Un livre de Wikilivres.

S'initier au Zend Framework

Présentation

Zend Framework

Le cadriciel Zend est né en 2005 à l'initiative de la société Zend [1]. C'est une collection de composants modulaires comme les autres bibliothèques comme PEAR, Zend Framework y ajoute les bonnes pratiques issues de l'expérience et de la théorie de l'informatique objet (design pattern) utilisés de longue date avec Java.

Des sociétés participent directement au développement comme Adobe comme pour le composant Zend_Amf qui est un connecteur entre des applications flash et le serveur en PHP.

La société Zend affiche ouvertement l'objectif de concurrencer Java et .Net dans le domaine des serveurs d'applications et des systèmes d'information. La licence BSD du framework autorise ainsi l'utilisation dans des applications propriétaires.

Pourquoi l'utiliser

La version 5 de PHP facilite l'industrialisation des développements, ainsi des solutions ont vu le jour comme

CakePHP, Symfony, Zend Framework, etc.

Un framework ou cadriciel et parfois « cadre de travail » est un ensemble de bibliothèques et d'outils pour aider le développeur mais en plus il impose une rigueur sur la façon de programmer justifiée par les bonnes pratiques et les conventions communes. Un framework est un gage de qualité car un code cohérent sera plus aisé à maintenir ou à faire évoluer.

Pour le développeur les bibliothèques fondamentales sont fournies et comme les connecteurs à la base de données, la persistance et la validation de données, la génération de formulaires, etc.

Notes

1. http://framework.zend.com

Concepts

Le patron de conception Modèle vue contrôleur (MVC)

Zend Framework comme tous les cadriciels majeurs des langages de scripts serveurs utilise le patron de conception Modèle-vue-contrôleur qui sépare le modèle de données, l'interface utilisateur et les traitements. Ces trois parties fondamentales se nomment : le modèle, la vue et le contrôleur[1].

Le modèle ne s'occupe que du traitements des données, les liens avec la base de données, lecture, insertion, mise-à-jour des tuples dans une base, vérifier que les données sont bien formatées (validation). La présentation des résultats ne se fait pas dans le modèle.

La vue correspond à la présentation des résultats, par exemple un tableau HTML, PDF pour un utilisateur ou sous forme XML pour fournir un programme distant. Les évènements et les actions de l'utilisateur (clic d'un bouton, liste de sélection, etc) sont gérés dans la vue. Cette séparation permet au graphiste de travailler sans avoir à se soucier des rouages de l'application.

Le contrôleur prend en charge le déroulement du programme. La liste des actions sera dans le contrôleur. Il synchronise les événements provenant de l'utilisateur vers la base de données.

Ce patron de conception existe traditionnellement dans les langages objet comme Java comme par exemple la bibliothèque graphique SWING et est employé généralement dans les langages web comme la plateforme Adobe/Flex ou la prochaine norme W3C de formulaires Xforms. Cette orientation a été perçue très tôt par les développeurs web notamment avec la popularité du framework Ruby on Rails. L'éditeur de PHP, Zend a pris la même orientation tout en conservant les développeurs ne souhaitant pas migrer vers MVC et permet ainsi d'utiliser ZF comme une bibliothèque classique.

Avantages et inconvénients de l'architecture MVC.

« Given enough eyeballs, all bugs are shallow » *(pour assez de globes oculaires, tous les bugs sont superficiels)* - Linus Torvalds.

L'avantage premier est la modularité, les objets sont réutilisables dans une autre application. Le cloisonnement ajoute en facilité de maintenance. De plus, la modification des traitements se fait

indépendamment de l'affichage (la vue). Il faut préciser que les développeurs qui sont impliqués dans l'amélioration du framework proposent des composants issus de la théorie informatique (comme les design patterns) ou le fruit de l'expérience par de bonnes pratiques de programmation.

L'inconvénient est une phase de conception plus longue et ces ajouts de couches multiplient le nombre de fichiers sur le serveur. Cela nécessite des outils spécifiques de débogage et de mécanismes d'amélioration des performances (cache). Le temps d'apprentissage est à prendre en compte pour les développeurs.

