Application GSB_FRAIS avec Symfony2

Description du thème

Propriétés	Description
Intitulé long	Gestion des frais de mission des visiteurs du contexte GSB réécrite en utilisant le framework Symfony2
Formation concernée	BTS SIO PPE4 PPE5
Matière	PPE, SLAM4
Présentation	Le support propose de réutiliser la couche modèle de l'application de gestion des frais du contexte GSB visible à : http://www.reseaucerta.org/cotecours/cotecours.php?num=561 en utilisant le modèle MVC du framework Symfony2
Notions	Modèle MVC, moteur de template de pages
Pré-requis	PHP, MVC
Outils	Framework Symfony2, environnement de développementNetBeans
Mots-clés	GSB, Modèle MVC, template, Framework, Symfony2
Durée	6 heures
Auteur(es)	Patrice Grand
Version	v 1.0
Date de publication	Février 2013

Présentation

L'objectif est ici de présenter et commenter l'application GSB version MVC re-développée en utilisant Symfony2. Nous nous attacherons à préciser certains points clés.

Il ne s'agit pas de faire un cours sur Symfony2 ; il n'existe pas aujourd'hui (février 2013) de livre sur la version 2 mais plusieurs sites font une présentation détaillée de découverte du produit largement suffisante pour débuter. Deux sites en particulier sont à consulter :

- http://symfony.com/fr/doc/current/book/index.html : le site officiel (en français)
- http://www.siteduzero.com/informatique/tutoriels/symfony2-un-tutoriel-pour-debuter-avec-le-framework-symfony2 : un support très bien fait sur le *Sitedu Zéro*.

Les choix

Symfony2 propose par défaut une architecture MVC ; de nombreuses fonctionnalités sont disponibles mais pas obligatoires. C'est ainsi que nous n'utiliserons pas le moteur ORM Doctrine (les avis sont largement partagés sur cette question et nous ne rentrerons pas ici dans le débat) ni la gestion proposée des formulaires.
C'est pourquoi nous conserverons totalement intacte la couche modèle avec sa classe PdoGsbpour l'accès aux données.
Nous utiliserons par contre le moteur de *templatetwig*pour les vues ; ce n'était pas non plus obligatoire, nous pouvions utiliser le PHP.

Installation de Symfony2

Télécharger, décompresser, copier les fichiers dans un répertoire créé à la racine de *www* : *Symfony*
Ajouter un *path* vers *php.exe* pour avoir accès à la console de commande (écrite en PHP).

Création de l'application : un *bundle*

Le *bundle* correspond à l'application à développer, sa création va générer une arborescence pour le développement et pour la mise en production ; c'est dans le *bundle* que nous écrirons tout le code. Une simple commande permet de générer tout cela ; il s'agit de la commande *console* située dans le répertoire *app*.

Lancer la console :

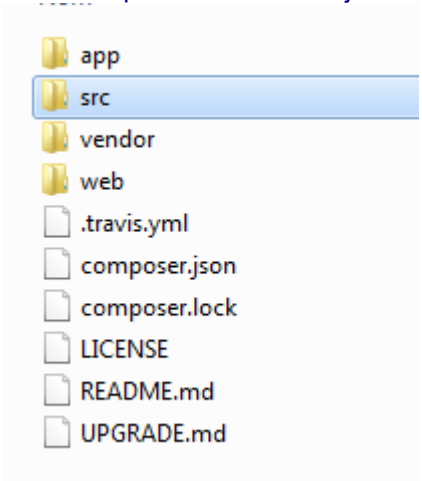
```
C:\wamp\www\Symfony\app>php console generate:bundle


  Welcome to the Symfony2 bundle generator


Your application code must be written in bundles. This command helps
you generate them easily.

Each bundle is hosted under a namespace (like Acme/Bundle/BlogBundle).
The namespace should begin with a "vendor" name like your company name, your
project name, or your client name, followed by one or more optional category
sub-namespaces, and it should end with the bundle name itself
(which must have Bundle as a suffix).

See http://symfony.com/doc/current/cookbook/bundles/best_practices.html#index-1
for more
details on bundle naming conventions.

Use / instead of \  for the namespace delimiter to avoid any problem.

Bundle namespace: Pg\GsbFraisBundle

In your code, a bundle is often referenced by its name. It can be the
concatenation of all namespace parts but it's really up to you to come
up with a unique name (a good practice is to start with the vendor name).
Based on the namespace, we suggest PgGsbFraisBundle.

Bundle name [PgGsbFraisBundle]: _
```

Appuyer sur*Entrée* pour conserver le *name* proposé.

Laisser le répertoire proposé (appuyer sur*Entrée*) source du code, indiquer que le format des fichiers de configuration sera **yml**, confirmer ensuite toutes les valeurs par défaut.

```
Bundle name [PgGsbFraisBundle]:

The bundle can be generated anywhere. The suggested default directory uses
the standard conventions.

Target directory [C:/wamp/www/Symfony/src]:

Determine the format to use for the generated configuration.

Configuration format (yml, xml, php, or annotation) [annotation]: yml

To help you get started faster, the command can generate some
code snippets for you.

Do you want to generate the whole directory structure [no]?


  Summary before generation


You are going to generate a "Pg\GsbFraisBundle\PgGsbFraisBundle" bundle
in "C:/wamp/www/Symfony/src/" using the "yml" format.

Do you confirm generation [yes]?


  Bundle generation


Generating the bundle code: OK
Checking that the bundle is autoloaded: OK
Confirm automatic update of your Kernel [yes]?
Enabling the bundle inside the Kernel: OK
Confirm automatic update of the Routing [yes]?
Importing the bundle routing resource: OK


  You can now start using the generated code!
```

Dans le répertoire *src* a été ajouté le répertoire *Pg* contenant l'application *GsbFraisBundle* :



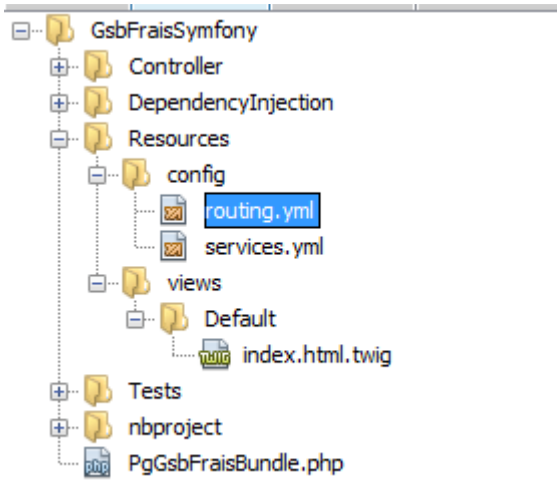
Un petit essai

Nous allons construire la page *home* qui va présenter plus tardla connexion et le menu de l'application. Commençons par faire un premier test.

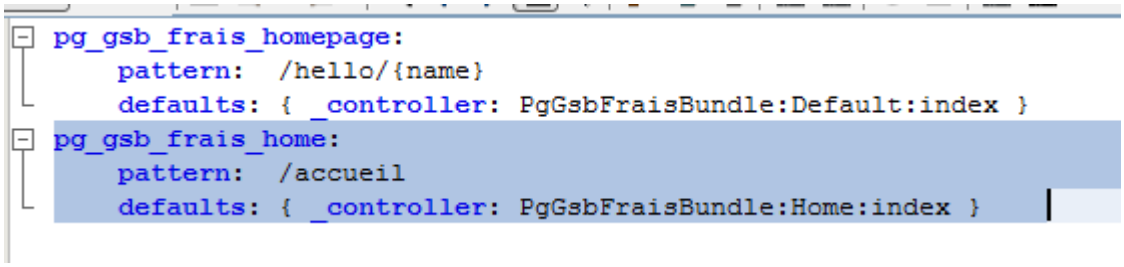
Création des routes

C'est le point d'entrée des pages ; une route a un nom logique et va établir un lien entre une URL et du code qui va s'exécuter à l'appel du lien. Ce code est nécessairement une méthode d'une classe *Controller*.

