B5 - Application Development

B-DEV-510

AREA

Automation platform of his digital life

Documentation

Version 1
01/03/2020

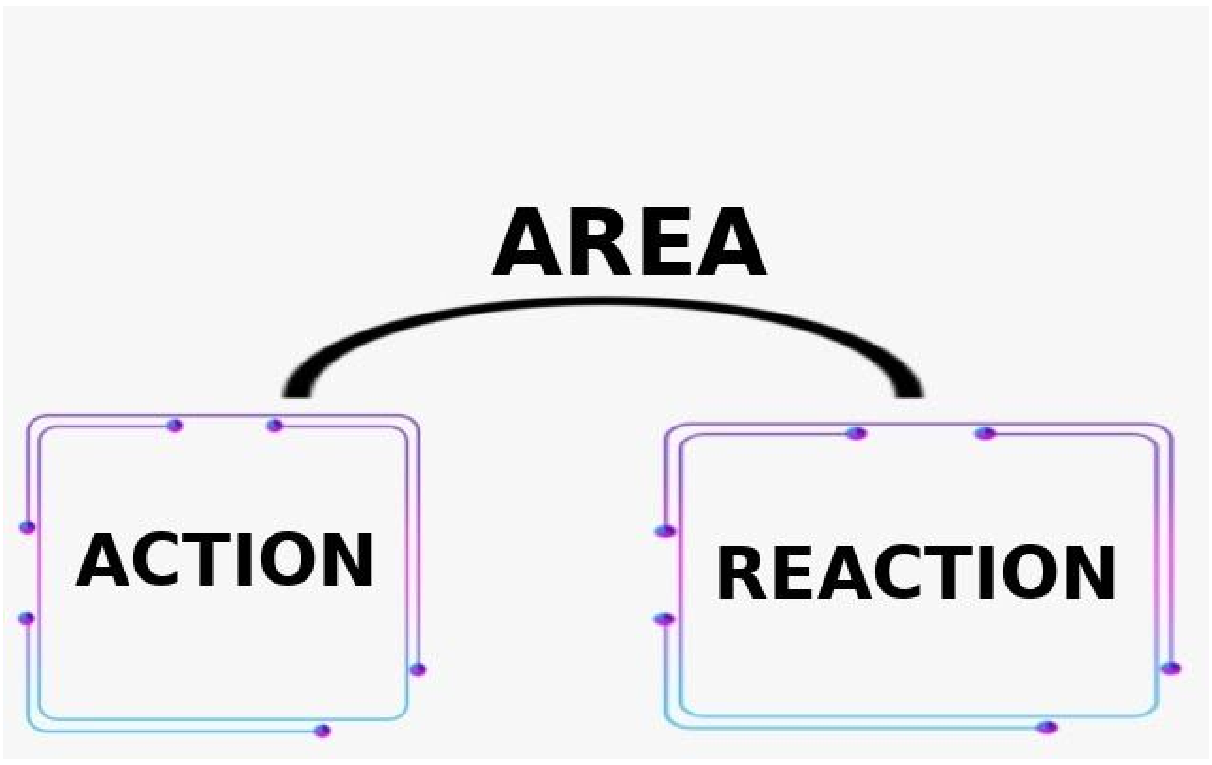
Summary

1. What is Area and how does it work?..3
    a. Project description...3
    b. Technological environment...4
    c. User management..4
    d. Description of the API..5
2. Flow diagram...7
3. AREA modules...8
    a. Organization of the modules...9
4. Actions / Reactions of AREA..10
    a. Description of Area's action and reaction scenarios……….10
5. Installation and Deployment...11

What is Area and how does it work?

AREA is a web and mobile application that provides automated online actions through the interconnection of different web services. The user can connect or register to different web services and then authorize AREA to access different services (modules). The user can therefore create connections between these services in the following way :

- An action occurs on module A
- A reaction takes place on a module B



By combining these 2 modules Action / REAction we obtain an AREA.

