REVERSE PROXY

This example of a reverse proxy has been done using NodeJs and ExpressJs. The reason for using this framework is that it is the one which I am more familiar with, avoiding the need to learn a new framework just for the purpose of the assignment.

First of all, the 2 servers were created: they are identical, running on different ports (the addresses are the same, but changing them will not affect the functionality of the program). There are just two simple operations done by the servers: “sumNumbers” and “getData”.

- **/sumNumbers**: it is a POST operation, that takes as input two numbers and returns the sum of these values. Plus, it returns the server address that computed this calculation, in order for the client to check which server was involved.
- **/getData**: it is a POST operation as well, and what it does is simply retrieve some random data taken from a mock API (*https://jsonplaceholder.typicode.com/*). This operation was implemented to test the caching in the proxy.

After these two servers were done, I have implemented the reverse proxy. Every request done by the client have to pass first through the proxy before reaching on the target servers. In this proxy many operations could be done, both to the requests and the responses. For example: when the “sumNumbers” operation is called, the proxy verifies which load balancing strategy to use in order to choose which server will handle the request. Two strategies are implemented: Random and Round Robin. The first one just randomly picks one of the servers from the array and passes on the request, the latter picks sequentially a server from the array for subsequent requests that arrive to the proxy. In both cases, headers are set, so that the server is able to read JSON data.

The proxy even gets the responses from the server before passing it back to the client. This is done only for the “getData” operation, so that it can store the response in the *cache*. The cache remains stored for only 5 seconds, so that if the same request is done within this timeframe, there is no need to query the server, it simply takes the stored response from the cache and returns it back to the client, saving bandwidth usage and reducing latency.

Caching could also be done on the client side, saving them directly into the browser, avoiding the need to reach the proxy, reducing even more the response time.

ReactJs was used for the frontend, just to simplify dynamic rendering, and Material-UI as the user-interface framework for quick-to-use table.

What follows is the list of the libraries that have been used for the assignment:

- *axios* v.0.21.1
- *cors* v.2.8.5
- *express* v.4.17.1
- *http-proxy* v.1.18.1
- *node-cache* v.5.1.2
- *react* v.17.0.2
- *nodemon* v.2.0.12

In order to test the program:

1. Create a new folder and clone the repository using the following commands on a git shell:
- git init
- git clone https://github.com/JasonShuyinta/reverse-proxy.git

2. Once the repository is downloaded, navigate to the main folder. For Windows users:
- cd reverse-proxy

3. Install all the necessary dependencies with the npm command:
- npm install

4. Once the installation is finished, open 4 terminals, navigate with each of them to the reverse-proxy folder and run the following commands:
- node server-one.js
- node server-two.js
- node reverse-proxy.js
- npm start

If the steps are done correctly, a web page should open at http://localhost:8080.

Useful links:

Server Cache in Node Js – YouTube https://www.youtube.com/watch?v=ipIGWZwxC7w&t=499s

http-proxy – Github Repository https://github.com/http-party/node-http-proxy#readme

Mock API - https://jsonplaceholder.typicode.com

https://dzone.com/articles/properly-measuring-http-request-time-with-nodejs

Author:

Jason Shuyinta