Development of a SaaS as an IT trainee

2016/02/19

Introduction

This is my second blog post about Pravatar and my third week among the Belighted team is already over, time flies so fast!

In this one, I'll share with you what seemed to be the most interesting things to talk about for me.

In order not to lose you with my explanations nor to overload you with details about the SaaS, I'll begin with a little summary of the main features of Pravatar at the time of writing, then I'll talk about the back-end and the front-end.

Pravatar

Pravatar now has a default page where guests can register to the service, manage their accounts and set default avatar.

Client snippet now allows an app to request different versions of an avatar, to upload an avatar for a profile and also to create accounts on the fly.

On the Back-end

The main tasks on the back-end were to continue building the API and adapting the app in order to suit the client needs, but it led me to make some major changes to the app.

Interactor

One says :

> Fat models skinny controllers.

But Philippe taught me that putting all the logic in models is not the solution either.
What we want is to keep in the models their own specialized logic, but we can extract the business logic in a fourth layer which is called an **Interactor**, and which reflects (as the creators of the gem say) what our application *does*.

I built an interactor to represent the *registration* process, it simply creates a `user` and its `account`

with given parameters, and returns a context with a success or a failure.
This *interactor* is then called by a *controller*, which will additionaly perform its more specific actions.

AWS S3

Despite how obscure it sounded to me, setting up S3 to store the *avatars* was quite fun.

The **fog** gem provides a cloud services library, which can easily be used with **CarrierWave** to uploads your files to S3.
My requirements were to keep the local storage for *testing*, and to target three different buckets for *development*, *staging* and *production*.

This was simply done by giving **fog** my AWS credentials, make **CarrierWave** use *file* or *fog* storage depending on the current environment, and define the right *buckets* for the non-testing environments.

Once configured, no changes were needed in the avatars implementation, **CarrierWave** works like a charm.

On the Front-end

Gulp : Comfy Testing & Workflow

Lastly, my javascript tests were located in the `spec` folder which normally concerns the RSpec tests on the Rails app only.
The project has two distinct sides but we wanted to keep the client one close the the server one, in order to easily maintain the whole package and simply because the **js snippet** is provided by the Rails app.

So I created a dedicated directory on the app's root for all the front-end app, and gave it the miraculous tool which is **Gulp**.

Gulp revealed itself a **freaking great** solution to a whole bunch of current or even future problems :

- How to easily launch tests while working on the app
- How to manage the bundling with jQuery everytime the snippet source change?
- As the tests and sources are written in coffeescript, how to avoid the early solution which was to use the rails assets precompiler?

The workflow on the widget now goes like this:

1. Launch gulp to `watch` for any changes in my .coffee sources, to compile them if it occurs
2. Write the tests
3. Write the snippet code, which will be compiled and bundled on the fly, and also automatically tested by gulp & karma with the tests I just wrote.

Sorry for hating you two weeks ago, testing.

Improvements on the Widget

Fetching the avatars

First of all, in order to move towards what the project will be later, the avatars are now fetched by the combination of an **account id** and an **external id** referencing the profile.
This means an application can register profiles for their users providing Pravatar their "*primary key*", no matter what they are (mails, usernames, ...).

Upload and default avatar

Account owners are now able to define a default avatar for their profiles, which will automatically be displayed if a requested profile doesn't exist or doesn't have its avatar.

An upload functionnality has been added to the widget, a pravatar `div` can now be used to define an upload zone for a requested user, existing or not.
If an upload occurs on a non-existing profile, it will be created on the fly by Pravatar on the related account.
This is moving towards the **plug'n'play** characteristic of Pravatar, an easy-to-use service and no constraints for the app which will use it.

Conclusion

Step by step, increment by increment, the SaaS is progressively taking shape.

It's only the beginning but I really enjoy working on it, and also learning valuable things every day and working with all the people at Belighted.