# React + Micro-Frontend Developer Challenge: "Modular People Portal" with Memory Game

## Technical Requirements

### Tech Stack:

- **Frontend:** ReactJS, TypeScript
- **Micro-Frontend Architecture**
- **State Management:** Redux or Context
- **Routing:** React-Router
- **Testing:** Jest (Unit), Cypress (E2E - bonus)
- **Authentication Layer:** JWT tokens for user authentication, token refresh handling
- **Performance Optimization:** Lazy-loading, code-splitting, SSR (optional)

### Persistence Layer:

- Use an **in-memory database** (Redis, or a simple object to track users and scores) or a **lightweight DB like SQLite** for data persistence.
- **Real-Time Communication:** Integrate **WebSocket** or **Server-Sent Events** for real-time leaderboard updates (Bonus).
- **Micro-Frontend Communication:** Efficient inter-MFE communication, including shared state management via event bus or shared Context.

## Objective

You are tasked with building a **Modular People Portal** using Micro-Frontends (MFEs). The portal will consist of multiple independently deployable components that interact seamlessly. The main components of the portal include:

1. **Auth MFE:** A login page where users authenticate using their username and password. After authentication, a JWT token is returned and stored for session management.

2. **Directory MFE:** After login, users can view a paginated table of users. The table supports:
    a. **Sorting** (client-side)
    b. **Search** (server-side)
    c. **Infinite scrolling or pagination**
    d. **Caching** user data (use local storage or session cache for performance optimization)
3. **Memory Game MFE:** A **NxN Memory Game** (customizable grid size) that allows users to play a memory matching game. The grid size is customizable via a configuration module (e.g., 3x3, 4x4, 5x5).
4. **Profile MFE:** View detailed information about other users (Bonus)
1. **Memory Game Leaderboard:** Display the leaderboard showing the top 10 users based on their performance metrics.
    a. Efficient leaderboard calculation (optimize for large datasets)

## Challenge Requirements

Requests for authentication, users and user info can be found at:
https://dummyjson.com/docs

### Authentication

**Use the login endpoint** for users to log in. **POST /auth/login** will accept the username and password and return a JWT token and user data.

### Directory

- **Fetch and display a list of users** in a paginated table.
- **Client-side Sorting:** Implement sorting by columns (Name, Username, Company, etc.).
- **Server-side Searching:** Use the search endpoint to search users by name or other fields.
- **Infinite Scrolling:** Load more data as the user scrolls down instead of pagination (optional but recommended for performance).
- **Caching:** Cache the user list data using localStorage or sessionStorage to avoid unnecessary re-fetching.

### Memory Game

- **Customizable Grid Size:** Users can select the grid size for the memory game (e.g., 3x3, 4x4, 5x5).

- **Game Mechanics:**
  - Display a grid of **cards** that are face down.
  - When a user clicks on a card, it should flip over.
  - The user needs to match pairs of cards by flipping them over.
  - After the user matches all the pairs, display a success message and the number of turns it took.
- **Memory Game Configuration:** Users can customize the grid size by selecting NxN (e.g., 3x3, 4x4, etc.). This should adjust the number of cards and the layout dynamically.

### Game Configuration Options:

- Grid size: **NxN** (e.g., 3x3, 4x4, etc.) - customizable.
- Timer: Set a time limit (optional).
- Turn counter: Count the number of turns it took to complete the game.

## Implementation Details

### Authentication:

- **POST /auth/login** will authenticate the user and return a JWT token.
- The token will be used for session management and to secure access to other protected routes (e.g., Directory MFE, Profile MFE).

### Game Session:

- **Game Mechanics:** The user will start a game and then stop a timer once it finishes.
- Store the start and stop times in-memory (use an in-memory object or SQLite for simplicity).
- Store the time and display it in a leaderboard with user and results.

### Memory Game:

- **State Management:** Manage the state of the cards, grid size, matched pairs, and turn counter.
- Use **React Context** to manage the game state globally (e.g., grid size, card states, and game progress).
- **Styling:** Use CSS Grid or Flexbox to layout the cards dynamically based on the selected grid size (NxN).

### Leaderboard:

- Maintain a leaderboard of players, ranking them by the target time.

## Deliverables

### 1. Code:

- Push your solution to a **Git repository**.
- Structure your application with **clear separation of concerns** (authentication, game session logic, leaderboard, memory game).
- Comment on your code and explain your thought process when necessary.

### 2. README:

- Provide a simple **README** with instructions on:
    - Setting up and running the application.
    - Describing the logic behind your leaderboard calculation and the memory game functionality.

### 3. Deployment:

- ○ It is preferred that your solution is already deployed and ready to consume (e.g., on Heroku, Netlify, DigitalOcean, AWS, etc.).
- ○ If possible, **deploy the solution** to a public URL and share the link in your submission.

## Using ChatGPT for Assistance

Feel free to use ChatGPT or other online resources for any assistance you may need. For example, using ChatGPT to generate boilerplate code, search for best practices, or get guidance on complex concepts is encouraged. However:

- **Document AI-generated solutions** (// generated-with-GPT).
- **Ensure** that your solution meets senior-level standards. The final solution should reflect your understanding and expertise.

## Submission Checklist

- **Repo** link shared.
- **README.md** with documentation and deployment.

- **Unit tests** ≥ 80% coverage badge.
- **Live demo URL.**