Backend Engineer Assessment

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 interviewer sets their own availability slot. For example, the interviewer Philipp is available next week each day from 9am through 4pm without breaks and the interviewer Sarah is available from 12pm to 6pm on Monday and Wednesday next week, and from 9am to 12pm on Tuesday and Thursday.

3. Each candidate sets their own 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 to 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 Philipp and Sarah, the response should be a collection of 1-hour slots: from 9am to 10am on Tuesday, from 9am to 10am on Thursday.

Please feel free to stop at any point when you believe it is reasonable. We recommend putting about 1–1½ hours of your time. Remember: the purpose of this test is not to implement a production-ready API, but to create a basis for our further discussion.

We will discuss the project from a few different point of views:

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 look at the data model, discuss possible alternatives, and perhaps exchange a few ideas on making the combination even stronger.

Engineering. We'll discuss 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 such things 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 easily; a repo on Github would be the best.

Meeting every single point is not necessary; in fact, we only expect just any two of them. We would love to know what you are strongest at. Remember, we are aiming to build a great cohesive team! There will be a lot of opportunities to learn and go through peer-driven quality review, so we care way more about what knowledge and experience you bring to the team.

Thank you for taking time and appreciate your enthusiasm in completing the challenge.
We are looking forward to our next interview that will follow up on this little project. Meanwhile, feel free to address any questions you might have.