Technological environment

The technology stack for this project includes :

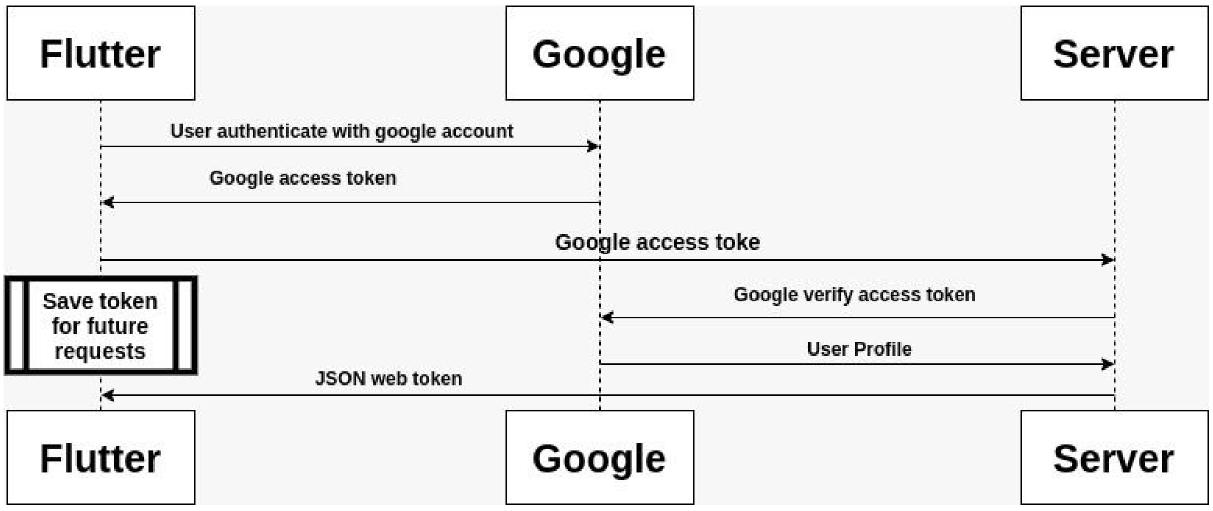
SERVER REST API	WEB & MOBILE CLIENT	DATABASE	BUILD & DEPLOYMENT

User management

Regarding user management, users can access the application by creating an account or by authenticating to the application using their Gmail account.
Non-authenticated users can create an account on a login page by choosing a username and password. Once the account is created, they can sign in to the web application.
Users can directly manage their connections to the different services on the page dedicated to them. Once the user logs in and authorizes the AREA application to access their account, the access tokens will be saved in a database that will allow automatic login each time the AREA is used.
On the AREA home page, the user can see what is available. The user is provided with a brief description of each scenario where they can choose to activate a specific scenario. Unavailable modules will require access authorization when the user activates the scenario and hasn't previously provided the necessary authorization.
For example, for Gmail, the module will not be available if the user has not previously authorized the AREA application to access their Gmail account information.
The user can choose to dis-activate a scenario and to sign out.

Description of the API

REST APIs replicate the way the web itself works in the exchanges between a client and a server.
The client-server principle defines the two entities that interact in an API REST: a client and a server, the same entities that communicate on the web. A client sends a request, and the server returns a response. The server must have as much information about the client as possible, because it is important that they are able to work independently of each other.
Being "stateless" means that the server has no idea of the state of the client between two requests. From the server's point of view, each request is a separate entity from the others. Then, the cache, for the REST APIs, uses the same principle as for the rest of the Internet: a customer must be able to store information without the need for constantly needing to ask the server for everything.
Server responses for REST APIs can be delivered in multiple
JSON (JavaScript Object Notation) formats are often used, but XML, CSV, or even RSS feeds are also valid.

