Thank you for taking the time to complete our assessment.  To make this process a little easier on everyone, we have designed this coding exercise for assessing candidates we are considering for the TypeScript and Node development team.  As part of the next step, below and attached is the coding exercise for you to complete and send back.  Please don’t hesitate to reach out with any questions.

The solution must be developed in Typescript Language and must be able to run on Node.js platform.

**Overview:**

Sequences of work processed by data streaming architectures can be broken up into three essential steps:

1. Consume
2. Transform
3. Produce

The Consume step is simply to read in a message of a specified format into the work processor.  The Transform step contains the bulk of the business logic: using the consumed data as input, this step acts on the data, eventually transforming it into resulting output of a specific format.  The final Produce step writes out the output to a destination.

**Exercise**

For this exercise, you will find a "logs" zipped folder attached with the exercise email.

{

    "id":"603cdba2-232c-4c15-a6ad-c427b9a1a199",

    "logs":[

        {

            "id":"e68c089d-0215-459f-b245-b520949b10d1",

            "email":"[first.user@domain.com](mailto:first.user@domain.com)",

            "message":"successfully completed processing after 5 milliseconds"

        },

        {

            "id":"e8bc9136-1ac9-4176-b0f8-a0b36a149a08",

            "email":"[another.user@example.com](mailto:another.user@example.com)",

            "message":"failed to process due to datastore exception"

        },

        {

            "id":"37f6c3b8-dc81-4921-b2d0-5f5d69f12ef8",

            "email":"[first.user@domain.com](mailto:first.user@domain.com)",

            "message":"failed to process due to datastore exception"

        }

     ]

}

After downloading and unzipping the logs folder containing "logs" files from the attachment, for each file:

1. Read in the json content from the file.
2. Produce the json content as a log message.
3. Consume that json message.
4. Transform the json message into a tally of all the number of logs for each unique email address.
5. Produce a message that includes that tally.  Here is an example of a tally message:

{

"logs\_id":"603cdba2-232c-4c15-a6ad-c427b9a1a199",

"tally":[

{

"email":"[first.user@domain.com](mailto:first.user@domain.com)",

"total":2

},

{

"email":"[another.user@example.com](mailto:another.user@example.com)",

"total":1

}

]

}

1. Maintain a global tally of all log counts for each unique email address
2. Consume each tally message produced by step 2.
3. Update the global tally according to the additions included in the message.
4. Print out the updated global tally.

Other notes:

You are free to use any 3rd party packages and libraries to accomplish the task. Just make sure they are tracked through the package.json file.

And that’s it!  When you are all done, upload the zip to Google Drive and share it with us (Please do not upload to a public repository!). Please make sure we can share it with others (allow editors to share).  Please also step back and ask yourself if your submission demonstrates a strong understanding of OO Design. Sometimes we race so fast to the problem, we don't step back and refactor to use the right OO design approach, so check for that. Also, if you have some improvements and suggestions on your own solution, do not hesitate to note that for us.

Feel free to send us any questions you might have.