Technical assessment

Import and display a company’s employees list

Thanks for your interest in Elevo. You will find thereafter a technical test which goal is to assess your knowledge of the tools and frameworks we use in our day-to-day development, in addition to your broader intuition, including product design.

Goal

We don’t expect you to necessarily know everything beforehand, nor to even answer all the questions or have them working perfectly (targeting 80% or something is alright).

The environment of this test is designed to be close to day-to-day things. Maybe you don’t know about something, but you will have to learn and solve it quickly with all the resources at hand.

Instructions

Duration: This test should take you around 3-4 hours (not including setuping libraries etc.).
How to submit: Just commit to a private GitLab repository and share the access to leo@elevo.io.
When you are finished: Just send us an email to let us know.

After the test, we collect everything in the Git repository as an answer, and we will get to discuss it together.

Part 1: Finding likely duplicates in a user list (around 1h)

Keep in mind this section is mainly to assess basic programming, basic API design, algorithmics and unit testing. Other considerations are less important.

The goal is to find possible duplicates in a list of users with similar names ignoring casing issues, in the most efficient manner time-wise. E.g. given

```
full_name
Delphine Chartier
Barbara Decaux
Adrien Sardaban
Delphine CHARTIER
sardaban adrien
Basile Dirac
adrien sardaban
```

Print

```
Delphine Chartier
Adrien Sardaban
Delphine CHARTIER
adrien sardaban
```

Questions

1. Write a Ruby script that finds likely users duplicates based on the casing of their full names.

   In particular, running your script in the current directory with the example input should give the example output.

   Don't worry too much about the parsing or being resilient to pathological inputs, the main interest here are the algorithmical implications and efficiency.

2. What is the time and space complexity of your program in O(n) notation? Can you find an asymptotically better version i.e. a better O(n) complexity?
   a. If you don't know what O(n) means, you can check out https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/. We will also get a chance to discuss it during debrief.
   b. If you are using a library, make sure you understand how this library works under the hood. If you don't know, maybe try to take a look at its source code (or make an educated guess).

3. How would you unit test it? You don't have to write the actual test code, rather examples of the test cases you would use.

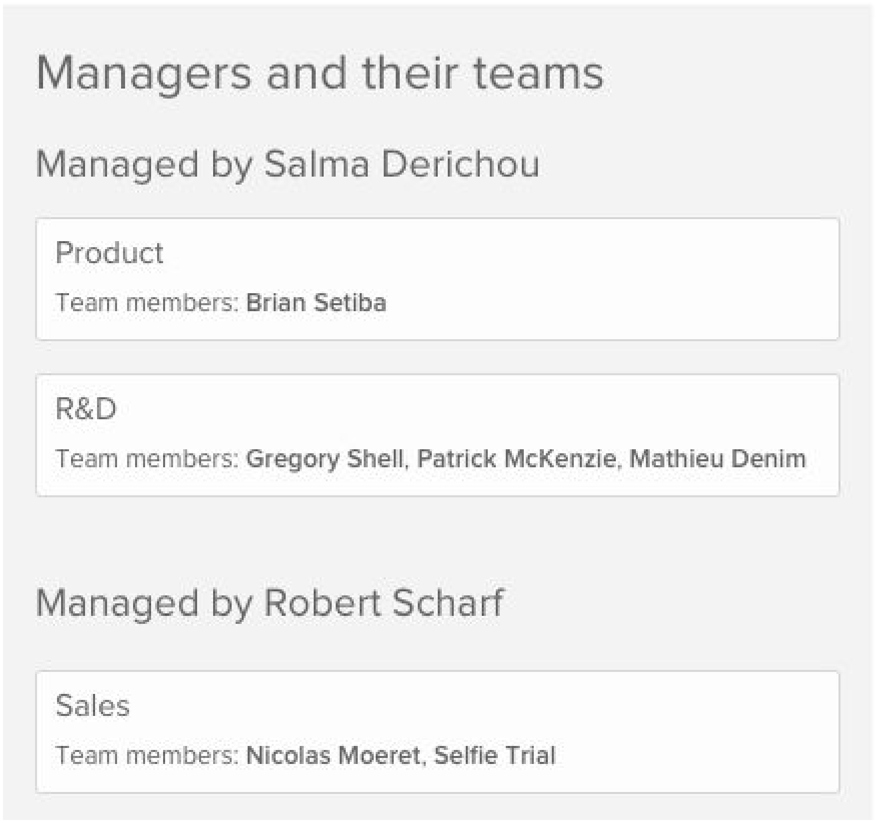
Part 2: Displaying employees as a hierarchy (around 2h)

Keep in mind this section is to assess the full-stack fluency, code organization and how compelling the result is to the end user, while working around potential ambiguities.

We will be more considerate of attempts to pick up React and/or Ruby on Rails even if you are not familiar with it.

The reason for this is that this is what we use and we think that even if you are not familiar with it, you should be able to pick it up quickly enough to be able to do this exercise (albeit not in a perfect or idiomatic way).

Our goal in this part is to display a company organization from a web-service according to this intentionally vague “specification”:


Managers and their teams

Managed by Salma Derichou

Product

Team members: **Brian Setiba**

R&D

Team members: **Gregory Shell, Patrick McKenzie, Mathieu Denim**

Managed by Robert Scharf

Sales

Team members: **Nicolas Moeret, Selfie Trial**

1. Create a small web server that serves a static JSON endpoint:

   ```
   GET /teams.json
   ```

   The latter should returns a hierarchy of users organized by teams:

   ```
   {
     "teams": [{
       "name": "Sales",
       "manager": "Robert Scharf",
       "members": ["Nicolas Moeret", "Selfie Trial"]
     }, {
       "name": "Product",
       "manager": "Salma Derichou",
       "members": ["Brian Setiba"]
     }, {
       "name": "R&D",
       "manager": "Salma Derichou",
       "members": ["Gregory Shell", "Patrick McKenzie", "Mathieu Denim"]
     }]
   }
   ```

2. Write a single page application that can be served as a static website (HTML CSS JS files, use whatever libraries you want) that requests this endpoint and shows the result according to the wireframe above.

3. We assume a SQL server that contains a table according to the following schema :

```
CREATE TABLE users (
  id INT NOT NULL PRIMARY KEY,
  team_name VARCHAR(255) NOT NULL,
  full_name VARCHAR(255) NOT NULL,
  manager_id INT,
  FOREIGN KEY (manager_id) REFERENCES users(id)
);
```

   Create a small HTTP server that produces the API endpoint above, now fetching data from the table.

4. Plug your frontend and backend together to confirm it works.

5. The table `users` described before is intentionally ambiguous with respect to the UI. A team could have multiple managers, for instance. How would you refactor the SQL schema?