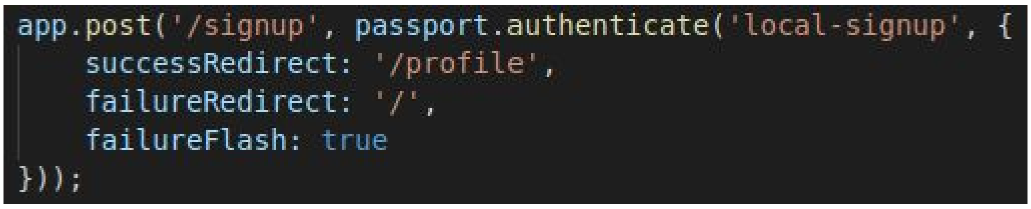


Authentication / Identification

Local login/signup
POST signup

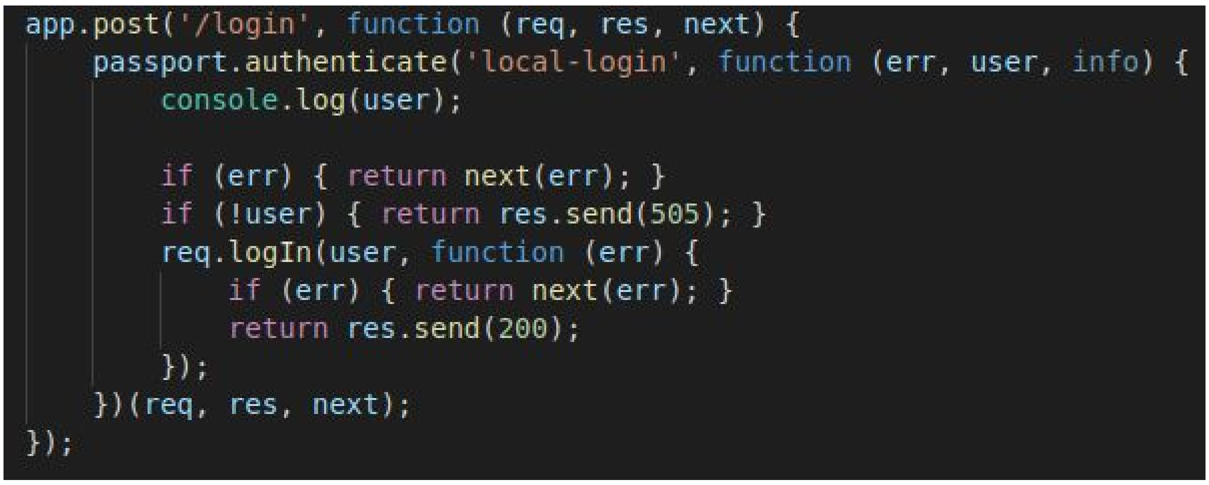
>localhost:8080/signup

> returns a status code 200 if case of success and 500 in case of error



```
app.post('/signup', passport.authenticate('local-signup', {
    successRedirect: '/profile',
    failureRedirect: '/',
    failureFlash: true
}));
```

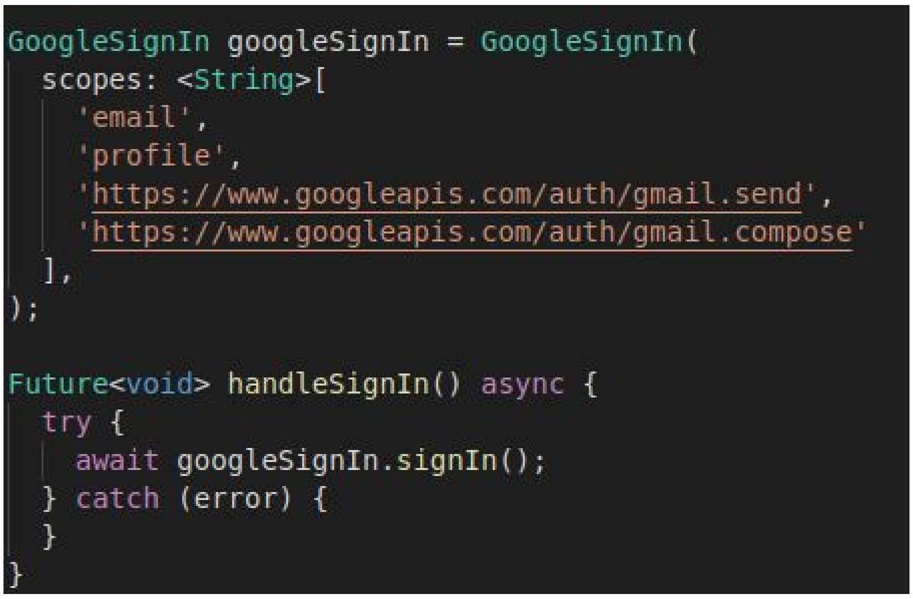
POST login
>localhost:8080/login
>returns status code 200 if case of success and 500 in case of error



```
app.post('/login', function (req, res, next) {
    passport.authenticate('local-login', function (err, user, info) {
        console.log(user);

        if (err) { return next(err); }
        if (!user) { return res.send(505); }
        req.logIn(user, function (err) {
            if (err) { return next(err); }
            return res.send(200);
        });
    })(req, res, next);
});
```