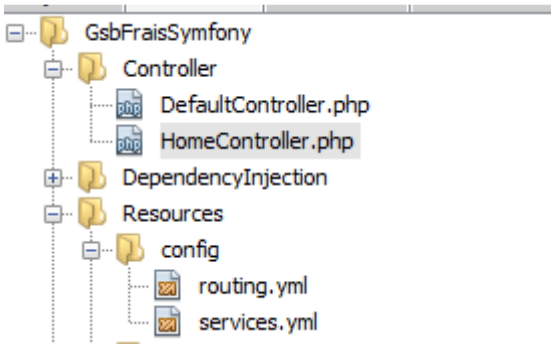
Allons dans le fichier de routes de l'application :



Ajoutons une nouvelle route (conservons celle présente par défaut provisoirement pour vérifier le format) :



Cette route s'appelle *pg_gsb_frais_home*, elle indique que toute URL (relative) de format */accueil* exécutera le code de la méthode *index* du Contrôleur *Home*.
Créons donc le contrôleur *HomeController* (le suffixe *Controller* est requis) :



Remarques :

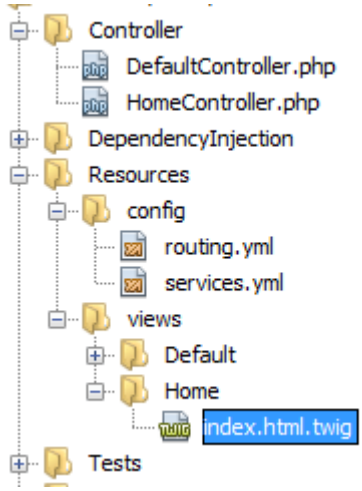
- Vous pouvez copier et transformer le code présent dans *DefaultController*.
- **Attention**, *GsbFraisSymfony* est le nom du projet (sous NetBeans), le répertoire physique est le nom du *bundle* : *GsgFraisBundle*.

```php
<?php
namespace Pg\GsbFraisBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class HomeController extends Controller
{
    public function indexAction()
    {
        return $this->render('PgGsbFraisBundle:Home:index.html.twig');
    }
}

?>
```

Remarques :

- La méthode **doit obligatoirement** se nommer *indexAction* puisque la route annonce une méthode nommée *index* (le suffixe *Action* est requis).
- La dernière ligne (*return*…) est très courante, elle demande à Symphony2 (S2 par la suite) de *présenter* le fichier annoncé : ce fichier se trouve dans le répertoire *Home* des *views* et se nomme *index.html.twig*. La *double* extension est obligatoire pour nommer le fichier *twig*.
- Il est préférable d'organiser son code en créant un répertoire du même nom pour les vues associées à un contrôleur, mais ce n'est pas obligatoire.

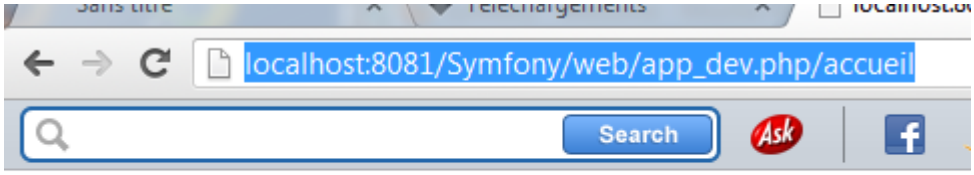
Créons la page annoncée (*index.html.twig*) dans un répertoire *Home* créé dans la partie *views*.



Le code est dans un premier temps au minimum :

```html
<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    </head>
    <body>
        <div>Bonjour ok</div>
    </body>
</html>
```

Normalement tout doit bien se passer :

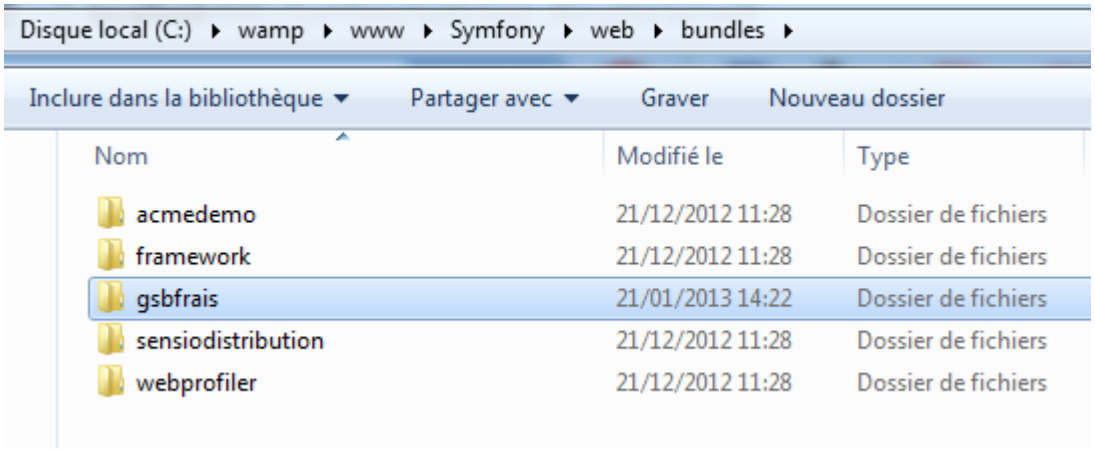


Bonjour ok

Nous voici prêts à démarrer…

Page d'accueil du site GSB

Nous allons commencer par intégrer des éléments de l'application *gsbMVC*.
Dans un premier temps, concentrons-nous sur l'unique page d'accueil. Copions les deux seuls ressources publiques du site GSB, l'image et le fichier CSS. Toutes les autres ressources, y compris le code,sont privées dans Symfony. Nous allons copier les deux répertoires de *gsbMVC* (*styles* et *images*) dans le répertoire *web* (public). Au préalable, créons un répertoire *gsbfrais* dans le dossier *bundles* :

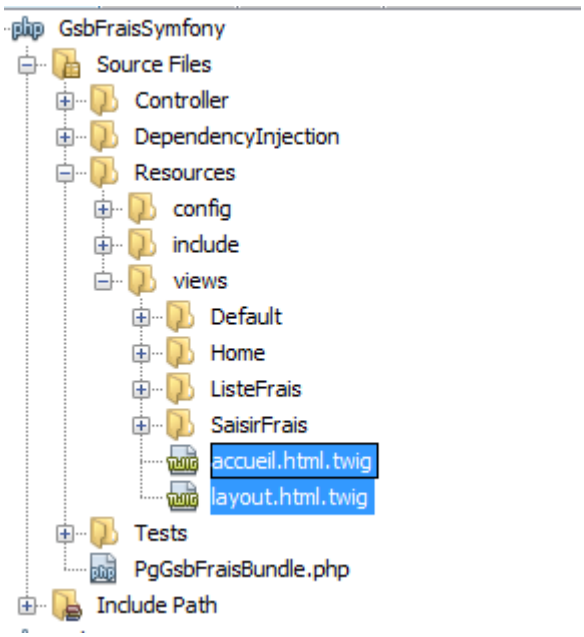


Toutes les pages comprennent le haut avec le logo et, dans certains cas, le sommaire présentant les liens disponibles.

