Frontend Engineering Intern: 2-Day Practical Assessment

1. Objective

This document outlines the requirements for a 2-day practical assessment. The objective is to build a two-page "User Directory" application to evaluate your proficiency in building, routing, and managing state within a modern frontend stack.

2. Core Technologies

The project must be built using the following technologies:

- **React** (with TypeScript)
- **React Router** (`react-router-dom`)
- **TanStack Query** (`@tanstack/react-query`)
- **Zustand**

3. API Specification

- **Endpoint:** `https://randomuser.me/api/?results=10`
- **Method:** `GET`
- **Documentation:** https://randomuser.me/documentation

---

4. Rules & Guidelines

1. **Original Work:** All code submitted must be your own original work.
2. **AI Tool Policy:** The use of AI-powered coding assistants for generating functional code or logic is strictly prohibited. You may use any resources for syntax lookup or documentation clarification.
3. **Commit Hygiene:** You must use clear, conventional commit messages.
4. **Code Review:** Upon submission, you must be prepared to provide a verbal code walkthrough. You will be expected to explain your architectural decisions, data flow, and state management choices.
5. **Library Restrictions:** Do not use any additional third-party libraries for state management or data fetching beyond those specified in Section 2.
6. **Focus:** The primary evaluation metrics are functionality, code quality, type safety, and correctness of state management. The UI must be clean and responsive, but pixel-perfect styling is a secondary concern.

---

5. Functional Requirements

The application must strictly adhere to the following functional and technical requirements.

5.1. Project Setup

- Initialize a new React project using a TypeScript template (Vite or CRA).
- Install and configure `react-router-dom`, `@tanstack/react-query`, and `zustand`.
- A `QueryClientProvider` must be set up at the root of the application.

5.2. Routing

The application must implement two (2) routes:

- **Home Page (`/`):** The main list/grid view of users.
- **User Detail Page (`/user/:id`):** A dynamic route for a single user. The `:id` parameter must be a unique identifier (e.g., login UUID or email).

5.3. Page 1: Home Page (`/`)

The Home Page must:

- Utilize the `useQuery` hook from TanStack Query to fetch the list of 10 users.
- Render a loading state while the query is in progress.
- Render an error state if the query fails.
- Map the fetched data to a list of reusable `<UserCard>` components.
- Each `<UserCard>` component must function as a `Link` that navigates to the corresponding `UserDetailPage`.

5.4. Page 2: User Detail Page (`/user/:id`)

The User Detail Page must:

- This dynamic route must display detailed information for the user corresponding to the `:id` parameter.
- The page must render the user's larger profile picture, full name, address, phone number, and email.
- A strategy for populating this page with the correct user data must be implemented.

5.5. Global State: Favorites (Zustand)

- A global Zustand store must be created to manage a list of "favorited" users.
- The store must expose state (e.g., an array of user IDs) and actions (e.g., `addFavorite`, `removeFavorite`).
- A "Favorite" button must be present on both the `<UserCard>` component (Home Page) and on the `UserDetailPage`.

- The button's UI must be bound to the global store, correctly reflecting the user's "favorited" status (e.g., a solid vs. an outlined icon).
- The button's `onClick` handler must dispatch the appropriate action to the Zustand store.
- State must be globally synced: an action on the Home Page must be reflected on the Detail Page and vice-versa.

---

6. Evaluation Criteria (Checklist)

- [ ] Project is built with React and TypeScript (`.tsx`) and is free of type errors.
- [ ] Application correctly renders a loading state on initial load.
- [ ] Home Page displays the list of 10 users from the API.
- [ ] Clicking a `<UserCard>` successfully navigates to the correct dynamic route (e.g., `/user/john.doe@example.com`).
- [ ] User Detail Page correctly displays all required data for the selected user.
- [ ] "Favorite" button is present on all required components.
- [ ] "Favorite" button UI correctly reflects the global state (e.g., favorited/not-favorited).
- [ ] "Favorite" button click events correctly update the global Zustand store.
- [ ] Global state is synced across both routes.
- [ ] Code is clean, well-organized, and follows React best practices.
- [ ] Git history demonstrates logical, incremental progress.
- [ ] The intern can successfully explain their code and architectural choices.

7. Bonus Challenge (Optional)

If all primary requirements are met, you may implement the following:

- Add a new route (`/favorites`) that displays a list of *only* the users present in the Zustand store.
- Add a simple search bar to the Home Page that filters users by name (client-side).

8. Submission

- **Deliverable 1:** A link to a GitHub repository containing the complete source code.
- **Deliverable 2:** Schedule a 15-minute code review and walkthrough session to present your solution.