Chargement et affichage des donnees :

Comment ça marche ?

La fonction _getGithubGistFiles

Chargement des données : La fonction _getGithubGistFiles

Quelles opérations sont réalisées par cette méthode de classe ?

Charge les différents code sources du gist hébergés sur Github :
Un gist peut contenir de un à plusieurs fichiers.
Les réponses des requêtes renvoyées par le serveur Github sont mise en cache dans le dossier app/cache/prod/githubapi.
à noter
en cas de modification des données sur le serveur github, un mécanisme interne au bundle knplabs/github-api met à jour ce cache.
aucun filtrage des données n'est effectué, c'est la réponse brute qui est stockée.

Convertit automatiquement le code source de chaque fichier php en texte formaté
Une fois les fichiers chargés en cache, une opération de conversion des données sur le code source est effectué sur chaque fichier.
Elle permet de *mettre en lumière* le code source et de créer les liens pointant vers la documentation officielle.
Afin d'accelerer chaque chargement de page et vu que chaque formatage est coûteux en temps processeur, tous les fichiers convertis sont mise en cache respectivement dans les répertoires adéquates.
à noter :
Cette conversion est réalisée par une classe spécifique pour les fichiers php (dû à un bug non résolu dans l'interprétation du flux par Geshi)

Le code :

1-PublicadminController.php

```
1.  <?php
2.  private function _getGithubGistFiles($id_gist) {
3.
4.          $service = $this->container->get('github_api');
5.
6.          $gists = $service->getClient();
7.
8.          /* commentaire : identifiant en dur pour test */
9.
10.         $id_gist = '5947930';
11.
12.         /* commentaire : réponse mise en cache dans app/cache/prod githubapi */
13.
14.         $gist = $gists->api('gist')->show($id_gist);
15.
16.         $h = new HighlightphpService();
17.
18.         $cache = $this->container->get('kernel')->getCacheDir() . '/highlightphp/' . $gist['id'];
19.
20.         /* commentaire : on créé le répertoire si il n'existe pas ,chaque gist possède son propre répe
21.
22.         if (!is_dir($cache)) {
23.             @mkdir($cache, 0777, true);
```

```
24.         }
25.
26.         foreach ($gist['files'] as $data) {
27.
28.             /* commentaire : conversion pour les fichiers php */
29.
30.             if ($data['language'] == 'PHP') {
31.
32.                 if (file_exists($cache . '/' . $data['filename'])) {
33.
34.                     $content_date_dreated = date(DATE_ISO8601, filemtime($cache));
35.
36.                     /* commentaire : si le code sur le serveur github est plus récent que le cache, on
37.
38.                     if ($gist['updated_at'] > $content_date_dreated) {
39.
40.                         $h->loadFile($data['raw_url']);
41.
42.                         $content = $h->toHtml();
43.
44.                         file_put_contents($cache . '/' . $gist['id'] . $data['filename'], $content);
45.
46.                     }else{
47.
48.                         $content = file_get_contents($cache . '/' . $data['filename']);
49.
50.                     }
51.                 } else {
52.
53.                     /* commentaire : initialisation de la convertion des fihiers et mise en cache */
54.
55.                     $h->loadFile($data['raw_url']);
56.
57.                     $content = $h->toHtml();
58.
59.                     file_put_contents($cache . '/' . $data['filename'], $content);
60.
61.                 }
62.
63.                 $files_data[] = array('filename' => $data['filename'], 'content' => $content, 'languag
64.
65.             } else {
66.
67.                 $files_data[] = array('filename' => $data['filename'], 'content' => $data['content'],
68.             }
69.         }
70.
71.         return $files_data;
72.     }
```

La vue *read*

Affichage des données : La vue *read*

Le multiple héritage grâce au moteur de template Twig

> **Appel dynamique des templates**
> Chaque article bénéficie d'un template personnalisé, enregistré en base de données, permettant une mise en page dédiée en fonction

des données à exposer.
Dans la vue *read*, c'est la ligne 19 qui se charge d'inclure le template adéquate.