C'est ce que nous avons traduit dans GSB_MVC par des vues distinctes (*en_tête* et *sommaire*) ; S2 (pour Symphony2) propose avec son moteur de *templates* (vues) *twig* un mécanisme d'héritage de vue ou *d'include* de vues. Ceci n'est pas propre à *twig* d'ailleurs.

Nous allons utiliser l'héritage. Une vue (*layout*) contiendra le logo, une autre (*accueil*) le logo et le sommaire.

Comme ces deux vues seront utilisées par plusieurs contrôleurs, nous avons choisi de les mettre à la racine des *views* :



Remarque : ne tenez pas compte des répertoires *ListeFrais* et *SaisirFrais* qui seront présentés plus loin (le support ayant été fait à la fin du développement... c'est souvent préférable☺).

Créons le fichier *layout.html.twig*qui va contenir ce qui se trouvait dans le fichier *v_entete.php*.
Remplaçons, dans le fichier *index.html.twig*par le code de la vue *v_entete.ph*p de l'application GSB_MVC :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
  <head>
    <title>Intranet du Laboratoire Galaxy-Swiss Bourdin</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <link href="{{ asset('bundles/gsbfrais/styles/styles.css') }}" rel="s
    <link rel="shortcut icon" type="image/x-icon" href="./images/favicon.
  </head>
  <body>
    <div id="page">
      <div id="entete">
        <img src="{{ asset('bundles/gsbfrais/images/logo.jpg')}}" id="log
        <h1>Suivi du remboursement des frais</h1>
      </div>
     </div>
          {% block menu %}
          {% endblock %}
          {% block body %}
                {% block bloc1 %}
                {% endblock %}
          {% endblock %}
    </body>
  </html>
```

Seules les lignes 7 et 13 sont modifiées : elles utilisent le langage *twig* et sa fonction *asset* qui permet de ne pas définir des URI"en dur".
Par contre, nous prévoyons des blocs qui seront surchargés dans les vues qui vont hériter du *layout*.

Remarques :

- La double accolade est interprétée par *twig* comme « met le contenu », c'est l'équivalent du < ?phpecho $quelquechose ?>
- Les balises {% %} sont interprétées comme « fait ce qui est défini », ici c'est de définir simplement des blocs.

Créons la page *accueil.html.twig*, copions-y la vue *v_sommaire.php* ; ceci doit donner à peu près cela :

```
1  {% extends "PgGsbFraisBundle::layout.html.twig" %}
2  {% block menu %}
3  <div id="menuGauche">
4       <div id="infosUtil">
5           <h3>
6              {%if(app.session.get('nom') is defined )%}
7                              Visiteur: bonjour {{app.session.get('nom')}}<br>
8              {%endif%}
9           </h3>
10      </div>
11         <ul id="menuList">
12            <li class="smenu">
13               <a href="index.php?uc=gererFrais&action=saisirFrais" title="Saisie fiche de frais ">S
14            </li>
15            <li class="smenu">
16               <a href="index.php?uc=etatFrais&action=selectionnerMois" title="Consultation de mes f
17            </li>
18            <li class="smenu">
19               <a href="index.php?uc=connexion&action=deconnexion" title="Se déconnecter">Déconnexio
20            </li>
21            <li class="smenu">
22               <a href="index.php?uc=validerFrais&action=voirFrais" title="valider les frais">Valide
23            </li>
24          </ul>
25      </div>
26 {% endblock%}
27
```

Remarques :

- Le bloc *extends* indique l'héritage.
- Notez la syntaxe du nommage du fichier père (*layout*), les deux points (::) indique **l'absence** du répertoire puisque le fichier est à la racine des *views*.
- Les lignes 6 à 8, assez simples à interpréter, rendent le même service que dans la vue *v_sommaire.php* le code *$_SESSION[nom'],* etc.
- Dans un templatetwig, nous accédons à la variable de session à partir de la classe *app*.
- Les URL pointées dans le menu seront modifiées plus tard.

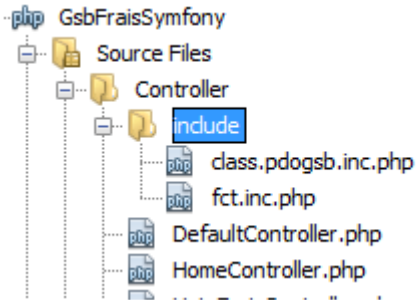
Testons en pointant sur l'accueil dans le contrôleur déjà utilisé :

```
8
9   class HomeController extends Controller
10  {
11      public function indexAction()
12      {
13          return $this->render('PgGsbFraisBundle::accueil.html.twig');
14
```

Nous voyons apparaître le logo et le sommaire (enfin j'espère…).

La connexion

Commençons par copier le répertoire *include* de l'application GSB_MVC dans le répertoire *Controller*.



Dans l'application GSB_MVC, une connexion est demandée dès l'arrivée sur le site. Nous allons traiter ce cas d'utilisation en reprenant une grande partie de la vue *v_connexion.php* et du *case* associé à la validation de la connexion.

Mais commençons à ajouter une nouvelle route :

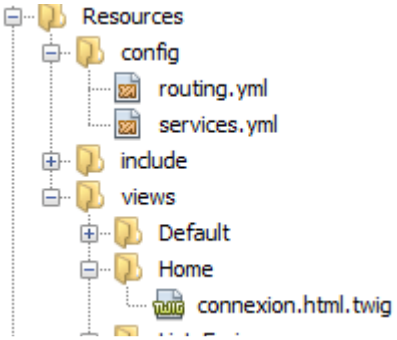
```
pg_gsb_frais_homepage:
    pattern:  /hello/{name}
    defaults: { _controller: PgGsbFraisBundle:Default:index }
pg_gsb_frais_home:
    pattern:  /accueil
    defaults: { _controller: PgGsbFraisBundle:Home:index }
pg_gsb_frais_validerconnexion:
    pattern:  /validerconnexion
    defaults: { _controller: PgGsbFraisBundle:Home:validerconnexion }
```

Remarque :

- La première route, créée par S2, peut être supprimée.

Il faut, comme annoncé, ajouter une méthode *validerconnexionAction* dans le contrôleur *Home* ; laissons vide cette méthode pour l'instant.

Passons au formulaire de connexion, il reprendra le code de la vue *v_connexion.php* :



```
{% extends "PgGsbFraisBundle::layout.html.twig" %}
{% block body %}
 {% block bloc1 %}
 <div id="contenu">
      <h2>Identification utilisateur</h2>
      {%if(message is defined )%}
            <div class ="erreur">
            <ul>{{message}}</ul></div>
      {%endif%}
<form method="POST" action="{{path('pg_gsb_frais_validerconnexion')}}">
        <p>
          <label for="nom">Login*</label>
           <input id="login" type="text" name="login"  size="30" maxlength="45">
        </p>
        <p>
            <label for="mdp">Mot de passe*</label>
            <input id="mdp"  type="password"  name="mdp" size="30" maxlength="45">
        </p>
         <input type="submit" value="Valider" name="valider">
         <input type="reset" value="Annuler" name="annuler">
      </p>
</form>
</div>
    {% endblock %}
    {% endblock%}
```

Remarques :

- La connexion hérite du seul *layout* (pas du menu).
- Les blocs *menu* et *bloc1* sont surchargés.
- Les lignes 6 à 9 seront expliquées plus loin.
- Notez la syntaxe du nommage de l'attribut action du formulaire : la fonction *path* s'occupe des répertoires (c'est bien pratique) et on se concentre sur l'essentiel : quelle route prendre ? C'est-à-dire quelle URL et quel code exécuter, c'est justement indiqué dans la nouvelle route construite.

Regardons maintenant le code de la méthode du contrôleur :

