Area Technical Documentation

Kylian BALAN - Nathan LECORCHET - Ghassane SEBAÏ - Rodolphe DUPUIS - Eliott PALUEAU

Summary

I. Project overview.

_____This is the technical documentation, it will detail all the technical parts and the code of the project.

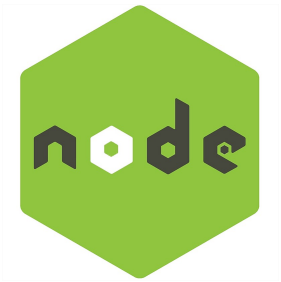
If you want to know how to use the application you are invited to read the provided user's guide.

The **AREA** project consists in the creation of a software suite that functions similarly to IFTTT and/or *Zapier*.
This software suite is a divided into 3 parts:

- A **server** to implement all the features.
- A **web application** to use the app from a browser.
- A **mobile application** to use the app from a phone.

This project uses the following languages:

- Server → NodeJs Express & MongoDb.
- Web application → VueJs / TypeScript.
- Mobile application → Flutter.




II. Libraries used.

For the design of the web application, this project uses the Css framework **Tailwind**.
This allows the use of custom classes for a better control of the css.

The Server is built with MongoDb and uses **PassportJs** to manage user authentication on the different services of the application.

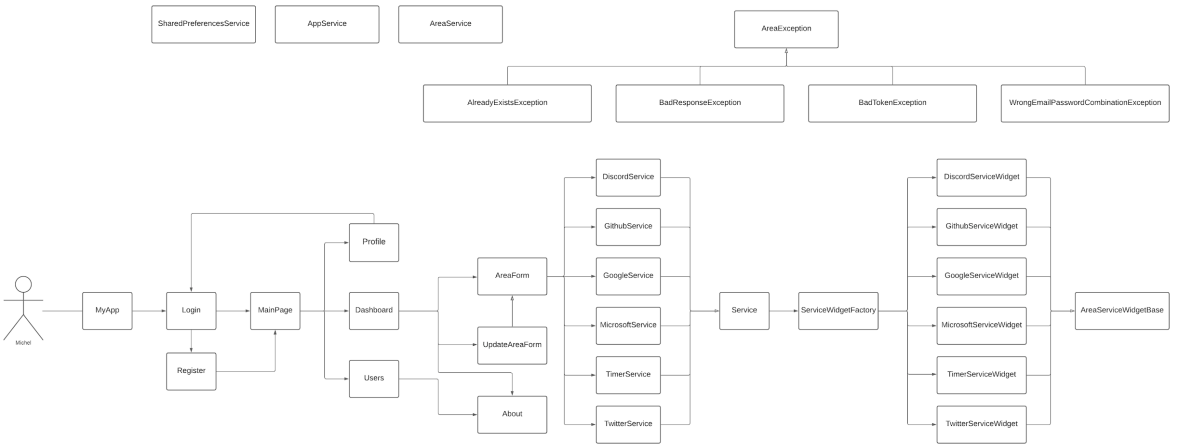
III. Project structure.

The project is structured in 3 directories each one contains the code for the corresponding part: *Front*, *Mobile* and *Server*.

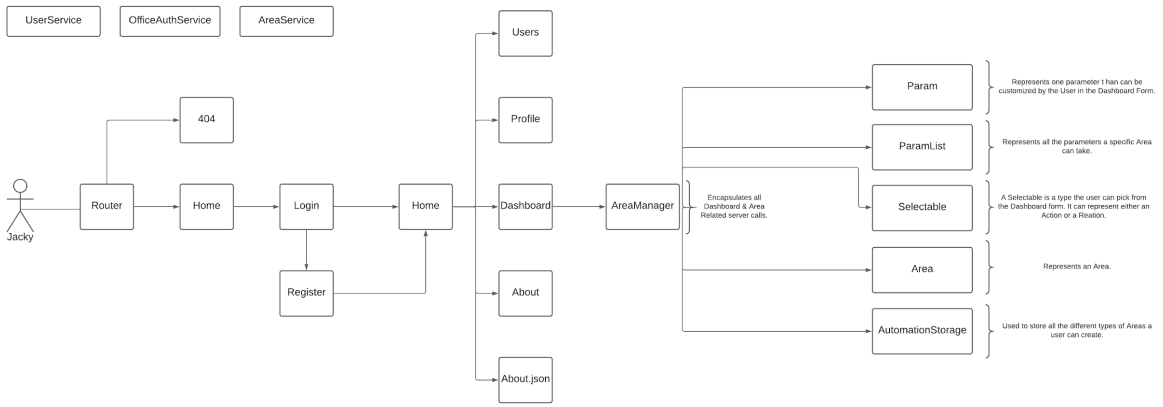
The project is built using **Docker**, this will be detailed in a next section of the document.

IV. Architecture diagram.

Mobile:



Web:



V. Apis used.

Here are the different APIs used for this project.

- Youtube API (https://developers.google.com/youtube/v3).
- Twitter API (https://developer.twitter.com/en/docs).
- Outlook API (https://docs.microsoft.com/en-us).
- Github API (https://developer.github.com/v3/).
- Discord API (https://discord.com/developers/docs/intro).

VI. Database structure.

This project uses MongoDB as database manager. It is integrated using NodeJS and Express to manage the interactions between the front server and the API server.

This database contains several models, such as **User** (including one model called Service), **AREA** (including two models **Action** and **Reaction**) and **ConnectSession** (used when the user connects to a service).