Zend Framework : Fonctionnement

L'apprentissage du fonctionnement du *framework* peut être un peu ardu, car cela nécessite une connaissance de quelques concepts de programmation orientée objet. Mais la consultation de la documentation, du *tracker* de *bugs*, des blogs des développeurs permettent d'en comprendre la globalité.

Le processus d'une requête est :

1. Le *bootstrap* (html/index.php) est le point d'entrée dans l'application, il s'agit de l'implémentation du design pattern contrôleur frontal[2] et d'un *singleton*. Toutes les requêtes passent par cet objet, à sa charge d'acheminer (*dispatcher*) vers les actions (suivant une route par défaut ou définie par le concepteur). Les réponses sont collectées par cet objet.
2. La requête peut d'abord être pré-traitée dans un *plugin*, cela permet au concepteur d'effectuer des traitements en tout début de requête.
3. La requête est routée, c'est à dire que l'on traduit l'URI d'entrée en lien vers le bon contrôleur. Le contrôleur est instancié, son constructeur init() est lancé.
4. La fonction preDispatch() est lancée si elle existe juste avant de lancer l'action : par exemple readAction(...).
5. Le contrôleur se termine et la fonction postDispatch() est lancée.
6. La réponse est renvoyée au visiteur.

Notes

1. Model-View-Controller (MVC), Trygve M. H. Reenskaug - http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html
2. A controller that handles all requests for a Web site., Martin Fowler, http://www.martinfowler.com/eaaCatalog/frontController.html

Prise en main

Comment l'utiliser

Téléchargement

- http://www.zend.com/fr/products/server/downloads.

Si le site propose de créer un compte pour pouvoir télécharger, il faut le faire cela prend moins d'une minute.

Installation

L'archive du framework se présente sous deux formes :

- full : le framework accompagné de la bibliothèque JavaScript Dojo et des tests unitaires
- minimal : le framework seul

Les composants sont dans *library/Zend/*

Il suffit de copier le dossier Zend et son contenu sur le serveur à l'emplacement des bibliothèques serveur.

Il faut mettre à jour la variable include_path du php.ini

Configuration Apache

Apache doit avoir le mod_rewrite présent et activé (a2enmode rewrite) ainsi que d'autoriser les directives par .htaccess : « AllowOverride » sera à « All » dans le fichier de configuration du site.

```
<Directory /var/www/monblog/html>
            DirectoryIndex index.php
            AllowOverride All
            Order allow,deny
            Allow from all
</Directory>
```

Mini blog

Exemple du mini blog.

Une première application après le traditionnel « Hello World! » est de créer un mini blog permettant d'afficher une liste de nouvelles, créer une nouvelle, effacer et la mettre à jour.

Ce premier objet Nouvelle a ainsi 4 actions : ajouter, voir, mettre à jour, effacer. Auxquelles on rajoute l'action voir tous les objets (index).

L'ébauche du contrôleur se présente :

```
public class IndexController extends Zend_Action_Controller

public function indexAction() { // listing de tous les objets }

public function createAction() {...}

public function updateAction() {...}

public function deleteAction() {...}
```

Pour chaque action il existe une vue : un tableau pour index, un formulaire pour ajouter et mettre-à-jour, un formulaire Oui/Non pour effacer.

Le modèle est implémenté par une table SQL nouvelle avec comme attributs (id, titre, corps, date).

La vue générée est est un gabarit XHTML habillé par une feuille de style CSS.

