Esercitazione: "Spese Tracker" con React e PocketBase

Obiettivo del Progetto

Realizzare una Single Page Application (SPA) in React per il tracciamento delle spese personali. L'applicazione dovrà implementare le funzionalità **CRUD** (Create, Read, Update, Delete) complete.

I dati dovranno essere salvati e recuperati da un'istanza backend **PocketBase** che configurerete localmente.

Requisiti Tecnici

- **Frontend:** React (utilizzando solo gli hook `useState` e `useEffect`, props e componenti).
- **Backend:** PocketBase (avviato in modalità `serve`).
- **Comunicazione:** `fetch` API nativa.

Task Generali

Di seguito sono elencati i macro-obiettivi da raggiungere. Siete liberi di organizzarvi e scomporre il lavoro come preferite, ma tutti i task devono essere completati.

1. Configurazione del Backend

Prima di scrivere codice React, è necessario un backend funzionante.

- Avviate **PocketBase** e accedete all'Admin UI.
- Definite una nuova collezione chiamata **`spese`**.
- Configurate la collezione con i campi necessari per tracciare una spesa (es: `descrizione`, `importo`, `data`).

> **Hint critico:** Per questo esercizio, semplificate l'autenticazione. Andate nelle "API Rules" della collezione `spese` e impostate i permessi "List", "View", "Create" e "Delete" su **"Anyone"**. Inserite anche 1-2 record di prova manualmente.

2. Visualizzazione delle Spese (Read)

Il primo task in React è leggere e mostrare i dati presenti sul backend.

- Nel componente principale (`App.js`), definite uno stato (`useState`) per contenere l'array delle spese.
- Implementate una funzione asincrona per eseguire il `fetch` dei dati dall'endpoint della vostra collezione PocketBase.
- Eseguite questa funzione solo al montaggio del componente.
- Renderizzate la lista delle spese nell'interfaccia.

> **Hint critico:** Ricordate che `useEffect` con un array di dipendenze vuoto (`[]`) esegue la funzione solo al *mount*. Inoltre, l'API di PocketBase restituisce l'elenco dei record dentro la chiave `items` dell'oggetto JSON di risposta.

3. Creazione di una Nuova Spesa (Create)

L'applicazione deve permettere all'utente di inserire nuove spese.

- Create un componente separato per il form (es. `FormSpesa.js`).
- Il form deve gestire il proprio stato interno per i campi di input.
- Alla sottomissione (`onSubmit`), il form deve passare i dati al componente genitore (`App.js`).
- Implementate in `App.js` la logica per eseguire il `fetch` con metodo **`POST`** verso PocketBase.
- L'interfaccia deve aggiornarsi automaticamente dopo l'aggiunta.

> **Hint critico:** Non eseguite il `fetch` (POST) *dentro* il componente Form. Il Form deve solo raccogliere i dati e invocare una funzione (ricevuta via *props*) dal genitore. Sarà il genitore (`App.js`) a gestire la logica di rete e l'aggiornamento dello stato globale.

4. Eliminazione di una Spesa (Delete)

Infine, l'utente deve poter eliminare una spesa esistente.

- Aggiungete un bottone "Elimina" per ogni spesa nella lista.
- Per una migliore organizzazione, potreste creare un componente `ItemSpesa.js`.
- Al click sul bottone, deve essere invocata una funzione (definita in `App.js`) che riceva l'ID della spesa da eliminare.
- Questa funzione deve eseguire un `fetch` con metodo **`DELETE`**.
- L'interfaccia deve aggiornarsi per riflettere l'eliminazione.

> **Hint critico:** L'endpoint per una richiesta `DELETE` è diverso da quello di `GET` (List). Deve includere l'ID specifico del record: `.../api/collections/spese/records/ID_DEL_RECORD`. Per aggiornare la UI, la soluzione più performante è usare `.filter()` sullo stato locale, invece di rieseguire un fetch completo.

Task Bonus: Dashboard Grafica

Se avete completato tutti i task principali, migliorate l'applicazione aggiungendo una visualizzazione dati.

- Calcolate e mostrate il **totale** delle spese.
- Scegliete una libreria per grafici (es. **Recharts** o **Chart.js** con `react-chartjs-2`).
- Installatela e create un nuovo componente `GraficoSpese.js`.
- Visualizzate i dati delle spese usando un grafico a torta (`PieChart`) o un grafico a barre (`BarChart`) per mostrare la distribuzione degli importi.