# Development of a SaaS as an IT trainee

*2016/02/08*

## Introduction

My name is Samuel Monroe, I'm a 23 years old student at EPHEC Louvain-la-Neuve.

As I'm currently in my last year of bachelor in IT, I had to find a company which would give me the opportunity of an internship during 15 weeks.

I then applied to Belighted, which was offering an internship among a young team of passionated and motivated web developers, working on various and interesting projects. It seemed to me that it would be the best place to acquire great professional experience and programming skills, but also to really enjoy these 15 weeks.

Here's where my story with Belighted begins.

### My background

EPHEC is more focused on networking stuff than development, moreover I started quite lately to learn my first programming language.

However, I went through Pascal, Java, C and PHP during my studies.
I wanted to learn other languages and framework to achieve some projects, which led me to **Ruby on Rails** and basic **Javascript**.

## The SaaS Project

**Pravatar**'s goal is to provide web applications with a simple and efficient way to manage their users' avatars, allowing each user to have a different and private avatar on each Pravatar-enabled app they use, just with a single line of javascript.

This will also be a solution to avoid setting up a specific storage service for every app created by Belighted only for avatars, delegating this to the Pravatar service.

The back-end would be running on a **Rails** app, providing an API and a way to manage the accounts.
The front-end would be a Javascript third-party widget, which could be included in a client's app with a `<script>` tag requesting the script from the rails app.

# My First Week

## On the Back-end

### First steps

I spent my first day setting up the **Rails** environment and getting a bit used to **RSpec** testing.

Philippe asked me to install *ActiveAdmin* and play with it, writing some test in a User-Story way.

### Bases of the Pravatar service

As I'll describe later, the client side will rely on a script (included in the client app) which will interact with the app's API.

So my first objectives for the back-end were to find a way to provide the script, build a basic API and write test on it.

### Providing the script

My main problem here was to ensure that the script would still be available by its name in production, like `http://app.io/assets/script.js`, and avoid a digest on the filename produced by the assets pipeline.

This could have been an irritating problem, but I found a gem allowing me to white-list assets which would be available with and without the digest.
The gem is `non-stupid-digest-assets`.

### Building the API

Kathleen introduced me to the **api-versions** gem to manage an API, then I created a controller called `profile_widgets_controller` inheriting from the `profile_controller` and controlling my profile resources.

This single controller in the API just provides a path to the *avatar* from a received *profile_id*.

### Testing the API

The test is pretty simple, creating a fake account and a fake profile, stubbing the request with a valid ID. Then verifying the response has a valid **200** status, and containing a well formatted body.

## On the Front-end

## Choosing the technology

My objective here was first to define which technology to use in order to develop the **third-party widget** that will be included in the client's app.

This inclusion solution meant:

- The widget should be simple to install
- As it would be written in some *Javascript* framework, the whole package should not interfere with the client configuration.

As the available WebComponents technologies seemed to cause version conflict for now, the solution I chose after some discussions with Simon and Maxime was to bundle my javascript code with his whole jQuery version using **Browserify**.

## The Widget istelf

For now the widget is pretty simple but correctly fulfill my first goal, fetching the pravatars by:

- Parsing the DOM searching for a div with the specific `data-type`
- Making a request to the API using a key value
- Appending an `img` tag to the div using the API response

The process, was launched by the `ready` event, so wrapped in a `$('document').ready` event listener (*This was going to get me into some trouble later*).
Finally, Philippe asked me to rewrite it in **Coffeescript**, never wrote any lines in Coffee before but I quickly get into it.

## Testing

This is the most **painful** part of my first week.

I set up my test environment with Karma and Jasmine (*Jasmine to keep the BDD syntax of RSpec*).
Also, I had to include `jasmine-jquery`, `mock-ajax`, and `jquery` to Karma.

My main problems were:
- Understanding how Karma works
- Beeing aware that I could not test the script without stubbing my API
- Finding a solution to test and control something that was waiting for the DOM ready event, without knowing exactly when this event was occuring

So, this is how I got the whole test working (for now) with the precious help of Michael and Philippe :

1. In a `beforeEach` section, I **appended** my special pravatar div into the DOM, and I **stubbed** the API to intercept the calls and provide a specific response to any of these calls.

2. As the snippet code was wrapped into the `$('document').ready` listener, I had to create a way to call this code when I wanted to.
   This has been achieved by creating a `window` object, and giving him a single method called `init()`.
   Then, the `$('document').ready` listener would trigger the `window.object.init()` method.
   (*Sounds so simple now...*)

# Conclusion

This first week as an intern was really great, Philippe gave me a lot of advice and taught me a lot of useful things.

Managing to have my Javascript tests running after so much struggle made me feel great and even if I'm sure I'll encounter great difficulties again, I'm really motivated to achieve great work.

I enjoy the fact I'll be working at Belighted every day, and that's exactly what I expected of my internship.