Thank you for your interest in joining the SalesLoft Engineering team! We take hiring and interviewing very seriously as we grow, and also pride ourselves on openness and transparency.

Our interview process consists of having candidates build a small application ahead of time, and then coming onsite to present it and pair program on adding a feature to it. It ends up being a really great real-world experience in what it's like to work here on a day to day basis.

We've broken the exercise into levels that get progressively more complex, so please spend the amount of time on it you feel reasonable. It isn't necessary to completely finish to be considered - we're more concerned with the fundamentals of good software design.

We'd also be delighted to see complete implementations if that's what you're comfortable with.

Please reach out with any questions at all along the way.

Instructions:

You will be creating an application in the language/framework of your choice that integrates with the public SalesLoft API. We've provided an optional Ruby+React starting point, but you are free to use whatever frameworks you want.

Review the documentation for our API here: https://developers.salesloft.com
SalesLoft Ruby+React Starting Point: https://github.com/SalesLoft/DevelopmentInterviewStarterKit

Challenge:

Create an application that lets you manage SalesLoft People data.

We will provide you with a Salesloft API key, please allow the API key to be loaded in your application through an environment variable. Please do not check the API key into version control, we will be checking for both of these.

Level 1: Show a list of People records that are available via the API. Display each Person's name, email address, and job title.

Level 2: Create a button that, when clicked, displays a frequency count of all the unique characters in all the email addresses of all the People you have access to, sorted by frequency count (the count below).

For example, if there was only 1 Person record returned by the API with the email address "jimmy_mcmahon@salesloft.com", it would show a frequency breakdown that looks like:

Character	Count
M	5
J	3
O	1
…	...

Level 3: Create a 2nd button that would show us a list of suggested possible **duplicate** People. A human can tell that "benoliv@salesloft.com" and "benolive@salesloft.com" are very likely the same person with just one of the email addresses having a typo. However we would like you to decide what might constitute a duplicate - up to you.

Level 4: Build a scalable enterprise-ready underlying architecture for this application that would scale to handle the function in Level 2 and 3 for trillions and trillions of records while returning results in a reasonable amount of time (< 200ms).

Notes/ Recommendations:

- Just kidding on level 4 - but be thinking about how you might tackle something like that when writing the code!
- Make commits often and at various stages to show your train of thought
- It is fine to use standard libraries/gems/plugins that go into building an application, but not anything that does the core logic in the counting or de-duping task for you, should that exist somewhere.
- Write code as if it were going to be a long term project with lots of users. After all, that's what our product needs to be.
- Use all the best practices you're familiar with - well tested, extendability, encapsulation, DRY code, readable, modifiable, scalable, etc, etc.
- The application itself does not need to fully work to be submitted - but you should get far enough into it to produce an amount of code that you'd be comfortable presenting to a team, talking through why you did certain things, etc.

*** When finished email the github url to nora.ignatius@salesloft.com**