# 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 développement NetBeans |
| **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 *Site du 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 PdoGsb pour l’accès aux données.

Nous utiliserons par contre le moteur de *template twig* pour les vues ; ce n’était pas non plus obligatoire, nous pouvions utiliser le PHP.

## Installation de Symfony2

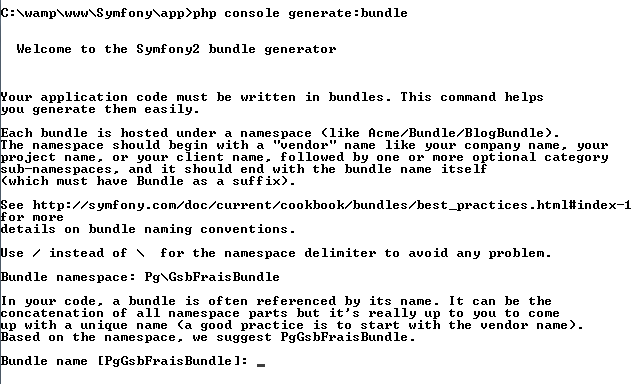
Télécharger, décompresser, copier les fichiers dans un répertoire créé à la racine de *www* : *Symfony*

Ajouter un *path* vers *php.exe* pour avoir accès à la console de commande (écrite en PHP).

## Création de l’application : un *bundle*

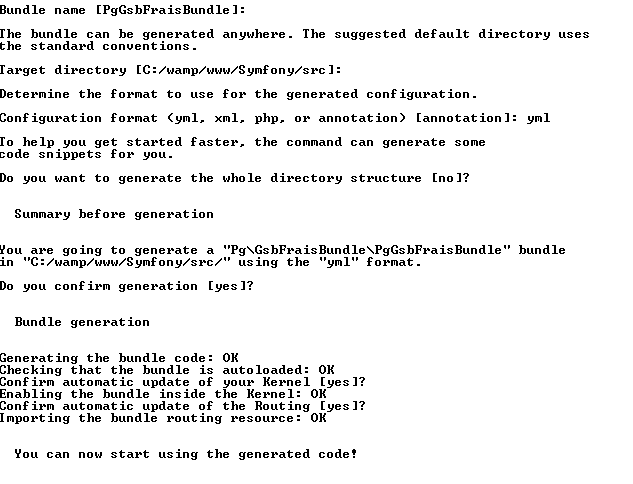
Le *bundle* correspond à l’application à développer, sa création va générer une arborescence pour le développement et pour la mise en production ; c’est dans le *bundle* que nous écrirons tout le code. Une simple commande permet de générer tout cela ; il s’agit de la commande *console* située dans le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

Lancer la console :

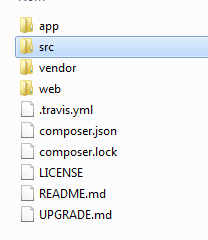


Appuyer sur *Entrée* pour conserver le *name* proposé.

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.



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



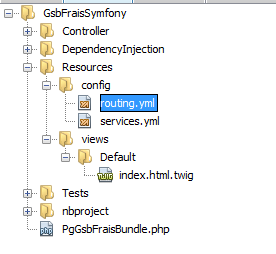
## Un petit essai

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

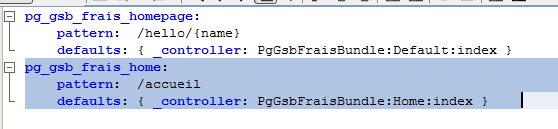
### Création des routes

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 :

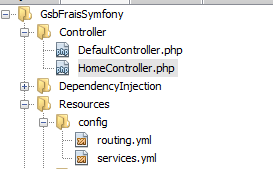


Ajoutons une nouvelle route (conservons celle présente par défaut provisoirement pour vérifier le format) :



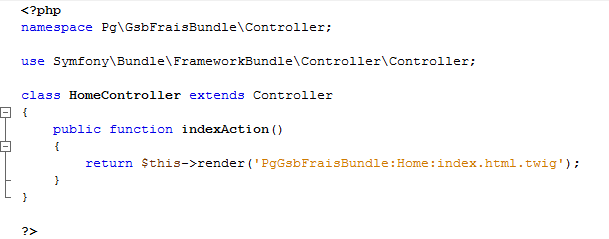
Cette route s’appelle *pg\_gsb\_frais\_home*, elle indique que toute URL (relative) de format */accueil* exécutera le code de la méthode *index* du Contrôleur *Home*.

