



Design of an application for time control

Final Project Memory

Software Development Specialization Diploma

Authors:

Petr Bezuglyy

Zsófia Kovács

Mercedes Rodriguez

Professor:

Raúl Novoa

14th June 2019

Index

1. Introduction
2. Goals
3. Resources
4. Methodology
5. Development analysis
6. Result
7. Conclusions

Time Control App- Final Project

1. Introduction

Final project for the Software Development Specialization Diploma (University San Jorge).

Movicoders ordered a web application in order to register its workers time entries, as it is mandatory since May 12th.

Royal Decree-Law 8/2019 of 8 March on urgent measures for social protection and the fight against job insecurity in the workplace was published in the Official State Gazette ("BOE").

Due to the entry into force of Royal Decree-Law 8/2019, of March 8, which includes regulatory reforms aimed at regulating the registration of working hours, as specified by the amendment of Article 34 of the Statute of Workers, adding a new Section 9.

This is the main novelty introduced by RDL 8/2019, which affects the modification of Workers' Statute article 34.

Article 34 of the workers' statute is extended by means of a new section 9, which establishes the business obligation to guarantee the daily record of the working day in which the articulation and minimum requirements of the same are regulated.

In this regard, the obligation is established that such records include both the beginning and the end of the working day, and this information must be recorded individually about each worker.

«9. THE COMPANY WILL GUARANTEE THE DAILY RECORD, WHICH SHALL INCLUDE THE CONCRETE SCHEDULE OF START AND TERMINATION OF THE WORKING DAY OF EACH WORKING PERSON, WITHOUT PREJUDICE TO THE TIME FLEXIBILITY THAT IS ESTABLISHED IN THIS ARTICLE.

THROUGH COLLECTIVE BARGAINING OR AGREEMENT OF THE COMPANY OR, IN ITS FAILURE, DECISION OF THE PREVIOUS ENTIRE CONSULTATION WITH THE LEGAL REPRESENTATIVES OF THE EMPLOYEES IN THE COMPANY, THIS JOURNAL REGISTRY WILL BE ORGANIZED AND DOCUMENTED.

THE COMPANY WILL KEEP THE RECORDS TO WHICH THIS PRECEPT REFERS FOR FOUR YEARS AND WILL REMAIN AVAILABLE TO THE WORKING PERSONS, THEIR LEGAL REPRESENTATIVES AND THE INSPECTION OF WORK AND SOCIAL SECURITY. »

From now on it will be mandatory for workers to sign at the beginning and end of their working day. The law establishes that the employer bears the responsibility of keeping a daily record, keeping the data for four years and making it available to the workforce, the works council and Labour Inspection, as well as choosing the concrete method to be implemented.

2. Goals

Project: Web application for time control, responsive and multilanguage.

Our main goal was to create a functional webpage following the customer's requirements.

The webpage's functionality:

User: be able to login with the credentials given by the company and be able to record the working day

Login: the credentials must be authenticated by the Movilizer Cloud

Responsive: be able to use in different devices

Multilanguage: implement different languages, in our case English and Spanish

3. Resources

Resources in order to make our Angular website:

- **NodeJS** (<https://nodejs.org/es/>)

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser.

- **Python 2.7.x** (<https://www.python.org/downloads/>)

Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high-level dynamic data types, and classes. Python combines remarkable power with very clear syntax.

- **Angular-cli** (<https://github.com/angular/angular-cli>)

The Angular CLI is a command-line interface tool that you use to initialize, develop, scaffold, and maintain Angular applications.

You can use the tool directly in a command shell, or indirectly through an interactive UI such as Angular Console.

- **TypeScript** (npm install -g typescript)

TypeScript is an open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and adds optional static typing to the language.

- **Tsun** (npm install -g tsun)

TSUN, a TypeScript Upgraded Node, supports a REPL (read–eval–print loop) and interpreter for TypeScript. When invoked without ts file argument, TSUN works as a repl where you can type in expression. When you pass ts file to TSUN in command line argument, TSUN will automatically run it with invisible compilation.

- **Visual studio code**

Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running, and version control.

- **Bootstrap**

Bootstrap is the most popular CSS Framework for developing responsive and mobile-first websites. It is a free and open-source CSS framework. It contains CSS- and (optionally) JavaScript-based design templates for typography, forms, buttons, navigation and other interface components.

- **Ngx-Translate** – for the multilanguage part

NGX-Translate is an internationalization library for Angular. It lets you define translations for your content in different languages and switch between them easily.

NGX-Translate is also extremely modular. It is written in a way that makes it really easy to replace any part with a custom implementation in case the existing one doesn't fit your needs.

- **Trello**

Trello is a collaboration tool that organizes the projects into boards.

- **Git-Hub**

GitHub is an open source code hosting platform for collaboration and version control. A GitHub repository can be used to store a development project.

It can contain folders and any type of files (HTML, CSS, JavaScript, Documents, Data, Images).

- **SourceTree**

SourceTree is a free Git desktop client for developers.

4. Methodology

First of all, we received the description of our final project: Time Control web app responsive and multilanguage.

We also received some recommendations to begin with our project. Raúl Novoa (our guide in the project) told us about the SCRUM methodology and how this could help us to manage all tasks to do working in team.

Scrum

Scrum is an agile way to manage a project, usually software development. It's a framework that allows collaborative work between teams. Scrum describes a set of meetings, tools, and roles that work in concert to help teams structure and manage their work.

Scrum also uses Sprints, which is a short, time-boxed period when a scrum team works to complete a set amount of work. Sprints are very important in agile methodologies.

"Sprints make projects more manageable, allow teams to ship high-quality work faster and more frequently, and gives them more flexibility to adapt to change."

<https://www.atlassian.com/agile/scrum/sprints> via @Atlassian

We first defined 3 points to start with:

1.- Read the Law that took effect from May about TimeControl and define the main requirements our application should meet

Reading the Law would take us to define our goals as we could extract from it that:

- Users should register their start and end time of work
- The company should keep these records for at least 4 years
- This information should be available for the users at any moment

2.- We had to investigate about the API to use

3.- Find a tool to help us to apply SCRUM. We needed to built the BACKLOG. Define and detail all the tasks our project involves defining how long can each of them take us and also the priority.

Raul talked us about JIRA, a tool to develop SCRUM in an efficient way. We first thought that we could implement SCRUM with this tool, but we were told that there weren't free licenses available for us and were told about using TRELLO instead.

For a better organization, we started to use Trello, which is a collaboration tool to organize projects into boards.

<https://trello.com>

It's very useful to see what's being worked on, who's working on what, and where something is in a process.

We implemented the **MoSCoW** method, in order to prioritize the tasks.