Starting with the **User** model, the schema contains different fields such as :

- `email` : email address (type: *String*)
- `password` : encrypted password (type: *String*)
- `displayName` : username (type: *String*)
- `isMicrosoftAuthed` : user connected using microsoft (type: *Boolean*)
- `connectData` : data retrieved when user is connected to a service (type: ***ServiceSchema***)

ServiceSchema is the model used to store datas related to a user's service, it contains three different parts :

- `accessToken` : accessToken of the service (type: *String*)
- `refreshToken` : refreshToken of the service (type: *String*)
- `data` : data related to the service (type: *Object*)

Next, The **AREA** model will store datas about the action (model Action) and the reaction (model Reaction) but also the user's id :

- `userId` : user id (type: *String*)
- `isMobile` : to know whether it is from mobile or not (type: *Boolean*)
- `action` : information about the action (type: ***ActionSchema***)
- `reaction` : information about the reaction (type: ***ReactionSchema***)

ActionSchema is considered as the definition of an Action :

- `service` : name of the service (type: *String*)
- `name` : name of the action (type: *String*)
- `data` : all the information of the action (type: *Object*)

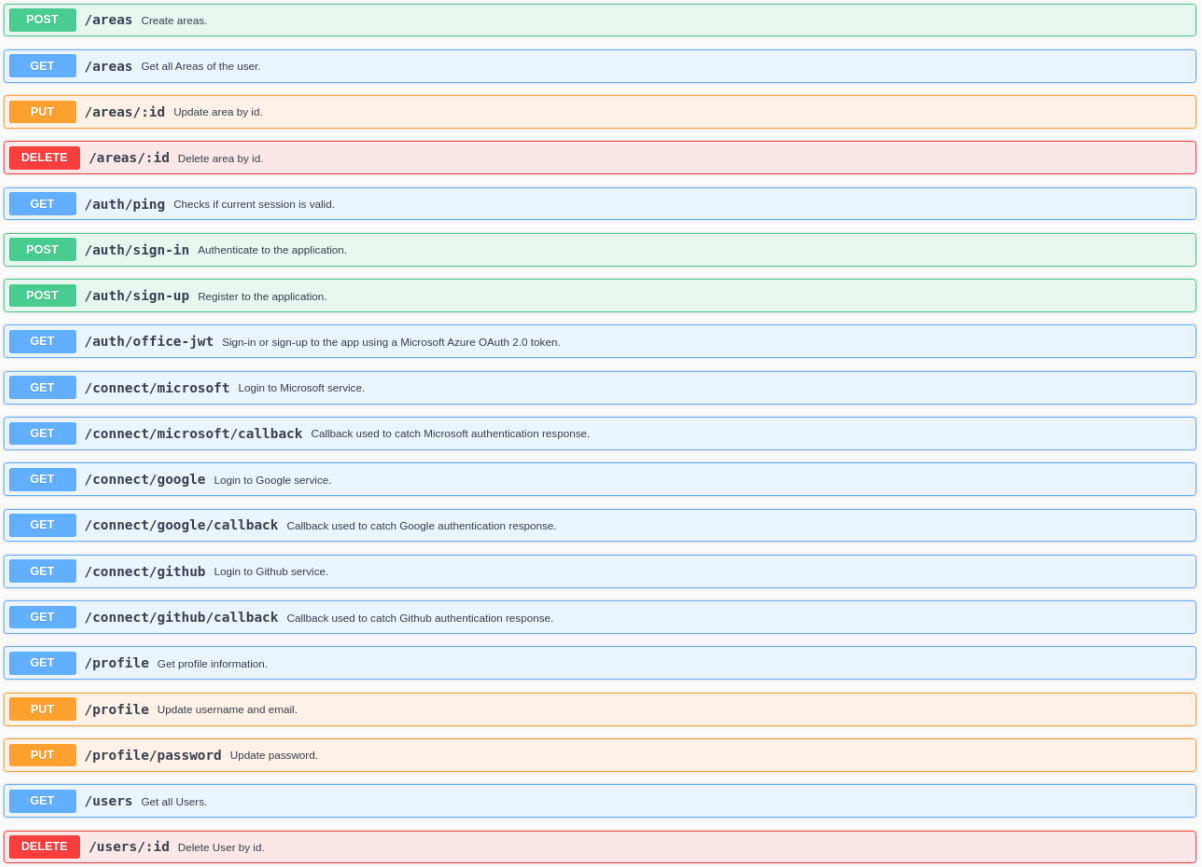
ReactionSchema is considered as the definition of a Reaction :

- `service` : name of the service (type: *String*)
- `name` : name of the reaction (type: *String*)
- `data` : all the information of the reaction (type: *Object*)

ConnectSession is the model we use to manage data when the user tries to connect to a service :

- `userId` : user id (type: *String*)
- `endpoint` : which endpoint the user is accessing (type: *String*)
- `isMobile` : to know whether it is a mobile request or not (type: *Boolean*)
- `data` : pass data through endpoints and callbacks (type: *String*)

VII. Server endpoints.



IX. Project build.

This project is built using Docker.
The project structure separates the back, the web and the mobile 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 (`/server/`, `/front/` & `/mobile/`) and build the entire project as well as launching the servers. It will also start and bind *mongod* to the port 27017 so the database can be used in the project.

Dockerfiles will simply build the three folders (server, front & mobile) so *docker-compose.yml* can launch the whole build.

Front server : `localhost:8081`
Back server : `localhost:8080`
Mobile apk download : `localhost:8080/client.apk`