Nouveau projet : monBlog

Dans votre IDE, créez un nouveau projet nommé 'monBlog', l'arborescence de ce projet sera par exemple :

```
monBlog/
|-- application
|   |-- configs
|   |-- controllers     <---- l'intelligence se trouve ici
|   |-- models          <---- la gestion des données, filtrage, validation, etc.
|   `-- views
|       |-- helpers
|       `-- scripts     <---- toutes les pages html se trouvent ici
|           |-- error
|           `-- index   <---- notre première page html
|-- library
|   `-- Zend -> /lien/vers/votre/ZendFramework-latest/library/Zend
|-- public
`-- tests
    |-- application
    `-- library
```

La bibliothèque fournit un script pour créer cette arborescence :

```
/lien/vers/votre/ZendFramework-latest/bin/zf.sh create project monBlog
...
```

Ce script situé dans le dossier bin/ de la bibliothèque. Sur les systèmes Unix, Il peut être appelé ultérieurement via un alias à placer à la fin de votre ~/.bash_aliases :

```
alias zf.sh='/lien/vers/votre/ZendFramework-latest/bin/zf.sh'
```

Mini blog/Modèle

Création dans la base de données

Exemple de création de base de données MySQL :

```
CREATE USER 'monblog'@'localhost' IDENTIFIED BY 'monblog';

GRANT USAGE ON *.* TO 'monblog'@'localhost' IDENTIFIED BY 'monblog';

CREATE DATABASE IF NOT EXISTS `monblog` ;

GRANT ALL PRIVILEGES ON monblog.* TO 'monblog'@'localhost';
```

Exemple de création de table :

```
CREATE TABLE nouvelle (
        id INT NOT NULL auto_increment,
```

```
        titre VARCHAR(200),
        corps TEXT,
        categorie_id INT,
        heuredate DATETIME,
        PRIMARY KEY (id)
);
```

La table étant créée il nous faut configurer l'application pour pouvoir y accéder.

Voici, un exemple de configuration pour accéder à la base, il est aussi possible de centraliser la configuration dans un fichier séparé (XML, ini, etc).

Placez les paramètres de connexion dans la fonction initDb() de l'initialiseur du bootstrap (application/Bootstrap.php) :

```
    /**
     * Initialize data bases
     *
     * @return void
     */
    public function initDb()
    {
        $params= array (        'host'     => '127.0.0.1',
                        'username' => 'monblog',
                        'password' => 'monblog',
                        'dbname'   => 'monblog');
        try {
                $db= Zend_Db::factory('PDO_MYSQL', $params);
                        Zend_Db_Table::setDefaultAdapter ( $db );
                } catch ( Exception $e ) {
                        exit ( $e->getMessage () );
                }
                Zend_Registry::set ( 'dbAdapter', $db ); // accessible partout
    }
```

Il est aussi possible de spécifier les paramètres de base de données dans le fichier application/configs /application.ini

Le modèle

ZF a choisit par défaut la correspondance objet-SQL vers Table[1](voir aussi : ORM), c'est-à-dire qu'une table SQL est un objet Zend_Db_Table, une ligne est Zend_Db_Table_Row et plusieurs lignes Zend_Db_Table_Rowset.

Pour ajouter cet objet placez-vous dans le dossier application/default/models, puis créez le fichier Nouvelle.php - n'oubliez pas la majuscule - (avec Zend Studio cela se fait par : bouton-droit New/Zend Framework Item/Zend Table) :

```
class Nouvelle extends Zend_Db_Table_Abstract {

        protected $_name = 'nouvelle';

}
```

Le framework va interroger la base pour connaître les méta-informations associées à la table et de ce fait il

va trouver 'id' dans le cas de cette table mais on peut la spécifier avec $_primary.

Le modèle créé, nous allons le charger dans son contrôleur et ensuite des actions.

Sinon, on peut créer un modèle à l'aide de Zend Tool à la racine du projet faire

```
zf create db-table Nouvelle nouvelle
```

Ce qui va créer un fichier Nouvelle.php placé dans l'arborescence application\models\DbTable contenant :

```
<?php

class Application_Model_DbTable_Nouvelle extends Zend_Db_Table_Abstract
{

    protected $_name = 'nouvelle';


}
```

Notes

1. *A single instance that handles the business logic for all rows in a database table or view.* http://martinfowler.com/eaaCatalog/tableModule.html

Mini blog/Contrôleur

Le contrôleur

Les actions par défauts seront la lecture d'une nouvelle, son ajout, sa modification, son effacement et le listing de toutes les nouvelles. Ses différentes actions sont accessibles directement par l'URL :

> http://www.monblog.zf/controlleur/action

Par défaut, c'est le routeur qui réécrit l'URI et redirige vers la bonne action.