M- MUST: Not legal without it, cannot deliver a viable solution without it

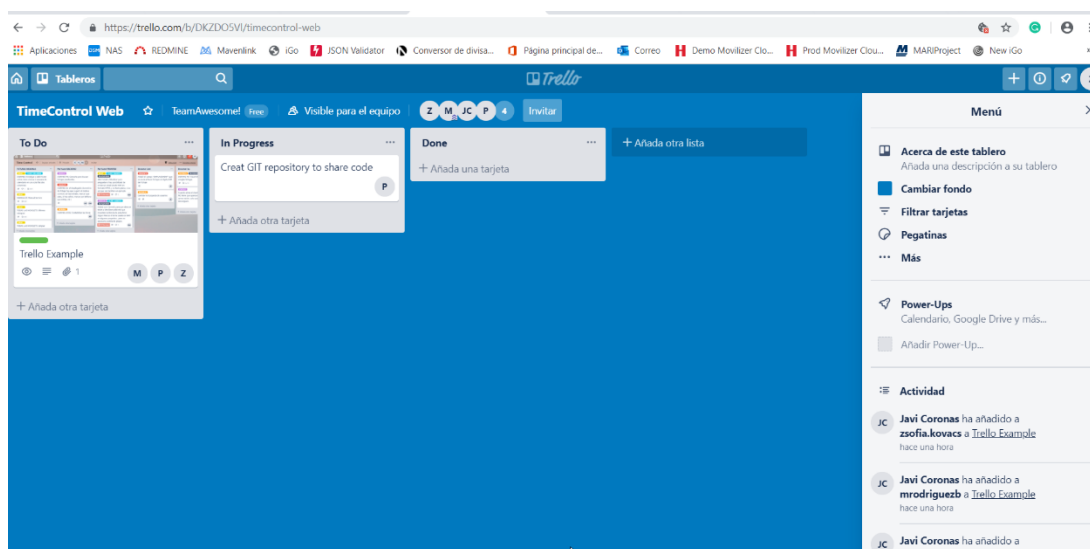
S-SHOULD: Important but not vital

C-COULD: Wanted or desirable but less important

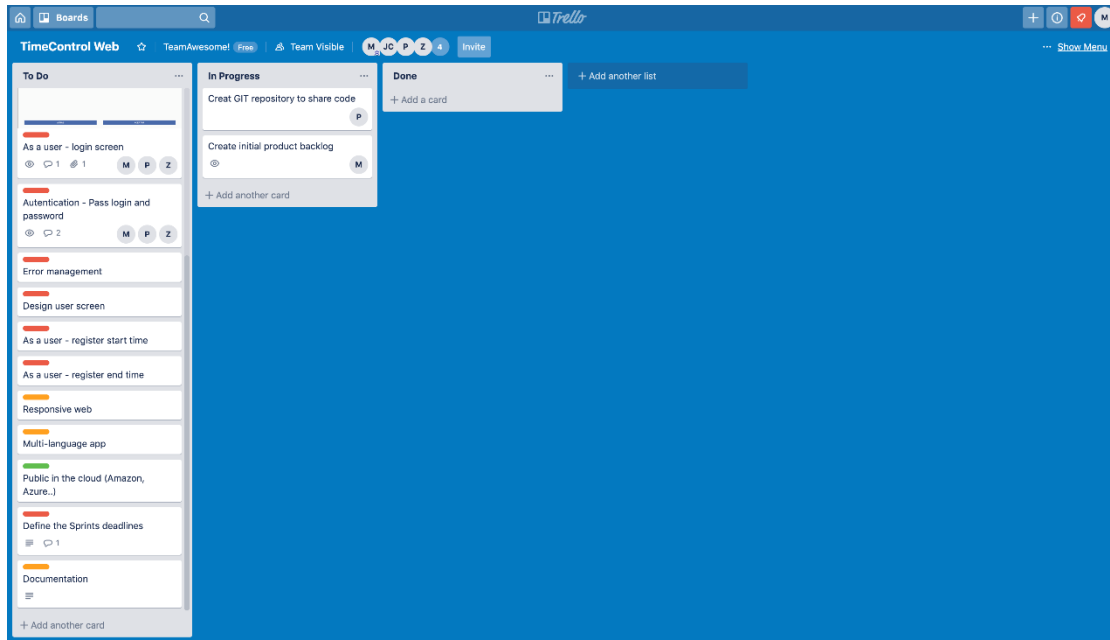
W-WOULD/WON'T: These are requirements which the project team has agreed will not be delivered (as part of this timeframe)

We used red colour for the MUST tasks, orange for SHOULD and COULD and green for WOULD.

Our Trello board in the first day:



We defined our first backlog this way.



We also established a couple of Sprint meetings.

In our first Sprint meeting we arrived to the point that our first and main task to do should be the Authentication-pass login user and password, the connection point with the “Movilizer API” as that task would be necessary to be able to complete the rest of tasks .

Also, we knew that this task would probably be the most challenging part of our project as our experience in connecting with API’s was limited. We asked about the Movilizer backend and we received some information in order to get the login.

In the connection process for the Login part, a double authentication is needed.

Authentication is using a “**Challenge-Response protocol**” which works with authentication through frontend-backend communication.

In the first communication of the frontend the EMAIL is sent, and in the second, the EMAIL and the answer of the challenge (RESP_DUEL) are sent.

In the first call where the EMAIL is sent, the Password is encrypted.

The backend responds with a unique DUEL value linked to the user, and a SALT value generated by each login request.

With these two values the frontend has to do the operations to calculate RESP_DUEL.

Now comes the second connection, login with the email and data is sent, and if everything is correct, the login is done.

If the credentials are incorrect, the backend sends an error message.

That's the information we received from the Backend developers about the Login process.

We tried first to connect to the backend applying what we had learned at class but after several unsuccessful tries we realized that task was going to be our first and more important stone in our way.

We received the same error messages all the time:

"Access to XMLHttpRequest at

'https://demo.movilizer.com/MovilizerDistributionService/m2m?dataformat=JSONCorrect&deviceAddress=@backend@dphuesca.es&password=qhxvnn70bsj22' from origin 'http://localhost:4200' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource"

We dedicate most time in our first week to this and also made a deep research in the web looking for information and thinking we could find the solution, but our research didn't arrive to a success in our connection to the backend.

We found a workaround; we installed a plugin in order to allow CORS.

CORS (Cross-Origin Resource Sharing) is a system, consisting of transmitting HTTP headers, that determines whether browsers block frontend JavaScript code from accessing responses for cross-origin requests.

The same-origin security policy forbids cross-origin access to resources. But CORS gives web servers the ability to say they want to opt into allowing cross-origin access to their resources.

Together with this research we structured our project and create the components and services we thought we could need.

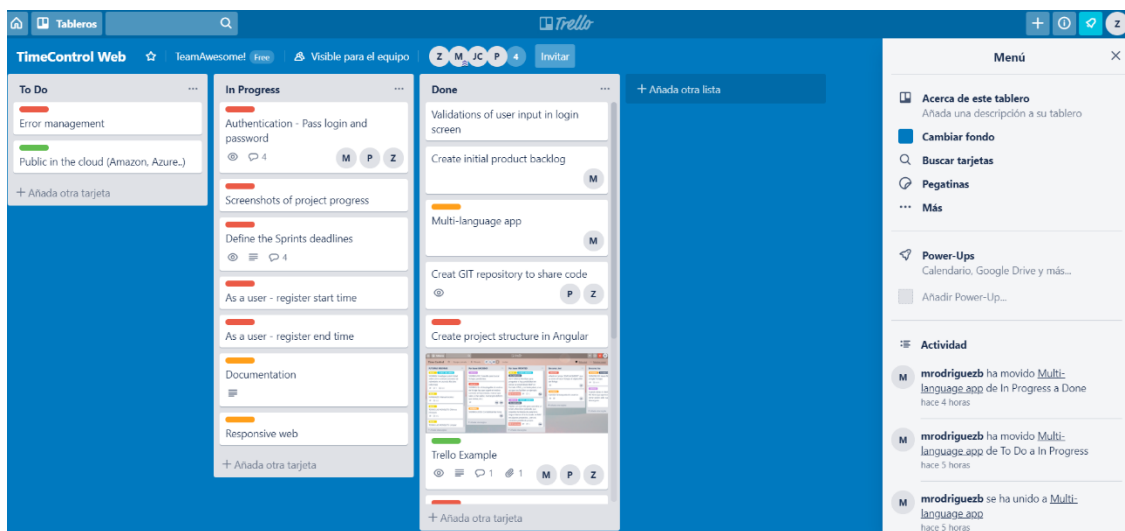
Finally, we realised that the movilizer API is not configured for external connections.

