# React Assignment: GitHub Dashboard

### Objective

Build a React-based GitHub Dashboard that displays dynamic information about a user's profile, repositories, and followers using the GitHub REST API. This assignment will test your understanding of RESTful APIs, pagination, search, sorting, error handling, and good React practices.

---

### Assignment Requirements

#### 1. User Profile Page

- **API Endpoint:** `https://api.github.com/users/{username}`
- **Features:**
  - Fetch and display the following user details:
    - Name
    - Username
    - Avatar
    - Location
    - Bio
    - Number of public repositories
    - Number of followers
  - Include a clear and responsive design.
  - Display a message if the username does not exist or if an error occurs (e.g., rate-limiting errors).

#### 2. Repositories Page

- **API Endpoint:** `https://api.github.com/users/{username}/repos`
- **Features:**
  - Fetch and display a list of public repositories, showing:
    - Repository name
    - Description
    - Number of stars
  - **Sorting:**
    - Allow users to sort repositories by stars (ascending or descending).
  - **Pagination:**
    - Implement pagination if the number of repositories exceeds the API's per-page limit (default: 30).
    - Allow users to navigate between pages.
  - **Error Handling:**
    - Handle and display API errors appropriately (e.g., no repositories found, API failures).

#### 3. Followers Page

- **API Endpoint:** `https://api.github.com/users/{username}/followers`
- **Features:**
    - Fetch and display the list of followers, showing:
        - Avatar
        - Name
        - Username
    - Show the total number of followers.
    - **Pagination:**
        - If the number of followers exceeds the API's per-page limit, implement pagination to navigate through the list.
    - **Error Handling:**
        - Handle and display API errors (e.g., no followers found, rate limits exceeded).

#### 4. Search Feature

- **Search Input:**
    - Provide an input field to dynamically search for a GitHub users.
    - Replace the `{username}` in API requests with the input value.
    - While typing, show a dropdown with search suggestions (if applicable) based on partial matches.
- **Error Handling:**
    - Display an appropriate message for invalid usernames or if the user does not exist.

---

**Additional Requirements**

- **Code Quality:**
    - Use modern React (e.g., functional components with hooks).
    - Structure the project in a modular way (components, services, styles).
    - Ensure the code is clean, readable, and well-documented.
- **Responsive Design:**
    - Ensure the application looks good on both desktop and mobile devices.
- **Performance:**
    - Implement efficient API usage and state management.
    - Cache responses (e.g., using local state) where applicable to reduce redundant API calls.
- **Styling:**
    - Use any CSS framework (e.g., TailwindCSS, Bootstrap) or custom CSS.
- **Optional:**
    - Use a state management library like Redux or Zustand.
    - Add a theme toggle (light/dark mode).