Muzz Backend Technical Exercise

Summary

This exercise is deliberately open ended, you can spend as little or as long on the exercise as you want. There is no obligation to spend more than an evening on this exercise but we encourage you to take as much time as you feel you need.

We'd like you to build a mini API that could power a very simple dating app.

The functionality is split into 3 parts and an optional bonus. Each part will involve writing some Go and building a small database.

You don't need to build a frontend interface.

Your API endpoints should be available locally - e.g. http://localhost/user/create

If you have any questions, please do not hesitate to ask.

Tools to use

Please implement the task using the latest version of Go.

You're free to use any database technology you want - at Muzz we use MySQL, DynamoDB, Elasticsearch, Redis and Redshift for different workloads.

Please containerize your work so it's testable without having to install any other tools (assume we've got Go and Docker installed as a minimum).

Part 1 - The Basics

i) Write an endpoint to create a random user at **/user/create**

- It should generate and store a new user.
- It should return these fields: *id, email, password, name, gender, date of birth.*

The expected response should be equivalent to:

```
{
    "result": {
        "id": <integer>,
        "email": <string>,
        "password": <string>,
        "name": <string>,
        "gender": <string>,
        "age": <integer>
    }
}
```

ii) Write an endpoint to login at **/login**

- Return an appropriate error if login fails.
- Ensure that all other endpoints are appropriately authenticated.

The expected response should be equivalent to:

```
{
     "token": <string>
```

```
}
```

iii) Write an endpoint to fetch profiles of potential matches at **/discover**

- It should return other profiles that are potential matches for this user - as there is no filtering login currently, just return all other registered users.
- Exclude profiles you've already swiped on.

The expected response should be equivalent to:

```
{
    "results": [
        {
            "id": <integer>,
            "name": <string>,
            "gender": <string>,
            "age": <integer>
        },
        ...
    ]
}
```

iv) Write an endpoint to respond to a profile at **/swipe**

- You should specify the other user id + a preference (YES or NO).
- It should store and return if there was a match (both users swipe YES).

The expected response should be equivalent to:

```
{
    "results": {
        "matched": <bool>,
```

```
        "matchID": <integer>
    }
}
```

Note: **matchID** must only be included if **matched** is **true**.

Part 2 - Filtering

i) Extend **/discover** to filter results by age and or gender.

ii) Extend **/discover** to sort profiles by distance from the authenticated user.

- You will need to add location to the user model.
- Add a "distanceFromMe" to the **/discover** results.

iii) Bonus: Extend **/discover** to sort profiles by attractiveness.

- You will need to come up with a ranking based on swipe statistics.

Deliverables

Please **email** us the following:

1. **README.md**
    a. Tell us how to set up & run your API.
    b. Include details that set you apart. Feel free to show off.

2. **Solution_Your_Name.zip**
    a. A ZIP folder containing your solution (code, schema etc).
    b. Be sure to include the .git repository.

Alternatively a GitHub link can be provided to allow us to clone the repository. Be sure the README.md file is present!

What are we looking for?

Be sure to:

- Detail all assumptions made in your implementation and decisions in your readme
- Show good architecture, brevity of code, comments and understanding of the problem
- Show you are a solid mid/senior engineer
- Show off some good language techniques you are aware of - always good to stand out in a good way!
- Don't over engineer things - KISS.

What will I get back?

We pride ourselves on respect and speed of decision making to you as a candidate. We promise to:

- Have senior engineers review your exercise ideally within 24-48 hours
- Provide detailed USEFUL feedback of the good, the bad and the ugly of your solution
- Progress to final interviews if we deem the exercise to be satisfactory
- Decision 24 hours after final interviews