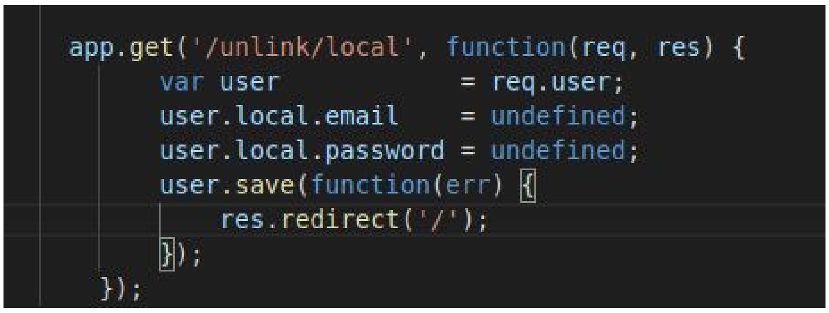
Signup with Google
Send to Google to do the authentication and get the access token and refresh token.



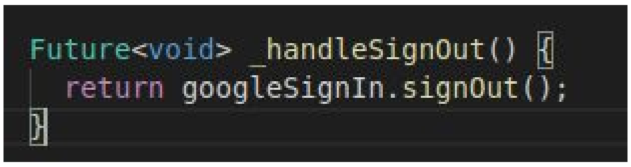
```
GoogleSignIn googleSignIn = GoogleSignIn(
  scopes: <String>[
    'email',
    'profile',
    'https://www.googleapis.com/auth/gmail.send',
    'https://www.googleapis.com/auth/gmail.compose'
  ],
);

Future<void> handleSignIn() async {
  try {
    await googleSignIn.signIn();
  } catch (error) {
  }
}
```

Logout
GET unlink
>localhost:8080/unlink/local
>returns status code 200 if case of success and 500 in case of error

