



Cake PHP : du gâteau pour tous les zéros !

Informations sur le tutoriel



Auteur : [Jibriiss](#)

Difficulté :

Licence :

[Plus d'informations](#)

Popularité

Visualisations : 680 139

Appréciation 0

des lecteurs : 7

11

47

64 personnes souhaitent voir ce tutoriel publié en livre !

[Vous aussi ?](#)

Publicité

Historique des mises à jour

- Le 06/12/2010 à 19:19:20
#3385 à #3389 : Corrections orthographiques
- Le 20/04/2010 à 14:15:44
Mise à jour pour CakePhp 1.3
- Le 22/02/2010 à 16:30:11
Correction orthographique.

Bonjour les zéros !

Ce tutoriel s'adresse aux *aficionados* du PHP. Vous avez lu le tutoriel de M@teo21, vous avez parcouru les tutoriels non officiels, maintenant vous savez bien tout, vous avez fait votre propre forum avec des fonctions spéciales pour des amis, un super forum. Seulement voilà, vous avez montré ça aux copains, à la famille et maintenant :

- la petite soeur veut que vous lui fassiez un blog juste pour elle ;
- le grand-père veut un site pour son club de collection de 33 tours ;
- la belle blonde du lycée aimerait bien que vous lui fassiez un site pour elle et ses copines ;
- le reste de la classe aimerait un site pour échanger les cours et mettre en ligne « anti-sèches » ou autres devoirs corrigés 🤔 ;
- le site olé olé de votre petit frère aurait besoin d'un coup de neuf ;
- sans oublier le voisin plombier qui est prêt à vous donner un petit billet si vous lui faites un beau site pour son entreprise.

Aïe, ça va me prendre des heures à réaliser, tout ça ! 😓

Surtout qu'en plus c'est toujours la même chose, il faut toujours refaire les mêmes formulaires, toujours le même système d'*upload* de fichiers, ça va être du copier-coller sans être sûr que ça s'adapte parfaitement à l'autre site. 😓

Ça serait bien s'il existait un truc pouvant accélérer le développement et me faire gagner du temps !

Eh bien ça tombe bien, vous êtes sur le bon tutoriel pour régler votre problème. Je vais vous apprendre à utiliser CakePHP, un ensemble de fonctions déjà codées qui, après un temps d'adaptation, vont vous faire gagner énormément de temps.

Par contre le numéro de la blonde ça serait bien de le faire circuler, hein ?



Pour suivre le cours (à partir de la 2^e partie) vous devrez connaître les bases de la POO pour PHP4. Si vous n'y connaissez RIEN de RIEN, j'ai fait une annexe pour ceux qui n'ont suivi que les cours de M@teo21. La POO de CakePHP est très simple, donc vous n'avez pas besoin d'être un expert de la POO pour l'utiliser. Suivez mon annexe et vous aurez les bases qu'il faut. Vous pouvez également trouver des tutoriels consacrés à la POO sur le Site du Zéro.

Ce cours est composé des parties suivantes :

- [Introduction](#)

- [Notre première application](#)
 - [Automatisation](#)
 - [Annexes](#)
-

• Partie 1 : Introduction

Je vais commencer par vous présenter CakePHP.

Je suis partisan de l'apprentissage par la pratique, mais là nous allons quand même être obligés de faire un tout petit peu de théorie.

Il y aura très peu de codage sur cette partie, on finira juste avec une application vide.

Mais nous aurons les connaissances de base, très importantes quand on travaille avec ce *framework*. On se prend la tête mais c'est pour gagner du temps par la suite, vous verrez...

Attention pas de panique, ça reste simple. Je vais essayer de suivre la philosophie du site : on commence tout à partir de zéro. 😊

o

[1\) Le framework](#)



- [Qu'est ce que c'est un "framework" ?](#)
- [Cake PHP](#)

o

[2\) Le MVC](#)



- [But de l'architecture MVC](#)
- [Le contrôleur](#)
- [La vue](#)
- [Le modèle](#)

o

[3\) Installation & configuration](#)



- [Les bons outils pour bien travailler](#)
 - [Mise en place](#)
 - [Un problème ?](#)
-

• Partie 2 : Notre première application

Allez, on va attaquer avec un exemple parlant.

On va réaliser une application qui va gérer une bibliothèque !

Tout ce qu'il y a de plus simple. Il y aura des exemplaires de livres, qui seront regroupés par genre littéraire. Les abonnés pourront les emprunter et les ramener. Pour l'instant c'est déjà pas mal.

Avant de commencer je vais vous dire que ce qu'on va faire là, Cake PHP peut le faire automatiquement (j'irai même jusqu'à dire « automagiquement »). Mais je trouve que c'est beaucoup plus simple de comprendre le système en créant soi-même les fichiers, surtout que ce n'est pas très dur. 😊

o

[1\) Afficher la liste des livres](#)



- [L'ADN : la table](#)
- [Le squelette : le modèle](#)
- [Les muscles : le contrôleur](#)
- [La peau : la vue](#)

o

[2\) Ajouter, modifier et supprimer des livres](#)



- [Ajouter un nouveau livre](#)
- [Effacer un Livre](#)
- [Éditer un livre](#)

o

3) TP : les auteurs



- [Énoncé](#)
- [Correction](#)
- [Le petit plus](#)

o

4) Relier les modèles



- [Les 4 types d'association](#)
- [À l'affichage](#)
- [À l'édition](#)

C'est fini pour cette première partie.

La prochaine partie ne va rien vous apprendre.

On va juste refaire la même chose, mais beaucoup plus vite !

• Partie 3 : Automatisation

À partir de là, on va commencer à utiliser les outils de génération de code de Cake PHP.

On va ainsi pouvoir créer des fichiers complets en tapant presque rien.

o

1) Encore plus rapide avec Scaffold !



- [Une nouvelle table pour notre exemple](#)
 - [Le modèle et le contrôleur](#)
 - [Utilité & limites](#)
-

• Partie 4 : Annexes

o

1) Conventions de nommage



- [Règles de base](#)
- [Le petit problème](#)
- [Tableau récapitulatif](#)

o

2) Introduction à la POO



- [Les exemples simples du début](#)
 - [Points particuliers](#)
 - [Héritage](#)
-

Partie 1 : Introduction

Je vais commencer par vous présenter CakePHP.

Je suis partisan de l'apprentissage par la pratique, mais là nous allons quand même être obligés de faire un tout petit peu de théorie.

Il y aura très peu de codage sur cette partie, on finira juste avec une application vide.

Mais nous aurons les connaissances de base, très importantes quand on travaille avec ce *framework*. On se prend la tête mais c'est pour gagner du temps par la suite, vous verrez...

Attention pas de panique, ça reste simple. Je vais essayer de suivre la philosophie du site : on commence tout à partir de zéro. 😊

Le framework

Je vais déjà vous présenter CakePHP.

Je vais essayer de rester le plus neutre possible lors de la rédaction du tutoriel, mais j'avoue que je suis un grand fan de ce framework alors je risque de ne pas être impartial.

Je l'utilise pour toutes mes applications PHP, même pour le blog de ma soeur qui, je dois l'avouer, n'utilise que 10% des capacités offertes par CakePHP. 🤖

Ce n'est pas grave, c'est juste pour le plaisir et je vais vous contaminer. 🐱

Qu'est ce que c'est un "framework" ?

Pour faire simple, c'est un gros tas de code qui est déjà là quand vous commencez votre projet : c'est la base de votre projet. Le *framework* propose une organisation : les fichiers sont rangés d'une façon précise qu'il faut respecter, ce qui est génial pour les personnes bordéliques comme moi. 😊

Il y a beaucoup de *frameworks* différents.

Certains sont légers, faciles à prendre en main mais ne proposent pas beaucoup de fonctions. D'autres au contraire sont de véritables machines à gaz mais au moins vous avez toutes les options déjà codées (et qui fonctionnent !) qui sont prêtes à être utilisées.

Il ne faut pas hésiter à réfléchir un bon moment et à prendre son temps pour choisir le *framework* de son application, c'est très frustrant de s'apercevoir après avoir fait une bonne partie du travail qu'on est obligé de coder une fonctionnalité qu'un autre *framework* aurait pu nous fournir facilement, ou bien de voir son application ralentie par des modules qu'on n'utilise même pas.

Cake PHP

Bon, c'est le sujet du tuto alors on va quand même en parler.

J'estime que CakePHP se situe dans la moyenne des *frameworks* ; ni trop gros, ni trop petit.

Vous disposerez déjà de toutes les options requises pour développer une application sérieuse :

- architecture MVC : si vous ne connaissez pas, j'en parle dans le prochain chapitre. C'est une pratique de programmation assez à la mode en ce moment et qui a l'avantage de bien organiser les fichiers en séparant la logique de la présentation et des données ;
- gestion du cache : les pages souvent affichées sont enregistrées au lieu d'être régénérées par PHP à chaque fois ;
- multilingue : si vous voulez que votre site soit compréhensible dans le Boutékistan Oriental, ça sera possible ;
- authentification et gestion des droits : vous pourrez interdire certaines actions à certains visiteurs ;
- un peu d'aide pour le code Javascript/Ajax grâce aux bibliothèques Javascript Prototype/Script-a-culous qui sont intégrées.



On peut noter que la gestion des droits est réputée pour être un peu difficile à utiliser, mais elle est très complète et elle permet de gérer des groupes d'utilisateurs et de leur donner des droits très spécifiques.

CakePHP fait partie des *frameworks* RAD : Rapid Developpement Application. Le but est de vous faire gagner un maximum de temps. Une fois que vous savez l'utiliser, vous pouvez littéralement réaliser une application fonctionnelle en quelques jours.

Il imite un peu le comportement de Ruby on Rails. J'ai souligné et mis en gras le mot "imite" parce qu'il est loin d'atteindre le niveau de RoR.

Vous avez déjà un minimum de configuration, au minimum vous avez juste à remplir un fichier (login/mdp de votre base de données) pour faire tourner votre site et c'est tout ! Cela est possible parce que vous devez respecter des règles très précises, sur le nom et l'emplacement de vos fichiers. Retenez juste que si vous suivez ces règles tout se passera bien.

Il y a également une console que vous pouvez utiliser pour générer automatiquement du code, ce qui vous fait une base de travail. Toujours pour vous faire gagner du temps.

Comme c'est indiqué sur leur documentation : tout est « **automagique** » ! 🧙

La documentation officielle, en anglais, se trouve à cette adresse <http://book.cakephp.org/>. Il y a une version française mais elle n'est pas finie et n'est

pas aussi précise que la version anglaise.