Connection can't be done directly, you can only make an access to the cloud through the portal. No access through an external call can use it.

Movilizer has its own structure and works based on its own language. All connections to the outside of the movilizer cloud are closed.

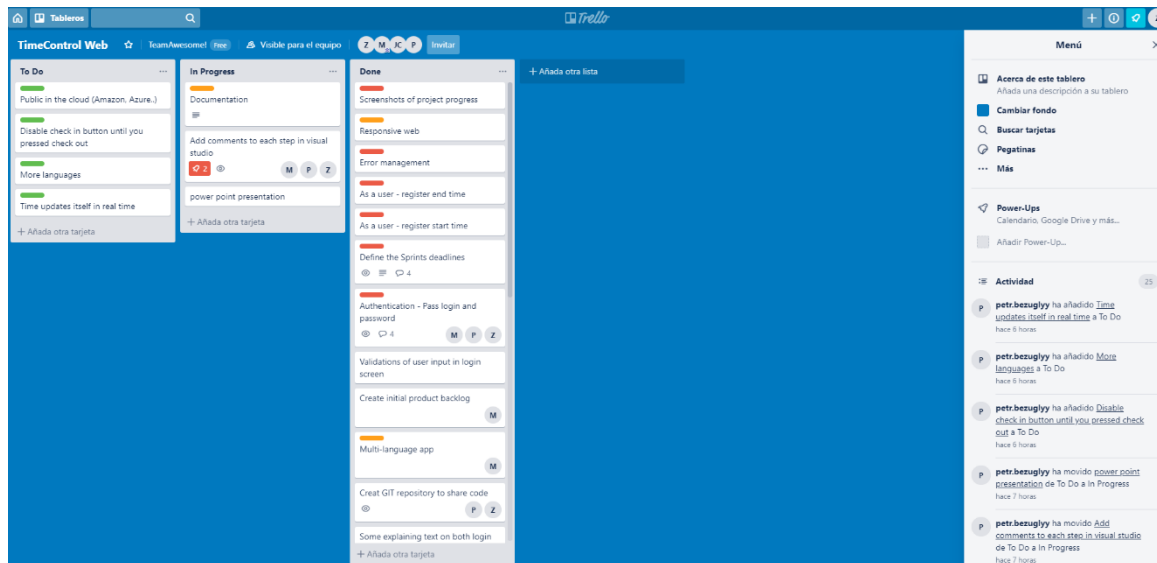
During our second week we decided we needed to advance somehow with the rest of tasks as we had dedicated a lot of time in trying to connect to the backend, but our effort didn't have results. We then began to take the rest of tasks.

This was our backlog after the second week.



Our Trello in the last working day.

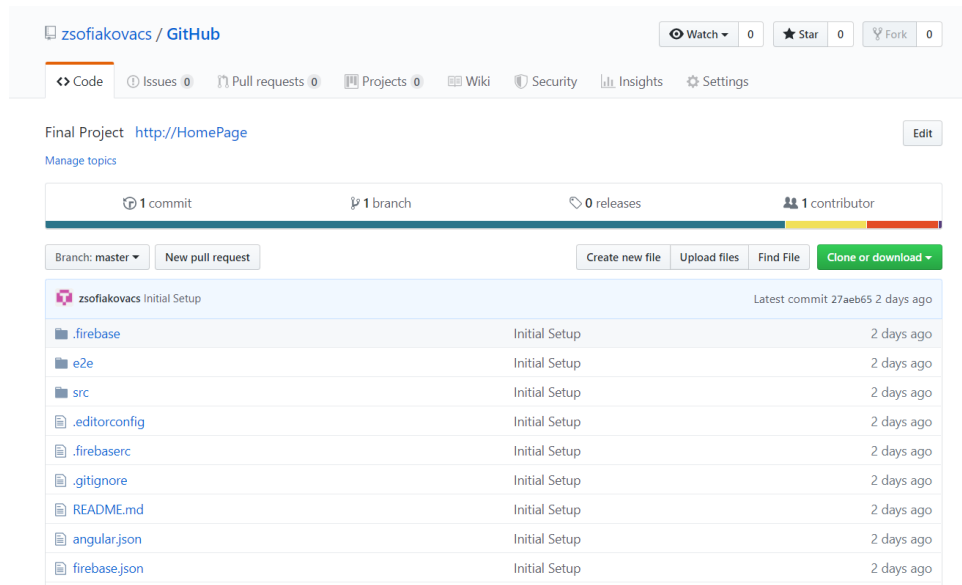
All the MUST tasks are done, only future improvements are left.



Also, as we were going to work together in remotely way, we decided to implement GitHub and SourceTree.

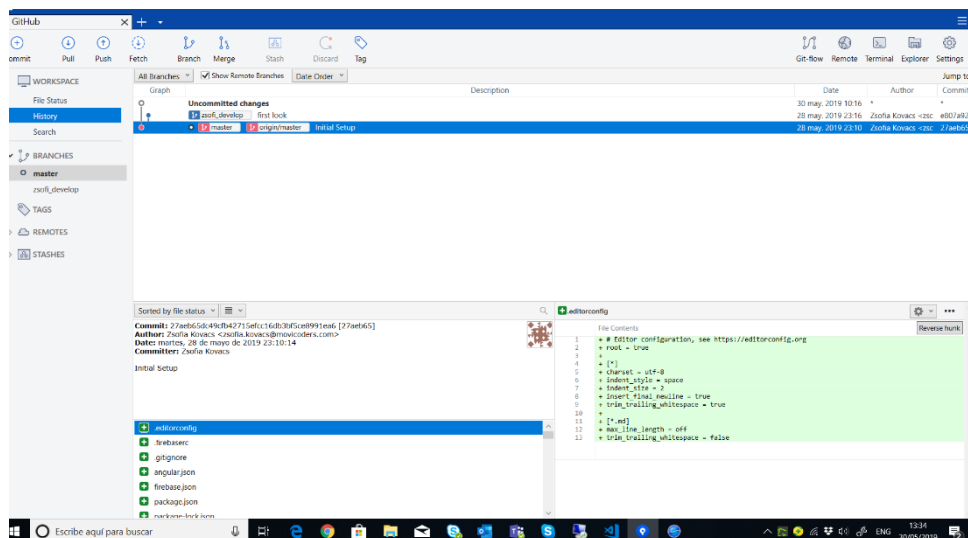
Although we hadn't used this tool before but considering SourceTree would reduce project duplicities, working with different versions of our project and that it would facilitate working remotely we dedicated part of our time to learn it and finally we were able to manage it.