```
    public function validerconnexionAction(){
            $session= $this->get('request')->getSession();
            $request = $this->get('request');
            $login =  $request->request->get('login');
            $mdp = $request->request->get('mdp');
            $pdo = PdoGsb::getPdoGsb();
            $visiteur = $pdo->getInfosVisiteur($login,$mdp);
            if(!is_array($visiteur)){
                return $this->render('PgGsbFraisBundle:Home:connexion.html.twig',array(
                   'message'=>'Erreur de login ou de mot de passe '));
            }
            else {
                $session->set('id',$visiteur['id']);
                $session->set('nom',$visiteur['nom']);
                $session->set('prenom',$visiteur['prenom']);
                return $this->render('PgGsbFraisBundle::accueil.html.twig');
            }
    }
}
```

Remarques :

- La ligne 29 récupère la session courante ; à noter que le contrôleur (*this*) y a directement accès. La gestion des sessions n'est pas très documentée sur le site officiel.
- La ligne 31 permet de récupérer une variable *postée* ; il faudrait remplacer *request* par *query* pour récupérer une variable « *getée* ».
- La ligne 36 est très intéressante : ceci permet de transmettre à la vue, grâce au second argument de *render*, un tableau de données construit dans le contrôleur. Le tableau ici

contient une unique clé/valeur, mais il pourrait contenir de nombreuses données, ce sera le cas plus tard. Nous comprenons maintenant le bloc lignes 6-9 qui affiche, dans le *templatetwig* présenté juste au-dessus, le message d'erreur éventuel construit dans le contrôleur. La variable *message*, clé de l'élément du tableau est directement accessible dans la vue.

- Les lignes 40 et suivantes enregistrent en session les données du visiteur.
- La classe contrôleur doit faire appel à différentes classes :

```php
<?php
namespace Pg\GsbFraisBundle\Controller;
require_once("include/fct.inc.php");
require_once ("include/class.pdogsb.inc.php");
use Symfony\Component\HttpFoundation\Session\Session;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use PdoGsb;

class HomeController extends Controller
{
```

Transformons un peu la méthode *indexAction* afin de ne pas redemander une connexion à un visiteur déjà connecté :

```php
public function indexAction()
{
   $session= $this->get('request')->getSession();
   if(estConnecte($session)){
          return $this->render('PgGsbFraisBundle:accueil.html.twig');
        }
    else
          return $this->render('PgGsbFraisBundle:Home:connexion.html.twig');
}
```

Remarque :

- La fonction *estConnecte* était présente dans GSB_MVC, sa signature est ici un peu transformée ; la collaboration de la programmation orientée objet (POO) de S2 avec du procédural pose quelques problèmes et n'est bien sûr pas documentée.

La déconnexion

Dans l'application GSB_MVC, le visiteur peut se déconnecter à partir du menu. Nous allons commencer par ajouter une nouvelle route :

```
5  pg_gsb_frais_home:
6      pattern:  /accueil
7      defaults: { _controller: PgGsbFraisBundle:Home:index }
8  pg_gsb_frais_validerconnexion:
9      pattern:  /validerconnexion
10     defaults: { _controller: PgGsbFraisBundle:Home:validerconnexion }
11 pg_gsb_frais_deconnexion:
12     pattern:  /deconnexion
13     defaults: { _controller: PgGsbFraisBundle:Home:deconnexion }
```

Ensuite ajouter la méthode *deconnexionAction* dans le contrôleur :

```
38      public function deconnexionAction(){
39          $session= $this->get('request')->getSession();
40          $session->clear();
41          return $this->render('PgGsbFraisBundle:Home:connexion.html.twig');
42
43      }
```

Passons maintenant au lien entre le menu et la méthode ; nous allons indiquer que ***le href est la route***, dans le fichier accueil qui contient les options du menu :

```
18          <li class="smenu">
19            <a href="{{path('pg_gsb_frais_deconnexion')}}" title="Se déconnecter">Déconnexion</a>
20          </li>
21          <li class="smenu">
22             <a href="index.php?uc=validerFrais&action=voirFrais" title="valider les frais">Valider les frais</a>
23          </li>
24        </ul>
25    </div>
```

Et c'est tout !!

Liste des frais

L'application GSB_MVC proposait une présentation des frais à partir d'une sélection des mois ; le cas d'utilisation associé est le suivant :

Scénario nominal :

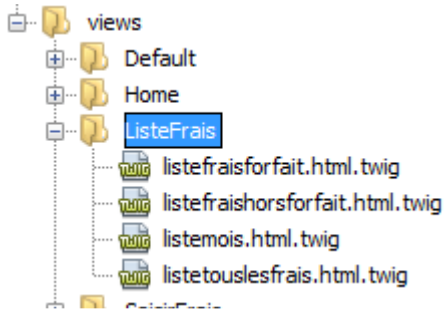
1. L'utilisateur demande à consulter ses frais.
2. Le système invite à sélectionner un mois donné.
3. L'utilisateur sélectionne un mois donné, puis valide.
4. Le système affiche l'état de la fiche de frais avec la date associée, les éléments forfaitisés – quantité pour chaque type de frais forfaitisé - et non forfaitisés – montant, libellé et date d'engagement - existant pour la fiche de frais du mois demandé.

Nous voyons les deux *réactions* du système. Dans le cas d'utilisation précédant (la connexion) nous avons créé autant de routes que de sollicitations du système. Ceci peut parfois entrainer de la redondance de code (gestion multiples des sessions, d'accès au modèle). Nous allons utiliser cette fois une seule route dont le code se distinguerapar la méthode HTTP utilisée, GET ou POST.

Commençons donc à définir notre nouvelle route :

```
pg_gsb_frais_homepage:
    pattern:  /hello/{name}
    defaults: { _controller: PgGsbFraisBundle:Default:index }
pg_gsb_frais_home:
    pattern:  /accueil
    defaults: { _controller: PgGsbFraisBundle:Home:index }
pg_gsb_frais_validerconnexion:
    pattern:  /validerconnexion
    defaults: { _controller: PgGsbFraisBundle:Home:validerconnexion }
pg_gsb_frais_deconnexion:
    pattern:  /deconnexion
    defaults: { _controller: PgGsbFraisBundle:Home:deconnexion }
pg_gsb_frais_listefrais:
    pattern:  /listefrais
    defaults: { _controller: PgGsbFraisBundle:ListeFrais:index }
```

Nous allons ensuite définir les *views*, quatre seront utilisées, leurs noms sont évocateurs de leurs rôles :



La dernière *listetouslesfrais* va *ramasser* les trois premières.

La liste des mois est assez simple :