Sachez que la documentation était un des points faibles du *framework* pendant un bon moment. Il fallait littéralement fouiller dans les fichiers du *core* (le coeur) du *framework* pour comprendre à quoi servait telle ou telle méthode. Certaines fonctions étaient même méconnues du public ! Maintenant, c'est beaucoup plus facile. 😊



Notez également que CakePHP n'est pas un petit projet personnel. C'est un *framework* sérieux utilisé dans certaines entreprises pour de grands sites. Par exemple le site de [Mozilla Addons](#) est fondé sur CakePHP. Ce n'est pas le plus utilisé mais il s'est fait sa place parmi les *frameworks* PHP.

Le MVC

But de l'architecture MVC

MVC signifie : **M**odèle, **V**ue, **C**ontrôleur (ça tombe bien, ce sont les mêmes initiales en français qu'en anglais pour une fois 😊).

Quand on fait une application en suivant le modèle "MVC" on sépare tout en 3 parties distinctes :

1. le **Modèle** qui s'occupe des données (en pratique, il communique avec la base de données et il sait où trouver les informations et comment les modifier) ;
2. la **Vue**, c'est tout le côté visuel, c'est ce que voit l'utilisateur. C'est là que se trouve le code XHTML ;
3. le **Contrôleur** effectue les traitements. C'est lui qui "comprend" la demande de l'utilisateur et il exécute cette demande (en utilisant les modèles) et ensuite il appelle une vue pour afficher le résultat à l'utilisateur.

C'est très rapidement expliqué mais je l'ai fait exprès, on verra en pratique comment ça se passe.



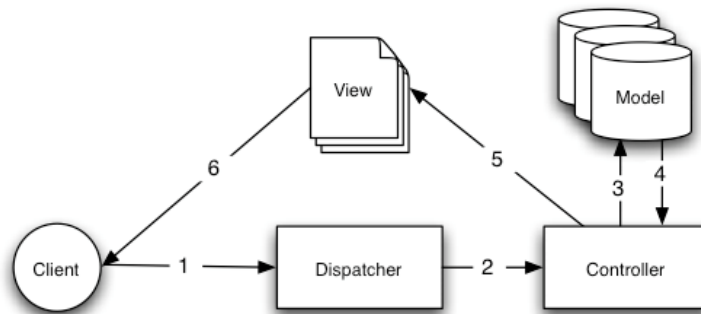
Si vous voulez plus de détails sur l'architecture MVC, [vincent1870](#) a écrit un tutoriel dessus : [Adopter un style de programmation clair avec le modèle MVC](#).

cysboy en parle aussi dans son cours sur Java : [Un pattern puissant : le pattern MVC](#).

Il y a également beaucoup d'articles sur Internet qui explorent le sujet à fond, mais ce n'est pas essentiel de les lire, en suivant juste ce tutoriel vous comprendrez le modèle MVC très facilement.

Déroulement d'une requête avec CakePHP

Allez, pour mieux comprendre je vais vous montrer comment cela se passe lorsque le client demande une page. Je vais vous coller le même schéma que l'on trouve sur la documentation de CakePHP :



Voilà : ça explique comment cela se passe lorsque l'on demande la liste des produits.

1. La requête arrive sur le dispatcher. Oui, bon, je ne vous en ai pas parlé ; c'est simplement un petit script qui sait quel contrôleur appeler.
2. On a demandé l'adresse "www.site.com/produits/afficher", eh bien on va aller chercher la fonction "afficher" du contrôleur "produits".
3. Le contrôleur demande au modèle "produits" la liste des produits.
4. Le modèle renvoie cette liste au contrôleur.

5. Le contrôleur envoie cette liste vers la vue.
6. Après avoir dessiné le code XHTML, la vue renvoie tout vers le client.

On explorera tout cela plus en détail par la suite, il y a d'autres choses à prendre en compte. Mais pour l'instant on n'utilisera que ces 6 étapes.

Le contrôleur

Le contrôleur est souvent appelé en premier dans une application.
Généralement quand on veut afficher une page, on fait pointer l'URL sur le contrôleur.

Son but est de traduire la demande de l'utilisateur, il fait souvent des tâches très simples :

- vérifier si l'utilisateur a le droit de faire cette action ;
- vérifier si l'utilisateur a bien envoyé les informations requises (s'il a rempli le formulaire correctement) ;
- afficher la bonne page en réponse (message d'erreur ou page de confirmation).

On va prendre un exemple.

Le client est sur un site d'e-commerce et il décide de regarder la liste des produits des clients VIP.

Le contrôleur "ArticlesController" reçoit la demande de l'utilisateur : il faut afficher la liste des produits, donc :

- il vérifie qu'il est bien connecté ;
- il demande au Modèle "Client" de lui donner les informations du client qui est connecté ;
- il vérifie grâce à ces informations que le client a le droit de voir les offres VIP ;
- il demande au Modèle "Article" de lui donner la liste des tous les produits VIP ;
- il envoie la liste des articles à la Vue "afficher_article_vip".

Voilà, son boulot est terminé, il a su expliquer aux modèles et à la vue quoi faire.

La vue

La vue c'est simplement l'affichage.

Quand la vue est appelée, tout le code "logique" a déjà été effectué. Il faut juste afficher les informations. Il reçoit du contrôleur les informations à afficher.

En pratique c'est du code XHTML avec un peu de PHP pour afficher les informations (seulement des fonctions *echo()* et quelques boucles *foreach* pour parcourir les tableaux d'informations).

Le modèle

Le plus important est le modèle : il est très souvent associé à une table de votre base de données, et fait le lien entre celle-ci et votre application. Il s'assure de l'intégrité et de la cohérence des données (trop classe la phrase 😊). En gros, il vérifie que vous ne remplissez pas la base de données avec des conneries (moins classe, mais on comprend mieux).

Normalement, on ne devrait faire aucune requête SQL en dehors du modèle, si on veut récupérer des informations on a des fonctions *find()*, si on veut enregistrer on a des fonctions *save()* qui savent quand il faut faire soit un INSERT soit un UPDATE. On n'a pas besoin de connaître SQL. 😊 Il suffit juste de régler le modèle, on lui explique qu'un client possède un profil, qu'il peut avoir effectué plusieurs commandes, et qu'il appartient à un groupe de clients qui contient d'autres clients. Avec ces informations, le modèle saura comment récupérer des informations comme un grand !



Le premier que je vois faire une requête SQL en dehors d'un modèle, je l'envoie dans la cave de M@teo21 !

Le modèle vérifie également les données lors des modifications : vous lui indiquez qu'un client a forcément un nom et un prénom, que son code postal est un nombre et que s'il habite en France son numéro de téléphone aura 10 chiffres ! Si ces conditions ne sont pas remplies il refusera l'insertion.

Toutes ces fonctions sont déjà codées dans le *framework* ! Tout ce qu'on a à faire, c'est de les utiliser. Enfin presque, il faut quand même lui indiquer comment il doit réagir. Notre principal boulot dans un modèle sera de lui dire comment il doit valider les données (*tel champ est obligatoire, celui-là doit avoir 3 lettres minimum et celui-là c'est seulement des chiffres...*). Mais on doit aussi lui indiquer les relations entre nos différentes tables. Si on lui indique *"un article du forum appartient à un auteur"* et que *"un auteur possède plusieurs articles du forum"*, quand on demandera un article le modèle Auteur saura tout seul que cet article correspond à un auteur et il vous renverra son nom !

Installation & configuration

Merci à Baboso et à Almaju pour leur commentaires très constructifs. J'ai intégré vos remarques dans le tuto !

Allez hop, on attaque la pratique.

Les bons outils pour bien travailler

En fait, il n'y a pas besoin d'outils pour travailler. 😊 Un bloc-note suffit.

Bon, on va quand même pas exagérer, utilisez un éditeur qui colore le texte et qui peut ouvrir plusieurs fichiers simultanément (Notepad++, Scite, etc.). L'idéal c'est quand même d'avoir un éditeur qui affiche la liste des fichiers dans une arborescence sur le côté, la plupart des EDI le font (Aptana Studio et Eclipse PDT sont gratuits).

Pour faire tourner votre script, il vous faut un serveur Web avec PHP et une base de données.
Sous Windows, EasyPHP et WAMP feront très bien l'affaire. Sous Linux vous avez LAMP qui fonctionne très bien !

Et en tout dernier, il vous faut... CakePHP, oui forcément ça aide. 😊 Vous pouvez le télécharger sur <http://cakephp.org/>.
Voilà [un lien direct](#), mais bon, vérifiez quand même qu'il n'y ait pas une nouvelle version depuis l'instant où j'ai écrit ces lignes...

Mise en place

Là c'est très simple. Le but c'est de gagner du temps, vous allez vite être mis dans le bain.

L'arborescence

Vous avez une archive, décompressez-la et mettez tous les répertoires tels quels là où vous voulez travailler.



Les utilisateurs de linux et de mac doivent faire attention à bien tout copier. Il y a des fichiers nommés .htaccess. Ce sont des fichiers cachés (car leurs noms commencent par un point). Prenez garde à ne pas les oublier (il y en a 5 en tout, mais les 3 plus importants sont à la racine, dans /app et dans /app/webroot)

Vous devez obtenir ceci :

On va commencer par les répertoires principaux :

- **/cake/** : contient les fichiers que CakePHP utilise. Au début on ne touchera pas du tout à ce répertoire, au risque de tout détraquer ;
- **/plugins/** : on n'utilisera pas les plugins mais c'est une fonction intéressante de CakePHP. Si vous avez une partie de votre site qui est terrible, vous pouvez rassembler juste les fichiers de cette partie et l'intégrer sur un autre site sous forme de plug-in indépendant. Une autre façon de ré-utiliser le code et de gagner du temps !
- **/vendors/** : ce sont des modules extérieurs téléchargés que vous pouvez intégrer à votre application ;
- **/app/** : contient votre application, vous travaillerez uniquement dans ce répertoire ;
- **/app/controllers/** : les contrôleurs ;
- **/app/models/** : les modèles ;
- **/app/views/** : les vues ;
- **/app/config/** : pour les fichiers de configuration ;
- **/app/locale/** : pour la localisation, les traductions ;
- **/app/webroot/** : c'est la racine de l'application ;
- **/app/webroot/css/** : les feuilles de style CSS ;
- **/app/webroot/img/** : les images ;
- **/app/webroot/js/** : les fichiers Javascript.

