Full Stack Developer Home Assignment

Introduction

Thank you for taking on our home assignment. This task requires the implementation of a **Users Web List Application**, encompassing both backend and frontend components.

- The **backend**, developed using **Node.js Web API**, interacts with an external API (ReqRes) and includes various endpoints, authentication, and CORS configuration.
- The **frontend** should provide a simple UI to interact with the backend.

Backend Requirements

Endpoints

The backend should expose the following endpoints:

1. **Retrieve All Users from database (new users that the user created) and from the external api. Should be 2 different lists in the response.**
    - **Route:** `/getUsers`
    - **Method:** `GET or POST`
2. **Retrieve Specific User**
    - **Route:** `/getUser/{id}`
    - **Method:** `GET`
3. **Create a New User**
    - **Route:** `/createUser`
    - **Method:** `POST`
4. **Update a User**
    - **Route:** `/updateUser/{id}`
    - **Method:** `PUT`
    - **Notes:**
        - Validate input data.
        - Check if the user exists before updating.
5. **Delete a User**
    - **Route:** `/deleteUser/{id}`
    - **Method:** `DELETE`

Authentication

- Implement authentication to ensure that only authenticated requests are accepted.
- You can choose any authentication method (e.g., JWT, OAuth, API keys).

Error Handling

- Return appropriate HTTP status codes:
    - **200 OK** / **204 No Content** for successful requests.
    - **400 Bad Request** for incorrect input data.
    - **404 Not Found** for non-existent resources.

CORS Configuration

- Configure CORS to allow requests only from the following origins:
    - `http://localhost`
    - `https://www.google.com`
    - `https://www.facebook.com`

Design Patterns and Code Quality

- Follow best practices for clean, efficient, and maintainable code.
- Utilize relevant design patterns where applicable.
- Consider creating a **User UML Diagram** to plan your implementation.
- Ignore the `support` object received from API calls – use only the `data` object.

Data Storage

- You can choose between:
    - Storing users in a database (e.g., MongoDB, PostgreSQL, MySQL, etc.).
    - You should store new created users in the chosen database.
    - Managing user data in an application-level cache.

Frontend Requirements

Features

The frontend should provide a simple and user-friendly UI to interact with the backend.
Use any UI framework/library of your choice (e.g., React, Angular, Vue).

***Should be a responsive UI to support all screens. (Do not use media query)**
Implement Redux state management within your code.
Make sure to provide user indications about errors and validations for best UX.

1. **List of Users (mark created users by a unique color/icon to your choice)**
   - Display a paginated list of users retrieved from the `/getUsers` endpoint.
2. **View User Details**
   - Clicking on a user in the list should display detailed information in a modal using the `/getUser/{id}` endpoint.
3. **Create User Form**
   - Provide a form to create a new user via the `/createUser` endpoint.
4. **Update User Form**
   - Implement a form to update user details using the `/updateUser/{id}` endpoint.
5. **Delete User Option**
   - Include an option to delete a user via the `/deleteUser/{id}` endpoint.
6. **Error Handling**
   - Display appropriate error messages for unsuccessful backend requests.

Submission

Please complete the assignment and upload both the **backend** and **frontend** solutions to your GitHub repository.

- **Submission Steps:**
  1. Push your code to a public or private GitHub repository.
  2. Reply to this email with the link to your repository.

Good luck! If you have any questions or need clarifications, feel free to reach out.