React Challenge

Timebox: *24 hours* from receiving the task.

Goal: Build a small but polished React app that explores a public dataset with great UX. AI use is allowed and encouraged.

The App You'll Build — *"Resource Explorer"*

A single-page React app that lists items from a public, no-auth API (examples below), lets users search, filter, sort, and view details, and lets them "favorite" items. Your app should feel fast, handle errors gracefully, and be easy to understand from the code.

Pick **one** dataset:

- **PokéAPI** (pokemon, abilities)
- **The Rick & Morty API** (characters, episodes)
- **Open Library** (books)
- Or any public, no-auth JSON API. If you're offline, load from a local JSON file served by the dev server.

> **Tip:** Choose an endpoint with built-in pagination or many results so search/filters have meaning.

Must-Have Requirements

1. **Project setup**
    - React (Next.js and TypeScript preferred).
    - Sensible file structure and component boundaries.
2. **Data list + detail view**
    - A list view with **pagination or infinite scroll**.
    - Clicking an item opens a **detail view** at a route like `/items/:id`.
3. **Search, filter, sort**

- **Debounced** search (e.g., 300–500ms) bound to the URL (e.g., `?q=pikachu`).
- At least one **filter** (e.g., status/species/type) and one **sort** option.
- URL reflects state so it's **shareable** and **reload-safe**.

4. **Favorites**

   - Toggle an item as a favorite from list and detail views.
   - Persist favorites in `localStorage` (or IndexedDB). Show a "Favorites" filter.

5. **Data fetching and state**

   - Handle **loading** (skeletons/placeholders) and **errors** (retry button).
   - **Cancel** in-flight requests when inputs change (e.g., using `AbortController` or library support).

Nice-to-Have (Pick 2–3)

- **Client caching** and background refetch (e.g., React Query/TanStack Query) or a simple custom cache.
- **Virtualized list** once there are 100+ items (e.g., `react-window`).
- **Optimistic UI** for favorite toggles.
- **Theme toggle** (light/dark) with persistent preference.
- **Code splitting** for the detail route.
- **Form** on the detail page (e.g., add a note for an item) with validation.
- **Basic E2E** smoke test using Playwright or Cypress for one happy path.
- **(Optional) Basic accessibility/keyboard support** for primary flows (labels, focus outlines, alt text) — nice-to-have, not required.

The Tricky Bits (intentional!)

- **URL is the source of truth** for search/filter/sort/page. Directly visiting a URL should recreate state.
- **Abort on change**: if the user is typing, previous fetches must be canceled to avoid race conditions.
- **Empty states**: show helpful empty/no-results messages (not just a blank screen).
- **Back/forward navigation** should not lose list scroll position or focus.

You don't have to perfect all of these—document trade-offs you made and why.

What We're Evaluating

- **Product thinking**: sensible defaults, useful empty/loading states.
- **Code quality**: component boundaries, naming, readability, comments where they add value.
- **React fundamentals**: hooks, effects, memoization, derived state vs. source of truth.
- **State & data**: URL/state sync, cancellation, error handling, caching.
- **A11y & UX**: keyboard, focus, semantics, affordances.

Submission Checklist

- ✅ Public Git repo (GitHub/GitLab) with clear commit history.
- ✅ **README** including:
  - How to run
  - Notes on architecture and trade-offs
  - If you ran out of time: what you'd ship next and why
- ✅ If possible, a hosted preview (Vercel/Netlify) — not required.

Starter Constraints (useful guardrails)

- You may use a small UI kit (Chakra, MUI, Tailwind) and a data-fetching lib (React Query, SWR) if you like.
- Avoid heavy state managers unless justified (Redux/Zustand only if really needed).
- Keep third-party dependencies minimal and purposeful.