React Pre Interview Coding Test

Introduction

Build the code as if this was just one of many tasks that you have to do in one day, using whatever libraries / frameworks you feel comfortable with unless explicitly instructed not to.

The tasks are fairly straightforward and should take no longer than an hour. Only spend enough time required to produce an appropriate, clean, testable and maintainable solution to the stated problem.
There are no major "gotchas" but there will be input and output that exist beyond the samples provided (as in the endpoint may return more or less records with varying values).

When finished, please create a repository and push the unzipped, raw source to a public bitbucket / github repository and then submit a link to this repository.
Please submit working source code to solve the problem along with any supporting code that you might have used in testing.

Task:

We have an api that gives us stock prices (and it's totally not just a random number generator**). Because this is super important data that is super duper important to our customers, we want to show them a log of up to date pricing as well as some aggregate historical data about their stocks. And because stock information is time sensitive, we want the log to update every 2 seconds.

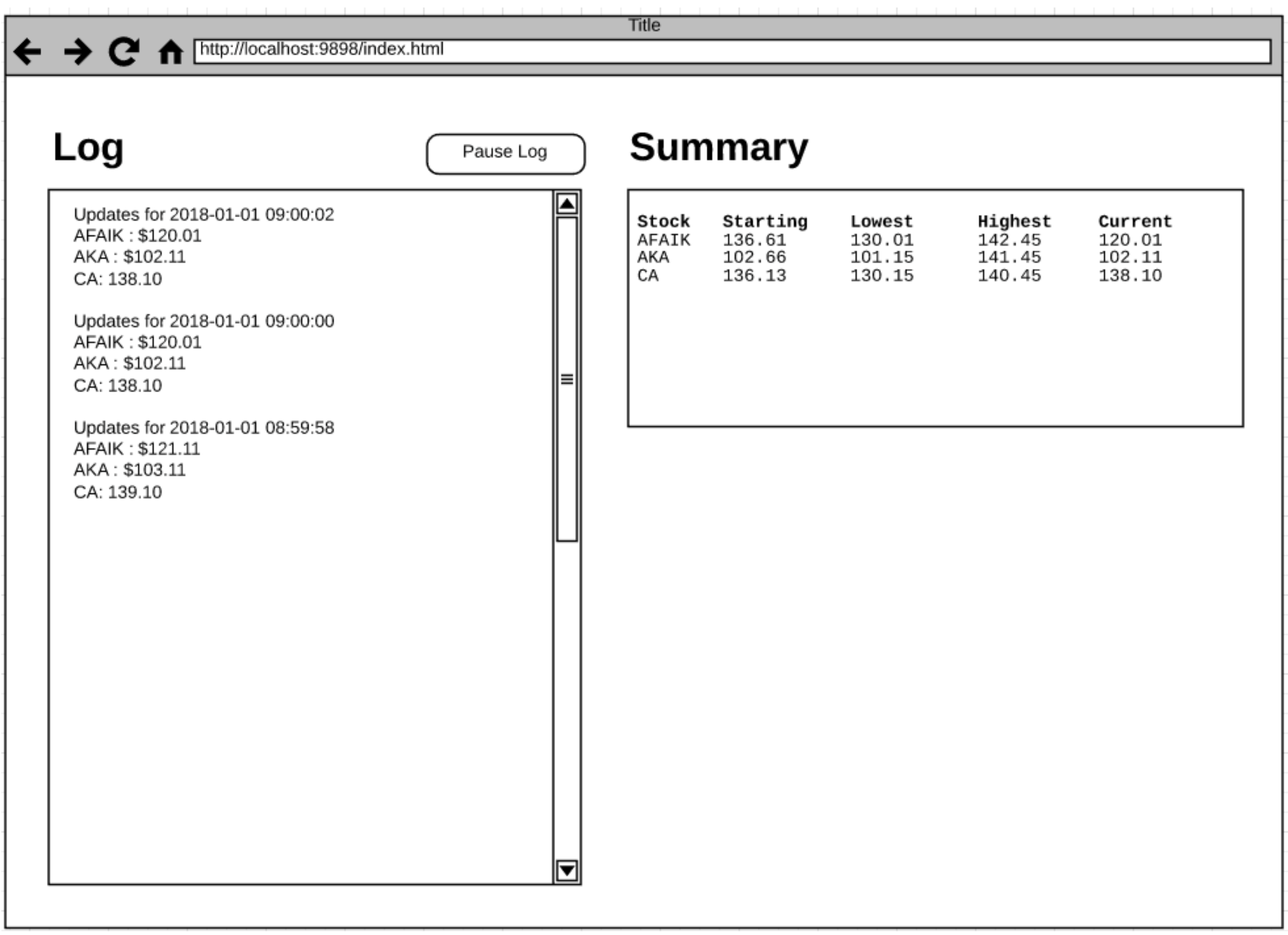
Requirements:

- Use Typescript.
- We have an endpoint at https://join.reckon.com/stock-pricing that returns data for our stock quotes. We expect you to create types for the api responses and then validate them (**Note**: using "as" does not count as validation). An example of the output is listed as *Appendix 1: Sample data*
- Given this data, create a page running in a Node.js server that will run when called at http://localhost:9898/index.html, use whatever UI layout framework you prefer and use boiler plate code where possible.
  Feel free to use any scaffolding / generating tools to get started if you want.
- Given the index.html rendered, we expect that 2 React components will be rendered on screen "Log" and "Summary". A sample screen layout is listed in *Appendix 2: Screen Layout.*
- Given these two React components, we expect to see them using some implementation hooks and/or context. Refrain from using 3rd party global state management libraries like Redux / Mobx
- Given the 'Log' Section in the rendered page, we expect to see this continually updated every **2 seconds** with the latest pricing information, with the newest information at the top of the screen
- GIven the 'Summary' Section, we expect to see an aggregate view of the data from the api that has been processed by the web app. This aggregate view should contain:
  - Starting Price
  - Lowest Price seen
  - Highest Price seen
  - Current Price
- Because there is a lot of data coming through the log, when we view the page, we expect to see a pause / resume button rendered as a seperate component.
- When the "Pause' button is hit we expect that the Log will **stop** updating, but we expect the summary view to **continue** to update and show current values
- When the "Pause" button is hit, we expect that the pause button will be have its text changed to 'Resume"
- When the "Resume" button is hit, we expect that the Log will again continue to update, but it will not list any stock updates since 'pause' was pressed.
- When the "Resume" button is hit, we expect that the text of this button will change to "Pause"

Appendix 1: Sample data

```
[
    {
        "code": "AFAIK",
        "price": 136.61
    },
    {
        "code": "AKA",
        "price": 102.66
    },
    {
        "code": "CAD",
        "price": 136.13
    },
    {
        "code": "GMAO",
        "price": 85.41
    },
...etc
]
```

Appendx 2: Example Sample Screen Wireframe



```
Updates for 2018-01-01 09:00:02
AFAIK : $120.01
AKA : $102.11
CA: 138.10

Updates for 2018-01-01 09:00:00
AFAIK : $120.01
AKA : $102.11
CA: 138.10

Updates for 2018-01-01 08:59:58
AFAIK : $121.11
AKA : $103.11
CA: 139.10
```

```
Stock   Starting    Lowest      Highest     Current
AFAIK   136.61      130.01      142.45      120.01
AKA     102.66      101.15      141.45      102.11
CA      136.13      130.15      140.45      138.10
```

** okay - it totally is just a random number generator for test purposes