Coding Challenge for mozok Internship

The idea of the challenge is to evaluate your skills in **Typescript**, **React**, and **NodeJS**, so that we can understand your knowledge better and create a more suitable internal academy for you.

The challenge is divided into two separate parts which you can find on the next pages, one for **Frontend** (Page 2) and one for **Backend** (Page 3). Ideally, you can solve both parts and connect them together, where the **Frontend** you are building requests the **API** you are building in the **Backend** part.

However, in case you have knowledge only in one specific part, feel free to do only that part which you feel most comfortable coding in, either **Frontend** or **Backend**. If you choose to do only **Frontend**, you would need to select and implement an already existing **API** from the ones specified below.

The coding challenge is expected to be completed within **3 days** of receiving the challenge by email.

Tips & Tricks:

- **Write simple and clean code**
  - Avoid long and complex implementations, divide and conquer
  - Use self-explanatory naming for variables, functions, classes etc.
  - Add comments only if necessary to explain custom logic

- **Feel free to research for solutions online**
  - Use Google, use libraries from NPM or other open source repositories
  - Try to understand them, do not simply copy-paste solutions
  - Apply them correctly for your use-case

- **There are no right, nor wrong answers**
  - We all here to learn and grow
  - Show us everything you’ve learned so far
  - Create the solution in ways how you think it should be done

Frontend Challenge:

Create a new React project in Typescript with Create React App or NextJS, the choice is yours. For the UI, feel free to use Material UI built components or custom styled components, whatever you prefer.
The website should have 2 pages with proper routing:

1. **List of Quotes Page, i.e. /quotes**
   a. Fetch the list of quotes from an **API** and display them in a Material UI or custom table;
   b. The table should consist of 3 columns, the **ID** of the Quote, the **Quote** itself, and the **Author** of the corresponding quote;
   c. Add a button to get a random quote, the button should redirect to the random quote page defined in (2.);

2. **Random Quote, i.e. /random-quote**
   a. Fetch a random quote from the **API**;
   b. Display the random quote in the centre of the screen with the author below it entered as well;
   c. Add a button to go back to the list, and a button to get a new random quote;

3. **Enrich the Quotes table with additional data**
   a. To make things interesting, for each row in the quotes table from **(1.)**, make an **API** call to https://nationalize.io/ by passing the first name from the Quote Author i.e. https://api.nationalize.io?name=michael for Michael;
   b. The above response will give you list of probabilities of Michael having a certain nationality with a list of nationalities;
   c. In a new table column i.e. **Nationality**, add a Flag Icon that corresponds to the nationality with the highest probability from the **API** response for that row. If no response is present for a name, you can use any placeholder icon or leave it blank.

The API used to fetch the list of quotes should be either the custom API from the Backend challenge above OR use the free API from https://zenquotes.io/ (see documentation).

- In case you have decided to go for both Frontend and Backend challenges, use your own API that you will build in the Backend challenge;
- In case you have decided to go only for the Frontend challenge, use the API from https://zenquotes.io/ and follow their documentation;

Backend Challenge:

Create a new Node server in Typescript with Express, Fastify, or NestJS, the choice is yours. Inside, create an API that consists of the following endpoints:

1. **Get Quotes from ZenQuotes and save them in a DB**
   a. The server on startup gets list of quotes, from the https://zenquotes.io **API** and saves the quotes in a **MongoDB** database.
   b. There shouldn’t be any duplicate quotes in the database, save only the new ones on each restart of the server.
   c. Optionally, there can be an endpoint that triggers a new fetch of fresh quotes once requested i.e. **/quotes/generate** to avoid the need of the server to be restarted.

2. **Get List of Quotes Endpoint, i.e. /quotes**
   a. The endpoint should fetch the previously saved list of quotes from the database;
   b. The endpoint should have an optional **limit** query parameter to limit the number of quotes sent to the client;
   c. Process the data from the database correctly and pass it to the response;
   d. Use proper http response codes for the API for success and failure;

3. **Get a Random Quote Endpoint, i.e. /quotes/random-quote**
   a. The endpoint should get one random quote from the previously saved quotes from the database;
   b. Process the data from the database and pass it to the response;
   c. Use proper http response codes for the API for success and failure;

Make sure the API returns the correct information both the **Quote** and the **Author**.

In case you have decided to go for both Frontend and Backend challenges, use the API built here for the Frontend challenge above to display the quotes.

Please provide the solution to the challenge on a public repository on GitHub and provide a link to the repository by replying to the email with a short description of the solution and the challenge you picked as a subject i.e. **Frontend Solution** or **Frontend + Backend Solution**.

Happy Coding!