CakePHP : Présentation

http://www.cakephp-fr.org/

Sommaire

1) PRÉSENTATION

Je vais tenter, tant bien que mal, dans ce tutoriel, de vous faire découvrir le framework PHP « ***CakePHP*** » qui va, je le pense, nous permettre de nous faciliter le codage du site, tout en ayant un cadre de développement « carré ».

CakePHP est un framework PHP entièrement libre, sous licence MIT, basé sur le motif de conception **MVC**.

Voici sa description sur le site officiel :

« *CakePHP est un framework de développement rapide pour PHP qui fournit une architecture extensible pour développer, maintenir et déployer des applications. Utilisant des motifs de conception bien connus tels MVC et ORM ainsi que le paradigme "convention plutôt que configuration", CakePHP réduit les coûts de développement et aide les développeurs à écrire moins de code.* »

2) STRUCTURE DE FICHIERS DE CAKEPHP

Après avoir téléchargé et extrait ***CakePHP***, voici les fichiers et répertoires que vous devriez voir :

- **app**
- **cake**
- **vendors**
- .htaccess
- index.php
- README

Vous remarquerez trois dossiers principaux :

1. Le dossier **app** sera celui où vous exercerez votre magie : c'est là que vous placerez les fichiers de votre application.
2. Le dossier **cake** est l'endroit où nous avons exercé notre propre magie. Engagez-vous personnellement à ne pas modifier les fichiers dans ce dossier. Nous ne pourrons pas vous aider si vous avez modifié le cœur du framework.
3. Enfin, le dossier **vendors** est l'endroit où vous placerez vos librairies PHP tierces dont vous avez besoin pour vos applications ***CakePHP***.

(sources : « Le CookBook de CakePHP »)

Le répertoire « **app** » :

Répertoires	Description
config	Contient les (quelques) fichiers de configuration utilisés par ***CakePHP***. Informations de connexion à la base de données, démarrage, fichiers de configuration de base et tout fichier du même genre doivent être rangés ici.
controllers	Contient vos contrôleurs et leurs composants.
locale	Stocke les fichiers pour l'internationalisation.
models	Pour les modèles, comportements et sources de données de votre application.
plugins	Contient les packages des Plugins.
tmp	C'est ici que ***CakePHP*** enregistre les données temporaires. La manière dont sont stockées les données actuelles dépend de la configuration que vous avez effectuée, mais ce répertoire est habituellement utilisé pour déposer les descriptions de modèles, les logs et parfois les informations de session.
vendors	Toutes classes ou librairies tierces doivent être mises ici, de sorte qu'il sera facile d'y accéder par la fonction vendors(). Les observateurs avisés noteront que cela semble redondant avec le répertoire "vendors" à la racine de l'arborescence. Nous aborderons les différences entre les deux lorsque nous discuterons de la gestion multi-applications et des configurations systèmes plus complexes.
views	Les fichiers de présentation sont placés ici : éléments, pages d'erreur, assistants, mises en page et vues.
webroot	Dans un environnement de production, ce dossier doit être la racine de votre application. Les sous-répertoires sont utilisés pour les feuilles de style CSS, les images et les fichiers Javascript.

(sources : « Le CookBook de CakePHP »)

3) UTILISATION

CakePHP est fortement basé sur ses **conventions de nommage**.

Si l'on souhaite l'utiliser convenablement, et ainsi, gagner du temps au niveau du codage, il faut absolument que tout le monde se conforme aux quelques conventions à respecter.

Exemple concret :

On souhaite créer un blog. Nous allons avoir besoin d'une table qui stockera les messages du blog, que nous nommerons « **message**s ». Ensuite, dans la logique MVC, nous allons avoir besoin de 3 éléments :

1) 1 **Table** (qui stockera les messages du blog)
2) 1 **Modèle**
3) 1 **Vue**
4) 1 **Contrôleur**

Comme dit au-dessus, ***CakePHP*** nous permet de gagner beaucoup de temps au niveau du codage. En suivant les conventions du framework, nous allons donc avoir, respectivement :

1) 1 **Table** : la table doit également suivre les conventions de nommage, vu que tout sera basé sur elle ensuite :
=> doit être un nom au pluriel. J'ai choisi ici « **message**s ».

