TrustBallot - WEB Project

Michele De Cillis

Table of Contents

How to install

Extract the project inside a folder. Make sure that the structure looks like this:

```
- api/
- data/
- web/
- README.md
```

Open a terminal inside the project and type:

```
    php -S localhost:9000
```

Open the project at this link.

Optional

By calling the file `populate.py` , it will create an admin user and 25 bots to quickly set up the platform.

```
    python3 populate.py
```

The admin user will be:

```
mail: admin@admin.com
pass: 12345678
```

Instead, the bots will have:

```
mail: bot-{n}@user.com
pass: 12345678
```

Report

Before proceding with coding, I decided to analyse well the two main parts of the project: the frontend and the backend.

For the backend, my goal was to create a system that could be the most generalised and "scalable" as possible, in order to have the possibility to ensure consistency among all the APIs and implement fast a new one, if I needed.

Since I was working with files, my first thought has been of creating a controller to work with the files:

1. **Atomicly**
2. **Easily**
3. **Error proof**
4. **Generally**

By atomicly, I mean that on each function call I would access the file, perform the operation and nothing else. No side effects. Easily: I call the function and I get what the function tells me to do. Error proof: I want that, if something goes wrong, I can know it with an error, but it doesn't interrupt my visite on the website. Generally: I wanted an "universal" function that would work for all my JSON files.

So, I've made some functions (inside the folder `/api/controller/`) that helped me to performs some basic operations like `get`, `update`, `delete` through my work on TrustBallot.

After that, I built a wrapper around them: that's where things gets more specific: I created a wrapping function for every API that, before communicating to the controller, would sanitize the data and make them safe to be stored.

Since I was handling a lot of APIs, I thought that the best way to do that would have been by creating another wrapper, that would take adventage of the standardization of the HTTP Status Code to, based on the return of the functions I previously wrote, to return an object that would contain:

```
{
    "code": "", // The HTTP STATUS CODE,
    "message": "", // The message to display inside the application
    "data": "" // The facultative data object returned by the server
}
```

At the end (or at the surface of the backend onion-structure, since it has multiple layers) there is a big switch statement that, based on the `API_NAME` argument, will detach each request to its own function.

Personally, it was one of the parts of the project where I had more fun, since I had to think of a proper structure that could be generalised for each data and could be sustainable during all the project.

On the other side, for the frontend, I used **AJAX** a lot to communicate with the APIs I previously created. I decided to use the `.php` extension just to easily handle the authentication (it's actually handled inside the header, that would redirect to the sign-in if the user is not authenticated).

Even here there's a wrapper around the AJAX call, because by default the website displays a small toast for the errors, in order to notify the users if something unexpected happens. I decided to use a wrapper because it would be better during the coding phase to have just one function that does something and that could be coherent through all the site.

The website also supports the multi-language through a file (`/web/js/translations/translations.js`) which contains all the labels inside the `data-translation` attributes that can be found inside the HTML tags, around the site. When a page is opened, jQuery iterates through all the `data-translation` values and it maps the labels with the translation inside the file.

For the styling the site uses some Bootstrap for some already premade components and then mostly custom CSS. Both CSS and JS are specificly for each page, even if they're sharing some common functions and selector respectly inside the `/web/js/main.js` and `/web/css/globals.css`.