Créons donc le contrôleur *HomeController* (le suffixe *Controller* est requis) :



Remarques :

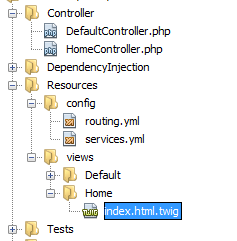
* Vous pouvez copier et transformer le code présent dans *DefaultController*.
* **Attention**, *GsbFraisSymfony* est le nom du projet (sous NetBeans), le répertoire physique est le nom du *bundle* : *GsgFraisBundle*.



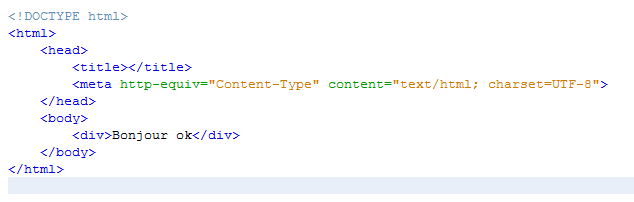
Remarques :

* 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.

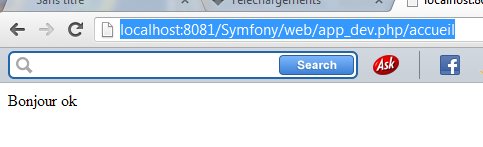
Créons la page annoncée (*index.html.twig*) dans un répertoire *Home* créé dans la partie *views*.



Le code est dans un premier temps au minimum :



Normalement tout doit bien se passer :

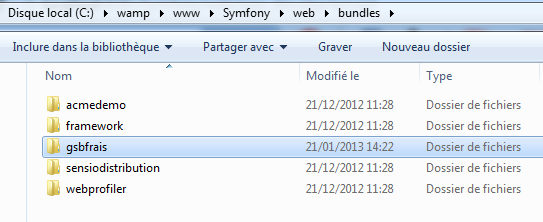


Nous voici prêts à démarrer…

## 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* :



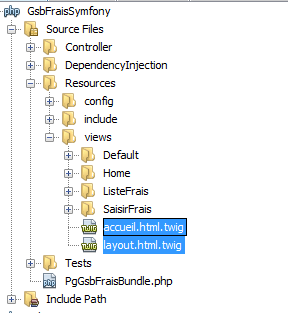
Toutes les pages comprennent le haut avec le logo et, dans certains cas, le sommaire présentant les liens disponibles.



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.

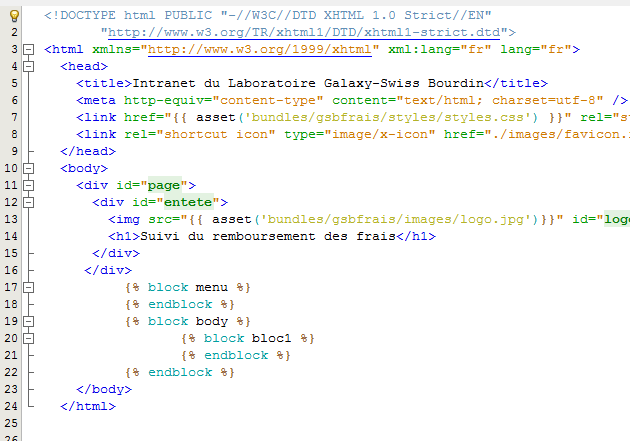
Comme ces deux vues seront utilisées par plusieurs contrôleurs, nous avons choisi de les mettre à la racine des *views* :



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 :



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 < ?php echo $quelquechose ?>
* Les balises {% %} sont interprétées comme « fait ce qui est défini », ici c’est de définir simplement des blocs.

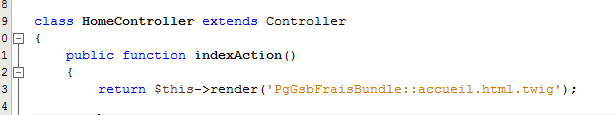
Créons la page *accueil.html.twig*, copions-y la vue *v\_sommaire.php* ; ceci doit donner à peu près cela :



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 template twig, nous accédons à la variable de session à partir de la classe *app*.
* Les URL pointées dans le menu seront modifiées plus tard.

