Countingup Pairing Exercise

Overview

Thank you for choosing to interview with Countingup for our Full Stack Developer role. We hope you find the experience interesting and rewarding.

Our technical interview takes the form of a two-part guided pairing exercise, where you will be expected to work with a member of our team to code a solution to the problems outlined below. There will then be a session of follow up questions, including a chance for you to ask us questions as well. Whilst the core problem to be solved is relatively trivial, the exercise is designed for you to showcase your current skills, as well as demonstrate how quickly you can learn something new.

Backend

Language selection

The first part of the exercise can be completed in a number of programming languages. We prefer languages that provide support for truly concurrent operations, languages we can support here are:

- Java
- C#
- Go (this is the primary backend language our team works with)

We can also support the following languages, though you should expect to answer follow up questions around why these do not provide support for truly concurrent operations, and how this makes the exercise easier:

- TypeScript or JavaScript (using node.js)

Task

In order to fit this exercise into the time available for an interview, we avoid asking you to implement an actual backend web service, and instead ask you to write a simple command line program that reads & writes files (the frontend exercise can then consume the output file). This program should:

- Read a list of URLs from a specific text file, one URL per line
    - (A sample text file with example URLs will be provided, but your solution should work for any such files.)

- For each URL in the file, in parallel (using the concurrency primitives provided by your language), fetch (HTTP GET) the URL recording:
    - the response status code (e.g. 200, 404)
    - the content length (from the response headers)
    - the time taken to complete the request in milliseconds (response time).
- Once all requests have completed, write the collected data to a json output file as an array of objects with the following format:

```
[
  {
    "url": "https://countingup.com",
    "statusCode": 200,
    "contentLength": 37203,
    "responseTime": 423
  },
  {
    "url": "https://www.gov.uk",
    "statusCode": 200,
    "contentLength": 35599,
    "responseTime": 283
  },
  ...
]
```

You should consider how you structure your solution to ensure it is testable, and write appropriate unit tests to verify correctness, using your language's unit testing framework.

Frontend

Language selection

Unlike the backend exercise, for the frontend exercise we are stipulating that the exercise be completed in TypeScript utilising the React library.

A skeleton React application will be provided as a starting point. You may find it helpful to familiarise yourself with the specifics of React ahead of the exercise, the library provides a handy tutorial, though this is not compulsory. If you have familiarity with another JavaScript framework (Angular, Vue) that should also be good preparation.

The skeleton application doesn't initially contain any TypeScript types but it's likely you'll need to define some basic structural types for the exercise. You shouldn't need anything more advanced than what's covered in this introductory document.

Task

The frontend exercise builds on the output of the backend exercise, and involves building a simple UI in React to display the data that was generated from that exercise. This can either be completed with your output file from the first exercise or, if required, a sample file will be provided.

Your React application should read the data from the json file, and then display this data as a table, including a header row with the field names. (The

file can be read from the local filesystem, and included within the application, there is no need to fetch it over http.)

You should add a summary row to the table that displays the average (mean) content length and average response time (you will need to calculate these). You should also add support for sorting the entries in the table by content length or response time, by clicking on the respective column headers.

You should consider how you structure your solution to ensure it is testable, and write appropriate unit tests to verify correctness. The Jest testing framework will be available as part of the skeleton React application provided.

A completed solution to the frontend exercise would look something like the following:

Countingup

URL	Status Code	Content Length ↓	Response Time
https://beta.companieshouse.gov.uk	200	19760	125
https://www.gov.uk	200	35624	108
https://countingup.com	200	37203	155
https://www.docker.com	200	82517	120
https://www.hashicorp.com	200	87719	190
https://stackoverflow.com	200	115022	383
https://twitter.com	200	311129	250
https://www.bbc.co.uk	200	319495	149
https://www.huckletree.com	301	447630	116
-	-	**161788.78**	**177.33**

Laptop setup

You should ensure you have your preferred IDE and toolchain setup for the language you have selected for the backend exercise (see below). For example a recent JDK and IntelliJ for Java.

For the frontend exercise you should ensure,

- Yarn 1.x is installed (https://classic.yarnpkg.com/en/docs/install). Note that we don't support npm or Yarn 2 in this exercise
- LTS version of Node.js installed (https://nodejs.org/en/download/)
- A suitable editor or IDE for web development. For example VS Code or WebStorm
- You have cloned the frontend exercise code from GitHub, https://github.com/Countingup/interview-exercises

Once you have the frontend exercise code on your laptop you can test everything is setup correctly,

1. Navigate to the `frontend` directory
2. Run `yarn` to install dependencies into `node_modules`
3. Run `yarn start`, this should open a browser at http://localhost:3000 showing a table with a single, hard coded row

Remote Interview Setup

If you're taking the exercise remotely you'll be on a video call with the interviewers. During the coding portions of the interview you'll need to share your screen on the call so we can see your code.

We appreciate that interviewing can be nerve-racking at the best of times and this can be made worse if you're taking this exercise remotely and your connection drops out, or we can't hear each other. To help keep everything running smoothly it's a good idea to

- Ensure your internet connection is stable, if practical use a wired (ethernet) connection
- If you're using a Mac make sure you've allowed Google Meet to access and share your screen, microphone and camera before the interview
- Ensure your microphone and speakers or headphones are working. Google Meet has a 'test' feature to try this out before the interview
- If possible find a quiet room with minimal background noise