> nouvelle/create redirigera vers la bonne page PHP de création de nouvelle. De même le passage de paramètre s'écrit par défaut de la forme :
> nouvelle/read/id/1

Le paramêtre « id » avec la valeur « 1 » sera accessible dans l'action. Idem pour :

> nouvelle/delete/id/123
> nouvelle/update/id/4321

Nous verrons ensuite plus comment personnaliser le routeur pour obtenir une URI plus lisible pour l'humain et les moteurs de recherche.

nouvelle/read/id/1 peut être réécrite en nouvelle/titre/Premiere_nouvelle

Ce nouvel URI a un titre écrit en français et en clair, c'est pour cela que l'optimisation pour les moteurs de recherche (SEO) peut se faire en parallèle du développement des fonctionnalités d'un site. Avant d'écrire ses actions, la première chose est de charger l'objet dès que l'on instancie le contrôleur. Cela se fait dans le « constructeur » init().

NB: init() aulieu de __construct() permet de ne pas appeler le constructeur de la super-classe. Sans redéfinition, init() est appelée sans aucun paramètre comme dernière instruction de __construct(). Notre objet devrait s'appeller Nouvelle comme l'objet qu'il manipule mais il aurait fallu voir immédiatement l'objet Router pour la réécriture d'URL. Ainsi pour des raisons pratiques, le contrôleur s'appelera Index, car dans le monde du web, l'index est la première chose affichée d'un site.

```
class IndexController extends Zend_Controller_Action
{
        private $_nouvelles;

        public function init()
        {
                $this->_nouvelles= new Nouvelle();
                $this->view->headTitle('Mon blog');
        }
...
}
```

L'objet est chargé et accessible depuis la variable privée de classe $_nouvelles.

Les actions Create Read Update Delete (CRUD)

Nous avons un contrôleur par objet, l'application va dans un premier temps créer les actions de base : c'est à dire : ajouter (create), lire (read), mettre-à-jour (update), effacer (delete) Cette liste d'action est plus connues sous l'acronyme CRUD (Create, Read, Update, Delete) auxquelles on peut rajouter : lister tous les objets (index) mettre à jour une liste d'objets (updatelist) effacer une liste d'objets (deletelist) Ces actions couvre le spectre des actions possible pour un objet mais l'application se devra aussi de pouvoir ajouter une nouvelle action ou d'invalider une existante. L'ordre d'explication des actions sera croissante en difficulté :

1. read : lire un enregistrement et l'afficher
2. create : formulaire d'ajout d'enregistrement
3. index : liste de tous les enregistrements
4. update : formulaire de modification
5. delete : effacement d'un objet

Mini blog/Actions

L'action lire (read).

Cette action prend l'identifiant de l'objet comme paramètre. Ce paramètre est envoyé dans l'URL par la méthode Http Get. Cet identifiant sert ensuite à consulter l'objet par la méthode find(..) :

```
$objet= $this->objet->find( $id )->current();
```

L'objet est ensuite transformé en tableau pour être envoyé à la vue :

```
$this->view->objet= $objet->toArray();
```

Cela donne dans le contrôleur la nouvelle action :

```
        public function readAction()
        {
                $id= (int) $this->_getParam('id');
                $this->view->nouvelle= $this->_nouvelles->find( $id )->current();
        }
```

La vue sera une simple liste à puce, à insérer dans :

application/default/views/scripts/index/read.phtml

```
<p>Nouvelle numero : <?php echo $this->nouvelle['id']; ?></p>
<ul><?php foreach ( $this->nouvelle as $cle => $valeur ) :      ?>
        <li><?php echo $cle . ' : ' . $valeur; ?></li>
<?php endforeach; ?>
</ul>
```

Cet action est accessible maintenant par l'URL :

http://www.monblog.zf/index/read/id/777

L'affichage ne produira qu'un NULL car nous n'avons pas encore enregistré de données. On peut tester en insérant des données directement en base comme ici :

```
INSERT INTO `nouvelle` (`id`, `heuredate`, `titre`, `corps`, `categorie_id`) VALUES
(777, '2008-05-29 18:25:56', 'La date fonctionne', 'La date est imposée\r\n', 1);
```

On peut aussi ajouter l'action create et sa vue.