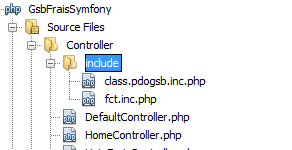
Testons en pointant sur l’accueil dans le contrôleur déjà utilisé :



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

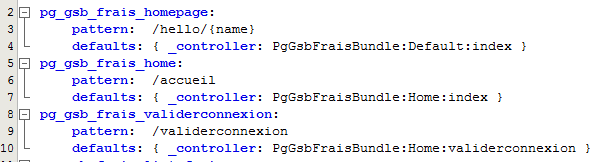
## La connexion

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 :

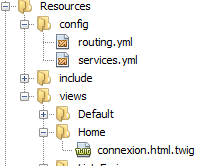


Remarque :

* 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* :

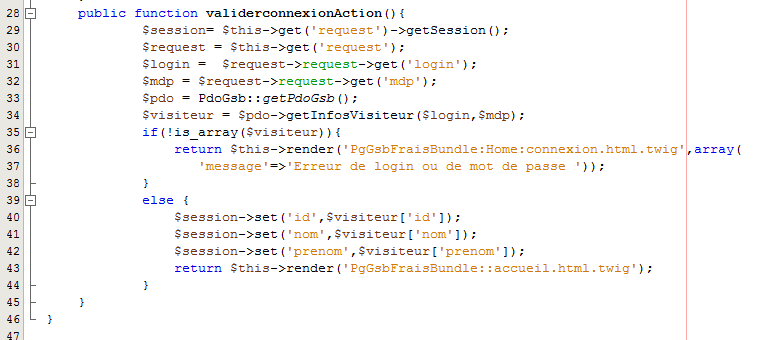




Remarques :

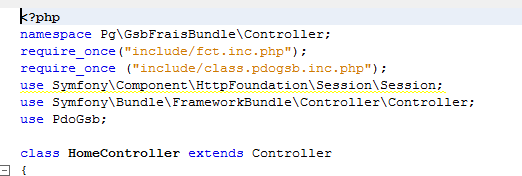
* La connexion hérite du seul *layout* (pas du menu).
* Les blocs *menu* et *bloc1* sont surchargés.
* Les lignes 6 à 9 seront expliquées plus loin.
* Notez la syntaxe du nommage de l’attribut action du formulaire : la fonction *path* s’occupe des répertoires (c’est bien pratique) et on se concentre 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 :

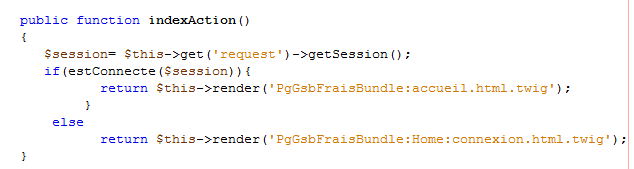


Remarques :

* 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 *template* *twig* 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 :



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

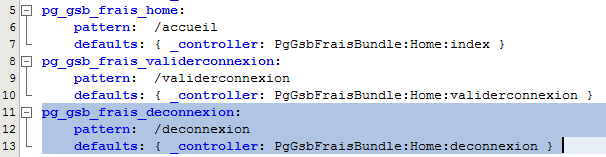


Remarque :

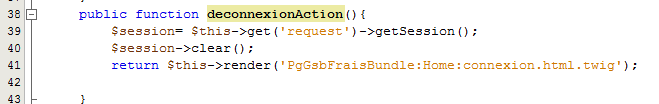
* La fonction *estConnecte* était présente dans GSB\_MVC, sa signature est ici un peu transformée ; la collaboration de la programmation orientée objet (POO) de S2 avec du procédural pose quelques problèmes et n’est bien sûr pas documentée.

## La déconnexion

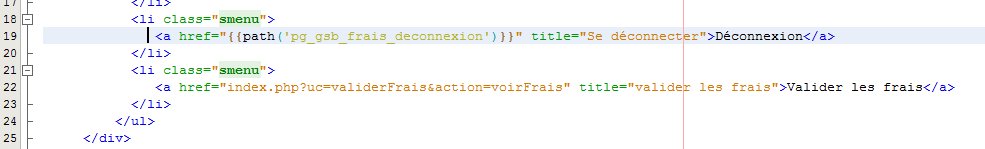
Dans l’application GSB\_MVC, le visiteur peut se déconnecter à partir du menu. Nous allons commencer par ajouter une nouvelle route :



Ensuite ajouter la méthode *deconnexionAction* dans le contrôleur :



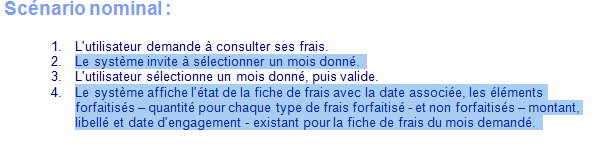
Passons maintenant au lien entre le menu et la méthode ; nous allons indiquer que ***le href est la route***, dans le fichier accueil qui contient les options du menu :



Et c’est tout !!