Vos fichiers devront être placés dans un répertoire bien précis. Mais les noms parlent d'eux-mêmes alors je ne pense pas que ça sera compliqué. Je parie même que tout le monde a compris que les modèles iront dans le répertoire "models" ! Très rapidement vous serez habitués et je n'aurai même plus besoin de vous dire dans quel répertoire mettre tel fichier. Vous verrez que même si vous êtes très bordélique, tout restera bien rangé parce que CakePHP ne se gênera pas pour vous envoyer un petit message d'erreur si un fichier ne se trouve pas à la bonne place.

Gardez aussi en tête qu'au niveau du XHTML, la racine est `/app/webroot/`. Donc si vous avez mis une image dans `/app/webroot/img/image.png`, pour y accéder il suffira de faire :

Code : HTML - [Sélectionner](#)

```


```

Configuration de la base de données

On va commencer par configurer, vous allez voir c'est simple. Renommez (ou mieux, copiez) le fichier `/app/config/database.php.default` vers `/app/config/database.php`. Ensuite, ouvrez `/app/config/database.php`. Vous devrez trouver une portion de code qui ressemble à ça :

Code : PHP - [Sélectionner](#)

```
<?php
var $default = array(
    'driver' => 'mysql',
    'persistent' => false,
    'host' => 'localhost',
    'login' => 'root',
    'password' => '',
    'database' => 'cakephp',
    'prefix' => '',
);
?>
```

- "driver" est à laisser tel quel si vous utilisez MySQL. Sinon, changez-le pour votre SGBD. Moi je tourne sous MySQL alors j'ai laissé "mysql" ;
- "persistent" indique si CakePHP doit garder la connexion à la BDD ouverte en permanence ou non, laissez à false ça marche très bien ; 😊
- "host" c'est le serveur de BDD, j'ai mis "localhost" car mon serveur tourne en local, je suppose que c'est pareil chez vous ;
- "login", "password" et "database" sont les informations classiques à fournir quand vous vous connectez à votre base de données. Mon login est "root" et mon mot de passe est vide.



"prefix" est pratique si vous êtes chez un hébergeur gratuit qui ne met à votre disposition qu'une seule base de données. Pour éviter que vos tables ne portent le même nom vous pouvez indiquer ici un préfixe qui sera ajouté devant tous les noms de vos tables. Moi j'ai laissé vide car je tourne en local, je peux créer autant de bases que je veux.

Une fois ceci fini, enregistrez le fichier.

URL Rewriting

Le rewriting, c'est un module du serveur Apache qui permet de faire des adresses "jolies".
Par exemple sur le Site du Zéro, quand on regarde le cours de M@teo21 on voit comme URL :
www.siteduzero.com/tutoriel-3-14668-un-site-dynamique-avec-php.html
au lieu de :
www.siteduzero.com/application/tutoriel.php?id_tuto=14668&page=3.
C'est quand même mieux, et surtout ça permet d'améliorer le référencement.

Pour activer le module rewrite de votre serveur Apache (sous windows, avec WAMP), trouvez le fichier de configuration "httpd.conf", vous pouvez l'ouvrir avec le bloc-note. Cherchez la ligne suivante :

Code : Autre - [Sélectionner](#)

```
#LoadModule rewrite_module modules/mod_rewrite.so
```

et retirez le dièse.

Code : Autre - [Sélectionner](#)

```
LoadModule rewrite_module modules/mod_rewrite.so
```



Le dièse signale en fait un commentaire, quand il était là Apache ignorait cette ligne et n'activait pas le service. Maintenant que vous l'avez retiré tout va s'activer. Si le dièse n'était pas là c'est que le module rewrite était déjà activé alors ne changez rien.

Si vous avez modifié le fichier, enregistrez-le et redémarrez le serveur.

Sous linux

Si vous êtes sous linux, activer le mod rewrite est plus simple.
Vous devez simplement taper ces commandes :

Code : Console - [Sélectionner](#)

```
a2enmod rewrite  
/etc/init.d/apache2 reload
```



a2enmod est la commande pour activer un module sur apache 2. **A**pache **2** **E**Nable **M**ODule. La première ligne active donc le module « rewrite ». La 2ème ligne permet de recharger la configuration du serveur sans le redémarrer.



Si vous n'êtes pas l'utilisateur root, n'oubliez pas le sudo devant chaque ligne.

Désactivation de l'URL Rewriting



Argh mais moi je n'ai pas accès aux réglages de mon serveur. Mod Rewrite n'est pas activé et j'ai aucun moyen de toucher aux fichiers de configuration.

Dans ce cas vous allez devoir faire tourner CakePHP sans le mod rewrite. Vous aurez accès à toutes les fonctions mais vous aurez juste des adresses "moches"...

Si vous ne pouvez pas activer le mod rewrite, effacez simplement les fichiers .htaccess.

Il y en a 3 :

- /.htaccess

- /app/.htaccess
- /app/webroot/.htaccess

Ensuite allez dans le fichier /app/config/core.php, vers le début du fichier vous devez décommenter la ligne suivante (retirez le // au début) :

Code : PHP - [Sélectionner](#)

```
<?php
//Configure::write('App.baseUrl', env('SCRIPT_NAME'));
Configure::write('App.baseUrl', env('SCRIPT_NAME'));
?>
```

Ça va permettre à CakePHP de retrouver les fichiers sans le rewrite.
Tout marchera parfaitement mais vos URL n'auront simplement pas la même forme.



Par la suite du tutoriel je supposerai que vous avez le mod rewrite activé, ceux qui n'ont pas pu l'activer devront simplement ne pas faire attention aux URL que je vais indiquer.

Premier affichage

Maintenant que tout est prêt, accédez à votre page !
Allez directement à l'URL de votre page.
Pour moi c'est : <http://localhost/cakephp/>.
Et là, oh miracle !



Argh chez moi ça ne marche pas...

Si ça ne fonctionne pas regardez plus bas, j'ai écrit une partie pour vous aider à régler les problèmes.

Normalement, à partir de là les carrés verts indiquent que tout est OK. Donc mon répertoire tmp est "writable" ça veut dire que CakePHP a bien les droits en écriture, c'est normal vu que je suis sous Windows qui n'est pas très chiant à ce niveau-là. Si vous êtes sous Unix et que CakePHP ne peut pas écrire dans ce répertoire, changez les droits pour autoriser l'écriture à tout le monde.

Code : Console - [Sélectionner](#)

```
chmod 777
```

L'idéal étant plutôt de donner les droits d'écriture seulement à PHP, en donnant les droits au seul groupe auquel il appartient. Mais ça nous ferait nous éloigner de notre sujet, un simple `chmod 777` suffira pour l'instant.

Le 2^e carré vert indique que le moteur de cache fonctionne correctement.

Les 3^e et 4^e carrés verts indiquent que vous avez bien réglé la base de données et que CakePHP arrive à s'y connecter sans problème.



Hep, là il y a deux carrés jaunes tout en haut, tu croyais pas qu'on allait les louper hein ?

Oui oui, 🤔 bon il y a deux "Notices", ce sont des petites alertes qui ne sont pas très importantes. Il s'agit de la `Security.salt` et `Security.cipherSeed`, de simples valeurs à modifier comme suit dans le fichier `app/config/core.php` :

Code : PHP - [Sélectionner](#)

```
<?php
/**
 * A random string used in security hashing methods.
 */
Configure::write('Security.salt', 'DYhG93b0qyJfIxfS2guVoUubWwvniR2G0FgaC9mi');

/**
 * A random numeric string (digits only) used to encrypt/decrypt strings.
 */
Configure::write('Security.cipherSeed', '76859309657453542496749683645');

?>
```

Remplacez simplement les valeurs par défaut par n'importe quelles autres valeurs. Attention, vous n'avez le droit que de mettre des chiffres dans la `cipherSeed`. Et surtout, ne changez cette valeur qu'une seule fois. Si vous la modifiez à nouveau alors que des utilisateurs se sont inscrits ça va foutre un peu la merde. 😊



Quand CakePHP stocke les mots de passe il les hache, c'est-à-dire qu'il les transforme en une autre chaîne de caractères pour qu'on ne puisse pas retrouver le mot original. Seulement des petits malins se sont amusés à hacher tous les mots du dictionnaire et à faire un dictionnaire inversé pour retrouver le mot original ! Pour contrer cela, CakePHP concatène cette chaîne de caractères après le mot de passe et il hache le tout, comme ça le dictionnaire inversé n'est plus d'aucune aide.

Normalement si vous rechargez la page après avoir modifié cette valeur, tout est dans le vert, vous êtes prêts pour le chapitre suivant !

Un problème ?

Si vous n'arrivez pas à faire afficher la page vous avez très probablement un problème avec le mod `rewrite` du serveur Apache.

.htaccess pas pris en compte

Il est possible que votre fichier `.htaccess` ne soit pas pris en compte par votre serveur apache2.

Windows / WAMP

Normalement, WAMP est réglé par défaut en prenant en compte le `.htaccess`. Mais on va quand même vérifier 🤔

Si vous avez installé Cake dans le répertoire "www" de base allez éditer le fichier `httpd.conf` (il se trouve dans `C:\wamp\bin\apache\Apache2.2.11\conf`). Essayez de trouver le bloc suivant :

Code : Apache - [Sélectionner](#)

```

<Directory "c:/wamp/www/">
#
# Possible values for the Options directive are "None", "All",
# or any combination of:
# Indexes Includes FollowSymLinks SymLinksifOwnerMatch ExecCGI MultiViews
#
# Note that "MultiViews" must be named *explicitly* --- "Options All"
# doesn't give it to you.
#
# The Options directive is both complicated and important. Please see
# http://httpd.apache.org/docs/2.2/mod/core.html#options
# for more information.
#
Options Indexes FollowSymLinks

#
# AllowOverride controls what directives may be placed in .htaccess files.
# It can be "All", "None", or any combination of the keywords:
# Options FileInfo AuthConfig Limit
#
AllowOverride all

#
# Controls who can get stuff from this server.
#

# onlineoffline tag - don't remove
Order Deny,Allow
Deny from all
Allow from 127.0.0.1

</Directory>

```

Il n'est peut être pas exactement pas pareil chez vous, le plus important c'est qu'il commence par <Directory "c:/wamp/www/"> (le répertoire où se trouvent vos fichiers).

Assurez vous que la directive "AllowOverride" est bien à "All".

Si ce n'est pas le cas, enregistrez et redémarrez WAMP.

Si vous avez créé un Alias, il suffit d'aller dans le menu de WAMP (clic gauche sur l'icone de WAMP, clic sur "Apache", clic sur "Alias Directories", clic sur l'alias que vous avez créé, puis clic sur "Edit Alias").

