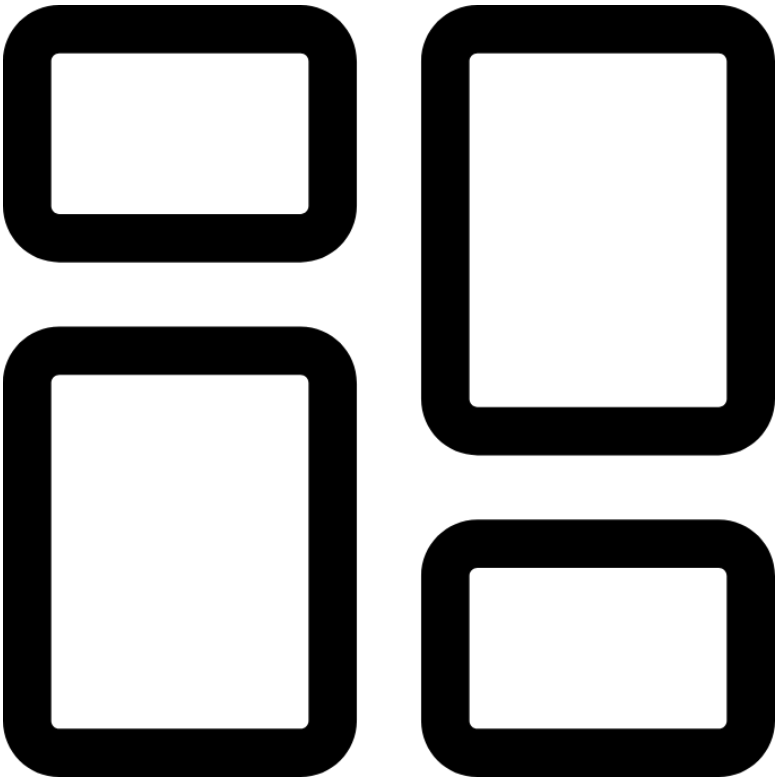
Dashboard Technical Documentation



Rodolphe DUPUIS - Eliott PALUEAU 11/2020

Summary

I. Project overview.

II. Libraries used.

III. Project structure.

IV. APIs used.

V. Database structure.

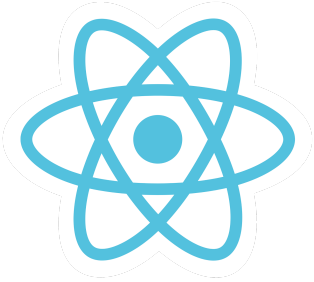
VI. Project build.

I. Project overview.

The **Dashboard** project consists in the creation of a web application.

This application is a customizable dashboard which integrates configurable widgets from different services.

This project uses the following languages:
Back-end → NodeJs
Front-end → React
Database → MongoDb




II. Libraries used.

For the Front-end part of the app, this project uses exclusively **Material UI** components, here are some notable components used:

- User input:
  - Text field
  - Button
  - Fab button
- Display:
  - Cards
  - Icon
  - Typography

III. Project structure.

The project is structured in 2 parts, *Front* and *Back*.

The *Back* part contains all code for the database, this part will be detailed in the next section of this documentation.

Front server : localhost:3000 / **Back server** : localhost:8080

The *Front* part contains the code for all the web application, it is divided in 2 big parts:

- The `assets/` folder, containing all the different icons used in the app and some default pictures.
- The `components/` folder, containing the code (in Js and Css) of all our pages and components.

The code is divided in 4 parts in the *components*/ folder:

- `Home/` -> Home page and 'Add widget' pop-up
- `Login/` -> Login/register page
- `Profile/` -> Profile pop-up and login pop-ups
- `Widgets/` -> All the different widgets

IV. Apis used.

Here are the different APIs and their use for each widget:

- Youtube API (https://developers.google.com/youtube/v3)
  - Get channel subscribers count.
  - Get channel last video.
- Spotify API (https://developer.spotify.com/documentation/web-api/)
  - Get artist top tracks.
  - Get user public playlists
- Github API (https://developer.github.com/v3/)
  - Get user public repositories.
  - Get repository last pushes.
- OpenWeather API (https://openweathermap.org/api)
  - Get city current weather information.

V. Database structure.

This project uses *MongoDB* as database manager because it offers easy model handling and storing. It is integrated with *NodeJS* and *Express* to manage the interactions between the front server and the database server.

This database uses a single model called **User** to store all the datas. This model contains several parts :

Email : email address (type: String)

Password : encrypted password* (type: String)

Username : username (type: String)

Data : array containing services **credentials** to keep the user logged in when he signs out, credentials stored are:

- Google accessToken.
- Spotify accessToken.
- Spotify refreshToken and its expiration time.
- Github accessToken.
- All the **widgets** added by the user

Note that all access and refresh tokens are stored as *Strings*.
Finally, the Data array stores all the **widgets** added by the user.

The front-end and back-end are working on different servers, so **axios** is used to call server routes and update the database from the front.

Every added widget is stored in the database. When the user signs out and comes back, he is able to get all his widgets without needing to interact with the services.

* Password: usage of bcrypt (https://github.com/kelektiv/node.bcrypt.js/) to encrypt the user's password in order to keep it safe in the database.

An `about.json` file is available on the database server (http://localhost:8080), giving all the information about the project.

VI. Project build.

This project is built using Docker.
The project structure separates the back and front servers, therefore, there is one Dockerfile in each folder with a docker-compose.yml file at the root.

The file *docker-compose.yml* will call the Dockerfile in each source (`/sources/front/` & `/sources/back/`) and build the entire project as well as launching the servers. It will also start and bind *mongod* to the port 27018 so the database can be used in the project.

Dockerfiles will simply build the two folders (front & back) so *docker-compose.yml* can launch the whole build.

To start the project, you'll need to go in the front (`/sources/front/`) and back (`/sources/back/`) folders and run ***npm install*** (or *npm i*) in order to install all the dependencies of each folder. Go back to the root of the project and run ***docker-compose up***, according to your Docker installation, you'll maybe need to add super doer permissions to this command. If ever the command fails, enter the ***sudo systemctl start mongod*** command.

Finally, open your favorite navigator and go to http://localhost:3000 to register to the application and start using it.

Thanks for reading, if you want to know how to use the application you are invited to read the user's guide.