Dashboard Technical Documentation

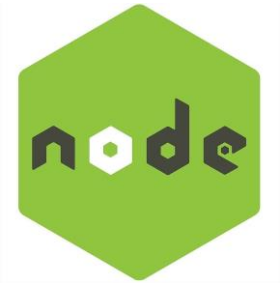
Delesan Srineevasan- Lucas Moritel 12/2021

Summary

I. Project overview.

The **Dashboard** project consists in the creation of a webapplication.
This application is a customizable dashboard which integratesconfigurable widgets from different services.

This project uses the following languages:
Back-end → NodeJs
Front-end → Angular
Database → Postgres



II. 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:4200/ **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 *app/* folder, containing the code (in Js and Css) of allour pages and components.

The code is divided in 3 parts in the *app/public/components* folder:

- `dashboard/` -> Home page and 'Add widget' pop-up
- `login/` -> Login page
- `register/` -> Register page
- `Components/widgets.*` -> All the different widgets

III. Apis used.

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

- YouTube API (https://developers.google.com/youtube/v3)
    - Get the channel information the user asks.
    - Make a search.

- Currency Freaks API (https://api.currencyfreaks.com/latest)
    - Get the conversion of a value between 2 currencies.

- Weathersstack API (http://api.weatherstack.com/current)
    - Get the weather in any city
    - Get the location of a city
    - Get the hour of the timezone of the city

- Reddit API (https://www.reddit.com/api/v1/)
    - Make a search in reddit
    - Get the posts of a subreddit
    - Get the subreddit you are sub to

- Riot API (https://euw1.api.riotgames.com/)
    - Get the information of a summoner
    - Get the last match of a summoner

- Nasa API (https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos)
    - Get picture taken by Curiosity

Ⅳ. 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)

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.

V. 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 (*/client/* & */server/*) and build the entire project as well as launching the servers. It will also start and bind *Postgresql* to the port 5432 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.

Finally, open your favorite navigator and go to http://localhost:4200 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.