Ici aussi, assurez vous que la directive "AllowOverride" est bien à "All", et redémarrer WAMP.

Si vous avez fait un virtual host, vous devez éditer "C:\wamp\bin\apache\Apache2.2.11\conf\extra\http-vhosts.conf". Mais si vous avez été capable d'activer les vhosts de WAMP, je considère que vous saurez vous débrouiller tout seul pour la suite 😊

Linux

Vous devez éditer le fichier suivant : /etc/apache2/sites-available/default.

Vérifiez que la directive "AllowOverride" est bien à "All" et pas à "None".

N'oubliez pas de dire à apache2 de charger les nouveaux réglages :

Code : Console - [Sélectionner](#)

```
/etc/init.d/apache2 reload
```

(avec un sudo, si vous n'êtes pas l'utilisateur "root")

Pas de CSS

Si la page s'affiche mais sans le CSS, c'est que le mod rewrite n'est pas du tout activé !!!

CakePHP: the rapid development php framework

Release Notes for CakePHP 1.2.0.7692 RC3.

[Read the release notes and get the latest version](#)

Notice (1024): Please change the value of 'Security.salt' in app/config/core.php to a salt value specific to your application [CORE\cake\libs\debugger.php, line 541]

Your tmp directory is writable.

The *FileEngine* is being used for caching. To change the config edit APP/config/core.php

Your database configuration file is present.

Cake is able to connect to the database.

Editing this Page

To change the content of this page, edit: APP/views/pages/home.ctp.

To change its layout, edit: APP/views/layouts/default.ctp.

You can also add some CSS styles for your pages at: APP/webroot/css.

Getting Started

[new CakePHP 1.2 Docs](#) [The 15 min Blog Tutorial](#)

More about Cake

CakePHP is a rapid development framework for PHP which uses commonly known design patterns like Active Record, Association Data Mapping, Front Controller and MVC.

Our primary goal is to provide a structured framework that enables PHP users at all levels to rapidly develop robust web applications, without any loss to flexibility.

- [Cake Software Foundation](#)
 - Promoting development related to CakePHP
- [The Show](#)
 - The Show is a weekly live internet radio broadcast where we discuss CakePHP-related topics and answer questions live via IRC, Skype, and telephone.
- [The Bakery](#)
 - Everything CakePHP
- [Book Store](#)
 - Recommended Software Books
- [CakeSchwag](#)
 - Getting your CakePHP apps. Ready-made to Go!

Relisez bien le tutoriel vous devez avoir effectué ces manipulations :

- décommenter la ligne dans le fichier httpd.conf ;
- redémarrer le serveur Apache.

Si ça ne marche toujours pas, vous avez le choix entre aller essayer de résoudre votre problème en posant des questions sur un forum, ou bien utiliser CakePHP sans le rewriting car votre problème est spécial. 😞

Internal Server Error

Si vous tombez sur une erreur 500 : "Internal Server Error", bonne nouvelle votre serveur Apache a probablement bien activé le module rewrite ! 😊

Quoi vous n'êtes pas contents ? 😞

Mais non ne paniquez pas, c'est juste une petite erreur de configuration.

En gros vous avez mis CakePHP dans un répertoire, alors qu'il s'attend à être présent sur la racine de votre site.



Hein ? Mais non moi j'ai fait un alias sur mon serveur, et j'ai seulement mis Cake PHP dans le répertoire root de cet alias, il est tout seul dans son site.

Oui mais bon en fait mauvaise nouvelle, certains serveur vous affichent ça comme étant un alias mais en fait il a juste fait un répertoire. 😞
C'est notamment le cas chez moi avec WAMP.

Il suffit simplement d'indiquer au mod rewrite qu'il doit récrire les URL selon le répertoire où il est installé. Vous allez devoir modifier les 3 fichiers .htaccess :

.htaccess (celui dans la racine du site)

Code : Apache - [Sélectionner](#)

```
<IfModule mod_rewrite.c>
RewriteEngine on
RewriteBase /cakephp
RewriteRule ^$ app/webroot/ [L]
RewriteRule (.*?) app/webroot/$1 [L]
</IfModule>
```

app/.htaccess

Code : Apache - [Sélectionner](#)

```
<IfModule mod_rewrite.c>
RewriteEngine on
RewriteBase /cakephp
RewriteRule ^$ webroot/ [L]
RewriteRule (.*?) webroot/$1 [L]
</IfModule>
```

app/webroot/.htaccess

Code : Apache - [Sélectionner](#)

```
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteBase /cakephp
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ index.php?url=$1 [QSA,L]
</IfModule>
```

Il suffit simplement d'ajouter une ligne juste après "RewriteEngine On".

Chez moi j'ai ajouté "RewriteBase /cakephp" parce que tout se trouve dans le répertoire que j'ai appelé « CakePHP ». Ça se voit dans mon URL : quand je veux y accéder, je dois taper `http://localhost/cakephp`.

Si vous avez installé votre site dans un sous répertoire, indiquez simplement l'adresse sur laquelle doit se baser l'URL rewrite...

Autre problème ?

Si vous avez un problème d'un autre genre, n'hésitez pas à m'envoyer un message ou à aller le signaler sur le forum, j'ajouterai un autre paragraphe à cette sous partie si d'autres soucis viennent déranger les zéros !

Partie 2 : Notre première application

Allez, on va attaquer avec un exemple parlant.

On va réaliser une application qui va gérer une bibliothèque !

Tout ce qu'il y a de plus simple. Il y aura des exemplaires de livres, qui seront regroupés par genre littéraire. Les abonnés pourront les emprunter et les ramener. Pour l'instant c'est déjà pas mal.

Avant de commencer je vais vous dire que ce qu'on va faire là, Cake PHP peut le faire automatiquement (j'irai même jusqu'à dire « automagiquement »). Mais je trouve que c'est beaucoup plus simple de comprendre le système en créant soi-même les fichiers, surtout que ce n'est pas très dur. 😊

Afficher la liste des livres

On va commencer par le plus simple.

Dans notre bibliothèque, on a des ordinateurs qui peuvent afficher la liste des livres qui sont présents.

Les abonnés, plutôt que de passer deux heures à parcourir les rayons, pourront simplement aller sur le PC voir si leur livre est là ou non.

L'ADN : la table

On va donc commencer par faire notre table "livres".

Exécutez cette requête SQL sur votre serveur MySQL (avec PhpMyAdmin par exemple, vous avez un onglet "SQL", vous n'avez qu'à coller la requête dedans) :

Code : SQL - [Sélectionner](#)

```
CREATE TABLE `livres` (  
  `id` INT NOT NULL AUTO_INCREMENT ,  
  `genre_id` INT NOT NULL ,  
  `auteur_id` INT NOT NULL ,  
  `title` VARCHAR( 255 ) NOT NULL ,  
  `date_parution` DATE NOT NULL ,  
  PRIMARY KEY ( `id` )  
 ) ENGINE = InnoDB CHARACTER SET utf8 COLLATE utf8_general_ci
```

Mais là je vais dire ST0000000000000P !



Stop ? Pourquoi ? Il n'y a rien à comprendre c'est une requête SQL de base alors que les zéros sont des cadors en SQL...

C'est pas sur la requête en elle même que je veux attirer votre attention, mais sur le nom des champs.

Je vous ai bien dit que travailler avec CakePHP vous impose de suivre des règles très strictes, et bien ça commence dès maintenant !

Déjà le nom de votre table est forcément au **PLURIEL** et en **MINUSCULES**, les mots sont séparés par des **UNDERScores** (barres de soulignement).

Des exemples de noms de table : *livres*, *abonnes*, *auteurs*, *grands_textes*, *profils_utilisateurs*...

Elle a pour identifiant (l'id, la PRIMARY KEY) un champ qui s'appelle forcément **id**.

Ensuite les clefs étrangères sont forcément nommées selon le format suivant : **<nom_table_singulier>_id**.

Dans notre exemple : *genre_id* qui correspondra au champ "id" de la table "*genres*" (toujours au pluriel). Et aussi *auteur_id* qui correspond au champ "id" de la table "*auteurs*" (encore au pluriel).



Attention je ne dis pas que vous ne pouvez pas appeler vos champs comme vous voulez, je dis simplement que si vous les appelez comme ça vous allez devoir configurer vos modèles pour leur dire à quoi correspond tel ou tel champ. Autant suivre les règles, on gagne beaucoup beaucoup de temps. On va supposer par la suite que vous avez suivi les règles, vous allez voir à quel point c'est facile d'afficher une page.

Ensuite on va ajouter un livre dans notre table :

Code : SQL - [Sélectionner](#)

```
INSERT INTO `livres` (  
  `id`,  
  `genre_id`,  
  `auteur_id`,  
  `title`,  
  `date_parution`  
)  
VALUES (  
  NULL , '1', '1', 'Moby Dick', '1951-10-18'  
);
```

Et voilà. Pour l'auteur et le genre j'ai mis l'ID 1 vu qu'on n'a pas encore fait ces tables. Normalement pour bien faire tout proprement on aurait du commencer par faire toutes les tables, mais ici le but c'est de vous apprendre à utiliser le *framework*.

Le squelette : le modèle

Vous avez bien suivi les règles de noms de CakePHP ? Alors là, vous allez voir à quel point la conception du modèle est difficile, attention, seulement les ingénieurs pourront comprendre cette partie :

Code : PHP - [Sélectionner](#)

```
<?php  
class Livre extends AppModel {  
    var $name="Livre";  
}  
?>
```

Et voilà sauvegardez ça dans : /app/models/livre.php et *basta*. 😊

On a créé une classe du même nom que la table (table "livres" => modèle "Livre" => fichier "/app/models/livre.php").

Attention, convention d'appellation oblige, vous devez mettre votre nom de modèle avec une **Majuscule**, et au **Singulier**. Vous devez l'enregistrer sans la majuscule car les noms de fichiers n'ont jamais de majuscules.

La seule chose à savoir c'est que votre classe hérite de la classe AppModel qui contient déjà toutes les fonctions de base. Vu qu'on ne lui spécifie rien du tout, elle a automatiquement utilisé les noms par défaut qui sont déjà réglés dans AppModel. C'est pour ça que je vous embête avec mes majuscules et mes noms au pluriel/singulier. Là on récupère le temps perdu car on n'a rien à écrire !

