Javascript Challenge for Oneflow

by Georgios Pligoropoulos <george@pligor.com>

Task 1 Coding Challenge

This is the git repository where the code has been uploaded:
https://github.com/pligor/expressjs_vuejs

oneflow_expressjs folder contains the server
vue_nightwatch folder contains the client

Use WebStorm Jetbrains IDE to open both projects
Hitting the run/play button first on the server you need to go to the browser and execute something similar to localhost:3000/load to load all the episodes from the json you had provided. This will create a saved.json file. This is a GET request and it could have been something else like a POST but for the simplicity of this technical exercise I kept it like a GET request so you can execute it from the browser directly without REST clients etc.

There are two routes:

- /load
- /episodes

Next step is to go to the client project and run it. This will make the browser open and display the web app with the all the Silicon Valley episodes loaded.

For executing tests on the server side you should use `mocha` command. Visit the corresponding testing folder and run `mocha (and then the javascript file that is necessary)` like episodes_unit_testing.js or load_testing.js

For executing tests on the client side you should first change the URL. Go to `vue_nightwatch/test/e2e/episodes.test.js` path open the file and replace `http://localhost:63342/vue_nightwatch/index.html` with whatever url the webstorm has generated when you hit the run/play button, on the client project. (Typically only the 63342 port should be different)
After saving this file you can go to the folder vue_nightwatch and execute `npm run e2e`.

I will keep this explanation short but feel free to send any questions you might have.

It was a pleasure to watch Vue.js becoming the best progressive js framework. I would like to thank Oneflow for giving me the chance to play with Vue.js in this technical challenge.

Task 2 Analysis Challenge

So here we must first read the the code to see what the original intention was. What the developer wanted to do. ← Being respectful to whoever reads your code this should have been in the comments and not let the reader do any guesswork.

It seems that users are being invited to a shop. From the message “user already invited to this shop” one would assume that the intention is to invite the user to a particular shop

Here we are taking the request body as it is and we are sending it using a post request to authUrl. But if we want to authenticate something we should not pass the entire body. This is unnecessary load to the server. We should instead pass on the minimum information. And this is only in the case where our architecture suggests that the authentication is at another server (here url.to.auth.system.com) and we should better do this to a server that is behind a firewall to our trusted intranet.

Then it seems that the http status codes have been greatly misused and the conventions have not been followed. “User already invited to the shop” this seems more like a conflict 409 than a 200 OK request. This is what would I expect from the auth server and this is also what I should return to the user: 409 conflict “user already invited to this shop”.
The situation gets worse when we are getting status 201 from the auth server meaning that the invitation has been created… but below we are managing error cases and nowhere in the error cases we are correcting this new entry that has been created.
The above suggests that ideally we should also fix the auth server implementation (if we have access to it)

One more thing to note is that we have callback inside a callback inside a callback which is not necessarily bad but it could become slightly more readable by using Promises.

We are also returning the invitation response outside the callbacks! which means that the invitation response will be returned regardless of what happens to User and Shop models.. So we are returning to the user that invitation ok but, for example, we might have left the shop non updated. Actually if the response is indeed not found then we are having a race condition of which of the two statements might be called first. In general bad practices to follow. The good practice is to write your code in a way where you follow one or the other path, not both.

Another thing to note is that we are not consistent. On one hand we are returning “error:true” along with the message “user already invited to this shop” but we are not returning an “error: true” when “no shop found”. Why would we force the developer the on

other side to keep track of two different approaches on error handling? We are making his life more difficult.
Something extra is that we are getting an error 500 when no shop is found. The 500 is usually used along with the generic message “server error” when we do not want the user to know of the exact error, typically for privacy reasons. However here since we are already explaining what is happening we would better use a more appropriate status code like 400 or 404 depending on how we like to define things.

In addition we see that there is business logic involving the shop its invitations and its users. This could become a method inside the Shop model that would be called something like “add_user_invitation” and this would make it possible to create also unit tests and thus make Shop testable.
Moreover having the shop containing a list with the full model of the User and not just the id is a smell of bad architecture. One would do that only if he or she had designed that one user belongs to only one shop. Only then it would make sense to a Shop fully contain a user. From this code it seems that the intention was for multiple users to be invited (belong) to multiple shops. Therefore having all these Shops maintaining a list of users could be highly erroneous but maybe this is how Mongoose abstracts things. If this is simply an abstraction of Mongoose then this is not an error and all is ok.

Finally we have that invitations are maintained both inside shops (invitations list) and in users (does invitation body exist?) and in auth server where we actually store a new invitation as this code suggests. Keeping all that in synchronization is not a good practice. The good practice is to keep the information in a single place. A simple approach would be to store whether the invitation happened or not inside Shop model and then query Shop model to check if invitation has happened already or not.

And of course the above analysis is based on various hypotheses. It would have been more clear if one knew the context.
Thank you