L'action ajouter (create).

Cette action ajoute un nouvel enregistrement en deux étapes : le formulaire vide est affiché, l'utilisateur saisie les données puis soumet le formulaire. Chaque élément possède un Validateur. Par exemple si l'utilisateur doit saisir un champ requis et qu'il soumet le formulaire sans avoir bien rempli ce champ, le formulaire est réaffiché pré-rempli avec les données précédemment saisies et un message d'erreur est affiché à coté du champ fautif.

```
        $titre= new Zend_Form_Element_Text ( 'titre' );
        $titre->setLabel ( 'titre' )->addValidator ( 'NotEmpty' )->setRequired ( true );
```

Le Framework propose ainsi un service pour examiner la conformité des données : caractères numériques, alpha-numériques, dates, adresse email, type de fichier et avec la possibilité de personnaliser son propre validateur.

La première étape est la création du formulaire, en considérant qu'il s'agit d'un objet fortement lié au modèle on va créer le fichier dans le dossier des modèles. Une autre approche est de considérer ce formulaire comme un objet utilisateur et de créer le fichier dans library/My/Forms/. Dans application/default/models /NouvelleForm.php :

```
class NouvelleForm extends Zend_Form {

        public function init() {
                $this->setName ( 'Nouvelle' )->setMethod ( 'post' );
                $this->setAttrib ( 'enctype', 'multipart/form-data' );

                $id = new Zend_Form_Element_Hidden ( 'id' );
                $this->addElement ( $id );

                $heuredate = new Zend_Form_Element_Hidden ( 'heuredate' );
                $aujourdhui = new Zend_Date();
                $date_format = 'YYYY-MM-dd HH:mm:ss';
                $heuredate->setValue( $aujourdhui->toString($date_format) );
                $this->addElement( $heuredate );

                $titre = new Zend_Form_Element_Text ( 'titre' );
                $titre->setLabel ( 'titre' )->addValidator ( 'NotEmpty' );
                $titre->addFilter ( 'StripTags' )->addFilter ( 'StringTrim' );
                $this->addElement ( $titre );

                $corps = new Zend_Form_Element_Textarea ( 'corps' );
                $corps->setLabel ( 'corps' )->addValidator ( 'NotEmpty' );
                $corps->addFilter ( 'StripTags' )->addFilter ( 'StringTrim' );
                $this->addElement ( $corps );

                $tab= array(    '1' => 'culture',
                                '2' => 'insolite',
                                '3' => 'sport' );
                $categorie_id = new Zend_Form_Element_Select('categorie_id');
                $categorie_id->setMultiOptions( $tab );
                $this->addElement( $categorie_id );

                // le bouton envoi
                $submit = new Zend_Form_Element_Submit ( 'submit' );
                $submit->setAttrib ( 'id', 'submitbutton' );

                $this->addElement ( $submit );
        }
}
```

La deuxième étape est la logique de gestion du formulaire dans l'action create :

```
        public function createAction()
        {
                $form= new NouvelleForm();
                $this->view->form= $form;
                $this->view->placeholder('title')->set('Ajouter une nouvelle');

            if ( $this->_request->isPost () )
            {
                $formData= $this->_request->getPost();
                if ( $form->isValid( $formData ) )
```

```
                {
                                $form->populate ( $formData );
                                $newRow= $this->_nouvelles->createRow( $formData );
                                try {
                                                $newRow->save();
                                        } catch (Exception $e) {
                                                Zend_Debug::dump($e, 'e');
                                        }
                        $this->_redirect ( '/index/index' );
                        } else {
                        $form->populate ( $formData );
                } //isValid()
                } //isPost()
        }
```

La vue associée dans application/default/views/scripts/index/create.phtml :

```
<?php echo $this->form; ?>
```

Maintenant on peut vérifier l'affichage de la nouvelle. Il faut noter auparavant l'id dans la base de donnée et ensuite appeler l'action read :

index/read/id/341

L'action lister tous les objets (index).