La ligne `$name="Livre"` est utilisée seulement pour la compatibilité PHP4, si vous tournez avec PHP5 vous pouvez la retirer ce qui nous donne :

Code : PHP - [Sélectionner](#)

```
<?php
// Pour PHP 5 seulement !
class Livre extends AppModel {

}
?>
```

On peut difficilement faire plus simple. 😊

Les muscles : le contrôleur

On va ensuite faire le contrôleur : chaque contrôleur est généralement associé à un modèle. Ce n'est pas obligatoire mais c'est très très souvent le cas. Dans notre cas on a une table "livres", un modèle "Livre" et on va avoir un contrôleur : "LivresController". Aïe encore une convention d'appellation : majuscules, pluriel et on ajoute le nom "Controller" à la suite...

Code : PHP - [Sélectionner](#)

```
<?php
class LivresController extends AppController {
    var $name="Livres";
}
?>
```

Le code ressemble pas mal au modèle, on hérite juste de AppController à la place de AppModel et on a toujours notre "var \$name" pour les vieux dinosaures qui se traînent en PHP4.

Enregistrez ce fichier dans : /app/controllers/livres_controller.php .

Donc pour les noms on va récapituler : table "livres" => modèle "Livre" dans "livre.php" => contrôleur "LivresController" dans "livres_controller.php".

Une fois habitué on trouve ça logique, je vous assure. 😊 Enfin, je vais quand même vous faire une annexe que vous pourrez imprimer comme « antisèche »...

Pour accéder à notre fameux "Livres" (je vais arrêter de dire "LivresController" parce que maintenant vous êtes des professionnels de CakePHP vous savez très bien que quand je dis "Livres" c'est forcément le contrôleur et quand je dis "Livre" c'est le modèle), il faut utiliser l'URL suivante : <http://localhost/cakephp/livres/>

Essayez et voilà ce que vous devriez obtenir :

CakePHP: the rapid development php framework

Missing Method in LivresController

Error: The action `index` is not defined in controller `LivresController`

Error: Create `LivresController::index()` in file: `app\controllers\livres_controller.php`.

```
<?php
class LivresController extends AppController {

    var $name = 'Livres';

    function index() {

    }

}
?>
```

Notice: If you want to customize this error message, create `app\views\errors\missing_action.ctp`.

CakePHP: Query took 75 ms						
Nr	Query	Error	Affected	Num. rows	Took (ms)	
1	DESCRIBE `livres`		5	5		

Et paf une erreur dans la tête...

Non vous n'avez rien fait de mal, c'est juste moi qui suis sadique. 😊

Je voulais vous montrer ce qui se passe quand on fait quelque chose de mal. Là en fait on a créé notre contrôleur, mais on ne lui a pas expliqué quoi faire ! Chaque fonction de notre contrôleur représentera une **action**. Pour effacer un livre ça sera par exemple : `http://localhost/livres/effacer` (ça appelle la fonction `effacer()` du contrôleur "Livres") ; pour afficher la liste d'un livre ça sera : `http://localhost/livres/liste` (idem, fonction `liste()` de "Livres")... Ici on n'a indiqué aucune action alors il va chercher la fonction `index()`.

Premier carré rouge : erreur l'action `index` is not defined (= gros fainéant tu n'as pas écrit la fonction `index()`).

Deuxième carré rouge : va la créer, et plus vite que ça ! Il vous aide quand même en vous disant dans quel fichier la créer.

Avec tous ces carrés rouges il n'a pas l'air de bonne humeur, moi je serais vous j'irais coder cette fonction assez rapidement. Allons-y de suite !

Code : PHP - Sélectionner

```
<?php
class LivresController extends AppController {
    function index() {
        $liste_des_livres = $this->Livres->find('all');
        $this->set('livres', $liste_des_livres);
    }
}
?>
```

J'ai viré le `var $name` parce que je suis en PHP5, je n'en ai pas besoin. 😊

On a écrit deux lignes ça suffit, c'est déjà assez !

En gros ça se passe en 2 étapes :


1. on appelle le modèle "Livres" et on lui dit de trouver tous les livres (`find('all')`, en anglais "Trouve tous") et on met tout ça dans la variable `$liste_des_livres` ;
2. on utilise la fonction "set()" du contrôleur. Cette fonction sert à envoyer des informations à la vue. Donc on envoie la liste des livres et on lui donne un petit nom "livres".




Notez que comme on a bien respecté les conventions d'appellation il sait que le contrôleur "Livres" est associé au modèle "Livres" et on peut l'utiliser. Si on avait voulu utiliser un autre modèle ou si on avait mal appelé nos fichiers/classes, on aurait été obligé de spécifier la liste des modèles à utiliser.











C'est tout ? Mais tu nous avais dit que le contrôleur appelait aussi la vue une fois qu'il a terminé !

Ah je vois qu'il y en a qui suivent à fond ! Effectivement le contrôleur appelle la vue quand il a fini. Mais c'est fait « automatiquement ».  Si on ne précise rien la vue est appelée, du moment qu'elle a le nom qu'il faut...


Enregistrez tout ça, on va créer la vue.

Vous pouvez toujours essayer d'appeler la fonction index du contrôleur "Livres" avec l'URL <http://localhost/livres/index> ou <http://localhost/livres> mais vous allez vous prendre des beaux carrés rouges dans la face, parce qu'on n'a pas encore fait la vue. 

La peau : la vue

-  **views**
 -  **elements**
 -  **errors**
 -  **helpers**
 -  **layouts**
 -  **livres**
 -  **index.ctp**
 -  **scaffolds**
- On va créer la vue qui sera appelée automatiquement par l'action "index" du contrôleur "Livres". Vous vous doutez bien qu'elle va avoir un nom et un emplacement très précis.
- Déjà on va se placer dans le répertoire "views", on va créer un dossier "livres", sans majuscule, et on va créer un fichier "index.ctp" (le nom de l'action).
- Pour chaque action du contrôleur on ira créer un fichier .ctp associé dans ce répertoire.
- Si un autre contrôleur a une action du même nom, ce n'est pas important puisque celle ci sera dans un autre répertoire (portant le nom de l'autre contrôleur, en minuscules).

On va maintenant éditer ce fichier .ctp comme si c'était un fichier XHTML / PHP.

Tout d'abord on sait qu'on a reçu des informations du contrôleur, mais sous quelle forme ? Allez, on va faire comme si je ne savais pas. 

Tout ce qu'on sait c'est que notre contrôleur nous l'a envoyé sous le nom "livres". Aller hop on va essayer ce truc :

Code : PHP - [Sélectionner](#)

```
<pre><?php print_r($livres); ?></pre>
```

Bon je triche, je sais d'avance que c'est un array, mais vous allez me pardonner ce petit raccourci !
Enregistrez ça dans /app/views/livres/index.ctp et allez sur l'adresse : <http://localhost/cakephp/livres>.

Vous devriez obtenir quelque chose comme ça :

Code : Autre - [Sélectionner](#)

```
Array
(
    [0] => Array
        (
            [Livres] => Array
                (
                    [id] => 1
                    [genre_id] => 1
                    [auteur_id] => 1
                    [title] => Moby Dick
                    [date_parution] => 1951-10-18
                )
            )
        )
)
```

On a un livre, donc il porte le numéro 0 (les arrays sont toujours numérotés à partir de zéro ne l'oubliez pas !). Dedans il y a un autre array "Livres" (le nom du modèle) qui contient tous les champs.

À partir de là vous devriez pouvoir me faire un beau tableau... Je vérifie vos copies dans 10 minutes !

Voilà ce que j'obtiens. Utilisez des <th> pour les cases du tableau qui représentent des titres et non des données, vous n'avez j'espère pas oublié tout ce que tonton M@teo21 vous a appris sur la sémantique XHTML j'espère, hein ?

Code : PHP - [Sélectionner](#)

```

<table>
  <tr>
    <th>Titre</th>
    <th>Date de Parution</th>
    <th>Auteur</th>
    <th>Genre littéraire</th>
  </tr>
  <?php foreach($livres as $livre): ?>
    <tr>
      <td><?php echo $livre['Livre']['title']; ?></td>
      <td><?php echo $livre['Livre']['date_parution']; ?></td>
      <td><?php echo $livre['Livre']['auteur_id']; ?></td>
      <td><?php echo $livre['Livre']['genre_id']; ?></td>
    </tr>
  <?php endforeach; ?>
</table>

```



Euh c'est quoi ce "endforeach;" ? Et tu n'as pas oublié les accolades du foreach ?

Non tout va bien. 😊

Je sais que la notation classique c'est :

Code : PHP - [Sélectionner](#)

```

<?php foreach($livres as $livre) { ?>
  <tr>
    <td><?php echo $livre['Livre']['title']; ?></td>
    <td><?php echo $livre['Livre']['date_parution']; ?></td>
    <td><?php echo $livre['Livre']['auteur_id']; ?></td>
    <td><?php echo $livre['Livre']['genre_id']; ?></td>
  </tr>
<?php } ?>

```

Mais je peux vous dire que quand on a plusieurs boucles et "if" imbriquées on se retrouve avec des trucs de ce genre :

Code : PHP - [Sélectionner](#)

```

<?php } ?>
  <?php } ?>
<?php } ?>

```

voire même pire, des :

Code : PHP - [Sélectionner](#)

```

<?php }}} ?>

```

On peut aussi noter les blocs de cette façon :

Code : PHP - [Sélectionner](#)

```
<?php
if($a==$b) :
    //blabla
else:
    //bloblo
endif;

foreach($tableau as $case):
    //blabla
endforeach;
?>
```

C'est un peu barbant le "end" à la fin, mais dans notre cas quand on mélange du XHTML et du PHP c'est plus lisible !

Voilà, fin de notre parenthèse, si vous avez bien enregistré votre vue dans /app/views/livres/index.ctp, vous pouvez aller voir le résultat !

<http://localhost/livres/index>, ou plus simple :

<http://localhost/livres>, je vous rappelle que si on ne précise pas d'action, CakePHP ira chercher l'action "index" par défaut.

CakePHP: the rapid development php framework			
Titre	Date de Parution	Auteur	Genre littéraire
Moby Dick	1951-10-18	1	1

Voilà, il n'y a qu'un livre dans notre base de données alors on n'a qu'une seule ligne. Mais le résultat est là. On a réussi à afficher notre tableau.

Le design c'est celui par défaut, on va le modifier par la suite, pour l'instant on a juste quelque chose de fonctionnel...

Notez que CakePHP nous affiche d'office un petit mode *debug* assez pratique en dessous, qui fait apparaître les requêtes passées à la base.

Vous voyez que ce n'est pas très optimisé puisqu'il effectue deux requêtes alors qu'une seule aurait été suffisante. Ce n'est pas très important pour l'instant.



On va continuer notre exemple, maintenant on va vouloir ajouter et retirer des livres depuis notre application !