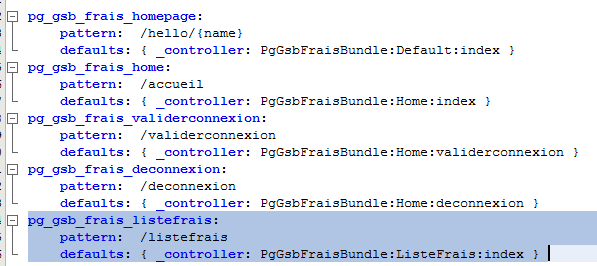
## Liste des frais

L’application GSB\_MVC proposait une présentation des frais à partir d’une sélection des mois ; le cas d’utilisation associé est le suivant :

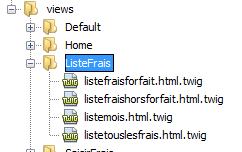


Nous voyons les deux *réactions* du système. Dans le cas d’utilisation précédant (la connexion) nous avons créé autant de routes que de sollicitations du système. Ceci peut parfois entrainer de la redondance de code (gestion multiples des sessions, d’accès au modèle). Nous allons utiliser cette fois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

Commençons donc à définir notre nouvelle route :

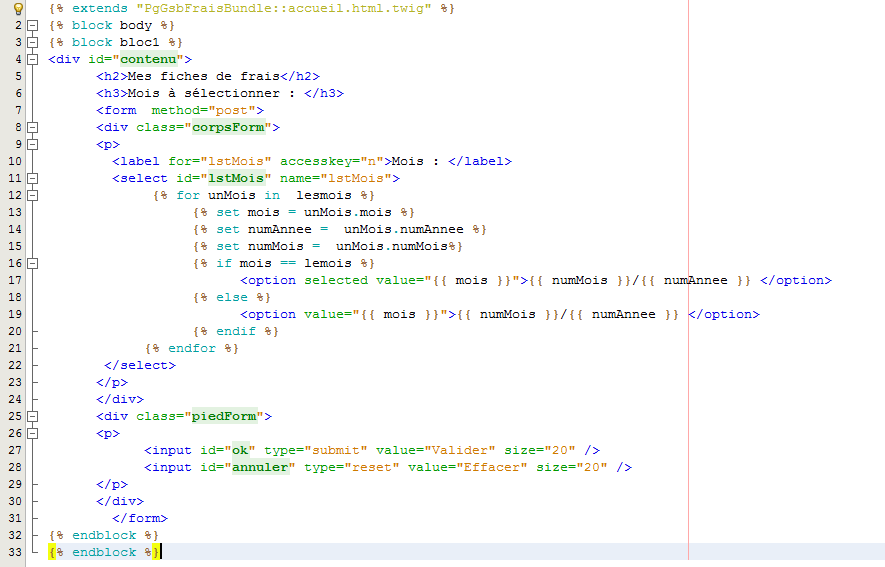


Nous allons ensuite définir les *views*, quatre seront utilisées, leurs noms sont évocateurs de leurs rôles :



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

La liste des mois est assez simple :

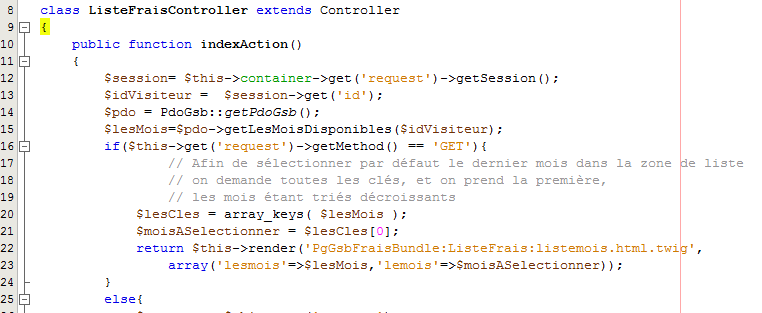


Cette vue hérite de l’accueil (logo+menu), le bloc1 est surchargé.

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.

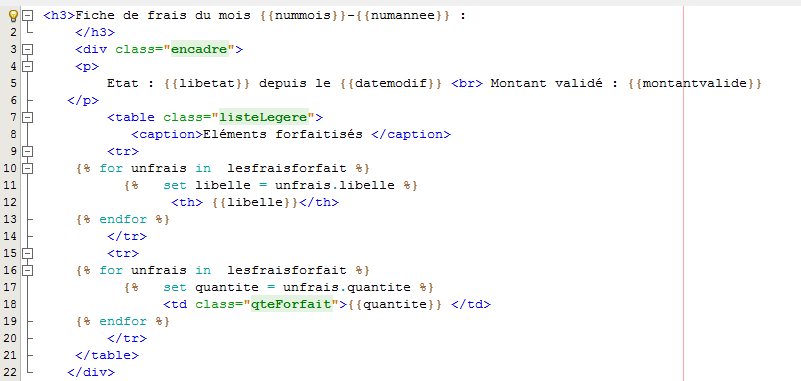
La partie du contrôleur associée à cette vue est :