Cette action liste tous les enregistrements d'un objets sous forme d'un tableau. Il y en a un par ligne. La logique dans l'action est beaucoup plus simple que celle dans 'create' par contre l'affichage des objets sera moins trivial.

```
    public function indexAction()
    {
        $this->view->nouvelles= $this->_nouvelles->fetchAll();
        $this->view->placeholder('title')->set('Liste de toutes les nouvelles');
    }
```

fetchAll() retourne un objet Rowset et c'est affecté à la variable de vue $nouvelles.

Ainsi dans la vue Html, $nouvelles sera par exemple :

```
object(Zend_Db_Table_Rowset)#38 (10) {
  ["_data:protected"] => array(24) {
    [0] => array(5) {
      ["id"] => string(2) "13"
      ["heuredate"] => string(19) "2008-05-29 18:25:56"
      ["titre"] => string(18) "La date fonctionne"
      ["corps"] => string(25) "La date est imposée"
      ["id_categorie"] => string(2) "17"
    }
    [1] => array(5) {
      ["id"] => string(2) "14"
      ["heuredate"] => string(19) "2008-05-30 10:35:14"
      ["titre"] => string(32) "La première nouvelle fonctionne"
      ["corps"] => string(112) "Dans la doc."
      ["id_categorie"] => string(3) "156"
    }...
```

Il y a ici un objet Rowset contenant 2 objets, que l'on affichera sous forme de lignes de tableau. Chaque colonne correspondant a un attribut. La vue associée sera donc un tableau dynamique car l'on ne connait pas à l'avance le nombre de colonnes.

```
<table>
<caption><?php echo $this->caption?></caption>
<thead>
                <tr><?php foreach ( $this->colonnes as $colonne ) : ?>
                        <th><?php echo $colonne?></th>
                        <?php endforeach; ?>
                </tr>
</thead>
<tfoot>
                <tr><?php foreach ( $this->colonnes as $colonne ) :     ?>
                        <th><?php echo $colonne?></th>
                        <?php endforeach; ?>
                </tr>
</tfoot>
<tbody>
                <?php foreach ( $this->nouvelles as $nouvelle ) : ?>
                <tr><?php foreach ( $nouvelle->toArray () as $col ) : ?>
                        <td><?php echo $col; ?></td>
                        <?php endforeach; ?>
                </tr>
                <?php endforeach; ?>
</tbody>
</table>
```

En résumé, on construit l'en-tête (thead), le pied (tfoot) puis finalement les données du tableau (tbody). Pour une lecture aisée, la liste des objets peut être découpée en pages, un formulaire permet de choisir le nombre d'objets par page. En utilisant le composant Zend_Paginator qui s'appuie sur les sessions PHP pour mémoriser les choix du client.

L'action mettre-à-jour (update).

Cette action prend un identificateur en paramètre, vérifie que les données du formulaire sont valides puis va mettre-à-jour un seul enregistrement. Le même formulaire que 'create' est utilisé sauf qu'il est pré-rempli avec les attributs de l'enregistrement à modifier. La logique de cette action est très proche de create. Cette fois-ci on récupère l'id en paramêtre GET, et on prérempli un formulaire. Par contre s'il y a des données POST, c'est que l'utilisateur a soumis le formulaire, dans ce cas nous mettons à jour l'enregistrement. La première étape est d'écrire les branchements sans mise-à-jour du modèle :

```
        public function updateAction()
        {
                $id= (int) $this->_getParam('id');
                $form= new NouvelleForm();
                $this->view->form= $form;

            if ( $this->_request->isPost () )
            {
                $formData= $this->_request->getPost();
                if ( $form->isValid( $formData ) )
                {
                        Zend_Debug::dump($formData, 'Validé !');
                        // à compléter ...
                }

            } else {
```

```
                $theRow= $this->_nouvelles->fetchRow ( 'id=' . $id );
                        $form->populate ( $theRow->toArray () );
            }
        }
```