Ajouter, modifier et supprimer des livres

Pour l'affichage vous savez faire mais je suppose que ça ne vous a pas bluffé. Une simple requête SQL "SELECT" basique aurait pu en faire autant. Pour l'ajout et surtout pour la modification vous allez vous apercevoir qu'avoir un *framework* a de grands avantages.

Nous avons déjà le modèle et le contrôleur qui sont là. Pour faire des trucs avec nos livres nous allons simplement ajouter de nouvelles actions au contrôleur !

Ajouter un nouveau livre

Aller hop ! Maintenant on est rodé, si on veut ajouter un livre, on crée une action "add" au contrôleur Livres. C'est parti :

Code : PHP - Sélectionner

```
<?php
class LivresController extends AppController {
    function index() {
        $liste_des_livres = $this->Livre->find('all');
        $this->set('livres', $liste_des_livres);
    }

    function add() {

    }
}
?>
```

Bon, on va mettre quoi dedans ? On verra cela après, on n'en est pas encore là.

Le formulaire

Ce qui est important c'est que cette fonction va appeler la vue /app/views/livres/add.ctp. On va créer cette vue qui contiendra un formulaire de création :

Code : PHP - [Sélectionner](#)

```
<?php
echo $this->Form->create('Livre');
echo $this->Form->input('title');
echo $this->Form->input('date_parution');
echo $this->Form->input('auteur_id');
echo $this->Form->input('genre_id');
echo $this->Form->end('Ajouter');
?>
```

On va quand même prendre un peu de temps pour expliquer tout ça.

`$this->Form` c'est un Helper, les Helpers sont là pour aider à la mise en page. En l'occurrence, FormHelper nous aide à créer des formulaires.

Le principe est simple :

- `$this->Form->create()` va créer le formulaire. On lui donne le nom du modèle : Livre. Comme ça il sait à quoi chaque champ correspond ;
- `$this->Form->input('title')` va créer un champ pour le titre ;
- `$this->Form->input('date_parution')` va créer un champ pour la date de parution. Mais forcément, il fait ça intelligemment notre petit Helper, il sait que 'date_parution' c'est une DATE donc il va nous afficher un joli sélecteur de date ;
- les deux `$this->Form->input()` suivants vont afficher des zones de texte classiques puisqu'il ne sait pas (encore) à quoi correspondent les auteur_id et genre_id ;
- le `$this->Form->end()` ferme le formulaire, le texte en paramètre sera inséré dans le bouton submit.



Quand CakePHP fait un formulaire d'ajout, il va automatiquement l'envoyer vers l'action `add()`. Si vous n'avez pas appelé votre action `add()` vous allez devoir ajouter un paramètre à `$this->Form->create()` pour lui indiquer où envoyer le formulaire.

Voilà c'est fini, on n'a plus qu'à récupérer les informations du formulaire.

Ajouter le livre

On retourne dans notre contrôleur, on avait laissé une fonction vide. On va lui dire maintenant quoi faire. On a deux cas possibles :

- le visiteur arrive sur cette page en ayant cliqué sur un lien, il doit alors voir le formulaire ;
- le visiteur a fini de remplir le formulaire, le livre doit être créé.

Il suffit de regarder si un formulaire a été envoyé, si c'est le cas on crée le livre. Les données du formulaire sont simplement envoyées dans la variable `$this->data` (parce qu'on a utilisé le Helper pour faire le formulaire). On va simplement vérifier si `$this->data` n'est pas vide, auquel cas on enregistre le nouveau livre, sinon on affiche la vue.

Code : PHP - [Sélectionner](#)

```
<?php
function add() {
    if (empty($this->data)==false) {
        $resultat = $this->Livre->save( $this->data );
        if ($resultat) {
            $this->flash(
                'Le livre vient d\'être ajouté',
                array('controller'=> 'livres', 'action'=>'index')
            );
        }
    }
}
?>
```

Pour enregistrer on utilise la méthode `save()` de notre modèle Livre. On avait déjà utilisé la méthode `find()` pour afficher les livres, là on utilise `save()`. On envoie simplement à cette fonction les données du formulaire, pas la peine de se prendre la tête. 😊 La fonction `save()` renvoie une valeur qui est égale à `false` si l'insertion n'a pas été effectuée. On stocke cette valeur dans `$resultat`.

Si l'insertion est réussie, on va afficher un petit message "Le livre vient d'être ajouté", avant d'être redirigé vers l'action "index" du contrôleur "Livres", on

est donc redirigé vers la liste des livres. `$this->flash()` est sympa car en ce moment on se trouve en mode "debug" donc `flash()` nous affiche le message et attend qu'on clique avant de continuer. Quand votre site sera fini, `flash` n'affichera rien et votre visiteur sera instantanément redirigé (il ne verra pas tous les messages de CakePHP).



On est d'office en mode "debug" dans CakePHP. C'est grâce à ça qu'on a tous les messages d'erreur bien expliqués, la liste des requêtes SQL et compagnie. C'est très pratique en développement mais quand le site est fini, on le désactive pour accélérer l'affichage des pages. On verra plus tard comment réduire le niveau du mode debug, ou carrément le désactiver.

Le détail de la fin

Il ne nous reste plus qu'une chose à faire : mettre le lien pour ajouter notre nouveau livre !

On va l'ajouter en bas de la liste des livres, dans `/app/views/livres/index.ctp` :

Code : PHP - [Sélectionner](#)

```
<table>
  <tr>
    <th>Titre</th>
    <th>Date de Parution</th>
    <th>Auteur</th>
    <th>Genre littéraire</th>
  </tr>
  <?php foreach($livres as $livre): ?>
    <tr>
      <td><?php echo $livre['Livre']['title']; ?></td>
      <td><?php echo $livre['Livre']['date_parution']; ?></td>
      <td><?php echo $livre['Livre']['auteur_id']; ?></td>
      <td><?php echo $livre['Livre']['genre_id']; ?></td>
    </tr>
  <?php endforeach; ?>
</table>

<?php
echo $this->Html->link(
  'Ajouter un livre',
  array('controller'=>'livres', 'action'=>'add')
);
?>
```

Ici encore on a utilisé un Helper, mais ce coup-ci ce n'est pas le FormHelper, mais le HtmlHelper.

`$this->Html->link()` permet de faire un lien. Donc on a fait un lien vers l'action 'add' du contrôleur 'livres'. Et le texte du lien sera "Ajouter un livre".



Mais on n'aurait pas pu simplement faire un `Ajouter ???`

Bien sûr que si, mais la bonne habitude pour tous les liens et formulaires, c'est d'utiliser les Helpers. Il y a une raison à cela : la redirection ! Vous savez que CakePHP gère l'url rewriting. Donc si jamais tout à la fin on décide que le lien `http://localhost/cakephp/ajouter` devra pointer sur la page d'ajout, vous allez devoir tout rechanger dans votre site. Là au moins on indique le contrôleur et l'action, même si on décide par la suite de changer la ré-écriture des URL, le Helper mettra le lien qu'il faut.

En avant, maestro !

Allez, maintenant on va tester ce que ça donne, allons-y : `http://localhost/cakephp/livres`.

On a notre petit lien et on va ajouter quelques livres pour avoir de quoi faire des tests pour les prochaines parties.

Faites vous plaisir !

CakePHP: the rapid development php framework			
Titre	Date de Parution	Auteur	Genre littéraire
Moby Dick	1951-10-18	1	1
The Da Vinci Code	2003-03-18	2	2
Deception Point	2001-11-15	2	2

Ajouter un livre

Effacer un Livre

Attention ça va être court. Il y a quelques différences avec ce qu'on a déjà fait :

- notre action va recevoir un paramètre, l'id du livre à effacer ;
- notre action ne va renvoyer aucune vue. On efface et on redirige.

Pour envoyer un paramètre à une action il suffit de l'ajouter dans l'url : <http://localhost/cakephp/livres/delete/2/>. C'est simple. 😊

Voyons maintenant comment ça se passe dans le contrôleur :

Code : PHP - [Sélectionner](#)

```
<?php
function delete($id) {
    $this->Livre->delete($id);
    $this->flash(
        'Le livre id=' . $id . ' a été effacé.',
        array('controller'=> 'livres', 'action'=>'index')
    );
}
?>
```

Le paramètre envoyé est récupéré. On utilise la fonction delete() du contrôleur pour effacer. C'est très simple.

Le lien de la fin

Il ne nous reste plus qu'à faire des petits liens pour effacer. Retour sur la vue livres/index.ctp :

Code : PHP - [Sélectionner](#)

```
<table>
<tr>
    <th>Titre</th>
    <th>Date de Parution</th>
    <th>Auteur</th>
    <th>Genre littéraire</th>
    <th>Action</th>
</tr>
<?php foreach($livres as $livre): ?>
    <tr>
        <td><?php echo $livre['Livre']['title']; ?></td>
        <td><?php echo $livre['Livre']['date_parution']; ?></td>
        <td><?php echo $livre['Livre']['auteur_id']; ?></td>
        <td><?php echo $livre['Livre']['genre_id']; ?></td>
        <td>
            <?php echo $this->Html->link(
                $this->Html->image('icone_effacer.png'),
                array('controller'=>'livres', 'action'=>'delete', $livre['Livre']
                    ['id']),
                array('escape' => false)
            ); ?>
        </td>
    </tr>
<?php endforeach; ?>
</table>
```

Ici j'ai utilisé le HtmlHelper pour afficher l'image. Notez que pour envoyer en paramètre l'ID du livre j'ai simplement ajouté cet ID dans l'array du lien sans aucune clef. Si j'avais voulu envoyer plusieurs paramètres je n'aurais eu qu'à les ajouter comme ça à la suite.

Pour l'image vous pourrez en trouver une ici : <http://www.iconfinder.com/search/?q=delete>. Placez votre image dans /app/webroot /img/icone_effacer.png.

Voilà le résultat :

CakePHP: the rapid development php framework				
Titre	Date de Parution	Auteur	Genre littéraire	Action
Moby Dick	1951-10-18	1	1	✖
The Da Vinci Code	2003-03-18	2	2	✖
Deception Point	2001-11-15	2	2	✖
Livre à effacer	2008-11-15	23	11111	✖

[Ajouter un livre](#)

Éditer un livre

Ici rien d'exceptionnel, on va se baser sur l'ajout pour faire la modification.