```
{% extends "PgGsbFraisBundle::accueil.html.twig" %}
{% block body %}
{% block bloc1 %}
<div id="contenu">
      <h2>Mes fiches de frais</h2>
      <h3>Mois à sélectionner : </h3>
      <form  method="post">
      <div class="corpsForm">
      <p>
        <label for="lstMois" accesskey="n">Mois : </label>
        <select id="lstMois" name="lstMois">
            {% for unMois in  lesmois %}
                {% set mois = unMois.mois %}
                {% set numAnnee =  unMois.numAnnee %}
                {% set numMois =  unMois.numMois%}
                {% if mois == lemois %}
                    <option selected value="{{ mois }}">{{ numMois }}/{{ numAnnee }} </option>
                {% else %}
                    <option value="{{ mois }}">{{ numMois }}/{{ numAnnee }} </option>
                {% endif %}
            {% endfor %}
       </select>
      </p>
      </div>
      <div class="piedForm">
      <p>
            <input id="ok" type="submit" value="Valider" size="20" />
            <input id="annuler" type="reset" value="Effacer" size="20" />
      </p>
      </div>
        </form>
{% endblock %}
{% endblock %}
```

Cette vue hérite de l'accueil (logo+menu), le bloc1 est surchargé.

Remarques :

- Le contrôleur doit être capable de fournir la liste des mois dans la variable *lesmois,* ligne 12, nous allons voir comment plus loin.
- Les lignes 13, 14 et 15 ne sont pas obligatoires ; nous pourrions directement afficher les champs de *unMois* dans les *options*.
- Notez la syntaxe pointée, pour objet.

La partie du contrôleur associée à cette vue est :

```
8   class ListeFraisController extends Controller
9   {
10      public function indexAction()
11      {
12          $session= $this->container->get('request')->getSession();
13          $idVisiteur =  $session->get('id');
14          $pdo = PdoGsb::getPdoGsb();
15          $lesMois=$pdo->getLesMoisDisponibles($idVisiteur);
16          if($this->get('request')->getMethod() == 'GET'){
17                  // Afin de sélectionner par défaut le dernier mois dans la zone de liste
18                  // on demande toutes les clés, et on prend la première,
19                  // les mois étant triés décroissants
20              $lesCles = array_keys( $lesMois );
21              $moisASelectionner = $lesCles[0];
22              return $this->render('PgGsbFraisBundle:ListeFrais:listemois.html.twig',
23                  array('lesmois'=>$lesMois,'lemois'=>$moisASelectionner));
24          }
25          else{
```

Remarques :

- La ligne 11 va filtrer la nature de la requête HTTP.
- Le *render* prend comme second argument un tableau qui peut contenir de nombreuses paires de clé/valeur.

La liste des frais au forfait ne pose pas de problèmes particuliers :

```
<h3>Fiche de frais du mois {{nummois}}-{{numannee}} :
    </h3>
    <div class="encadre">
    <p>
        Etat : {{libetat}} depuis le {{datemodif}} <br> Montant validé : {{montantvalide}}
    </p>
        <table class="listeLegere">
           <caption>Eléments forfaitisés </caption>
        <tr>
    {% for unfrais in  lesfraisforfait %}
          {%   set libelle = unfrais.libelle %}
                <th> {{libelle}}</th>
    {% endfor %}
        </tr>
        <tr>
    {% for unfrais in  lesfraisforfait %}
          {%   set quantite = unfrais.quantite %}
               <td class="qteForfait">{{quantite}} </td>
    {% endfor %}
        </tr>
    </table>
    </div>
```

La liste des frais hors forfait est aussi relativement simple :

```
<table class="listeLegere">
            <caption>Descriptif des éléments hors forfait -{{nbjustificatifs}} justificatifs reçus -
        </caption>
              <tr>
                 <th class="date">Date</th>
                 <th class="libelle">Libellé</th>
                 <th class='montant'>Montant</th>
              </tr>
           {% for unFrais in  lesfraishorsforfait %}
                 {% set date = unFrais.date%}
                 {% set libelle = unFrais.libelle%}
                 {% set montant = unFrais.montant%}
                 <tr>
                     <td>{{date}}</td>
                     <td>{{libelle}}</td>
                     <td>{{montant}}</td>
                 </tr>
             {% endfor %}
    </table>
```

La dernière vue, *listetouslesfrais* sera montrée plus loin ; voyons d'abord la partie du contrôleur qui gère le POST du formulaire :

```
        else{
            $request = $this->get('request');
            $leMois =  $request->request->get('lstMois');
            $lesFraisHorsForfait = $pdo->getLesFraisHorsForfait($idVisiteur,$leMois);
            $lesFraisForfait= $pdo->getLesFraisForfait($idVisiteur,$leMois);
            $lesInfosFicheFrais = $pdo->getLesInfosFicheFrais($idVisiteur,$leMois);
            $numAnnee =substr( $leMois,0,4);
            $numMois =substr( $leMois,4,2);
            $libEtat = $lesInfosFicheFrais['libEtat'];
            $montantValide = $lesInfosFicheFrais['montantValide'];
            $nbJustificatifs = $lesInfosFicheFrais['nbJustificatifs'];
            $dateModif =  $lesInfosFicheFrais['dateModif'];
            $dateModif =  dateAnglaisVersFrancais($dateModif);
            return $this->render('PgGsbFraisBundle:ListeFrais:listetouslesfrais.html.twig',
                array('lesmois'=>$lesMois,'lesfraisforfait'=>$lesFraisForfait,'lesfraishorsforfait'=>$lesFraisHorsForfait,
                    'lemois'=>$leMois,'numannee'=>$numAnnee,'nummois'=> $numMois,'libetat'=>$libEtat,
                        'montantvalide'=>$montantValide,'nbjustificatifs'=>$nbJustificatifs,'datemodif'=>$dateModif));

        }
```

Rien de bien nouveau, c'est à quelques détails le code de l'application GSB_MVC.

La méthode *render* appelle la vue *listetouslesfrais*, regardons cette vue qui rassemble les trois vues précédentes :

```
{% include 'PgGsbFraisBundle:ListeFrais:listemois.html.twig' with {'lesmois':lesmois,'lemois':lemois} %}

{% include 'PgGsbFraisBundle:ListeFrais:listefraisforfait.html.twig' with {'lesfraisforfait':lesfraisforfait,'numannee':numannee,
'nummois':nummois,'libetat':libetat,'montantvalide':montantvalide,'datemodif':datemodif} %}

{% include 'PgGsbFraisBundle:ListeFrais:listefraishorsforfait.html.twig' with {'lesfraishorsforfait':lesfraishorsforfait,
'nbjustificatifs':nbjustificatifs} %}
```

Lorsque nous utilisons la relation *include* entre *template*, nous pouvons passer des arguments à la vue appelée grâce à la clause *with* ; ainsi la variable *lesmois* pourra être disponible dans la vue *listemois*.

Traçons le parcours de cette donnée :

Etape 1 : valorisation dans le contrôleur

```
$lesMois=$pdo->getLesMoisDisponibles($idVisiteur);
```

Etape 2 : passage à la vue *listetouslesfrais* par **render**, toujours dans le constructeur

```
return $this->render('PgGsbFraisBundle:ListeFrais:listetouslesfrais.html.twig',
    array('lesmois'=>$lesMois,'lesfraisforfait'=>$lesFraisForfait,'lesfraishorsforfait'=>$lesFraisHorsForfait,
        'lemois'=>$leMois,'numannee'=>$numAnnee,'nummois'=> $numMois,'libetat'=>$libEtat,
            'montantvalide'=>$montantValide,'nbjustificatifs'=>$nbJustificatifs,'datemodif'=>$dateModif));
```

Etape 3 : la vue *listetouslesfrais* appelle la vue *listemois* en lui passant la liste de mois.