Screenshot from our project in GitHub:

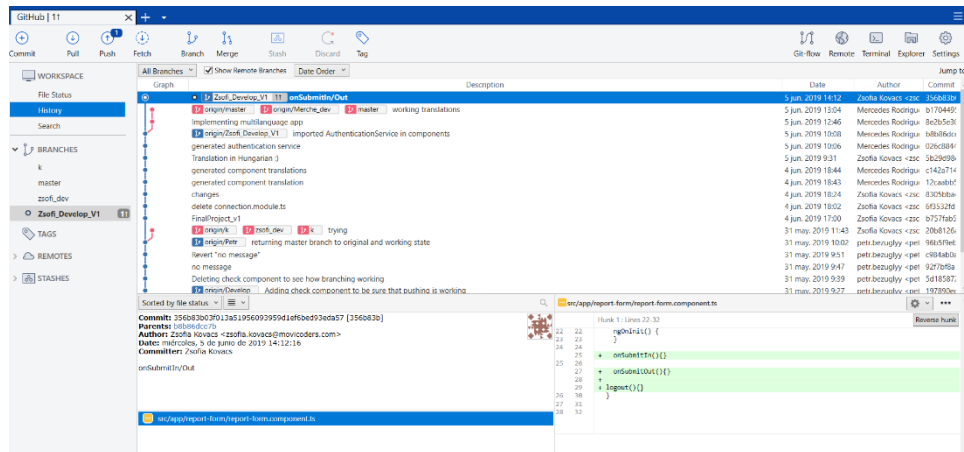


Screenshots from our project in SourceTree

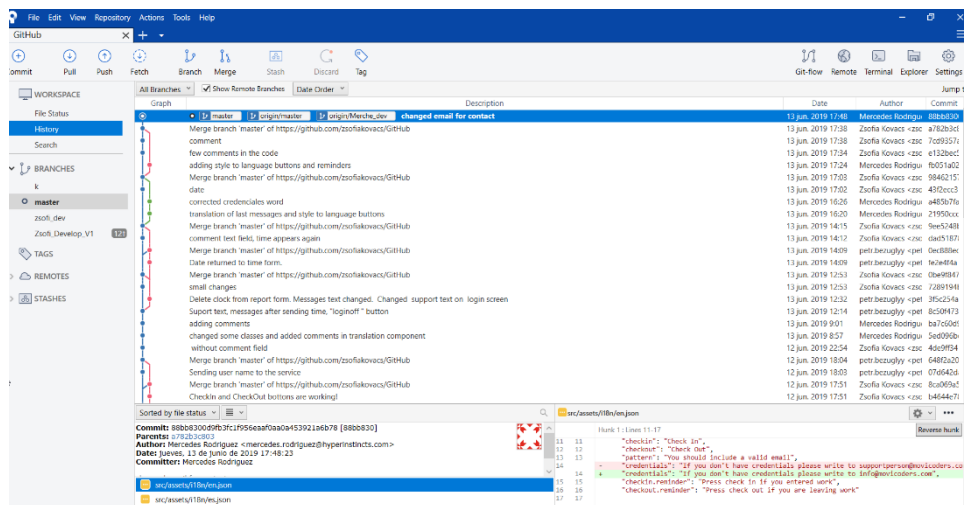
In the beginning:



When finally, we figured out how it worked, and how we could make our own branches, make commits, push our code or pull our colleagues' code, this program was very useful for us.



Finishing the app, that's how our SourceTree looked like.



As part of the SCRUM methodology, a daily meeting was organized with other members of the company who support the backend for the webpage. They asked us how we were going with the app, they sent us all the information they had, like json template for the login and time checking.

We also organized a daily meeting between ourselves. 5 min check-up, discussion of ideas, tasks and problems. As we work in different locations, we also used Skype to follow up each other's work and with Trello application we could follow up easily how the tasks were going.

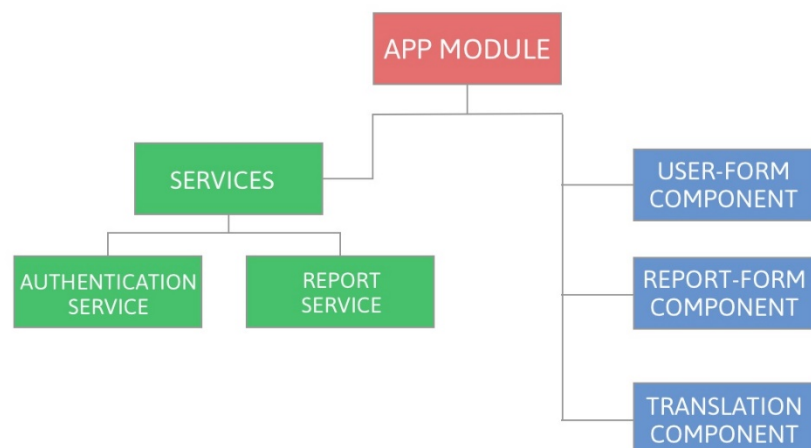
5. Analysis Development

The beginning of a project is challenging, there are many things to think about, a good organisation is crucial.

We were thinking first in the structure of our project and we arrived to this organisation.

Project structure in Angular:

- 3 Components: UserForm ,ReportForm and Translation
- 1 Services folder: report.service, authentication.service



The obligation is established that such records include both the beginning and the end of the working day, and this information must be recorded individually about each worker.

The worker should be able to login with his own credentials and send his records.

The first step was to decide the structure of the app.

We created 2 components and later one more, as we added the multilanguage function.

We started with creating simple draft with only two lines and one button. Screenshot of this draft shown below:

home works!

Name:

Password:

Login

For creating this form, we decided to use Template Driven Forms. We did this way for a few reasons:

1.- Official documentation recommended using Template Driven Forms (TD Forms) for things where information is not changing over time such as login pages, employees forms etc.

2.- It's easier to share info between services with TD Forms.

In developing forms part of our app, we get into trouble of sharing information between service and Reactive Driven Forms. This was another reason to use TD Forms.

3.- Doing validations in the HTML part is easier and saved us a lot of time.

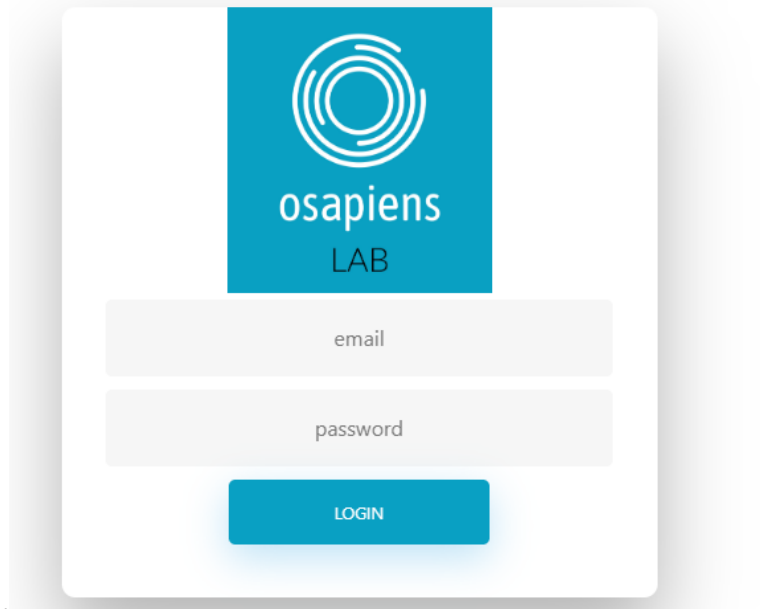
Since we are validating only static information, tools supported by TD forms were more than enough for us.

To share information between forms we created a service that would store information for different components. For that purpose, we created a class called "User". Inside of it, we set the parameters of this class.

Then inside the service, we created an Object variable called model and give it a user type.