2) 1 **Modèle** : « app/models/**message**.php »
=> convention de nommage : « ***<nom_table_au_singulier>*.php** »

3) 1 **Vue** : « app/views/**message**s**/index.ctp** »
=> convention de nommage : « ***<nom_table>*/*<nom_vue>*.ctp** »
1 dossier du nom du modèle qui va contenir toutes les vues liées à ce contrôleur.
Nous avons ici une 1ère vue « **index.ctp** ». Je détaillerai ce nom un peu plus loin.

4) 1 **Contrôleur** : « app/controllers/**message**s**_controller.php** »
=> convention de nommage : « ***<nom_table>*_controller.php** »

Récapitulatif de la répartition des fichiers :

- **app**
  - **models**
    - message.php
  - **views**
    - message**s**/index.ctp
  - **controllers**
    - message**s**_controller.php
- Table « message**s** » (nom au pluriel).

Dans tout ce qui va suivre, **l'emploi de couleurs n'est pas anodin**. Veillez à établir les liens qu'il existe entre les différentes couches afin de saisir comment fonctionne le framework.

La convention de forme des URL utilisées par le framework est la suivante :

« www.exemple.com/<nom_contrôleur>/<nom_méthode>/<param1>/<param2>[/<paramX>]* »

Exemple : « www.exemple.com/monControleur/maMethode/maValeur1/maValeur2 »
=> Cette URL appellera la méthode « **maMéthode()** » du contrôleur « **monControleursController** »
avec comme paramètres, respectivement : maValeur1 et maValeur2
⇔ **« maMethode(maValeur1, maValeur2) ; »**.

Voyons maintenant le contenu de la table et des 3 fichiers modèle, vue et contrôleur associés à cet exemple.

Contenu de la table « messages » :

```
/* D'abord, créons la table des messages : */
CREATE TABLE messages (
   id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   titre VARCHAR(50),
   corps TEXT,
   cree DATETIME DEFAULT NULL,
   modifie DATETIME DEFAULT NULL
);

/* Puis insérons quelques messages pour les tests : */
INSERT INTO messages (titre,corps,cree)
   VALUES ('Le titre', 'Voici le contenu du post.', NOW());
INSERT INTO messages (titre,corps,cree)
   VALUES ('Encore un titre', 'Et le contenu du post qui suit.', NOW());
INSERT INTO messages (titre,corps,cree)
   VALUES ('Le retour du titre', 'C\'est vraiment excitant, non ?', NOW());
```

Fichier modèle « message.php » lié à la table « messages » :

```
<?php

class Message extends AppModel {

   var $name = 'Message'; // À déclarer dans chaque modèle avec comme valeur le nom du modèle

}

?>
```

Commentaire :

Tout modèle doit étendre (hériter de) la classe « **AppModel** ».

Fichier contrôleur « messages_controller.php » associé :

```
<?php

class MessagesController extends AppController {

	// À déclarer dans tout contrôleur avec comme valeur le nom du contrôleur
	// sans «Controller » à la fin
	var $name = 'Messages';

	function index() {

		$this->set('messages', $this->Message->find('all'));

	}

	function afficher($id = null) {

		$this->Message->id = $id;

		$this->set('msg', $this->Message->read());

	}

}

?>
```

Commentaires :

1) Tout contrôleur doit étendre (hériter de) la classe « **AppController** ».

2) La méthode **index()** : méthode par défaut qui sera appelée lorsque l'URL « **/messages/** » ou « **/messages/index** » sera saisie dans le navigateur.
La méthode **afficher()** suit la même démarche : elle sera appelée lorsque l'URL « **/messages/afficher** » sera saisie dans le navigateur.

3) La méthode **set()** de la classe parente « **AppController** » permet d'associer une valeur à une variable de la forme : **set(<variable>, <valeur>)**, qui pourra ensuite être exploitée dans la vue sous la forme « **$<variable>** ».