```
{% include 'PgGsbFraisBundle:ListeFrais:listemois.html.twig' with {'lesmois':lesmois,'lemois':lemois} %}
```

Cette liste est maintenant disponible dans *listemois*.

La saisie des frais

C'est le cas d'utilisation le plus délicat de l'application.

Deux routes sont ajoutées :

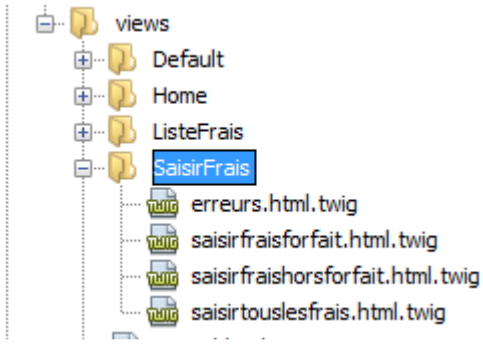
```
pg_gsb_frais_home:
    pattern:  /accueil
    defaults: { _controller: PgGsbFraisBundle:Home:index }
pg_gsb_frais_validerconnexion:
    pattern:  /validerconnexion
    defaults: { _controller: PgGsbFraisBundle:Home:validerconnexion }
pg_gsb_frais_deconnexion:
    pattern:  /deconnexion
    defaults: { _controller: PgGsbFraisBundle:Home:deconnexion }
pg_gsb_frais_listefrais:
    pattern:  /listefrais
    defaults: { _controller: PgGsbFraisBundle:ListeFrais:index }
pg_gsb_frais_saisirfrais:
    pattern:  /saisirfrais
    defaults: { _controller: PgGsbFraisBundle:SaisirFrais:index }
pg_gsb_frais_validerfraishorsforfais:
    pattern:  /validerfraishorsforfait
    defaults: { _controller: PgGsbFraisBundle:SaisirFrais:validerfraishorsforfait }
```

Une troisième sera ajoutée un peu plus loin.

Notre nouveau contrôleur doit au moins contenir les méthodes annoncées dans les routes :

```php
<?php
namespace Pg\GsbFraisBundle\Controller;
require_once("include/fct.inc.php");
require_once ("include/class.pdogsb.inc.php");
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Session\Session;
use PdoGsb;
class SaisirFraisController extends Controller
{
    public function indexAction()
    {...}
     public function validerfraishorsforfaitAction(){...}

```

Comme pour le cas d'utilisation précédent, nous définirons plusieurs vues :



Une vue d'erreurs est nécessaire car des saisies sont demandées.

Dans le contrôleur, pour la première soumission et la validation des frais au forfait, le filtre se fera sur la requête HTTP (comme dans le cas précédent) :

```php
use PdoGsb;
class SaisirFraisController extends Controller
{
    public function indexAction()
    {
        $session= $this->get('request')->getSession();
        $idVisiteur =  $session->get('id');
        $mois = getMois(date("d/m/Y"));
        $numAnnee =substr( $mois,0,4);
        $numMois =substr( $mois,4,2);
        $pdo = PdoGsb::getPdoGsb();
        if($pdo->estPremierFraisMois($idVisiteur,$mois)){
                            $pdo->creeNouvellesLignesFrais($idVisiteur,$mois);
        }
        $request = $this->get('request');
        $lesErreursForfaits = array();
        if($this->get('request')->getMethod() == 'POST'){
                $lesFrais = $request->request->get('lesFrais');
                if(lesQteFraisValides($lesFrais)){
                     $pdo->majFraisForfait($idVisiteur,$mois,$lesFrais);
                }
                else{
                     $lesErreursForfaits[]="Les valeurs des frais doivent être numériques";
                }
         }
        $lesFraisForfait = $pdo->getLesFraisForfait($idVisiteur,$mois);
        $lesFraisHorsForfait = $pdo->getLesFraisHorsForfait($idVisiteur,$mois);
        return $this->render('PgGsbFraisBundle:SaisirFrais:saisirtouslesfrais.html.twig',
                array('lesfraisforfait'=>$lesFraisForfait,'lesfraishorsforfait'=>$lesFraisHorsForfait,'nummois'=>$numMois,
                    'numannee'=>$numAnnee,'leserreursforfait'=> $lesErreursForfaits,'leserreurshorsforfait'=>null));
    }
    public function validerfraishorsforfaitAction(){
```

Remarques :

- Une grande partie du code peut être récupérée à partir de GSB_MVC.
- La gestion des erreurs est un peu différente, pas d'appel de fonction mais gestion d'un tableau directement dans le code. Un tableau est utilisé ici car l'unique vue d'erreurs attend un tableau.
- On *passe* les erreurs hors forfait alors que ceci concerne les frais hors forfait ; ceci permet de d'utiliser la même vue *conteneursaisirtouslesfrais* qui abesoin des erreurs des frais hors forfait.

La vue des frais au forfait est la suivante :

```
<div id="contenu">
      <h2>Renseigner ma fiche de frais du mois {{nummois}}-{{numannee}}</h2>
      <form action="{{path('pg_gsb_frais_saisirfrais')}}" method="POST">
          <div class="corpsForm">
              <fieldset>
                <legend>Eléments forfaitisés</legend>
                     {% for unfrais in  lesfraisforfait %}
                         {% set idfrais = unfrais.idfrais %}
                         {% set libelle = unfrais.libelle %}
                         {% set quantite = unfrais.quantite %}
                        <p>
                            <label for="idFrais">{{libelle}}</label>
                            <input type="text" id="idFrais" name="lesFrais[{{idfrais}}]" size="10" maxlength="5" value="{{quantite}}" >
                        </p>
                     {%endfor%}
              </fieldset>
          </div>
          <div class="piedForm">
          <p>
            <input id="ok" type="submit" value="Valider" size="20" />
            <input id="annuler" type="reset" value="Effacer" size="20" />
          </p>
          </div>
      </form>
```

Pour la validation des frais hors forfait, le code suit la même logique :

```
    }
    public function validerfraishorsforfaitAction(){
                $session= $this->get('request')->getSession();
                $idVisiteur =  $session->get('id');
                $mois = getMois(date("d/m/Y"));
                $numAnnee =substr( $mois,0,4);
                $numMois =substr( $mois,4,2);
                $pdo = PdoGsb::getPdoGsb();
                $request = $this->get('request');
                $dateFrais = $request->request->get('dateFrais');
                $libelle = $request->request->get('libelle');
                $montant = $request->request->get('montant');
                $lesErreursHorsForfait = valideInfosFrais($dateFrais,$libelle,$montant);
                if (count($lesErreursHorsForfait)==0){
                        $pdo->creeNouveauFraisHorsForfait($idVisiteur,$mois,$libelle,$dateFrais,$montant);
                }
                $lesFraisForfait= $pdo->getLesFraisForfait($idVisiteur,$mois);
                $lesFraisHorsForfait = $pdo->getLesFraisHorsForfait($idVisiteur,$mois);
                return $this->render('PgGsbFraisBundle:SaisirFrais:saisirtouslesfrais.html.twig',
                array('lesfraisforfait'=>$lesFraisForfait,'lesfraishorsforfait'=>$lesFraisHorsForfait,'nummois'=>$numMois,
                    'numannee'=>$numAnnee,'leserreursforfait'=>null,'leserreurshorsforfait'=> $lesErreursHorsForfait));

    }
```

Les remarques sont les mêmes que précédemment.

La vue des frais hors forfait est la suivante (extrait ici) :