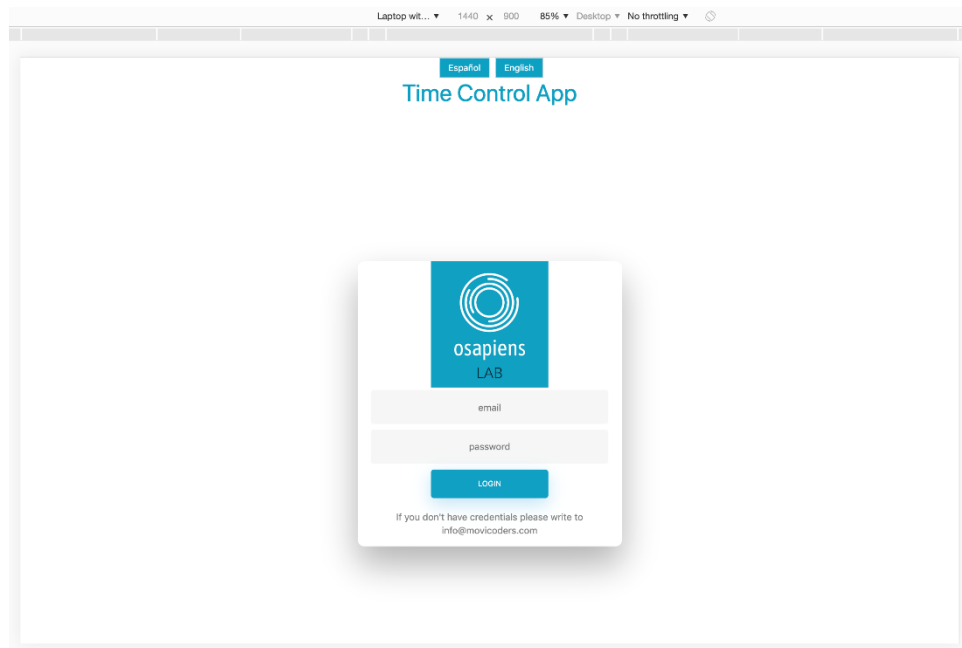
That way we were able to get information from our login form and send it to the report form by using login as name parameter from User class.

After that we moved into designing user screen using the colours and logo of Osapiens LAB

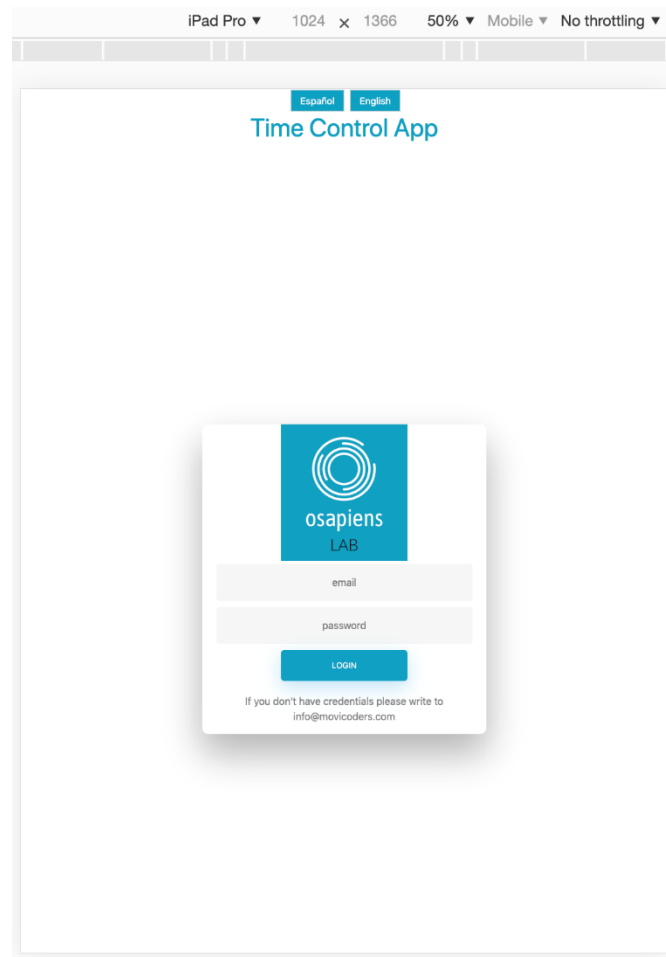


We also installed bootstrap to help us to make the application responsive. Here we include three screenshots showing these views from different devices.

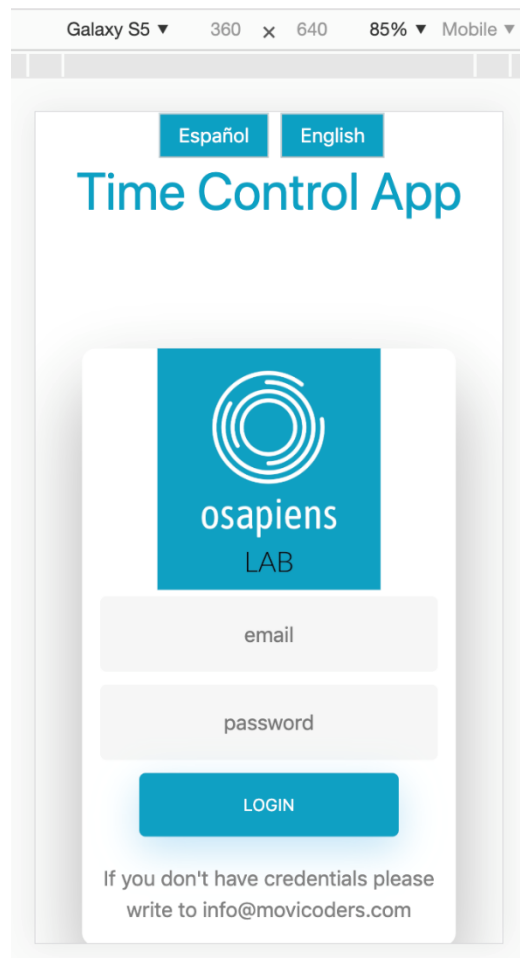
View from a laptop device:



View from a tablet device:



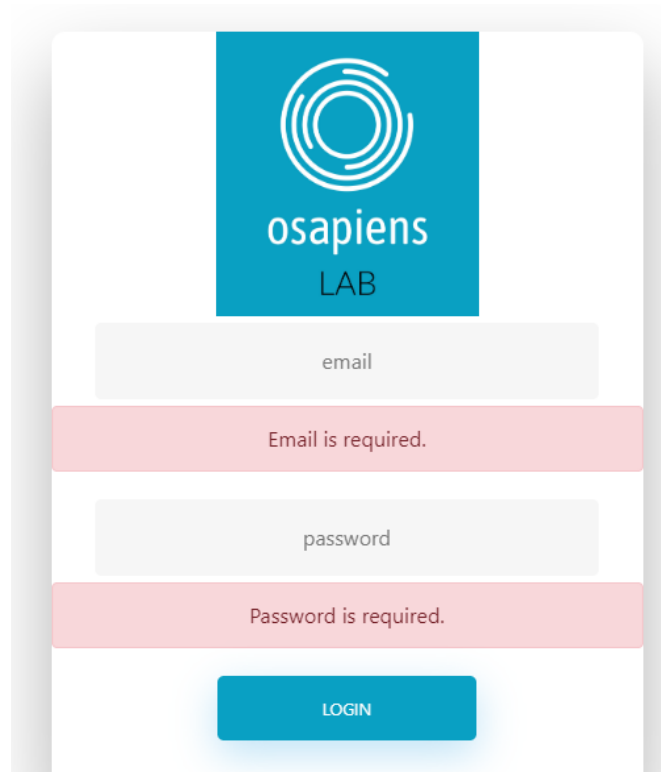
View from a mobile device:



After that step we went through validations:

If email and password fields are touched but left empty, warnings appear.