4) Du fait du respect des conventions, il est très simple d'accéder au modèle associé par la forme : « **$this -> <Modèle>** », et d'y appliquer des méthodes pré-faites pour gagner du temps, comme ici, par exemple, la méthode **find('all')** qui aura pour résultat de retourner tous les tuples de la table liée au modèle, ainsi que tous les champs.
Par contre, dans la méthode **afficher()**, on utilise la méthode **read()**. Celle-ci permet de faire une projection sur un seul (ou plusieurs) champ(s).

Fichier vue n°1 « index.ctp » :

```
<h1>Les messages du Blog</h1>
<table>
	<tr>
		<th>Id</th>
		<th>Titre</th>
		<th>Créé le</th>
	</tr>

	<!-- C'est ici que nous bouclons sur le tableau $messages afin d'afficher les informations des différentes messages stockés en base de données -->
	<?php foreach ($messages as $msg): ?>
	<tr>
		<td><?php echo $msg['Message']['id']; ?></td>
		<td>
			<?php echo $html->link($msg['Message']['titre'], "/messages/afficher/".$msg['Message']['id']); ?>
		</td>
		<td><?php echo $msg['Message']['cree']; ?></td>
	</tr>
	<?php endforeach; ?>
</table>
```

Commentaires:

1) Toutes les vues doivent se trouver dans le dossier portant le nom du contrôleur associé, et avoir le nom de la méthode liée suivi de l'extension « **.ctp** » (exemple : ici, « **index.ctp** » est la vue qui sera appelée lorsque l'URL suivante sera demandée « www.exemple.com/**messages**/**index** » ou « www.exemple.com/**messages**/ » . Ainsi, la vue liée à la méthode **afficher()** devra porter le nom « **afficher.ctp** ».

2) Ici, afin d'afficher tous les messages stockés en base, nous faisons un *foreach* sur la variable-tableau « **$messages** », qui a été définie dans le contrôleur via l'instruction « **$this->set('messages', $this->Message->find('all') ;** ». Nous pouvons ensuite accéder aux différents champs à l'aide de la structure « $msg['**<nom_modèle>**']['**<nom_champs_table>**'] ».

3) Dans cette vue, nous utilisons un outil proposé par ***CakePHP,*** permettant de générer des liens hypertextes facilement, via l'instruction suivante :
< ?php echo **$html->link(**$msg['**Message**']['**titre**'], "/posts/view/".$msg['**Message**']['**id**']**)**;

Il s'utilise de la forme : **$html->link('<intitulé_lien>', '<url_du_lien>')** ;

CakePHP met à disposition tout un tas d'outils similaires, permettant de nous simplifier la vie, tout en faisant un code propre et respectant le MVC.

Fichier vue n°2 « afficher.ctp » :

```
<h1><?php echo $msg['Message']['titre']; ?></h1>

<p><small>Créé le : <?php echo $msg[' Message ']['cree']; ?></small></p>

<p><?php echo $msg['Message']['corps']; ?></p>
```

Commentaire :

Cette vue sera appelée lorsque, par exemple, l'URL « www.exemple.com/**messages**/**afficher**/1 » sera demandée, et passera à la méthode **afficher()** la valeur « 1 » comme ID de message, afin d'aller chercher le message correspondant dans la table, et l'afficher.

4) CONCLUSION

À mon avis, ce framework nous permettrait de développer dans un cadre structuré, efficacement et simplement. Je rappelle que ceci n'est qu'une proposition, et ne nous engage à rien.

CakePHP offre tout un tas d'outils qui permettent de nous faciliter le travail, en proposant des outils de vérification de données, gestion de sessions, connexion à un espace privilégié, envoi de mails, etc .

Je vous invite à essayer de réaliser le petit exemple donné sur le « *CookBook de CakePHP* » (que vous trouverez à l'adresse http://book.cakephp.org/fr/view/219/Blog), afin de constater la facilité et le peu de code nécessaire pour produire un début de blog.

Ce tutoriel expose l'essentiel des choses générales à connaître pour utiliser efficacement ce framework, et met surtout en avant les conventions qu'il utilise.

Merci de votre lecture. Je reste à disposition pour toutes questions complémentaires.