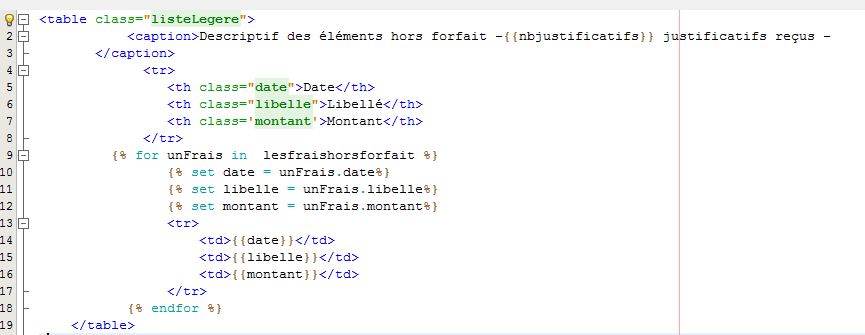
Remarques :

* 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.

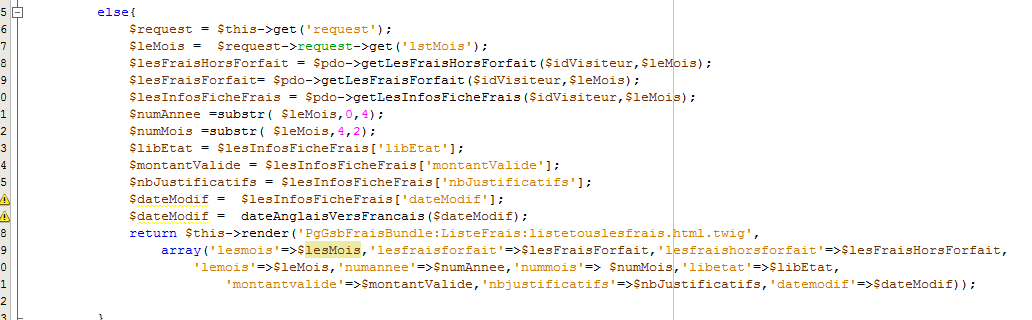
La liste des frais au forfait ne pose pas de problèmes particuliers :



La liste des frais hors forfait est aussi relativement simple :

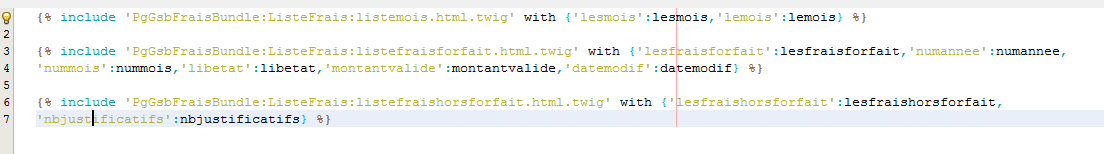


La dernière vue, *listetouslesfrais* sera montrée plus loin ; voyons d’abord la partie du contrôleur qui gère le POST du formulaire :



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 :



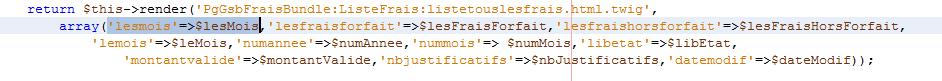
Lorsque nous utilisons la relation *include* entre *template*, nous pouvons passer des arguments à la vue appelée grâce à la clause *with* ; ainsi la variable *lesmois* pourra être disponible dans la vue *listemois*.

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



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

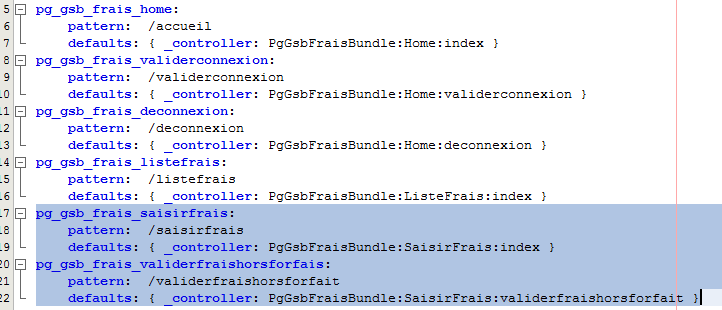


Cette liste est maintenant disponible dans *listemois*.

