Frontend-Aufgabe: Personal Finance Tracker

Aufgabenstellung

Stand: 26. Oktober 2025

1 Ziel der Aufgabe

Ziel dieser Aufgabe ist es, Ihre Fähigkeiten in moderner Frontend-Entwicklung mit React und TypeScript zu prüfen — mit Fokus auf Datendarstellung, Offline-Funktionalität, State-Management und guter UX. Die Applikation arbeitet vollständig clientseitig: alle Daten werden lokal (localStorage / IndexedDB / Dateien) gespeichert. Keine serverseitigen Abhängigkeiten.

2 Kurzbeschreibung

Entwickeln Sie eine Single-Page-Applikation „Personal Finance Tracker“, mit der Nutzer Ausgaben und Einnahmen erfassen, kategorisieren, Budgets setzen und Auswertungen (Monatsübersicht, Kategorie-Breakdown, Sparziele) einsehen können. Die App soll robust typisiert, zugänglich und responsiv sein sowie grundlegende Tests enthalten.

3 Anforderungen

1. Framework und Tooling
    - React + TypeScript (z. B. Vite + React + TS oder Next.js ohne Server-APIs).
    - Paketmanager: npm oder yarn.
    - Nützliche Scripts in `package.json`: `dev`, `build`, `start`, `test`, `lint`.
2. Daten und Persistenz
    - Transaktionen (Einnahmen/Ausgaben), Konten und Kategorien in lokalen Datenstrukturen.
    - Persistenz via `localStorage` oder optional `IndexedDB` (z. B. idb).
    - Optional: Import/Export von CSV-Dateien (Transaktionen).
    - Keine Remote-APIs.
3. Kern-Funktionalitäten
    - Erfassen von Transaktionen: Datum, Betrag, Typ (Ein-/Ausgabe), Kategorie, Konto, Notiz/Tag.
    - Kategoriesystem: feste Kategorien (z. B. Miete, Lebensmittel) und Subkategorien; Kategorieverwaltung (CRUD).
    - Budgetierung: Monatsbudget pro Kategorie mit Warnungen bei Überschreitung.
    - Auswertungen:
        - Monatsübersicht (Einnahmen vs. Ausgaben).
        - Kategorie-Breakdown (Tortendiagramm oder Balken).

  - Verlauf (Zeitreihe) für Konto-Saldo oder Kategorieausgaben.
- Filter und Suche: Zeitraum, Konto, Kategorie, Volltextsuche in Notizen.

4. UI / Styling
   - Responsives Design (Mobile Desktop).
   - Klarer, minimalistischer Look (z. B. Tailwind, CSS Modules oder Styled Components).
   - Wichtige UI-Elemente: Header mit Navigation, Dashboard, Formulare zum Erfassen, Filter-Sidebar, Transaktionsliste, Footer.
   - Microinteractions: Bestätigung beim Löschen, Undo für kürzlich gelöschte Transaktionen.
5. Accessibility (A11y) (Optional)
   - Semantische HTML-Elemente, Tastaturbedienbarkeit (Formulare, Tabellen, Modals).
   - ARIA-Attribute für komplexe Controls (z. B. modale Import-Dialoge).
   - Farbkontrast prüfen (WCAG AA). Nicht nur Farbe zur Informationsvermittlung verwenden (z. B. Symbole).
6. Typisierung
   - Vollständige TypeScript-Typen für Datenmodelle, Komponenten und Hooks.
   - Keine Verwendung von `any`.
7. Tests
   - Unit-Tests für Kernlogik (Saldo-Berechnung, Budget-Warnungen, CSV-Parser).
   - Komponententests (React Testing Library) für wichtige UI-Flows (Transaktion hinzufügen, Filter, Import).
   - Testlauf via `npm test`.
8. Dokumentation
   - README mit Setup, Architektur-Kurzbeschreibung, Datenformat und Instruktionen zu Import/Export.
   - Hinweise zur Persistenz (welcher Speicher verwendet wird) und zur Testausführung.

4 Optionale Erweiterungen

- PWA-Funktionen (Service Worker) für Offline-Nutzung.
- Automatische Kategorisierung mittels einfacher Regeln (z. B. Payee-Mapping).
- Mehrwährungs-Unterstützung und Wechselkurs-Snapshot (lokal gepflegt).
- GraphQL-Abstraktion für internen State-Layer (nur clientseitig).
- Visuelle Financial Goals / Sparziel-Tracker mit Fortschrittsbalken.
- Storybook für UI-Komponenten.
- Kontenverwaltung: mehrere Konten (z. B. Giro, Cash, Kreditkarte) mit Salden (manuell oder berechnet).
- CSV-Import/Export: Import von Transaktionen (Mapping UI) und Export der gefilterten Ansicht als CSV.
- Zustandshandling: React Context oder leichter Store (z. B. Zustand) — strikt typisiert.
- Lade-/Fehlerzustände: simulierte Ladezeit beim Initialisieren/Importieren von Daten; Fehlerbehandlung bei ungültigen CSVs.

Viel Erfolg bei der Umsetzung!