While warnings are on Login button is blocked and user cannot proceed.



The image shows a login form for 'osapiens LAB'. At the top is a blue square logo with a white circular icon and the text 'osapiens LAB'. Below the logo is a light gray input field labeled 'email'. A red warning bar appears below the email field with the text 'Email is required.'. Below the email field is another light gray input field labeled 'password'. A second red warning bar appears below the password field with the text 'Password is required.'. At the bottom of the form is a blue button labeled 'LOGIN'.


When login entered password is checked. If user entered less that minimum length alert above appear. Button is still blocked.

After all requirements met warnings disappeared and user can proceed.

Español

English

Time Control App



You should include a valid email

LOGIN


Enter your company email and a password and press login button. To get one please write to supportperson@movicoders.com

Email validation is also included, the email must contain “@” symbol and “.”

Español

English

Time Control App



You should include a valid email


Enter your company email and a password and press login button. To get one please write to supportperson@movicoders.com

The correct email format:

Español

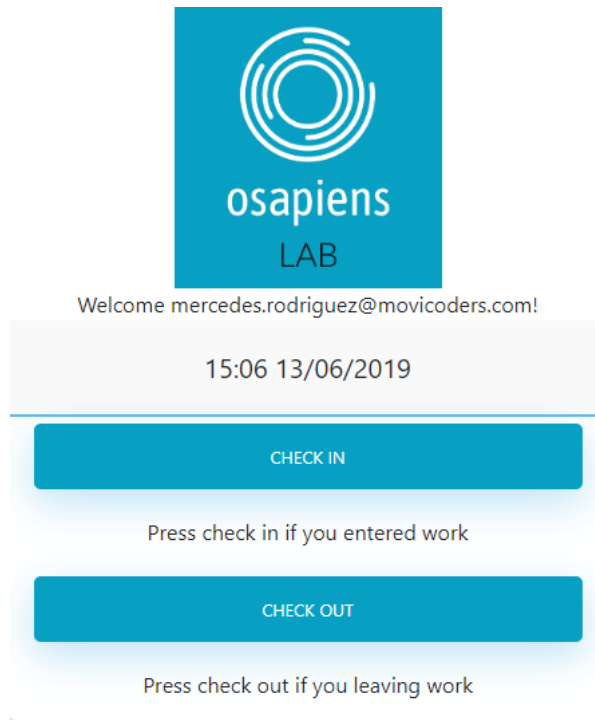
English

Time Control App



Enter your company email and a password and press login button. To get one please write to supportperson@movicoders.com

Once the correct credentials are sent, the user passes to the Check In/ Check Out page.



User's name is shown after the login.

We have also implemented the time, we think it's better for the user to see when he checks in or out.

One of the main characteristics our app should cover is that we needed a multilanguage app, so we investigated how to do this. We installed Ngx-Translate.

NGX-Translate is an internationalization library for Angular. It lets you define translations for your content in different languages and switch between them easily.

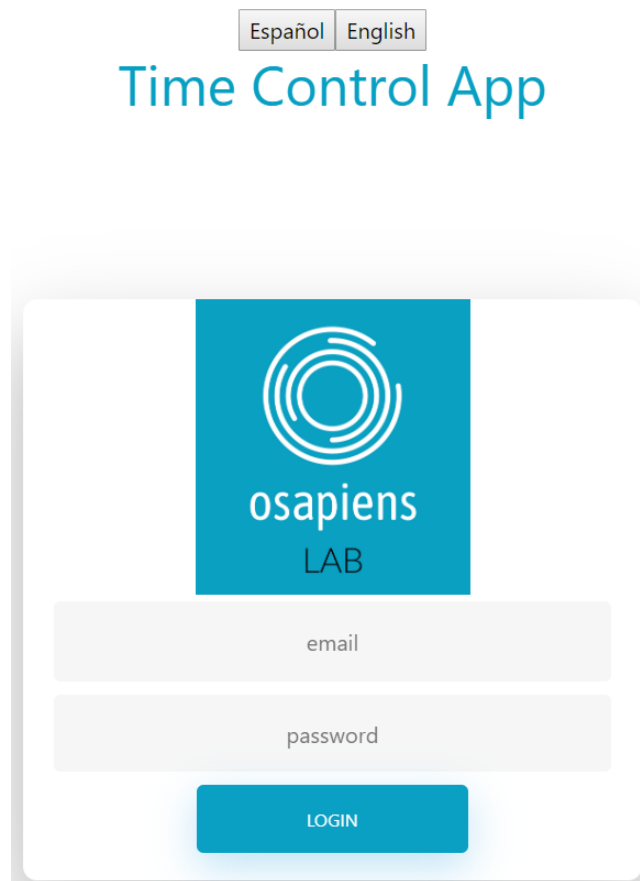
The NGx-translate/core contains the core routines for the translation: The TranslateService and some pipes.

The Ngx-translate/http-loader loads the translation files from the webserver.

We created a component called Translations that is the one in charge of changing the language.

First we had to inject TranslateService in the constructor and the next step was to set the default language of our application, in our case, English, using `translate.setDefaultLang('en')`

Each language is stored in a separate .json file. We created 2 files to have our app in English and Spanish.



Our app should connect to the backend, in our case it is: Movilizer cloud

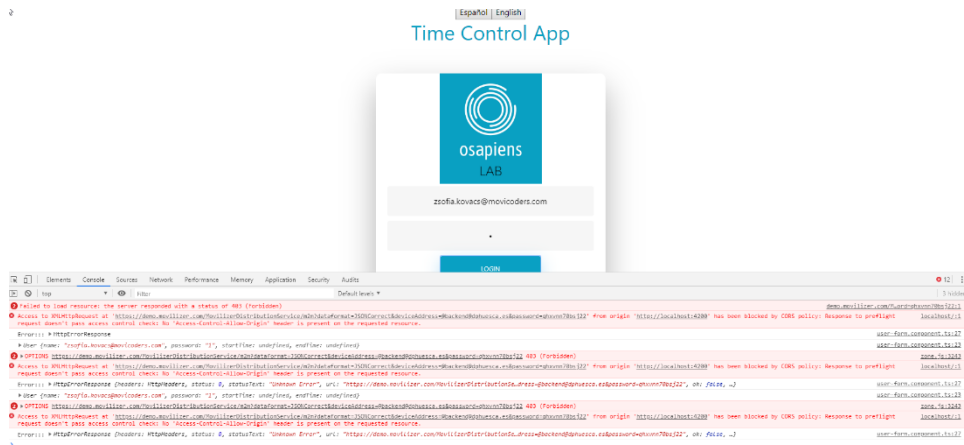
We tried to connect, but because of privacy policy, we weren't able to.

```

✖ *XMLHttpRequest https://demo.movilizer.com/MovilizerDistributionService/mse/dataformat=JSON&redirectdeviceaddress=backend@phuesca.es&password=qhvm70sj22_005 (+0ms)
✖ Access to XMLHttpRequest at 'https://demo.movilizer.com/MovilizerDistributionService/mse/dataformat=JSON&redirectdeviceaddress=backend@phuesca.es&password=qhvm70sj22' from origin 'http://localhost' blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.
✖ *HttpErrorResponse {headers: HttpHeaders, status: 0, statusText: "Unknown Error", url: "https://demo.movilizer.com/MovilizerDistributionService/mse/dataformat=JSON&redirectdeviceaddress=backend@phuesca.es&password=qhvm70sj22", ok: false}

```

So we had to find out how to make the connection.



