.NET Developer - Test Assignment

Goal: Develop a .NET Core / .NET 5+ component (e.g., a class library) that interacts with a public API to fetch, process, and potentially cache data. This simulates integrating with and managing data from an external provider, a common task when extending larger platforms.

Scenario: Imagine your platform needs to display user information or configuration data fetched from an external system. You need to build a reliable service component to handle this interaction. We will use the public https://reqres.in/API as a stand-in for this external system.

Core Requirements:

1. API Client Implementation:
    a. Create a client service to interact with the following reqres.in endpoints:
        - GET https://reqres.in/api/users?page={pageNumber} (to get paginated lists of users)
        - GET https://reqres.in/api/users/{userId} (to get details for a single user)
    b. Use HttpClient (preferably via IHttpClientFactory if integrated into a host, or demonstrate proper HttpClient usage otherwise).
    c. Implement using async/await correctly.
2. Data Modeling & Mapping:
    a. Define internal C# models (POCOs/DTOs) to represent the relevant user data returned by the API (e.g., id, email, first_name, last_name).
    b. Map the JSON response from the API to your internal models.
3. Service Layer:
    a. Create a service class (e.g., ExternalUserService) that encapsulates the logic for fetching data.
    b. Implement methods like:
        - Task<User> GetUserByIdAsync(int userId)
        - Task<IEnumerable<User>> GetAllUsersAsync() (Handle pagination internally to fetch all users across available pages).

4. Configuration:
    a. Make the base URL (https://reqres.in/api/) configurable (e.g., read from appsettings.json if using a host, or demonstrate how it would be configured).
5. Error Handling & Basic Resilience:
    a. Handle potential errors gracefully:
        - API request failures (network issues, timeouts).
        - Non-success HTTP status codes (e.g., 404 Not Found when requesting a specific user).
        - Deserialisation errors.
    b. Return meaningful results or throw specific exceptions to indicate different failure modes.
    c. Consider (even if just commented or discussed in README) how you might add basic retry logic for transient network errors.

Deliverable:

1. A link to a Git repository (e.g., GitHub, GitLab) containing your solution:
2. A .NET Class Library project containing the core logic (client, service, models).
3. A separate Unit Test project.
4. Optionally, a simple Console Application project that demonstrates how to use the class library.
5. Clear instructions in a README.md on how to build/test the solution, and any design decisions made.
6. A video walkthrough of how you solved this problem.

Bonus Points:

1. Implement basic caching for API responses (e.g., in-memory cache with configurable expiration).
2. Implement actual retry logic using a library like Polly.
3. Demonstrate advanced configuration handling (e.g., options pattern).
4. Structure the solution following Clean Architecture principles.

Time Suggestion: Aim to spend roughly 4-6 hours on this task. The focus should be on the service component's quality, design, robustness, and testability.