```
div
<div id="contenu">
<table class="listeLegere">
           <caption>Descriptif des éléments hors forfait
       </caption>
            <tr>
               <th class="date">Date</th>
                                <th class="libelle">Libellé</th>
               <th class="montant">Montant</th>
               <th class="action"> </th>
            </tr>
        {% for unfrais in  lesfraishorsforfait %}
            {% set libelle = unfrais.libelle %}
            {% set date = unfrais.date %}
            {% set montant = unfrais.montant %}
            {% set idfrais = unfrais.id %}
            <tr>
               <td>{{date}}</td>
               <td>{{libelle}}</td>
               <td>{{montant}}</td>
               <td><a href="{{path('pg_gsb_frais_supprimerfraishorsforfait',{'id' : idfrais})}}"
                                onclick="return confirm('Voulez-vous vraiment supprimer ce frais?');">Supprimer ce fr
            </tr>
        {%endfor%}

    </table>
      <form action="{{path('pg_gsb_frais_validerfraishorsforfais')}}" method="post">
      <div class="corpsForm">

          <fieldset>
            <legend>Nouvel élément hors forfait
            </legend>
            <p>
              <label for="txtDateHF">Date (jj/mm/aaaa): </label>
              <input type="text" id="txtDateHF" name="dateFrais" size="10" maxlength="10" value=""  />
            </p>
            <p>
```

Remarques :

- Les lignes 12 à 15 ne sont présentes que pour la lisibilité des lignes suivantes.
- La ligne 20 sera commentée plus loin.
- Ce code reprend le code initial de GSB_MVC en grande partie.

La vue des erreurs est particulièrement simple :

```
<div class ="erreur">
<ul>
 {% for erreur in  leserreurs %}

      <li>{{erreur}}</li>

{%endfor%}
</ul></div>
```

La vue qui rassemble les trois vues est la suivante :

```
{% extends "PgGsbFraisBundle::accueil.html.twig" %}
{% block body %}
{% block bloc1 %}
 {% include 'PgGsbFraisBundle:SaisirFrais:erreurs.html.twig' with {'leserreurs':leserreursforfait} %}
{% include 'PgGsbFraisBundle:SaisirFrais:saisirfraisforfait.html.twig' with {'lesfraisforfait':lesfraisforfait,
'nummois':nummois,'numannee':numannee} %}

{% include 'PgGsbFraisBundle:SaisirFrais:erreurs.html.twig' with {'leserreurs':leserreurshorsforfait} %}
{% include 'PgGsbFraisBundle:SaisirFrais:saisirfraishorsforfait.html.twig' with {'lesfraishorsforfait':lesfraishorsforfait} %}
{% endblock %}
{% endblock %}

```

Remarques :

- La vue hérite de l'accueil et surcharge le bloc1.
- C'est bien cette vue qui est appelée dans le contrôleur.
- La vue *erreur* est appelée deux fois avec des *arguments* différents.

Suppression d'un frais

Nous allons avoir l'occasion de voir la notion de route paramétrée.

Regardons la nouvelle route créée :

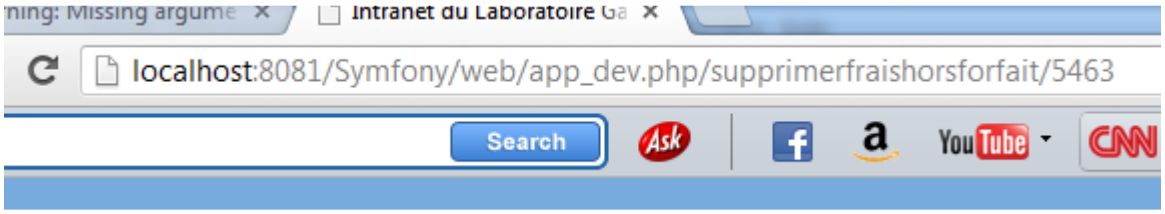
```
18      pattern:  /saisirfrais
19      defaults: { _controller: PgGsbFraisBundle:SaisirFrais:index }
20  pg_gsb_frais_validerfraishorsforfais:
21      pattern:  /validerfraishorsforfait
22      defaults: { _controller: PgGsbFraisBundle:SaisirFrais:validerfraishorsforfait }
23  pg_gsb_frais_supprimerfraishorsforfait:
24      pattern:  /supprimerfraishorsforfait/{id}
25      defaults: { _controller: PgGsbFraisBundle:SaisirFrais:supprimerfraishorsforfait }
26
```

Le modèle d'URL fait référence à un paramètre *id* qui apparaîtra derrière un slash. Ce paramètre prend la valeur construite dans la vue *saisirfraishorsforfait* présentée plus haut à partir de l'identifiant du frais à supprimer :

```
11          {% for unfrais in  lesfraishorsforfait %}
12              {% set libelle = unfrais.libelle %}
13              {% set date = unfrais.date %}
14              {% set montant = unfrais.montant %}
15              {% set idfrais = unfrais.id %}
16              <tr>
17                 <td>{{date}}</td>
18                 <td>{{libelle}}</td>
19                 <td>{{montant}}</td>
20                 <td><a href="{{path('pg_gsb_frais_supprimerfraishorsforfait',{'id' : idfrais})}}"
21                              onclick="return confirm('Voulez-vous vraiment supprimer ce frais?');"
22              </tr>
23          {%endfor%}
```

La ligne 20 valorise *id* à partir du parcours du tableau des frais.

A l'exécution l'URL prendra la forme suivante :



Poursuivons par l'écriture de la méthode *supprimerfraishorsforfait* annoncée dans la route :

```
61
62      public function supprimerfraishorsforfaitAction($id){
63                  $session= $this->get('request')->getSession();
64                  $idVisiteur =  $session->get('id');
65                  $mois = getMois(date("d/m/Y"));
66                  $numAnnee =substr( $mois,0,4);
67                  $numMois =substr( $mois,4,2);
68                  $pdo = PdoGsb::getPdoGsb();
69                  $pdo->supprimerFraisHorsForfait($id);
70                  $lesFraisForfait= $pdo->getLesFraisForfait($idVisiteur,$mois);
71                  $lesFraisHorsForfait = $pdo->getLesFraisHorsForfait($idVisiteur,$mois);
72                  return $this->render('PgGsbFraisBundle:SaisirFrais:saisirtouslesfrais.html.twig',
73                  array('lesfraisforfait'=>$lesFraisForfait,'lesfraishorsforfait'=>$lesFraisHorsForfait,'nummois'=>$numMois,
74                      'numannee'=>$numAnnee,'leserreursforfait'=>null,'leserreurshorsforfait'=>null));
75      }
76
77  }
```

La méthode prend comme argument le paramètre de l'URL. Le mécanisme de paramétrage de route permet de récupérer de manière élégante une variable que nous aurions passée dans l'URL.

Il nous reste un petit travail à faire : vérifier que le numéro de frais passé est bien celui d'un frais du visiteur pour le mois courant. En effet, rien n'interdit de construire une URL avec n'importe quel numéro de frais ! Nous avions négligé cet élément dans l'application GSB initiale ☺

Il va nous falloir intervenir pour une fois dans le modèle.

Ajoutons une nouvelle requête :

```
public function estValideSuppressionFrais($idVisiteur,$mois,$idFrais){
                $req = "select count(*) as nb from lignefraishorsforfait
                where lignefraishorsforfait.id=:idfrais and lignefraishorsforfait.mois=:mois
                and lignefraishorsforfait.idvisiteur=:idvisiteur";
                $stmt = PdoGsb::$monPdo->prepare($req);
                $stmt->bindParam(':idfrais', $idFrais);
                $stmt->bindParam(':mois', $mois);
                $stmt->bindParam(':idvisiteur', $idVisiteur);
                $stmt->execute();
                $ligne = $stmt->fetch();
                return $ligne['nb'];

}
```

Modifions un peu le code du contrôleur :