## La saisie des frais

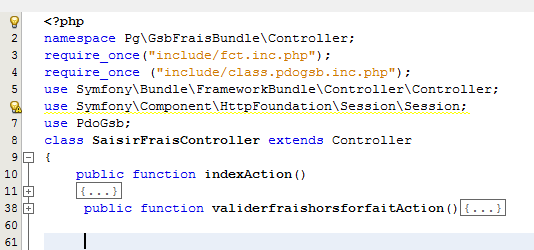
C’est le cas d’utilisation le plus délicat de l’application.

Deux routes sont ajoutées :

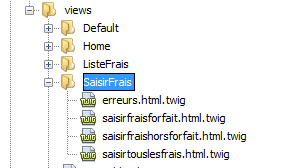


Une troisième sera ajoutée un peu plus loin.

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



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) :



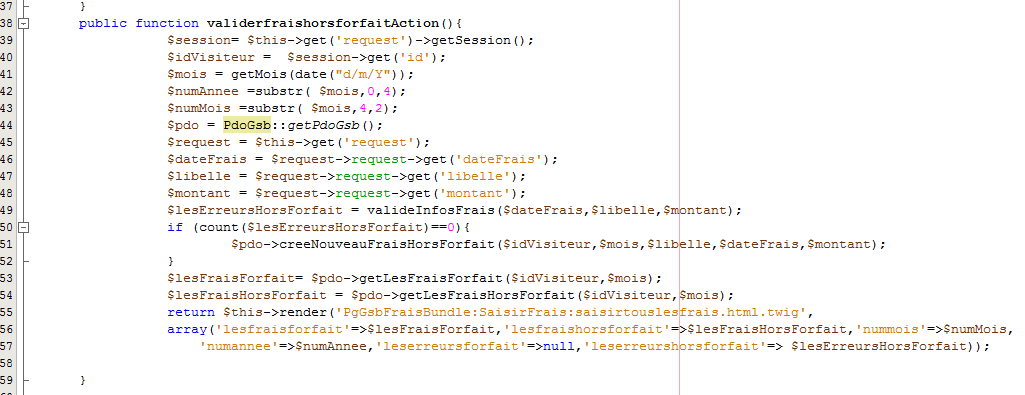
Remarques :

* 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 *conteneur* *saisirtouslesfrais* qui a besoin des erreurs des frais hors forfait.

La vue des frais au forfait est la suivante :

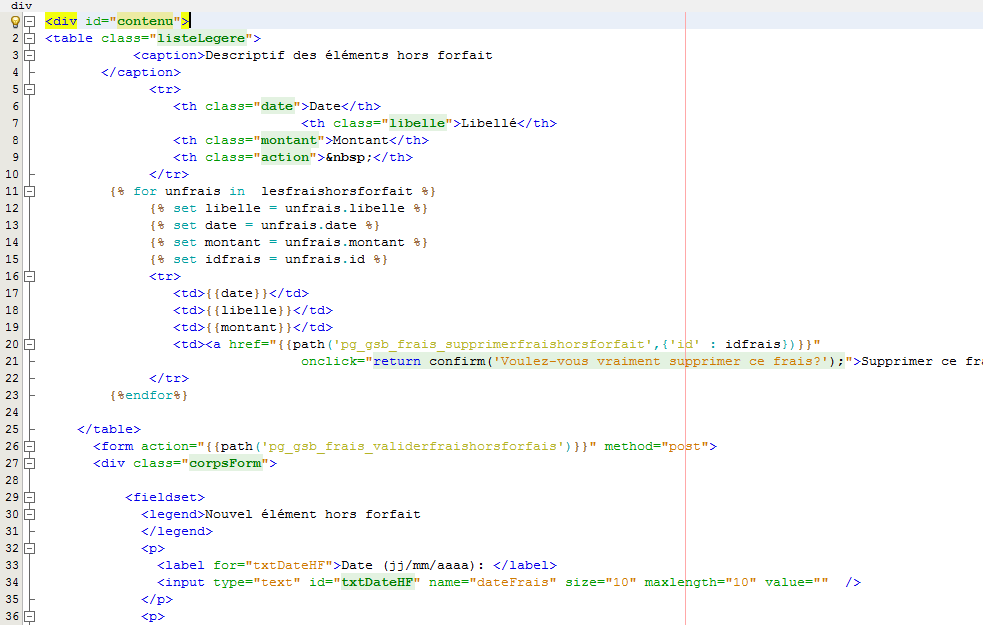


Pour la validation des frais hors forfait, le code suit la même logique :



Les remarques sont les mêmes que précédemment.

