Dear Deepgram Engineering Candidate,

In preparation for your upcoming interview, we would like you to prepare the following coding project.

Build a simple API server to handle user audio projects. Your server should provide endpoints that allow a user to perform the following actions:

1. POST raw audio data and store it.
   For example:
   - `$ curl -X POST --data-binary @myfile.wav \ http://localhost/files`
   - `Add metadata to this file when uploading`
2. GET a list of stored files, GET the content of stored files, and GET metadata of stored files, such as the duration of the audio. The GET endpoint(s) should accept a query parameter that allows the user to filter results. Results should be returned as JSON.
   For example:
   - `$ curl http://localhost/list?maxduration=300`
   - `$ curl http://localhost/download?name=myfile.wav`
3. GET a summary of the uploaded file using an LLM
   - `$ curl http://localhost/info?name=myfile.wav`
   - `*It is completely fine to mock out the integration of the summary.`

Please consider the specific routes above as examples, and feel free to design the API as you see fit, provided that it covers the specified POST and GET functionality. As you encounter decisions in your implementation, please feel free to make and document assumptions as needed.

When you arrive for your interview, we will ask you to present your code and elaborate on design decisions you made, things you would have done differently if you had more time, what you learned about libraries you used, and more. Additionally, we will ask you to build upon the code you've worked on to add functionality.

Your code should minimally be able to handle the GET and POST requests described above. However, because we want this to be a point of departure for a slightly bigger project you will do during your in-person interview, here are some features and questions you might want to think about:

1. If this project was bigger, how would you use an llm orchestration framework?
2. How would you validate and monitor the quality of AI outputs to prevent errors or hallucinations?
3. When would you use RAG?
4. If tomorrow, we wanted to support multiple AI LLM providers, how would you architect the backend to make swapping models easy?
5. AI inference can be expensive and slow. What strategies would you use in the backend to cache, batch, or route requests to balance cost and performance?
6. How do you want to store audio data? For the purposes of this interview, just keeping them in

memory is fine, but how else would you want to keep and serve audio data?
7. How would you handle user authentication and data security?
8. How would you handle data integrity? How do you make sure that users can't break your API by uploading rogue text data?

Thanks for your time and your earnest effort on this small project. We look forward to seeing what you've done.

The Deepgram team