Take-home assignment for backend engineers

Hello, we really enjoyed our conversation and would like to ask you to complete a little take-home assignment. Working on your own and then presenting a solution would be a part of your work with us, so we would like to learn more about your style.

This challenge is:

- **Build an interview calendar API.**

It's okay to only implement the API and skip UI altogether. It's also fine to skip the authentication. The purpose is not to build a solid API but for you to demonstrate your coding skills and engineering practices.

There may be two roles that use this API, a candidate and an interviewer. A typical scenario is when:

1. An interview slot is a 1-hour period of time that spreads from the beginning of any hour until the beginning of the next hour. For example, a time span between 9am and 10am is a valid interview slot, whereas between 9:30am and 10:30am it is not.
2. Each of the interviewers sets their availability slots. For example, the interviewer David is available next week each day from 9am through 4pm without breaks and the interviewer Ingrid is available from 12pm to 6pm on Monday and Wednesday next week, and from 9am to 12pm on Tuesday and Thursday.
3. Each of the candidates sets their requested slots for the interview. For example, the candidate Carl is available for the interview from 9am to 10am any weekday next week and from 10am to 12pm on Wednesday.
4. Anyone may then query the API to get a collection of periods of time when it's possible arrange an interview for a particular candidate and one or more interviewers. In this example, if the API is queries for the candidate Carl and interviewers Ines and Ingrid, the response should be a collection of 1-hour slots: from 9am to 10am on Tuesday, from 9am to 10am on Thursday.

Remember: the purpose of this test is not to implement a production-ready API but to create basis for our further discussion.

We will assess the project from a few different points:

Architecture. You're welcome to use any tools, frameworks, libraries, and third-party services, and organize the code in a way that you believe suits best. Here, we'll look at the data model, discuss possible alternatives, and perhaps exchange a few ideas on making the combination even stronger.

Engineering. We'll assess this point based on the actual code: how it's organized and written, if it's easy to get picked up by a new engineer in a team (or in community, if we talk about open source). Of course, it's important that the code is readable, but we'll go beyond that and talk about coherence of the codebase and its hypothetical (or potential) future.

Tooling. How does a completely unfamiliar user with basic technical background get the API, install it, add a couple users and slots and get time overlaps for various combinations of candidates and interviewers? What programs are expected to be available on user's computer before installing the tool? How could that user potentially do a micro-deploy on their own machine?

Documentation. Good code is one that documents itself, but sometimes it isn't enough. We'll discuss things such as quick start guide, code style and contribution guideline. It's important that a new team member that joins the team and gets introduced to the codebase, or another team that develops integration, has really good time exploring the code and the API itself.

Please put it somewhere so we could try it. A repo on GitHub would be the best.

We're using Spring Boot as a backend framework, but you can use this one or any other. The only thing we ask is to use Java.

Thank you for taking the time, and we are looking forward to our next interview that will follow up on this little project.