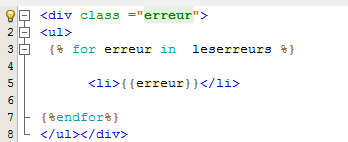
La vue des frais hors forfait est la suivante (extrait ici) :



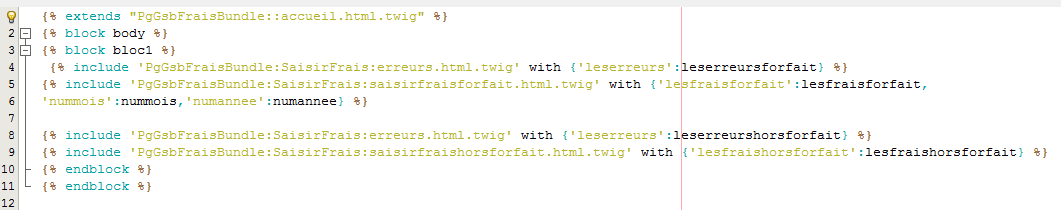
Remarques :

* 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 :



La vue qui rassemble les trois vues est la suivante :



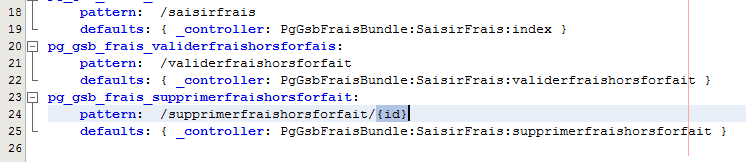
Remarques :

* 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.

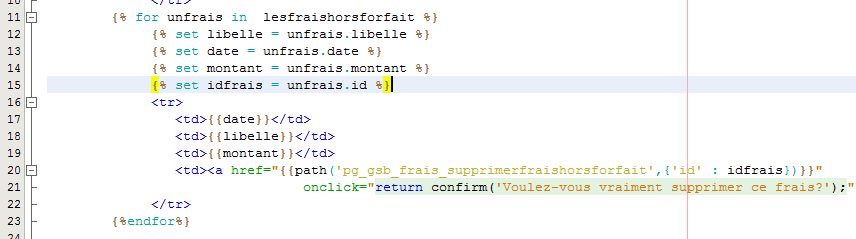
## Suppression d’un frais

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

Regardons la nouvelle route créée :

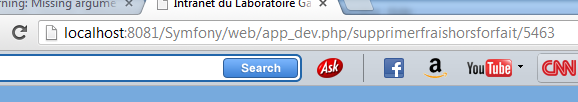


Le modèle d’URL fait référence à un paramètre *id* qui apparaîtra derrière un slash. Ce paramètre prend la valeur construite dans la vue *saisirfraishorsforfait* présentée plus haut à partir de l’identifiant du frais à supprimer :

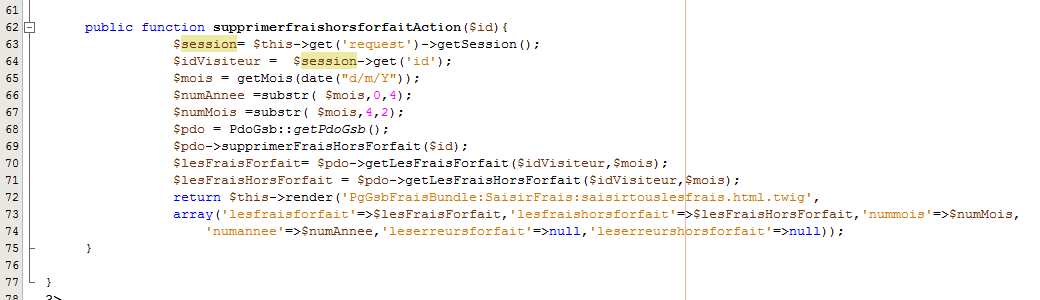


La ligne 20 valorise *id* à partir du parcours du tableau des frais.

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



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

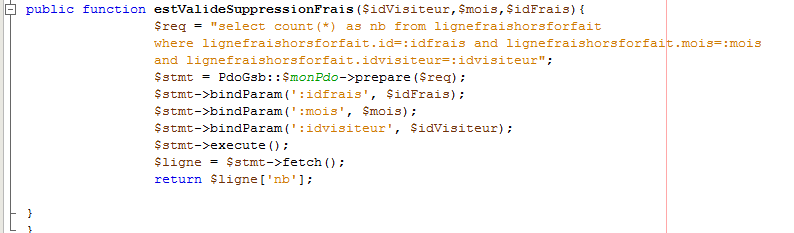


La méthode prend comme argument le paramètre de l’URL. Le mécanisme de paramétrage de route permet de récupérer de manière élégante une variable que nous aurions passée dans l’URL.

