Pravatar : Developper's Guide

Création d'un compte utilisateur

Pour pouvoir utiliser Pravatar sur une autre application, il est nécéssaire de posséder :

- Un projet sur Pravatar
- La clé `API` associée à ce projet

La création de projet passe par la création d'un compte utilisateur sur le site web de Pravatar.
Une fois un compte utilisateur créé, la section `My Account` permet la création de multiple projets et de la clé **API** associée.

Regénération de la clé API

En cas de clé API compromise, il est possible de générer une nouvelle clé API pour un projet donné dans cette section **My Account**.

Avatar par défaut

Un avatar par défaut peut-être défini pour un projet donné via l'option `Edit Project`, cet avatar par défaut sera utilisé par le service pour tout profil ne possédant pas d'avatar.

Implémentation de Pravatar

Via la gemme et le snippet

Des applications web simples permettent l'utilisation du snippet javascript, et une application Rails peut inclure la gemme pour encore plus de facilité.

L'inclusion du snippet via le placement d'un tag `<script src="http://pravatar.url/pravatar.js"></script>` dans les headers html de l'application.

La gemme nécéssite le placement de la clé API du projet et le nom du projet sous les constantes *pravatar_key* et *pravatar_project* dans `secrets.yml`. Elle permet ensuite l'utilisation de deux helpers pour générer les zones d'affichage et d'upload des avatars :

- `pravatar_display(external_id, options = {})` : Permet la définition d'une zone d'affichage via l'id utilisateur et permet le placement d'options pour ajouter des attributs au tag

`<img>` généré.
- `pravatar_upload(external_id)` : Permet la définition d'une zone d'upload via l'id d'un utilisateur.

Via l'API

Des solutions utiliant des frameworks tels qu'Angular ne sont pas adaptées pour l'utilisation de la gemme et du snippet javascript.

Néamoins l'utilisation de l'API permet d'utiliser assez simplement le service de Pravatar sur l'application.

Display

L'affichage d'un avatar se fait sur l'entrée **GET** `/api/project/:name/profile/:id` de Pravatar.

Celle-ci requiert uniquement le nom du projet et l'id du profil dans l'url, et renvoie la réponse suivante :

```
data = {
        'type' => avatar_type (profile or project_default),
        'attributes' => {
            'avatars' => {
                'thumbnail' =>  thumbnail_avatar_link,
                'normal' => normal_avatar_link,
                'large' => large_avatar_link,
                'original' => avatar_link,
            }
        }
    }
```

Upload

L'upload d'un avatar se fait sur l'entrée **POST** `/api/project/:name/profile/:id` de Pravatar

Cet appel requiert un *header* http `Authorization` conçu avec la libraire `JWT`, formé des éléments suivants :

- **payload** : JSON object forme d'un `external_id` qui est l'id du profil ciblé, un `project_name` qui est le nom du projet utilisant Pravatar et d'un `datetime` formé avec le Time.now() du serveur de l'application.
- **key** : L'API key du projet en question.
- **algorithm** : HS256

Les données d'images qui doivent être envoyées en paramètres d'une requête d'`update` suivent la spécification JSON:API et se présentent sous la forment suivante :

```
'data' => {
              'type' => 'profile',
              'attributes' => {
                  'avatar' => {
                      'data' => image_b64_data,
                      'name' => image_name
                  }
              }
          }
```

Suite à un appel réussi sur cette entrée, la réponse émise par le serveur est typiquement celle reçue par un `show`, ce qui permet de mettre à jour l'affichage du profil immédiatement après upload en utilisant cette réponse :

```
data = {
        'type' => avatar_type (profile or project_default),
        'attributes' => {
            'avatars' => {
                'thumbnail' =>  thumbnail_avatar_link,
                'normal' => normal_avatar_link,
                'large' => large_avatar_link,
                'original' => avatar_link,
            }
        }
    }
```

Développement sur Pravatar

Snippet Javascript

Le snippet client est localisé dans le directory `client`.

Les technologies principales utilisées pour ce script sont :

- Browserify: pour la décomposition du script en modules et l'intégration isolée de jQuery.
- Karma/Jasmine pour le testing dont la configuration se trouve dans le `karma.conf.js`
- Gulp pour l'automatisation de tâches. Le fichier de config est `gulpfile.js`

Sources

Les sources du snippet se trouvent dans le dossier `src/pravatar/coffee`, ces fichiers coffeescript sont séparés en modules intégrés via Browserify.

Les tests sont également écrits en coffeescript et intègrent des modules de Pravatar pour certains éléments

de tests spécifiques, ils couvrent les fonctionnalités d'upload et de download des avatars. Ils se trouvent dans le dossier `src/test`.

Compilation et testing

La compilation et le bundling Browserify sont simplifiés via des commandes Gulp présentes dans le `gulpfile.js` :

- `make_snip` : Compile les sources coffeescript du snippet et lance Browersify sur le résultat Javascript, l'output final est ensuite renvoyé dans le dossier public sous le nom `pravatar.js`.
- `make_test` : Compile les sources coffeescript des tests pour pouvoir les faire tourner par Karma/Jasmine.
- `watch` : Lance une surveillance de l'ensemble des sources coffee pour lancer une compilation/bundling sur chaque changement.
- `test` : Lance la suite de test Karma une fois
- `tdd` : Lance la suite de tests à chaque changements des sources.

Backend Rails

Le backend Rails comporte trois facettes différentes.

User Interface

L'interface utilisateur est celle qui permet aux clients de s'enregistrer et de créer des projets afin d'implémenter le service Pravatar.

Elle est conçue comme une application Rails classique, un `interacteur` a été ajouté en plus sur le MVC.

Administration

Le panel d'administration est conçu avec `ActiveAdmin`, seul un utilisateur avec l'attribut `admin` à *TRUE* peut accéder à l'administration via `/admin`.

Cet accès est protégé par la méthode d'authentification `authenticate_user_admin!` présente dans l'`application_controller` et spécifié dans l'initialize d'ActiveAdmin.

API

L'API est conçue avec la gemme `api-versions` et possède deux méthodes liées aux deux actions possibles sur un avatar :

- **Show** pour récupérer un avatar sur base d'un id unique propre à la gestion utilisateur du client.
- **Update** pour mettre à jour l'avatar sur base de l'id unique et de données b64 de l'image souhaitée.

L'update sur un profil non-existant conduit à sa création en base de données.

`ActiveModel Serializer` a été inclus dans le projet pour la création de réponses conformes aux spécifications JSON:API.

Les tests API situés dans `spec/requests` contiennent des exemples précis des structures de données utilisées pour la communication avec l'API.