La seconde et dernière étape est de compléter avec la mise-à-jour du modèle :

```
                ...
                unset($formData['submit']);
                $where= 'id = '. $id;
                $this->_nouvelles->update( $formData, $where );
                $this->_forward( 'index' );
```

Les données postées sont nettoyées des données non pertinentes au modèle (valeur du bouton 'submit'). On exécute ensuite la requête de mise-à-jour. Il faut garder à l'esprit que les données provenant des paramêtres d'URL comme 'id' doivent être contrôlées pour éviter tous problèmes de sécurité. La vue est la même que pour 'create', ainsi dans application/default/views/scripts/index/update.phtml :

```
<?php echo $this->form; ?>
```

Il faut par contre modifier la vue 'index' pour y ajouter une colonne 'modifier' dans les noeuds thead et tfoot :

```
<th>Modifier</th>
```

Dans le noeud tbody ce sera un lien vers l'objet avec son id en parametre :

```
<td><a href="index/update/id/<?php echo $nouvelle->id; ?>">modif.</a></td>
```

Pour que le lien vers l'URI aboutisse il est souvent conseillé d'utiliser la balise 'Base' dans l'en-tête Html :

```
        <head>
                ...
            <base href="http://www.monblog.zf/">
```

Pour terminer ce tour d'horizon des actions de base, il reste l'action d'effacement : 'delete'.

L'action effacer (delete).

Cette action prend l'identifiant comme paramètre et se déroule en deux temps : au premier appel l'objet à effacer est affiché suivit d'un formulaire avec boutons pour confirmer ou non l'effacement. Si l'utilisateur clique sur 'Non', il est redirigé vers l'action 'index' qui liste tous les enregistrement de l'objet courant. Dans le cas de la confirmation, l'objet est effacé.

```
        if ( $this->_request->isPost () ) {
                        if ( $this->_hasParam('Oui') ) {        // on efface
                                $where= 'avion_id = ' . $id;                    // id uni
                                $this->objet->delete ( $where );
                                $this->view->message='Effacé(e)';
```

La première étape est de créer le formulaire de confirmation ainsi l'action delete peut s'écrire ainsi :

```
public function deleteAction()
{
        $id= (int) $this->_getParam('id');

        $confirmForm= new Zend_Form();
        $hid= new Zend_Form_Element_Hidden('id');
        $hid->setValue( $id );
        $hid->setLabel('Voulez-vous supprimer la nouvelle num. ' . $id . ' ?');
        $oui= new Zend_Form_Element_Submit('Oui');
        $oui->removeDecorator('DtDdWrapper');
        $non= new Zend_Form_Element_Submit('Non');
        $non->removeDecorator('DtDdWrapper');
      $confirmForm->addElements( array($hid, $oui, $non) );
      $this->view->confirmForm = $confirmForm;

      // à compléter ...
}
```

Ici le formulaire est programmé mais il aurait pu être tout autant créé directement en Html dans la vue. La vue s'en trouve d'ailleurs ici réduite au minimum :

```
<?php echo $this->confirmForm; ?>
```

On doit maintenant ajouter la logique d'effacement :

```
if ( $this->_request->isPost () )
{
        if ( $this->_hasParam('Oui') )
        {       // on efface
                $where= 'id = ' . $id;                  // id unique
                $this->_nouvelles->delete ( $where );
        }
$this->_forward( 'index' );
} else {
        if ( $id > 0 ) {
$this->view->nouvelle= $this->_nouvelles->fetchRow ( 'id=' . $id )->toArray();
        }
} //isPost()
```

Pour compléter, la vue on récapitule les attributs de la nouvelle. On copie/colle le code source de la vue 'read' à la suite du formulaire ce qui au final est :

```
<?php echo $this->confirmForm; ?>
<p>Nouvelle numero : <?php echo $this->nouvelle['id']; ?></p>
<ul><?php foreach ( $this->nouvelle as $cle => $valeur ) :      ?>
        <li><?php echo $cle . ' : ' . $valeur; ?></li>
<?php endforeach; ?>
</ul>
```