We were still not able to connect because of CORS policy.

The suggested solution was to use a servlet, in order to connect our app with the Backend.

For the connection, a service was created, named authentication.service.ts

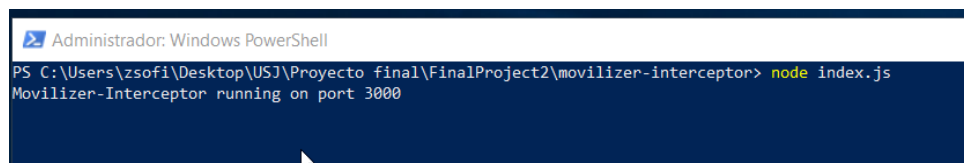
Movilizer Interceptor was finally created to intermediate between our app and the Backend.

It's a local server that acts as an intermediary backend to make the connection with the Movilizer backend (Movilizer Application FrameWork).

The Interceptor should be started with the following command in the console:

Node index.js

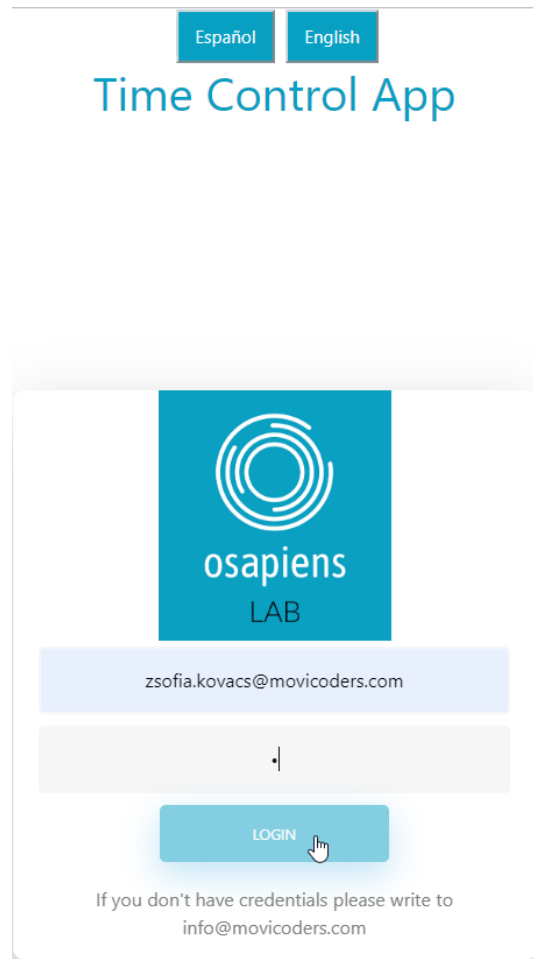
Once it's running, we can open our app and start with Login process.



This interceptor and part of the code was given to us, as unfortunately we couldn't do it by our own.

Login method was created, and data could be sent to the Backend via json format.

After using the correct credentials, the user can login.



In the screenshot below, the 2 calls can be seen. The first with the email, and the second with the email and RESP_DUEL (response of the challenge).

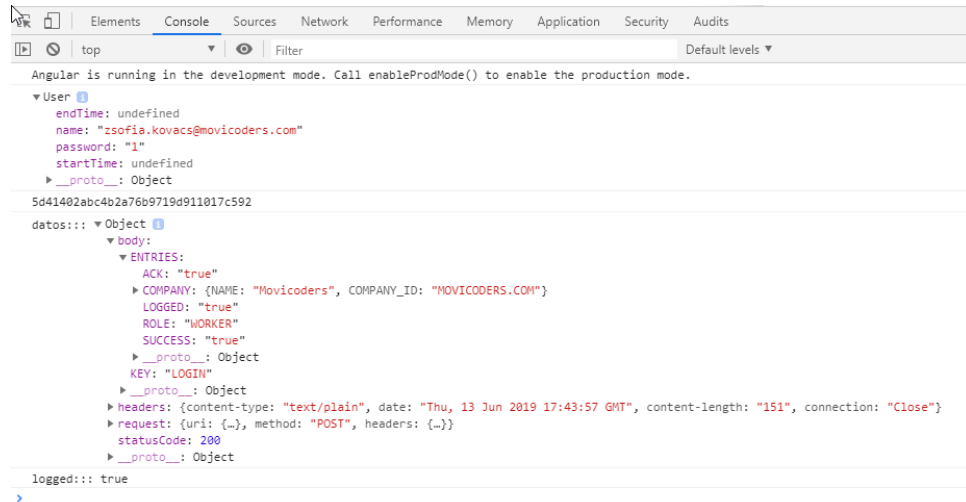
```

Administrator: Windows PowerShell
PS C:\Users\zsofi\Desktop\USJ\Proyecto final\FinalProject2\movilizer-interceptor> node index.js
Movilizer-Interceptor running on port 3000
Request:::POST
POST
https://demo.movilizer.com/MovilizerDistributionService/m2m?dataFormat=JSONCorrect&deviceAddress=@backend-web@dphuesca.es&password=ajsw3xhu8s183
{ EMAIL: 'zsafia.kovacs@movicoders.com' }
PETICION HECHA
Request:::POST
POST
https://demo.movilizer.com/MovilizerDistributionService/m2m?dataFormat=JSONCorrect&deviceAddress=@backend-dd@dphuesca.es&password=qhxvnn70bsj22
{ EMAIL: 'zsafia.kovacs@movicoders.com',
  RESP_DUEL: '70b44029719044873a4ac7c42007090' }
PETICION HECHA
  
```

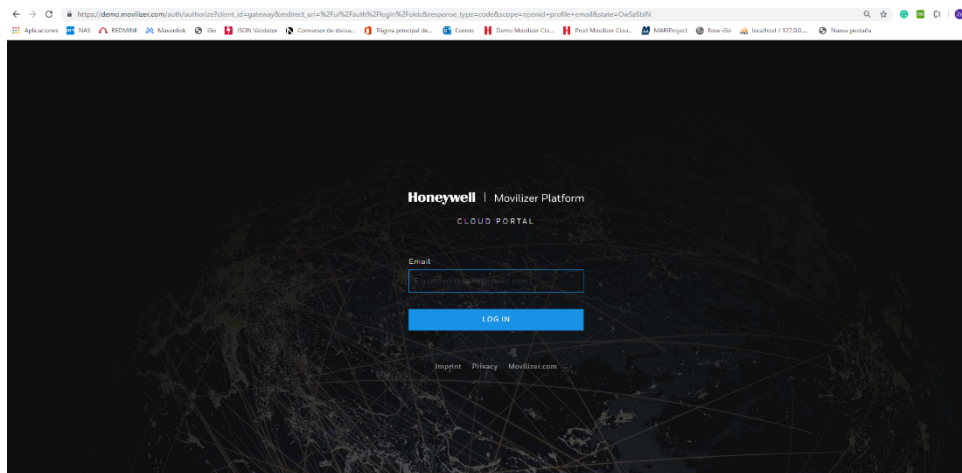
The screen after a successful login. The credentials are authenticated and are correct, so the user can continue with his time registration.