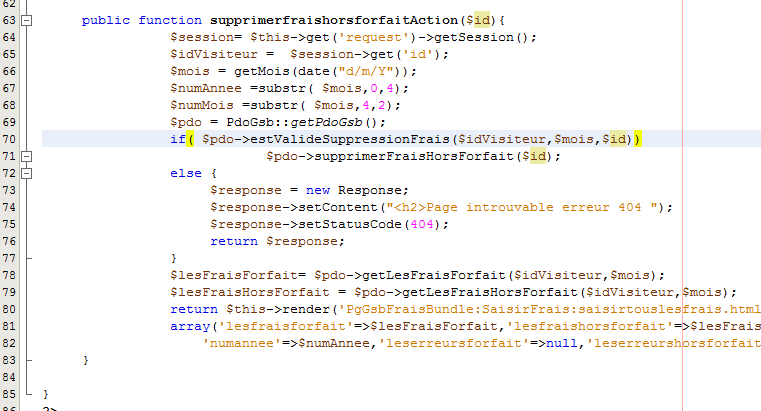
Il nous reste un petit travail à faire : vérifier que le numéro de frais passé est bien celui d’un frais du visiteur pour le mois courant. En effet, rien n’interdit de construire une URL avec n’importe quel numéro de frais ! Nous avions négligé cet élément dans l’application GSB initiale ☺

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

Ajoutons une nouvelle requête :



Modifions un peu le code du contrôleur :



Remarques :

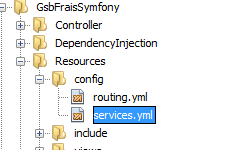
* La ligne 73 utilise un objet *Response* qui va générer une page 404.
* Nous avons jusqu’ici retourné des pages *twig* grâce à la méthode *render* ; cette méthode est un raccourci pour la création d’un objet *Response* et son exploitation par le moteur de *template* *twig*.

## Pour aller un peu plus loin : notion de service

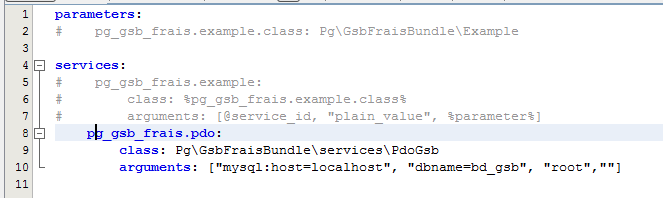
L’application fonctionne ; nous avons conservé la couche modèle (excepté l’ajout signalé au-dessus) en l’état. Néanmoins, l’appel dans les contrôleurs à la classe d’accès aux données n’est pas tout à fait satisfaisant : il y a une trop forte dépendance entre *PdoGsb* et le code. Il est possible de supprimer 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* :



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 :

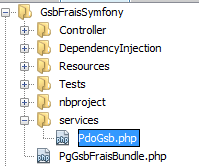
*$pdo = $this->get('pg\_gsb\_frais.pdo');*

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.

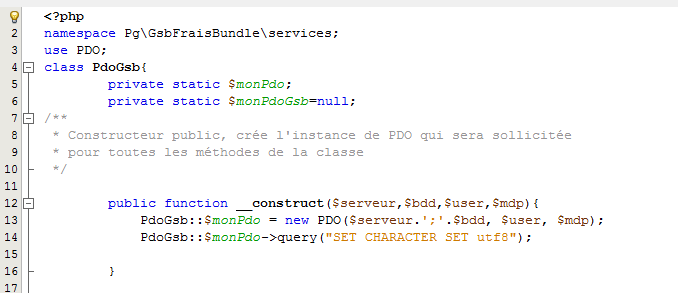
Nous allons profiter des propriétés des services pour ne pas laisser en *dur* dans la classe (comme c’était le cas dans GSB\_MVC) les arguments de la connexion.

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** :



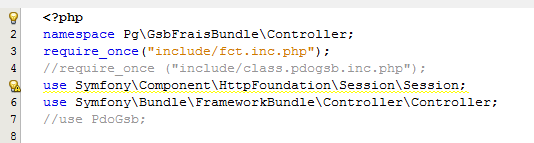
Reproduisons le code modifié de la classe :



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).

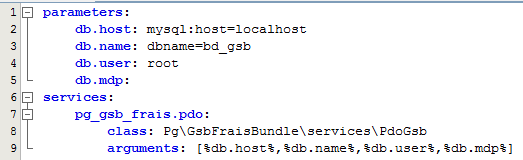
Dans les contrôleurs, toute référence à la classe *PdoGsb* disparait :



L’appel au service est celui présenté plus haut :



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 :



## En guise de conclusion

Le passage du *modèle pédagogique* *GSB\_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).