Portfolio Dashboard: Technical Summary and Challenges

Overview

The Portfolio Dashboard is a full-stack Next.js 15-based web application designed to allow users to monitor investment holdings in real-time. It features:
- Real-time portfolio updates using polling (every 15s)
- Sector-wise breakdown of performance
- Dynamic UI using TailwindCSS and ShadCN components
- Dark/light theme support
- Mock APIs for stocks and sectors

Key Challenges & Solutions

1. Real-Time Data Refresh without WebSockets

- **Challenge:** Ensure the portfolio view and sector analysis stay up-to-date without the complexity of socket connections.
- **Solution:** Used `setInterval` to fetch updated data every 15 seconds on both the portfolio table and sector summary components. These updates simulate real-world value fluctuations.

2. Seamless User Experience During Data Load/Failures

- **Challenge:** Provide meaningful feedback to the user while data is loading or if a fetch fails.
- **Solution:** Show animated loaders with icons (`lucide-react`) during loading. Display styled error alerts with retry buttons. Maintain cached data to prevent UI flashing.

3. Managing Complex State (Data, Loading, Errors)

- **Challenge:** Synchronize multiple states (e.g., data, loading, errors, last update) effectively within each component.
- **Solution:** Used `useState` and `useEffect` hooks to create robust data fetch logic. Separated concerns between fetching, data storage, and rendering for clarity.

4. Component Reusability and Scalability

- **Challenge:** Ensure UI components like tables, cards, buttons, and tabs are modular, maintainable, and style-consistent.
- **Solution:** Used `shadcn/ui` and Radix UI primitives. Built custom hooks and component libraries. Adopted TailwindCSS with a comprehensive config for extended theming.

5. Theming & Dark Mode Support

- **Challenge:** Enable easy switching between light and dark modes with minimal overhead.

- **Solution:** Utilized `next-themes` with a `ThemeProvider`. Defined light/dark color tokens using CSS custom properties and configured Tailwind accordingly.

6. Data Mocking and Testing

- **Challenge:** Represent realistic but static financial data without a backend.
- **Solution:** Built mock endpoints for stocks and sectors with logic to simulate market fluctuations for visual dynamism.

7. Directory and Module Structure

- **Challenge:** Maintain a clean codebase that scales as features grow.
- **Solution:** Organized code under clearly named folders such as `app/`, `components/`, `lib/`, `hooks/`, and `styles/`.

Final Thoughts

This project prioritizes clarity, user feedback, and modular design. While mock data is currently used, the structure is ready for real API integrations and enhanced with advanced features like authentication, persistent storage, and chart visualizations.

Technologies Used

Next.js 15, React 18, TailwindCSS, ShadCN/UI, Radix UI, TypeScript, Lucide Icons, Recharts.

Author: Hritik Kumar