```
62
63      public function supprimerfraishorsforfaitAction($id){
64                  $session= $this->get('request')->getSession();
65                  $idVisiteur =  $session->get('id');
66                  $mois = getMois(date("d/m/Y"));
67                  $numAnnee =substr( $mois,0,4);
68                  $numMois =substr( $mois,4,2);
69                  $pdo = PdoGsb::getPdoGsb();
70                  if( $pdo->estValideSuppressionFrais($idVisiteur,$mois,$id))
71                              $pdo->supprimerFraisHorsForfait($id);
72                  else {
73                       $response = new Response;
74                       $response->setContent("<h2>Page introuvable erreur 404 ");
75                       $response->setStatusCode(404);
76                       return $response;
77                  }
78                  $lesFraisForfait= $pdo->getLesFraisForfait($idVisiteur,$mois);
79                  $lesFraisHorsForfait = $pdo->getLesFraisHorsForfait($idVisiteur,$mois);
80                  return $this->render('PgGsbFraisBundle:SaisirFrais:saisirtouslesfrais.html
81                  array('lesfraisforfait'=>$lesFraisForfait,'lesfraishorsforfait'=>$lesFrais
82                      'numannee'=>$numAnnee,'leserreursforfait'=>null,'leserreurshorsforfait
83      }
84
85  }
```

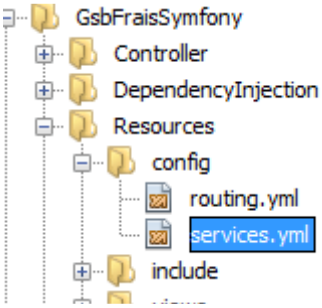
Remarques :

- La ligne 73 utilise un objet *Response* qui va générer une page 404.
- Nous avons jusqu'ici retourné des pages *twig* grâce à la méthode *render* ; cette méthode est un raccourci pour la création d'un objet *Response* et son exploitation par le moteur de *templatetwig*.

Pour aller un peu plus loin : notion de service

L'application fonctionne ; nous avons conservé la couche modèle (excepté l'ajout signalé au-dessus) en l'état. Néanmoins, l'appel dans les contrôleurs à la classe d'accès aux données n'est pas tout à fait satisfaisant : il y a une trop forte dépendance entre *PdoGsb* et le code. Il est possible de supprimer cette dépendance en utilisant un ***service*** et non pas directement une classe ; ainsi nous pourrons changer de *service* sans modifier le code.
Un service est une classe et peut être directement utilisé dans les contrôleurs. Nous avons déjà utilisé des services, par exemple en écrivant *$request = $this->get('request')* nous faisons appel au service de nom *request* et ceci dans le contrôleur.

Pour créer un service, il faut définir ses attributs dans le fichier *services.yml*



Ajoutons ce nouveau service dans le fichier *services.yml* :

```
1   parameters:
2   #    pg_gsb_frais.example.class: Pg\GsbFraisBundle\Example
3
4   services:
5   #    pg_gsb_frais.example:
6   #        class: %pg_gsb_frais.example.class%
7   #        arguments: [@service_id, "plain_value", %parameter%]
8       pg_gsb_frais.pdo:
9           class: Pg\GsbFraisBundle\services\PdoGsb
10          arguments: ["mysql:host=localhost", "dbname=bd_gsb", "root",""]
11
```

Remarques :

- S2 avait mis en commentaire un format de création de service (ligne 5-7).
- Le service a un nom qui permettra d'y accéder, ici *pg_gsb_frais.pdo.*
- La classe doit préciser le chemin d'accès (ligne 9).
- Les arguments sont ceux du constructeur.

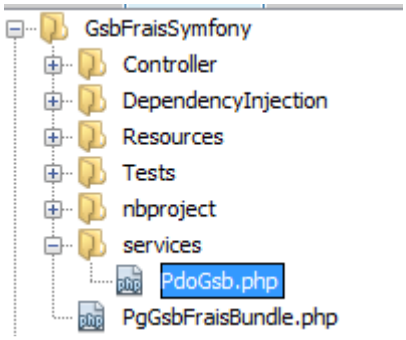
Ainsi au lieu d'écrire *$pdo = PdoGsb::getPdoGsb();*
Nous allons faire appel au service :
$pdo = $this->get('pg_gsb_frais.pdo');

Nous *injectons* par l'intermédiaire d'un service la classe d'accès aux données. Ce mécanisme permet de découpler le lien contrôleur/persistance. Nous pourrions ainsi présenter un service différent sans intervenir sur le code du contrôleur.

Nous allons profiter des propriétés des services pour ne pas laisser en *dur* dans la classe (comme c'était le cas dans GSB_MVC) les arguments de la connexion.

Par contre, notre classe *PdoGsb* doit être un peu modifiée car l'appel au service se traduit par l'appel du constructeur de la classe *PdoGsb* et cette classe utilisait à l'origine un constructeur *statique* (singleton oblige).

Commençons par ajouter un répertoire de services à la base de notre *bundle*, copions le fichier *class.pdogsb.inc.php* dans ce répertoire **et renommons ce fichier du nom de la classe** :



Reproduisons le code modifié de la classe :

```php
<?php
namespace Pg\GsbFraisBundle\services;
use PDO;
class PdoGsb{
        private static $monPdo;
        private static $monPdoGsb=null;
/**
 * Constructeur public, crée l'instance de PDO qui sera sollicitée
 * pour toutes les méthodes de la classe
 */

        public function __construct($serveur,$bdd,$user,$mdp){
            PdoGsb::$monPdo = new PDO($serveur.';'.$bdd, $user, $mdp);
            PdoGsb::$monPdo->query("SET CHARACTER SET utf8");

        }
```

Remarques :

- Le *namespace* est requis avec le chemin physique.
- La clause *use* permet d'utiliser la classe PDO.
- Les champs statiques initiaux ont disparu.
- Le constructeur est devenu public et prend comme arguments ceux qui sont annoncés dans la définition du service (cf plus haut).

Dans les contrôleurs, toute référence à la classe *PdoGsb* disparait :

```php
<?php
namespace Pg\GsbFraisBundle\Controller;
require_once("include/fct.inc.php");
//require_once ("include/class.pdogsb.inc.php");
use Symfony\Component\HttpFoundation\Session\Session;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
//use PdoGsb;

```

L'appel au service est celui présenté plus haut :

```
//$pdo = PdoGsb::getPdoGsb();
$pdo = $this->get('pg_gsb_frais.pdo');
$visiteur = $pdo->getInfosVisiteur($login,$mdp);
```

Dans la définition des services, l'item *parameters* peut être utilisé pour définir les paramètres des services. Voici une nouvelle version de la définition du service :

```
parameters:
    db.host: mysql:host=localhost
    db.name: dbname=bd_gsb
    db.user: root
    db.mdp:
services:
    pg_gsb_frais.pdo:
        class: Pg\GsbFraisBundle\services\PdoGsb
        arguments: [%db.host%,%db.name%,%db.user%,%db.mdp%]
```

En guise de conclusion

Le passage du *modèle pédagogiqueGSB_MVC* au modèle plus professionnel se fait assez naturellement ; nous avons réutilisé la partie modèle (en très grande partie) ainsi que la plupart des vues existantes (en intégrant le langage *twig* et ses relations –héritage, inclusion-). L'effort a porté davantage sur la gestion des contrôleurs et les routes associées. Nous n'avons pas exploité une fonctionnalité de S2, la gestion des formulaires qui peut être automatisée à partir d'un modèle de données (fourni sous forme de classe ou pas).