interview-challenge.md

FullStack Coding challenge

Welcome to our Coding Challenge. We can't wait to see, what you will create!

Why are we doing this?

We want to value your time. So, before putting you on long job interviews, we first check if it's worth it for you ;).

With this test, we:

- Evaluate your coding abilities
- Estimate your technical experience
- Understand how you design a solution
- Give you an opportunity to impress us with your developing skills

What we will take an eye on:

Your developing skills will be scored on:

- Coding standard, comments and style (Readability, i.e. lintintg)
- Overall solution design (Clean architecture is more important than 100% working business logic)
- Testing strategy (Testability)
- How comfortable you are with TypeScript
- Appropriate use of source control (Standard commit messages)

Here's what you should do:

- Please use **Typescript**, **NestJS** and **React**
- Please put the challenge results in a public code repository hosted on Github
- Once the test is completed, please share the Github repository URL to the Recruiting team so we can review your work

Timeline

Please hand in your results within **5 days**, starting the day after you've received this link. If you can't start immediately or, for some reason, you need more time, please send us a message.

Here comes the challenge - DIY a Gmail of your own 

- Create a solution that reads, write and set messages status that mimics and email client and server
- The messages should be shown in a web browser as user friendly as you can make it and mimic an email client (i.e. Gmail, Hotmail, Outlook, etc)
- It is mandatory that the data is stored in a database and exposed from a server (RESTful API, GraphQL)
- The frontend client has to show real-time interactions from the server *(hint: Websockets, GraphQL subscriptions, polling, Server-Sent Events)*

Let's talk about technical details

Backend:

For the backend of the app, please use **Nest.js** to build a server that interacts with a **MongoDB database**.

The email data would be stored in a collection in the database, with each email represented as a document. The API would provide endpoints for the various actions required by the app, such as fetching a list of emails, adding a new email, updating the read/unread status of an email, deleting an email.

Each time a new message is received the server should handle a way to update the frontend client. For the real-time update of messages, choose the best tool you can find to handle the connection between the server and the client. ***When a new email is received, the list of emails displayed in the app should be updated.***

Frontend:

For the frontend of the app, please use **React** to build a single-page application (SPA). The app should handle the routing between different views (Inbox, Sent and Trash). When the user clicks on an email the app navigates to a single email view and displays the email's content. The app would also have a compose email view, which would be displayed in a modal when the user clicks the "Compose" button.

To handle the real-time update of emails, choose the best tool you can find to create a real-time connection to the server. When the server sends an update, the frontend app would update the list of emails displayed in the app.

Code Readability:

To ensure the code is easy to read and understand, please follow best practices for writing clean and readable code, such as using descriptive and meaningful variable names and using appropriate formatting and indentation.

Testability:

To ensure the code is testable, please write unit tests for the backend using your preferred testing framework and write integration tests for the frontend using a testing library. Write end-to-end tests using a tool of your choice to test the overall functionality of the app.

Use any other third party library of your choice, if needed.

Our minimal requirements:

Please remember that we are not expecting a fully functional web application! Just your thoughts on project structure and architecture and the features it will have. In fact we won't even open the browser. However, when running for example "npm run dev" or "npm test", no coding errors should appear.

- The overall (clean) architecture of the app
- One full feature of the challenge (for example Email Listener, Compose Email, Read Email, List Emails)
- Test structure for the feature presented

- Usage of Docker (Separation of services)

** Thanks for taking the time to read all this! We're super excited to see your results. Happy coding :) **