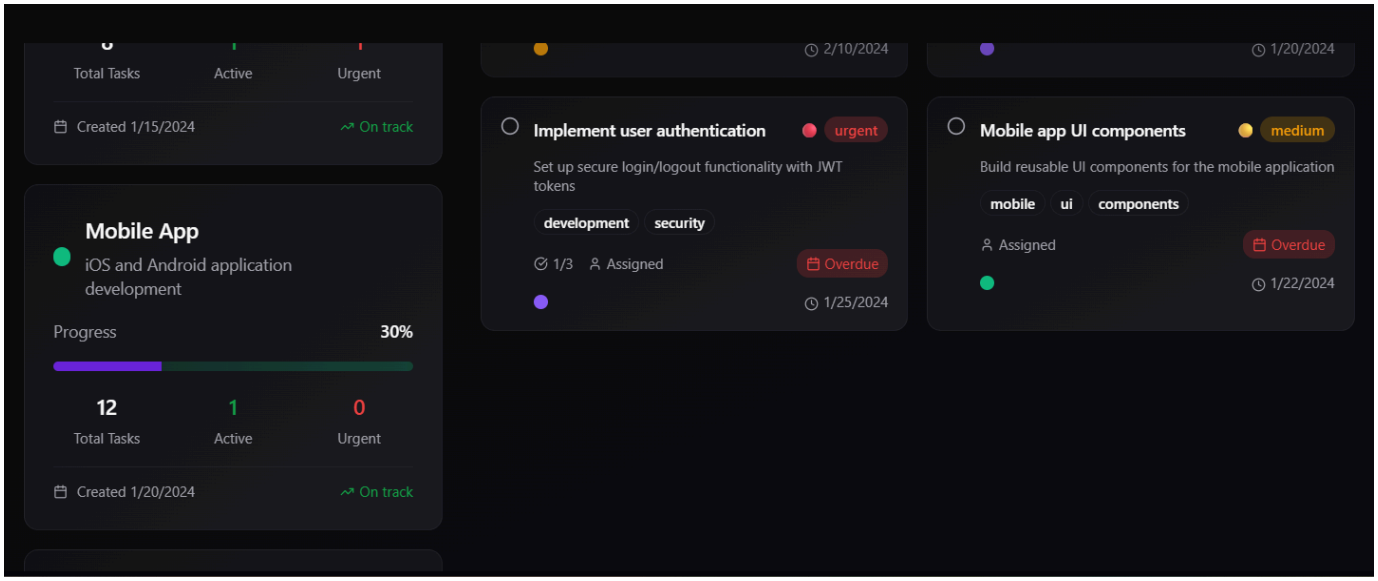
```tsx
import { CheckCircle, TrendingUp, AlertTriangle, Clock } from "lucide-react";
import { SidebarProvider, SidebarTrigger } from "@/components/ui/sidebar";
import { DashboardSidebar } from "@/components/DashboardSidebar";
import { StatsCard } from "@/components/StatsCard";
import { ProjectCard } from "@/components/ProjectCard";

const Index = () => {
  const statsData = [
    {
      title: "Total Tasks",
      value: 4,
      icon: CheckCircle,
      variant: "primary" as const,
    },
    {
      title: "Completion Rate",
      value: "50%",
      icon: TrendingUp,
      variant: "success" as const,
    },
    {
      title: "Urgent Tasks",
      value: 1,
      icon: AlertTriangle,
      variant: "destructive" as const,
    },
    {
      title: "Overdue",
      value: 2,
      icon: Clock,
      variant: "warning" as const,
    },
  ];

  return (
```