`{% include "foxp2projectsBundle:Common:" ~ template ~ ".html.twig" %}`

Le code :

2-read.html.twig

```
1.  {# \Symfony\src\foxp2\projectsBundle\Resources\views\Publicadmin\read.html.twig #}
2.  {% extends "foxp2projectsBundle:Common:layout.html.twig" %}
3.  {% block application %}
4.  {% block headerapp %}
5.  <div id="nav-bar">
6.      <div class="row">
7.          <div class="span9">
8.              <div id="header-container">
9.                  <a id="backbutton" class="win-backbutton" href="{{ path('publicadmin') }}.{{
    app.request.getRequestFormat}}"></a>
10.                 <div class="dropdown">
11.                     <h1 class="header-dropdown accent-color">{{ 'Read this article'|trans }} <small>
    : {{ titrearticle }}</small></h1>
12.                 </div>
13.             </div>
14.         </div>
15.
16. {% endblock headerapp %}
17. {{ parent() }}
18.
19. {% include "foxp2projectsBundle:Common:" ~ template ~ ".html.twig" %}
20.
21.
22. {# reste du code #}
```

La vue *articlewizard*

Affichage des données : La vue *articlewizard*

Comment cette vue expose les données

Traitement des données chargées dynamiquement : initialisation
Le plus gros du travail est effecué par cette vue.
Elle expose les données provenant de la base données :
- description complète des articles (titre /sous titre / etc ...)
Elle expose les données provenant des serveurs github :
- Conversion du code source *à la volée*
En fonction du langage, via un tag Twig (*highlight()*), elle fait appel à des méthodes de classe internes à geshi et les mets en cache.
Dans la vue, cela se résume à 4 lignes de code :

```
{% if gvalue.language == 'PHP' %}
{{ gvalue.content|raw }}
{% else %}
{{ highlight(gvalue.content,gvalue.language|lower,'geshi') }}
{% endif %}
```

Traitement des données chargées dynamiquement : via le cache
Lorsqu'un article a déjà été lu, et que chaque fichier code source a déjà été converti, Geshi se charge d'appeler les fichiers mis en cache.
Résultat : la page se charge plus vite.

Le code :

3-articlewizard.html.twig

```twig
{# \Symfony\src\foxp2\projectsBundle\Resources\views\Common\articlewizard.html.twig #}
<div class="row-fluid wizard">
    <div  class="span3" style="position:fixed">
        <div class="well" id="wizard-steps-container">
            <h2>{{ soustitre }}</h2>
            <ul id="wizard-steps">
{% for key,value in dbdata%}
                    <li>
                        <a href="#{{ key }}">{{ key+1 }} : {{ value.titlearticle|raw }}</a>
                    </li>
{% endfor %}
                </ul>
            </div>
        </div>
        <div class="span9 pull-right">
            <section id="wizard-step-content">
{# première boucle sur les données provenant de la base de données #}
{% for key,value in dbdata%}
                <div id="{{ key }}">
                <article>
                {# description de l'article #}
                {{ value.shortdescription|raw }}
                </article>
    {# deuxième boucle sur les données provenant de github avec une intersection avec celles provenant de la bdd #}
    {% for gkey, gvalue in gdata %}
        {# chaque description d'article possède un code source #}
        {% if key == gkey %}
            <h1 class="text-info lead"><strong>Le code : </strong><small class="pull-right fox-sous-titre">{{ gvalue.filename }}</small></h1>
            {% if gvalue.language == 'PHP' %}
                {# Mise en lumière du code stocké en cache traité en amont via la méthode de classe _getGithubGistFiles #}
                {{ gvalue.content|raw }}
            {% else %}
                {# Mise en lumière dynamique du code via Geshi : html5, twig, javascript, sql #}
                {{ highlight(gvalue.content,gvalue.language|lower,'geshi') }}
            {% endif %}
        {% endif %}
    {# fin de la deuxième boucle #}
    {% endfor %}
                 </div>
{# fin de la première boucle #}
{% endfor %}
             </section>
          </div>
</div>
```