Contrôleur

Code : PHP - [Sélectionner](#)

```
<?php
function edit($id) {
    $this->Livre->id = $id;
    if (empty($this->data)) {
        $this->data = $this->Livre->read();
    } else {
        if ($this->Livre->save($this->data)) {
            $this->flash(
                'Le livre id=' . $id . ' a été modifié',
                array('controller' => 'livres', 'action' => 'index')
            );
        }
    }
}
?>
```

1. On récupère l'id et on indique au modèle qu'on travaille avec le livre id=\$id.
2. Si \$this->data est vide alors le formulaire n'a pas été envoyé, ça veut dire qu'on va afficher le formulaire, donc on demande au modèle de charger les informations depuis la base de données, avec la fonction read(). On remplit alors \$this->data avec ces valeurs.
3. Sinon c'est qu'on a rempli le formulaire, alors on enregistre avec save() et on redirige vers la liste des livres.



La méthode save() du modèle sert pour l'insertion et pour la mise à jour. Ici notre \$this->data contiendra l'ID du livre à mettre à jour, c'est ce qui indique à CakePHP qu'il doit faire une mise à jour, et non une insertion.

Notez qu'à la fin, \$this->data est forcément renseigné. Soit on a envoyé une \$id donc il est rempli par la fonction read() du Modèle, soit l'enregistrement a échoué. Dans le deuxième cas, \$this->data contient toujours les valeurs que l'utilisateur a entrées.

La vue

/app/views/livres/edit.ctp :

Code : PHP - [Sélectionner](#)

```
<?php
echo $this->Form->create('Livre');
echo $this->Form->input('title');
echo $this->Form->input('date_parution');
echo $this->Form->input('auteur_id');
echo $this->Form->input('genre_id');
echo $this->Form->input('id', array('type' => 'hidden'));
echo $this->Form->end('Editer');
?>
```

Ici on a simplement ajouté un nouveau champ qui contient l'ID, en indiquant que ce champ est "hidden" on s'assure qu'il n'apparaîtra pas. On a également

modifié le label du bouton *submit*, ainsi que l'action appelée par le formulaire.

Ce que vous ne savez pas, c'est que le FormHelper récupère les valeurs de `$this->data` pour pré-remplir le formulaire. Donc il n'y a rien d'autre à faire. 🤖



Tu nous avais pas dis que CakePHP envoyait les données du formulaire à l'action `add()` ? Si on laisse comme ça on va ajouter un nouveau et pas éditer !

Mais non pas de panique, tout a été intelligemment pensé, si jamais il y a un champ "id" dans le formulaire (ce qui est le cas), alors CakePHP sait que c'est une édition, pas un ajout ! Il va alors envoyer les données du formulaire vers l'action `edit()`. C'est « automagique ». 🧙

Le lien

Dans la vue `index.ctp`, faites un lien à côté de l'icône effacer, là c'est simplement du copier/coller je vous laisse faire comme des grands.

Ensuite vous n'avez plus qu'à tester. 😊

Voilà, maintenant on a les 4 fonctions de base pour ajouter/retirer des livres.

TP : les auteurs

Premier TP, il est assez simple : vous allez faire pareil, tous seuls comme des grands, pour les auteurs.

Enoncé

Eh bien dans l'énoncé je ne sais pas trop quoi mettre, vous allez simplement refaire la même chose, mais avec les auteurs.

Je vais juste vous donner les informations que je souhaite que vous preniez en compte :

- un nom (ça comprends le prénom et le nom) ;
- une date de naissance ;
- une nationalité (un champ texte) ;
- une description (un champ TEXT).

N'oubliez pas qu'il faut aussi un champ `id`. 😊

C'est parti !

Correction

Table SQL

Code : SQL - [Sélectionner](#)

```
CREATE TABLE `auteurs` (  
  `id` INT NOT NULL AUTO_INCREMENT ,  
  `name` VARCHAR( 255 ) NOT NULL ,  
  `date_naissance` DATE NOT NULL ,  
  `nationalite` VARCHAR( 255 ) NOT NULL ,  
  `description` TEXT NOT NULL ,  
  PRIMARY KEY ( `id` )  
 ) ENGINE = InnoDB CHARACTER SET utf8 COLLATE utf8_general_ci
```



Mettez bien "name" dans le champ du nom, ça nous sera utile par la suite. Encore des règles de noms à suivre. 🤖

Modèle

/app/models/auteur.php :

Code : PHP - [Sélectionner](#)

```
<?php
class Auteur extends AppModel {
}
?>
```

Si vous vous êtes trompé sur celui-là, vous irez faire un stage dans la cave de M@teo21. 😊

Contrôleur

/app/controllers/auteurs_controller.php :

Code : PHP - [Sélectionner](#)

```
<?php
class AuteursController extends AppController {
    function index() {
        $this->set('auteurs', $this->Auteur->find('all'));
    }

    function add() {
        if (empty($this->data)==false) {
            if ($this->Auteur->save( $this->data )) {
                $this->flash('L\'auteur vient d\'être
ajouté', array('controller'=>'auteurs', 'action'=>'index'));
            }
        }
    }

    function edit($id) {
        $this->Auteur->id = $id;
        if (empty($this->data)) {
            $this->data = $this->Auteur->read();
        } else {
            if ($this->Auteur->save( $this->data )) {
                $this->flash('L\'auteur id='.$id.' a été
modifié', array('controller'=>'auteurs', 'action'=>'index'));
            }
        }
    }

    function delete($id) {
        $this->Auteur->delete($id);
        $this->flash('L\'auteur id='.$id.' a été
effacé.', array('controller'=>'auteurs', 'action'=>'index'));
    }
}
?>
```

Hop, ça m'a pris 30 secondes à faire, avec deux Find/Replace de mon éditeur. 😊

Les vues

/app/views/auteurs/add.ctp :

Code : PHP - [Sélectionner](#)

```
<?php
echo $this->Form->create('Auteur');
echo $this->Form->input('name');
echo $this->Form->input('date_naissance');
echo $this->Form->input('nationalite');
echo $this->Form->input('description');
echo $this->Form->end('Ajouter');
?>
```

/app/views/auteurs/index.ctp :

Code : PHP - [Sélectionner](#)

```

<table>
  <tr>
    <th>Nom</th>
    <th>Date de Naissance</th>
    <th>Nationalité</th>
    <th>Action</th>
  </tr>
  <?php foreach($auteurs as $auteur): ?>
    <tr>
      <td><?php echo $auteur['Auteur']['name']; ?></td>
      <td><?php echo $auteur['Auteur']['date_naissance']; ?></td>
      <td><?php echo $auteur['Auteur']['nationalite']; ?></td>
      <td>
        <?php echo $this->Html->link(
          $this->Html->image('icone_effacer.png'),
          array('controller'=>'auteurs', 'action'=>'delete', $auteur['Auteur']
            ['id']),
          array('escape' => false)
        ); ?>
        <?php echo $this->Html->link(
          $this->Html->image('icone_editer.png'),
          array('controller'=>'auteurs', 'action'=>'edit', $auteur['Auteur']
            ['id']),
          array('escape' => false)
        ); ?>
      </td>
    </tr>
  <?php endforeach; ?>
</table>

<?php echo $this->Html->link('Ajouter un Auteur', array('controller'=>'auteurs', 'action'=>'add')); ?>

```

/app/views/auteurs/edit.ctp :

Code : PHP - [Sélectionner](#)

```

<?php
echo $this->Form->create('Auteur');
echo $this->Form->input('name');
echo $this->Form->input('date_naissance');
echo $this->Form->input('nationalite');
echo $this->Form->input('description');
echo $this->Form->input('id', array('type'=>'hidden'));
echo $this->Form->end('Editer');
?>

```

Le petit plus

Allez, je vais vous récompenser d'avoir fait tout ce travail en vous montrant un petit truc assez sympa et rapide à faire !

On va modifier le modèle Auteur.

/app/models/auteur.php :

Code : PHP - [Sélectionner](#)

```
<?php
class Auteur extends AppModel {
    var $validate = array(
        'name' => array(
            'rule' => array('minLength', 3),
            'required' => true,
            'allowEmpty' => false,
            'message' => 'Un nom doit au moins faire 3 lettres'
        ),
        'description' => array(
            'rule' => array('minLength', 10),
            'message' => 'Veuillez remplir une description un peu plus
longue.'
        ),
        'nationalite' => array(
            'rule' => 'alphaNumeric',
            'message' => 'Ce n\'est pas une nationalité.'
        ),
        'date_naissance' => 'date'
    );
}
```

En lisant vous devriez à peu près comprendre à quoi ça sert. 😊

Essayez maintenant d'ajouter un nouvel auteur, en laissant tous les champs vides.

Voilà une fonctionnalité très sympathique du *framework*. Vous êtes sûrs, même si vous faites des insertions à plusieurs endroits de votre application, que le modèle vérifiera que les données respectent bien ces règles.

Plus tard j'écirai une annexe pour expliquer le système de validation des données, en attendant vous pouvez en apprendre plus sur le site officiel :

<http://book.cakephp.org/view/1143/Data-Validation> (en anglais).

Voilà c'est fini.

Je ne vous cache pas que ce TP était une façon cachée de mettre à notre disposition une deuxième table afin d'attaquer la suite du tutoriel. 😊

Relier les modèles

D'un coté on a les auteurs, de l'autre on a leur livres... il ne reste plus qu'à relier les deux. On a déjà prévu un champ "auteur_id" dans notre table "livres". Si on travaillait sans *framework* on aurait été obligé de modifier toutes les pages pour ajouter une jointure à la requête SQL. Avec CakePHP il suffit juste d'ajouter une valeur.

D'abord on va faire un peu de théorie pour vous expliquer les différents type d'association possibles, puis on va créer notre association entre les modèles "Livre" et "Auteur".

Les 4 types d'association

Quand vous associez un modèle à un autre vous devez spécifier de quel type est cette association.

Il y en a quatre :

- hasOne ;
- hasMany ;
- belongsTo ;
- hasAndBelongsToMany.

hasOne

"hasOne" = "possède un".

Exemple : « Utilisateur hasOne Profil ».

Pour qu'une association hasOne fonctionne il faut que l'autre modèle possède un champ <nom_modele>_id.

Dans notre exemple : « Utilisateur hasOne Profil », il faut un champ "utilisateur_id" dans la table "profils".

hasMany

"hasMany" = "possède plusieurs".

On va prendre un exemple de notre application test : « Auteur hasMany Livre ».

Il faut que la table "livres" possède un champ "auteur_id" (ça tombe bien, c'est justement le cas 😊).

belongsTo

"belongsTo" = "appartient à".

C'est le complément naturel de "hasOne" et "hasMany".