Il faut comme cela l'a été pour 'update' modifier la vue 'index' pour y ajouter une colonne 'supprimer' dans les noeuds thead et tfoot :

```
<th>Supprimer</th>
```

Dans le noeud tbody ce sera un lien vers l'objet avec son id en parametre :

```
<td><a href="index/delete/id/<?php echo $nouvelle->id; ?>">suppr.</a></td>
```

D'autres actions sont envisageables comme les actions d'effacement (deletelist) ou de modification de masse (updatelist) mais cela dépasse le cadre de ce tutoriel.

Mini blog/URL

Réécriture d'URL, SEO, Zend_Router.

Actuellement l'URI de lecture d'une nouvelle est en anglais et a des paramêtres numériques :

index/read/id/1

Ceci a deux inconvénients, un moteur de recherche va par exemple mal indexer nos articles pour des recherches en français. L'utilisation d'*id* numériques est pratique mais n'a aucune valeur pour l'indexation. Les sites avec un rang d'indexation élevé ont des URI explicites comme :

index/fr/article/L'automne-est-de-retour

La première étape est de tester une redirection d'URL plus simple comme :

index/read/id/714 vers : nouvelle/714

Dans le Bootstrap il suffira d'ajouter (ou dans Initializer.php : initRoutes()) :

```
        $routeur= $this->_front->getRouter();
        $routeur->addRoute( 'nouvelle',
                new Zend_Controller_Router_Route('nouvelle/:id',
                        array(        'controller'    => 'index',
                              'action'                => 'read' ) )
        );
```

Avec $this->_front l'instance du Zend_Controller_Front. Qui est normalement déjà disponible si l'on a créé un projet ZF avec un EDI, sinon il faut rajouter :

```
$this->_front = Zend_Controller_Front::getInstance();
```

La nouvelle route étant créé nous pouvons modifier la vue de l'index pour rajouter un lien vers une nouvelle lorsque l'on clique sur le titre. Dans le noeud tbody de la vue 'application/default/views/scripts/index /index.phtml' :

```
<tbody>
<?php foreach ( $this->nouvelles as $nouvelle ) : ?>
<tr><?php foreach ( $nouvelle->toArray () as $col ) : ?>
<td><a href="<?php echo $this->url( array('id' => $nouvelle->id), 'nouvelle' ) ?>"><?php
```

L'aide de vue (View Helper) Url() construit automagiquement l'URL réécrite, on peut ainsi changer ad nauseam.

La deuxième étape est de modifier cette route pour y intégrer le titre de la nouvelle. Le titre de la nouvelle est une phrase, il nous faut détecter cette phrase à l'aide de l'expression régulière.

Conclusion

Ce rapide tour d'horizon du cadriciel en utilisant l'exemple simple d'une gestion de nouvelle a montré les avantages de cette nouvelle méthode de programmation et la part grandissante des patrons de conceptions dans le développement web moderne.

Il est recommandé de consulter les livres traitant ces sujets en profondeur, de se familiariser avec la documentation officielle et d'établir une veille technologique par l'abonnement aux forums, blogs des développeurs.

Webographie

- La documentation : http://framework.zend.com/manual/fr/
- L'API : http://framework.zend.com/apidoc/core/
- Le forum francophone : http://www.z-f.fr

Et aussi :

- « Programmation PHP », oeuvre collective. - Wikibooks.
- « Cours PHP », oeuvre collective. - Wikiversity, département de la Programmation informatique.

Bibliographie

- français « Développement Php avec le Zend Framework » : Julien Pauli. - Eyrolles, Cahiers du programmeur. - 400 pages.
- anglais « Zend Framework in Action » : Rob Allen. – Manning Publications. – 425 pages ; http://www.manning.com/allen/

Récupérée de « https://fr.wikibooks.org/w/index.php?title=S%27initier_au_Zend_Framework

/Version_imprimable&oldid=484221 »