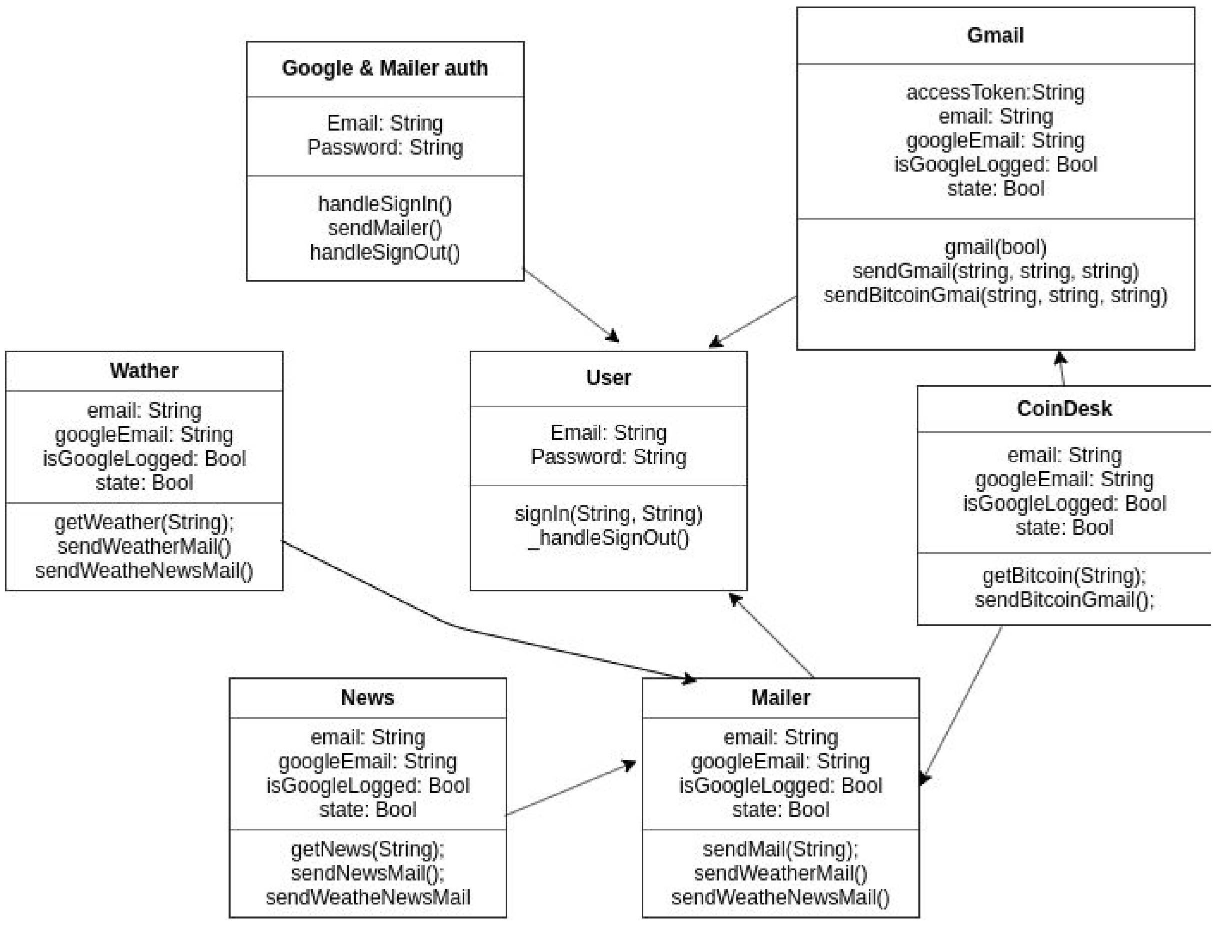

```
app.get('/unlink/local', function(req, res) {
    var user            = req.user;
    user.local.email    = undefined;
    user.local.password = undefined;
    user.save(function(err) {
        res.redirect('/');
    });
});
```

Signout
When the user is logged in using Google Sign in


```
Future<void> _handleSignOut() {
  return googleSignIn.signOut();
}
```

Flow diagram

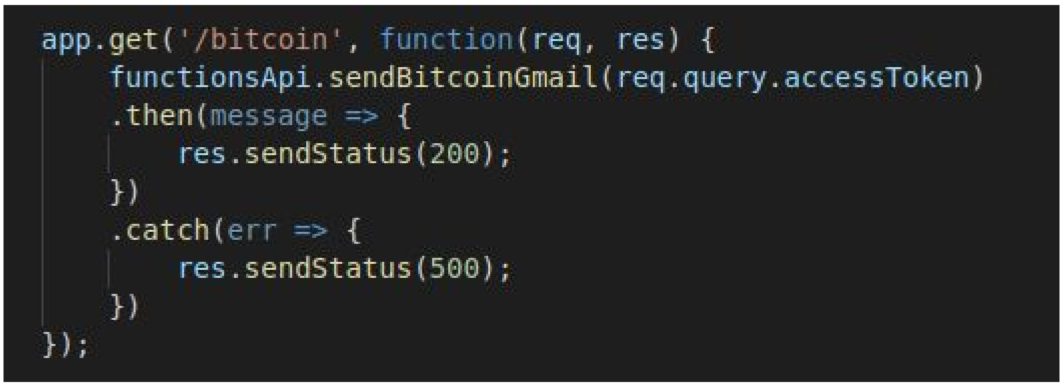
Each service has an API key, a base url and its widgets. Each widget has an API key, a base url, a rate a refresh time (60 minutes by default).
All information is retrieved via the REST APIs. A RequestApi function taking in parameter the complete URL of the request and a callback function allowing to perform these requests.
A scheduler periodically checks whether the user makes changes to the scenarios and listens for changes in the actions of the selected scenarios. Based on this information, the scheduler determines whether to trigger the reaction or not.



