CLIPBOARD
HEALTH

See FAQ

Hey Yuxiao Sun, we're excited for you to interview with Clipboard Health!

This challenge is a chance to show your programming skills and understand the kind of problems you'd be working on day-to-day.

Fetch the most active workplaces

This assessment involves fetching and processing data from the public API of a web app. The tech stack is Typescript/Node, NestJS/Express, and Prisma/PostgreSQL. Unlike many assessments, we're trying to evaluate your overall skills, not just programming. As a result, you might find that, unlike other assessments, the prompt is a bit vague and requires deeper thinking about the business domain. That's intentional!

You'll have 24 hours to complete the assessment. You should expect to spend anywhere from 25 minutes (strong engineer with an AI copilot) to 75 minutes (junior engineer working independently).

If you're not familiar with TypeScript, that's fine! This assessment doesn't require any TypeScript-specific skills, knowledge, or concepts, although it may be helpful to make sure you understand the TypeScript `async` keyword before starting. Since this is an async assessment, you're welcome to use any resources, including online searches or AI assistance, in completing the assessment.

Be aware that solving this assessment with AI still requires clever prompting and understanding of the business domain, and that if you pass the assessment, we will be pushing on your understanding of any AI-generated code in a live interview.

Start Assessment

Fetch the most active workplaces

Repository https://github.com/hatch...

In this assessment, you will be submitting your solution using `git`. To do so, you will need to clone the repo linked above, and push all your changes to the `dev` branch (which is also the default branch). You will be submitting this assessment by creating a pull request between the `dev` and the `main` branch.

If you need help with how to clone the repo to work on the project locally, see this **link for help**.

Implement the top-workplaces script to fetch ...

Our PM wants to solicit feedback from the customers who have had the most experience using our platform in order to learn more about what they like and don't like, understand worker quality, etc.

We don't currently have an analytics solution in place, so as a stopgap, our PM has asked us to pull analytics on the top 3 workplaces and the top 3 workers from our running app in production in order to contact them.

More specifically, we want the currently active workplaces with the most shifts completed, and the currently active workers with the most shifts completed. For more context on the way our business and application operates, please see `README.md`.

Since these are one-offs, we're building them as a script which we can run locally against our production app. A placeholder for the first script is in `src/scripts/top-workplaces.ts`. You can run this script by running `npm run start:topWorkplaces` in the root directory.

A placeholder for the second script is in `src/scripts/top-workers.ts`. You can run this script by running `npm run start:topWorkers` in the root directory.

Your task is to implement these script using the existing public web API for the app. Aside from possible bugfixes, avoid modifying the existing APIs or adding new APIs.

The output of the script after running `npm run start:topWorkplaces` should be formatted like so:

```
[
  { name: "Martian Hydro", shifts: 10 },
  { name: "Luna Greens", shifts: 7 },
  { name: "Red Diamond Mines", shifts: 6 }
]
```

The output of the script after running `npm run start:topWorkers` should be formatted like so:

```
  [
    { name: "Dmitri David", shifts: 10 },
    { name: "Chike Williams", shifts: 7 },
    { name: "Fatima Khan", shifts: 6 }
  ]
```

Exercise Notes

- We'll run an automated test suite against your solution. As a result:
  - `npm run start:topWorkplaces` ***must*** run your topWorkplaces script.
  - `npm run start:topWorkers` ***must*** run your topWorkers script.
  - Your script output **must** adhere strictly to the format shown above. While in the real world a PM would be able to interpret a slightly different result, for this task we can't support that.
  - Because we run an automated test suite against your output, **failing to clean up debug statements that use console.log will result in an automatic failure.**
- As much as possible, treat this PR submission the way you would your first PR at a new company. At Clipboard, we care about:
  - Completing the given task accurately
  - Committing high quality code that is easy to review, maintain, and extend
  - Providing helpful PR descriptions to make code easy to review and help future developers understand the context
- Focus on making your solution functional rather than optimizing for performance
- We recognize that it's now commonplace to use AI copilots as a routine part of an engineer's job and expect you might leverage them while solving this problem, no problem! However, in the live interview you should expect that we will ***rigorously dig*** to understand and ***scrutinize*** how you got to your solution, will ***bias toward skepticism*** if you can't explain and iterate on your solution on your own, and ***will ultimately reject candidates*** who are over-dependent on AI.

Submitting Your Solution

Open a pull request from `dev` into `main` once you are ready to submit. Do **not** merge the pull request. Here is a step-by-step guide on how to create a pull request.

When you have completed this task, click Ready To Submit. READY TO SUBMIT