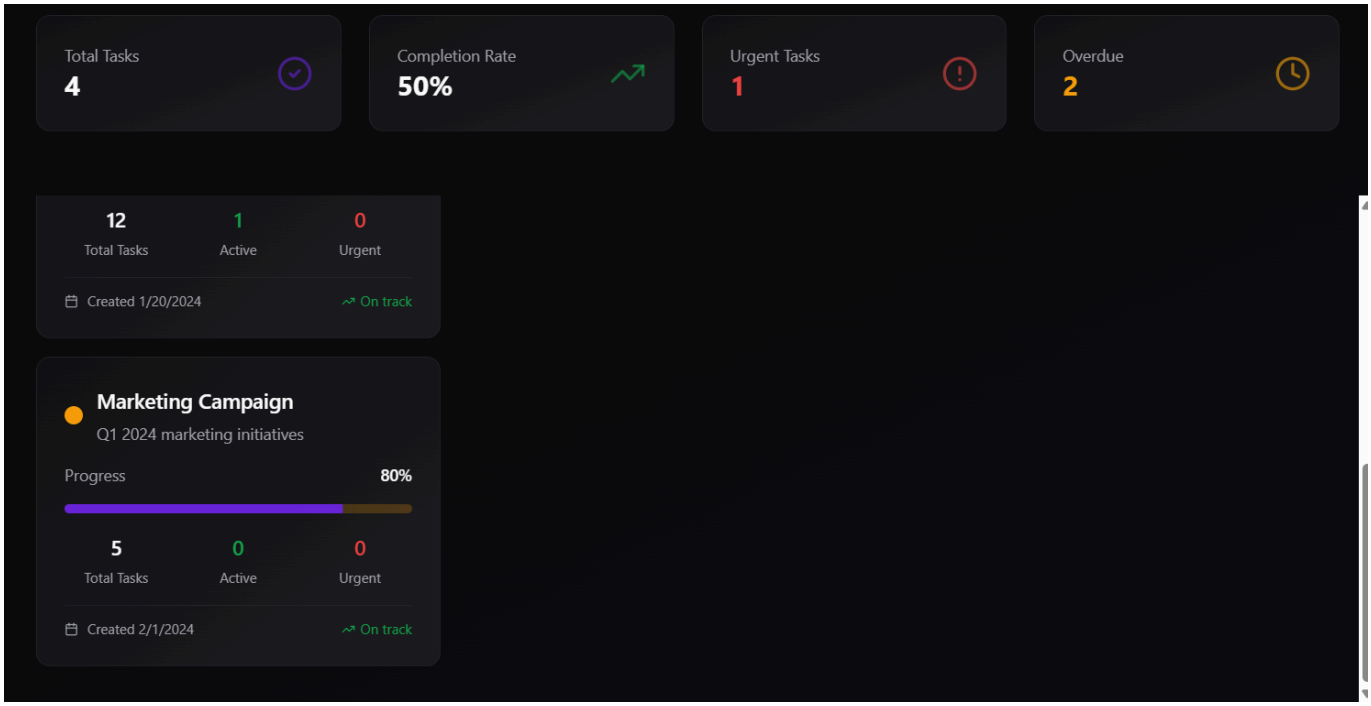
```
<SidebarProvider>
  <div className="min-h-screen flex w-full bg-background">
    <DashboardSidebar />

    <div className="flex-1 flex flex-col">
      {/* Header */}
      <header className="h-16 border-b border-border bg-card/50 backdrop-blur-sm sticky top-0 z-10">
        <div className="flex items-center h-full px-6">
          <SidebarTrigger className="mr-4" />
          <h1 className="text-xl font-semibold text-foreground">Dashboard</h1>
        </div>
      </header>

      {/* Main Content */}
      <main className="flex-1 p-6 space-y-6">
        {/* Stats Grid */}
        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">
          {statsData.map((stat) => (
            <StatsCard key={stat.title} {...stat} />
          ))}
        </div>

        {/* Projects Section */}
        <div className="space-y-4">
          <h2 className="text-lg font-semibold text-foreground">Active Projects</h2>
          <div className="grid grid-cols-1 lg:grid-cols-2 xl:grid-cols-3 gap-6">
            <ProjectCard
              title="Marketing Campaign"
              description="Q1 2024 marketing initiatives"
              progress={80}
              totalTasks={5}
              activeTasks={0}
              urgentTasks={0}
              createdDate="2/1/2024"
              status="on-track"
```

Output


TaskFlow
Complex State Management
Search tasks...
Filters
Dashboard
Tasks
2
Calendar
Team
Projects
Web Development
1 active • 65% complete
Mobile App
1 active • 30% complete
Marketing Campaign
0 active • 80% complete
Settings
Dashboard
Manage your tasks with complex state orchestration
Board
List
Calendar
Total Tasks
4
Completion Rate
50%
Urgent Tasks
1
Overdue
2
Projects
3
Web Development
Building the new company website
Progress
65%
8
Total Tasks
1
Active
1
Urgent
Created 1/15/2024
On track
Mobile App
iOS and Android application development
Tasks
4 of 4
Content strategy document
medium
Develop comprehensive content strategy for social media and blog
content
strategy
marketing
2/2
Assigned
Overdue
2/10/2024
Design landing page wireframes
high
Create detailed wireframes for the new landing page including all sections and responsive layouts
design
wireframes
2/2
Assigned
Overdue
1/20/2024
Implement user authentication
urgent
Set up secure login/logout functionality with JWT tokens
development
security
1/3
Assigned
Overdue
Mobile app UI components
medium
Build reusable UI components for the mobile application
mobile
ui
components
Assigned
Overdue
1/22/2024


Projects
3
Web Development
Building the new company website
Progress
65%
8
Total Tasks
1
Active
1
Urgent
Created 1/15/2024
On track
Mobile App
iOS and Android application development
Tasks
4 of 4
Content strategy document
medium
Develop comprehensive content strategy for social media and blog
content
strategy
marketing
2/2
Assigned
Overdue
2/10/2024
Design landing page wireframes
high
Create detailed wireframes for the new landing page including all sections and responsive layouts
design
wireframes
2/2
Assigned
Overdue
1/20/2024
Implement user authentication
urgent
Set up secure login/logout functionality with JWT tokens
development
security
1/3
Assigned
Overdue
1/25/2024
Mobile app UI components
medium
Build reusable UI components for the mobile application
mobile
ui
components
Assigned
Overdue
1/22/2024


Total Tasks
Active
Urgent
Created 1/15/2024
On track
Mobile App
iOS and Android application development
Progress
30%
12
Total Tasks
1
Active
0
Urgent
Created 1/20/2024
On track
2/10/2024
1/20/2024
Implement user authentication
urgent
Set up secure login/logout functionality with JWT tokens
development
security
1/3
Assigned
Overdue
1/25/2024
Mobile app UI components
medium
Build reusable UI components for the mobile application
mobile
ui
components
Assigned
Overdue
1/22/2024



Conclusion

Managing complex state in React applications is essential for building scalable and maintainable user interfaces. This project demonstrated two popular state management techniques: **Redux** and **Context API**. Redux provides a more structured and scalable approach ideal for large applications with complex state logic, while Context API offers a simpler, built-in solution suitable for medium to small-scale apps.