TSX & Typed Components

Your Task:
Build a `BudgetTracker` component that:

- Tracks income and expenses in different currencies.
- Shows net balance in a selected currency.
- Uses `useReducer` for state management.
- Implements type-safe props for currency conversion rates.

Your Turn!

1. Create a `PortfolioSummary` functional component that:
    - Receives a typed array of assets (`Asset[]`) as props.
    - Renders the total value and average percentage change.
2. Create an `AssetEditor` class component that:
    - Has typed state for `name` , `symbol` , `value` , and `change` .
    - Accepts a callback prop `onUpdate` (typed) to update an asset.
    - Resets the form after submission.

Routing in React: Type-Safe Route Parameters with React Router & TypeScript

Your Turn!

1. Define a route `/doctors/:doctorId/patients/:patientId` and a `DoctorPatientDetails` component.
2. Use a typed interface for params and extract them in the component.
3. Validate that both IDs are present and numeric; display an error if not.
4. Add a link from a doctor list to a specific doctor/patient page, passing the IDs as parameters.

State Management in React: Context Providers & Zustand (with TypeScript)

Your Turn!

1. Create a Zustand store for notifications:
    - Each notification has `id` , `message` , `type` (`'info' | 'error' | 'success'`), and `read: boolean` .
    - Add actions: `addNotification` , `markAsRead` , and `clearNotifications` .
2. Use the store in a NotificationList component to display unread notifications and mark them as read.

Advanced State Management with Zustand: Middleware, Persistence, and Async Patterns

Your Turn!

1. Create a persisted Zustand store for user session:
    - Fields: `userId: string` , `token: string` , `expiresAt: number`
    - Only persist `userId` and `token` , not `expiresAt`
    - Add a migration to handle a new field, `role: 'admin' | 'user'` (default ‘user’), in version 2.
2. Use devtools and immer middleware for a note history log:

    - Actions: `addHistoryEntry` , `clearHistory`
    - Log each entry as `{ noteId: string, action: string, timestamp: number }`
3. Combine Zustand and React Query:
    - Fetch a list of collaborators from an API.
    - Store collaborators in Zustand.
    - Display collaborators in a component, updating automatically when data is fetched.

Zustand Slices & Modular State Architecture: Scaling a Collaborative Design Platform

Your Turn!

1. Create a `notificationsSlice` :
    - Fields: `notifications: { id: string; message: string; read: boolean }[]`
    - Actions: `addNotification` , `markAsRead` , `clearNotifications`
2. Add the slice to the main store.
3. Build a `NotificationsPanel` component that displays unread notifications and lets users mark them as read.