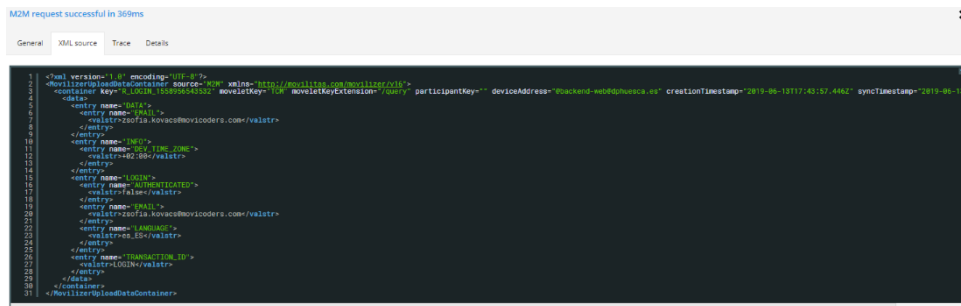


In the console we can check that we received the expected information.



We can check the Movilizer Cloud in order to see that our login was successful.

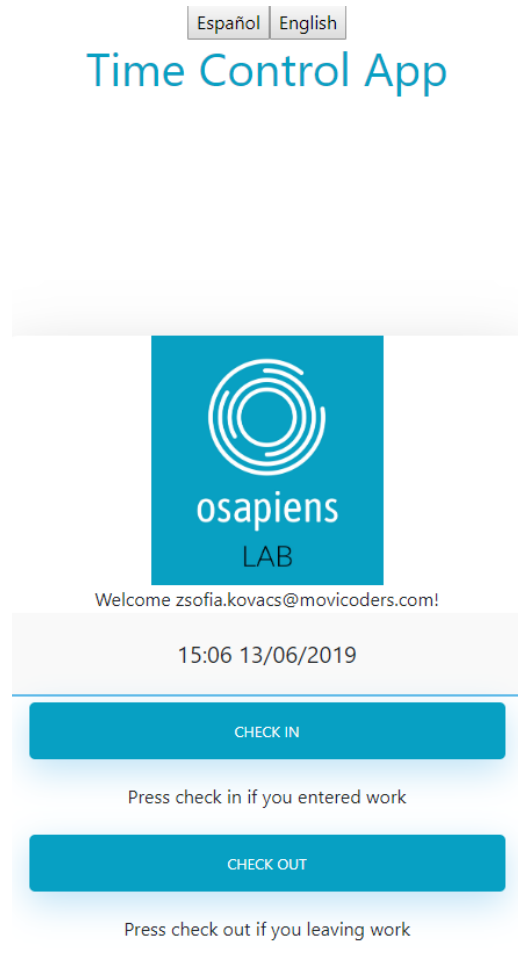




If we try to login with incorrect credentials, we receive an error message in Spanish, that's because the backend uses Spanish for the communication.



The main goal of this app is, to be able to send the entry time and check out time as per Royal Decree-Law 8/2019, so we created 2 buttons, CHECK IN and CHECK OUT.



Once the button is pressed, the information is sent via localhost:3000 to the Backend.

In the console we can observe that the connection is done, and the data is sent.

Check In:

```
POST
https://demo.movilizer.com/MovilizerDistributionService/m2m?dataFormat=JSONCorrect&deviceAddress=@backend-web@dphuesca.es&password=ajsw3xhu8s183
{
  REASON_ID: 0,
  DATETIME_SIGNING: 2019-06-13T11:30:47.000+02:00,
  SIGNING_TYPE: WORKING,
  REASON_OPERATION: WORKING,
  EMPLACEMENT: OUTSIDE_OFFICE,
  COMPANY_ID: MOVICODERS.COM,
  WORKER_ID: zsofia.kovacs@movicoders.com
}
PETICION HECHA
```

Check Out:

```
Request:::POST
POST
https://demo.movilizer.com/MovilizerDistributionService/m2m?dataFormat=JSONCorrect&deviceAddress=@backend-web@dphuesca.es&password=ajsw3xhu8s183
{
  REASON_ID: 0,
  DATETIME_SIGNING: 2019-06-13T11:31:14.000+02:00,
  SIGNING_TYPE: WORKING,
  REASON_OPERATION: LEAVING,
  EMPLACEMENT: OUTSIDE_OFFICE,
  COMPANY_ID: MOVICODERS.COM,
  WORKER_ID: zsofia.kovacs@movicoders.com
}
PETICION HECHA
```

To send the information, we used 2 different Json format, the difference was that in the Check In the operation reason was an Input and, in the Check Out it was an Output.

We made this way, because this is how the backend need to receive the information.

All the information that the user sends can be found in the Movilizer Cloud.

M2M request successful in 582ms

General XML source Trace Details

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <MovilizerCloudMessageContainer source="GSM" value="https://movilizer.com/movilizer/15">
3   <container Key="R_ASD_SESSION_159689288228" moveletKey="TO" moveletKeyExtension="/query" participantKey="" deviceAddress="Backend-web@huusca.es" creationTimestamp="2019-06-12T21:35:59.836Z" syncTimestamp="">
4     <data>
5       <entry name="DATA">
6         <entry name="SESSION_ID">
7           <valstr>0</valstr>
8         </entry>
9         <entry name="DATE-TIME-STAMPING">
10          <valstr>2019-06-12T21:35:59.836Z</valstr>
11        </entry>
12        <entry name="STANDING_TYPE">
13          <valstr>STANDARD</valstr>
14        </entry>
15        <entry name="SESSION_OPERATION">
16          <valstr>INUI</valstr>
17        </entry>
18        <entry name="EMPLOYMENT">
19          <valstr>POUSIDE_OFFICE</valstr>
20        </entry>
21        <entry name="WORKER_ID">
22          <valstr>MOVICODERS.COM</valstr>
23        </entry>
24        <entry name="WORKER_ID">
25          <valstr>cofia.kovacs@movicoders.com</valstr>
26        </entry>
27      </entry>
28      <entry name="IN_Z">
29        <entry name="IN_TIME_ZONE">
30          <valstr>+02:00</valstr>
31        </entry>
32      </entry>
33      <entry name="LOGIN">
34        <entry name="AUTHENTICATION">
35          <valstr>true</valstr>
36        </entry>
37        <entry name="MAIL">
38          <valstr>cofia.kovacs@movicoders.com</valstr>
39        </entry>
40        <entry name="GSM">
41          <valstr>WORKER</valstr>
42        </entry>
43        <entry name="LANGUAGE">
44          <valstr>es</valstr>
45        </entry>
46        <entry name="COMPANY">

```

Download

6. Results

With our app we have achieved our main goal, a functional web application.

Our app complies with the requirements of the law and the client. The user is able to login, and send his entry or exit time.

The app is multilanguage, which means that the language can be switched to English or Spanish.

Responsive form is also implemented, which also was our goal.

In general we created a simple app, without any complications for the user when he needs to register his work day.

7. Conclusions

This project was an interesting experience for all of us. We were able to use everything we learnt during the course and we also learnt so many new things.

We had to investigate about how to connect to an API, how Movilizer works, how to implement translation, and in general, how Angular works. Although we spent the last module experiencing with Angular, there are so many things to learn about it.

Work in team needs a good organisation and communication all the time. We were 3 in the team, so we could divide the tasks or help to each other if somebody got problems.

We are very satisfied with the result, we like our app very much, all hours worked have been worth it!

As we had limited time, we couldn't implement everything we wanted.

We have ideas for future improvements. We think that would be a good idea implement the following features:

- Enter like admin to create new workers or see detailed information about the time worked in a period
- View the Historial work of the logged user
- Fix the logs
- Implement more languages
- Deploy/Public in a cloud service (like Azure...)
- Time updates itself in real time

We want to thank the University San Jorge, Osapiens Foundation and Movicoders for the opportunity to participate and study in this course.

We have learnt so many interesting things, and we have a long way to go on learning.