Twister

Aldan CREO MARINO, Maxime MILLARD

May 8, 2022

1 Work distribution

In this case, the work distribution has not been balanced.

- Maxime MILLARD: In the first TMEs, some changes to the code (see `https://github.com/ACMCMC/twister/commits?author=MaximeMllrd`).
- Aldan CREO MARINO: Almost all of the work (see `https://github.com/ACMCMC/twister/commits?author=ACMCMC`).

2 Found issues

We have seen that the `Mocha` tests didn't finish by themselves, because the sample server code that we were provided was not properly closed after running the tests. For this reason, `Mocha` saw that a resource was still being used, so it didn't finish. We had to find a solution for this issue.

We also had to rewrite all the components when we started using *redux*. That's because they need to become functional components, which they were not at first.

3 Remaining work to do

All the compulsory parts have been covered. In fact, we've implemented extra functionality in our project (described below).

4 Extra functionality

The extra functionality we talked about in the video is searching among our contacts only.

Other aspect to highlight is that the app is able to perform likes and remove them on tweets.

Moreover, we also added an endpoint to remove users, which takes care of removing their related messages first too. This endpoint is not accessible from the client application; it is only exposed as another endpoint of the User service.

We implemented the database as a *docker-compose* instantiation, for better portability. To run it, it's enough to run `docker-compose up -d` at the `server` subdirectory.

5 Different choices

We have decided to use `Mongoose` as our ORM, because it's more tightly integrated with `MongoDB`, plus it allows for schema-checking.

We have decided to use `redux` in the client, because it provides a cleaner (though more complex) approach to keeping the state of the application. We think that it is more didactical to learn this technology as it requires a higher level of understanding.

Also, our API on the server is not a REST API. That is motivated by the fact that we were suggested to keep a session open using `express-session`. For this reason, the state is maintained (you need to log in before performing other operations).

We also decided to mix the Friends API and the User API, because they are actually very closely related. In fact, both of them mean changing the same collection in the database.

We also decided to implement separate stylesheets per module, to keep a better separation. There is a common stylesheet that applies to the entire project, however, in the root of the client.