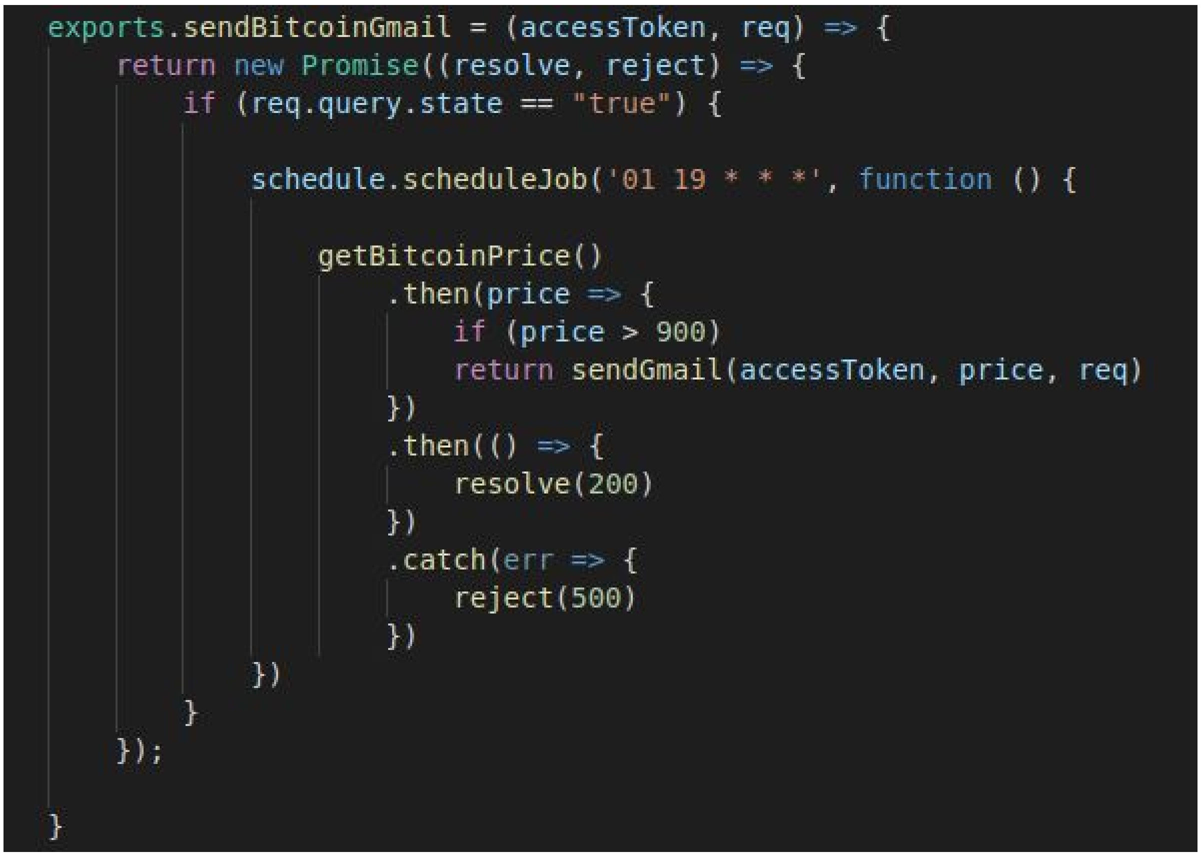
AREA modules

Area provides various services implemented in the form of individual modules. Each service is exported as a function and is then accessed in the corresponding route. These modules are composed of different functions that listen to the action, its changes and then determine the corresponding reaction.

All requests made to the different API urls are implemented as promises and the scenarios are implemented as a chain of promises.



```
app.get('/bitcoin', function(req, res) {
    functionsApi.sendBitcoinGmail(req.query.accessToken)
    .then(message => {
        res.sendStatus(200);
    })
    .catch(err => {
        res.sendStatus(500);
    })
});
```



```
exports.sendBitcoinGmail = (accessToken, req) => {
    return new Promise((resolve, reject) => {
        if (req.query.state == "true") {

            schedule.scheduleJob('01 19 * * *', function () {

                getBitcoinPrice()
                    .then(price => {
                        if (price > 900)
                        return sendGmail(accessToken, price, req)
                    })
                    .then(() => {
                        resolve(200)
                    })
                    .catch(err => {
                        reject(500)
                    })
            })
        }
    });

}
```

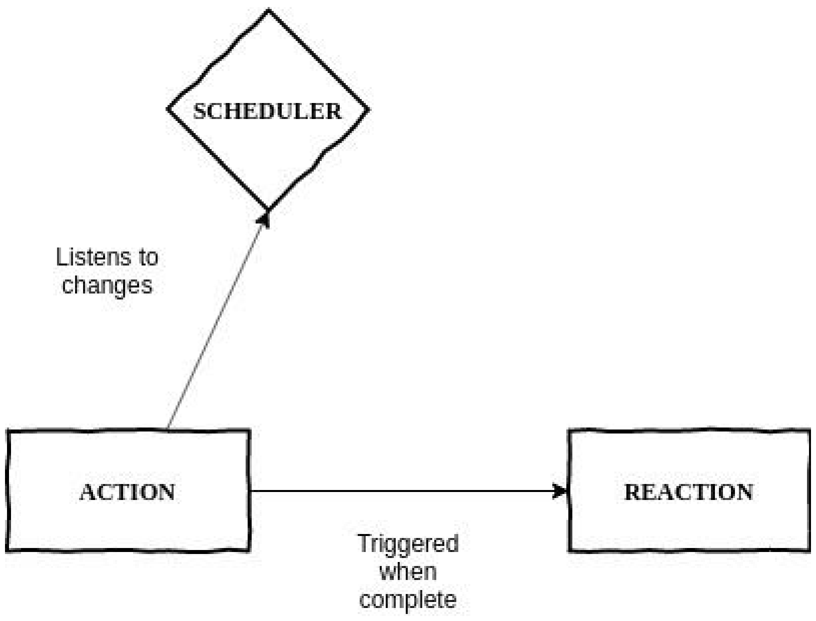
Organization of the modules

The AREA project uses a MongoDB database. Each time a module is added, the developer must set up a specific installation in order to integrate it into the database in the form of a module template.
Each module selected by the user must be connected to the user ID registered in the database for each user.

Actions / Reactions of AREA

Area is composed of one or more actions and a reaction. Each scenario is sent to a specific endpoint in the server API that manages the entire logical part of the scenario implementation.
The server sends requests to several APIs depending on the scenario, gets a result, analyzes the result and determines when to trigger the reaction or not. The requests are implemented using promises that are resolved when the specific conditions of the scenario are met. A complete scenario is implemented as a chain of promises.

A scheduler implemented in Nodejs checks the execution of all scenarios selected by the user. It periodically checks the state of the actions and triggers or not the specific reaction. This scheduler also listens to any change in the user selected scenarios and stops their execution if the user has disconnected a specific scenario.



At the end of the execution of a scenario, a status code 200 is returned, indicating that the scenario was executed without errors. If an error occurs, the function returns a status code of 500.

Any new implementation of modules or scenarios must follow the previous line of implementation.

Installation and Deployment

Docker is used to deploy and build Area. Run **docker-compose up --build** and launch the project in emulator or web browser.

MEMBERS:

Joan KABELLO
Joana KARANXHA
Endri BIMBLI