Si « Auteur **hasMany** Livre », alors « Livre **belongsTo** Auteur ». C'est logique : un auteur possède des livres, alors le livre appartient à un auteur.

Pour que **belongsTo** fonctionne, il faut que dans votre table il y ait un champ autremodèle_id.

« Livre **belongsTo** Auteur », il faut un champ "auteur_id" dans la table "livres" (exactement comme pour hasMany).

hasAndBelongsToMany

"hasAndBelongsToMany" = possède plusieurs et appartient à plusieurs.

hasAndBelongsToMany, affectueusement appelé HABTM, permet des liaisons plus complexes que vous utiliserez surtout pour gérer des catégories ou des tags.

Par exemple « Catégories **HABTM** produits ».

Dans une catégorie il peut y avoir plusieurs produits.

Un produit peut appartenir à plusieurs catégories différentes.

HABTM ne peut fonctionner que grâce à une table intermédiaire, qu'on appelle « Table de Lien », *Join table*. Conventions obligent, vous allez lui donner un nom très précis, composé des noms des deux tables, par ordre alphabétique, séparés par un *underscore*.

Dans notre exemple ça sera la table *categories_produits*. Cette table contiendra deux champs : *category_id* et *produit_id*.

Récapitulatif

Les associations vont par deux. Vous ne pouvez avoir que ces trois couples de liaisons.

« Model_1 **hasOne** Model_2 ».

« Model_2 **belongsTo** Model_1 ».

« Model_1 **hasMany** Model_2 ».

« Model_2 **belongsTo** Model_1 ».

« Model_1 **HABTM** Model_2 ».

« Model_2 **HABTM** Model_1 ».

À l'affichage

Voilà, maintenant que vous avez tout compris, on va pouvoir s'amuser. On a un modèle Livre et un modèle Auteur. Si vous avez bien suivi jusqu'ici on a : « Auteur **hasMany** Livre », et « Livre **belongsTo** Auteur ».

Il suffit maintenant de simplement l'indiquer dans les modèles :

/app/models/livre.php :

Code : PHP - [Sélectionner](#)

```
<?php
class Livre extends AppModel {
    var $belongsTo = 'Auteur';
}
?>
```

/app/models/auteur.php :

Code : PHP - [Sélectionner](#)

```
<?php
class Auteur extends AppModel {
    var $hasMany = 'Livre';
    var $validate = .....
}
?>
```

C'est tout. À partir de maintenant Auteur et Livre sont reliés. On va alors tester ça. Dans la liste des livres, on va essayer d'afficher directement le nom de l'auteur plutôt que l'id...

/app/views/livres/index.ctp :

Code : PHP - [Sélectionner](#)

```
...
<td><?php echo $livre['Livre']['title']; ?></td>
<td><?php echo $livre['Livre']['date_parution']; ?></td>
<td><?php echo $livre['Auteur']['name']; ?></td>
<td><?php echo $livre['Livre']['genre_id']; ?></td>
...
```

Si vous avez bien fait correspondre les ids des auteurs et des livres, vous voyez maintenant les noms s'afficher.

À l'édition

Ça serait bien que pendant l'édition on puisse aussi sélectionner notre auteur directement plutôt qu'en entrant son id.

Mais c'est à présent possible !

Il faut juste lui envoyer la liste de tous les auteurs au moment où on affiche le formulaire.

Ajoutez simplement :

Code : PHP - [Sélectionner](#)

```
<?php
$this->set('auteurs', $this->Livre->Auteur->find('list'));
?>
```

Juste à la fin des action edit() et add().

Et voilà maintenant on a une jolie liste déroulante. Essayez d'ajouter des auteurs : ils apparaîtront automatiquement dans la liste. C'est « automatique ».



Auteur

Herman Melville
 Dan Brown
 Victor Hugo
 Balzac
 William Shakespeare



Ah oui mais là c'est même carrément trop magique ! 🤖 Comment le modèle sait-il qu'il faut afficher le nom dans la boîte de sélection ? Il aurait très bien pu afficher la date de naissance ou la nationalité !

C'est parce qu'on a bien respecté les règles. Quand il doit afficher sa liste, CakePHP va rechercher un champ "name" ou "title". Vu que notre auteur a un "name" tout va bien. Sinon CakePHP aurait affiché juste l'id des auteurs et on aurait été obligé de rajouter une ligne pour lui indiquer quel champ afficher.

C'est fini pour cette première partie.

La prochaine partie ne va rien vous apprendre.

On va juste refaire la même chose, mais beaucoup plus vite !

Partie 3 : Automatisation

À partir de là, on va commencer à utiliser les outils de génération de code de Cake PHP.

On va ainsi pouvoir créer des fichiers complets en tapant presque rien.

Encore plus rapide avec Scaffold !

Allez, maintenant vous allez rire. 😊

Enfin j'espère que vous n'allez pas m'en vouloir surtout. Je vais vous dire que ce qu'on a fait ne sert à rien. 🙄

Enfin ça ne sert pas à rien mais on aurait pu aller **beaucoup BEAUCOUP** plus vite !

Vous allez découvrir ce qu'est le scaffold. Scaffold signifie "échafaudage" (d'où l'icône), en fait on va dire à CakePHP que l'on n'a pas trop le temps de faire toutes ces fonctions *index*, *add*, *edit*, etc. ; il va le faire à notre place !

Une nouvelle table pour notre exemple

Bon eh bien on va continuer à avancer sur notre application de gestion de bibliothèque. On a des livres et leurs auteurs. Maintenant, ce qui serait bien, c'est qu'on puisse emprunter des livres. Mais avant, il faut savoir qu'il peut y avoir plusieurs fois le même livre. On va donc devoir gérer les exemplaires des différents ouvrages. Les abonnés ne vont pas emprunter un livre, mais un exemplaire.

Je vais vous donner la table qu'on va créer :

Code : SQL - [Sélectionner](#)

```
CREATE TABLE `exemplaires` (  
  `id` int(11) NOT NULL auto_increment,  
  `livre_id` int(11) NOT NULL,  
  `created` datetime NOT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

C'est tout ce qu'il y a de plus simple. En gros un exemplaire correspond à un livre, donc j'ai mis un champ "livre_id" qui correspondra au champ "id" de la table "livres".



J'ai ajouté un champ "created" qui représentera la date à laquelle l'exemplaire a été ajouté dans la base de données. Comme on lui a donné le nom "created", CakePHP le reconnaît et il sait que c'est la date de création. De même, si vous appelez un champ "modified" CakePHP le mettra à jour à chaque modification.

Le modèle et le contrôleur

Là, attention ça va être très rapide.

</app/models/exemplaire.php>

Code : PHP - [Sélectionner](#)

```
<?php  
class Exemplaire extends AppModel {  
    var $belongsTo = 'Livre';  
}  
?>
```

« Exemplaire belongsTo Livre ».

Un exemplaire appartient à un livre.

/app/controllers/exemplaires_controller.php

Code : PHP - [Sélectionner](#)

```
<?php  
class ExemplairesController extends AppController {  
    var $scaffold;  
}  
?>
```

Voilà. On a juste indiqué à CakePHP que ce contrôleur est un scaffold, c'est parti...



Euh tu n'as pas oublié les vues ?

Non je n'ai rien oublié, pas besoin de vues, scaffold a ses propres vues, il est là pour nous faire gagner du temps !
Allez voir le résultat : <http://localhost/cakephp/exemplaires/>.

CakePHP: the rapid development php framework

Actions
[New Exemplaire](#)
[List Livres](#)
[New Livre](#)

Exemplaires

Id	Livre	Created	Actions		
1	Le vieil homme et la mer	2010-04-20 01:28:09	View	Edit	Delete
2	Le vieil homme et la mer	2010-04-20 01:28:09	View	Edit	Delete
3	Le vieil homme et la mer	2010-04-20 01:28:13	View	Edit	Delete

Page 1 of 1, showing 3 records out of 3 total, starting on record 1, ending on 3

<< previous | next >>

Voilà, on a la liste de tous les exemplaires (j'en ai rajouté quelques uns). Avec la pagination, le nom du livre...

Vous avez en bas la possibilité d'ajouter un exemplaire, ou ajouter un livre. Tout est fait tout seul. Ça permet déjà de remplir facilement la base de données. 😊



Quand on clique sur le titre du livre ça fait une erreur !

En effet, regardez le lien, quand on clique sur un livre notre navigateur va chercher la page suivante : <http://localhost/livres/view/2>

L'action "view" c'est une page où l'on voit les détails d'un livre (ici le livre id=2). Nous n'avons pas écrit cette page parce que ça ne vous aurait rien appris de plus et surtout parce qu'on affiche déjà tous les détails dans la liste. Cela étant, CakePHP suppose qu'il y a une action "view" alors il fait un lien vers celle-ci, notez qu'il a également fait une page "view" pour chaque exemplaire, cliquez sur le lien "view" au bout de la ligne vous verrez qu'il n'y affiche rien de spécial parce que notre table exemplaires n'a pas beaucoup de champs, ils sont déjà affichés dans la liste.

Utilité & limites

Scaffold, comme son nom l'indique, permet vraiment de faire très très rapidement une base pour commencer à faire votre application.

Vous allez vous en servir pour faire le *Back Office* (le "back") de votre application. *Back Office* c'est quoi ? C'est l'outil d'administration de l'application. Au début, on n'a pas le temps de faire des formulaires, surtout qu'on n'est pas sûr qu'on va vraiment s'en servir, certains passeront à la trappe, c'est du temps perdu. La bonne méthode à adopter c'est :

1. créer les tables (après avoir fait un schéma de celles-ci sur papier, pour ceux qui ont déjà fait de l'analyse) ;
2. créer les modèles avec leurs relations (hasMany, belongsTo, HABTM...) ;
3. créer les contrôleurs avec scaffold ;
4. tester si tout fonctionne. On ne va quand même pas commencer à faire des pages si nos modèles, constituant la base de notre application, n'ont pas été testés !

C'est très rapide et vous avez déjà un outil d'administration qui est au point. Vous pouvez commencer à faire le *Front Office* (la partie publique du site, où les visiteurs se baladent).

Par la suite, vous pourrez remplacer les scaffolds par vos vraies pages d'administration.



Pourquoi remplacer ? Ça me va très bien, il suffit juste de modifier un ou deux trucs et ça sera parfait !

Hé oui mais on arrive aux limites de scaffold, là. C'est comme un vrai échafaudage. On s'en sert pour construire l'application mais ce n'est pas fait pour être utilisé en permanence, si vous décidez de modifier certaines valeurs d'un contrôleur "scaffoldé", il va avoir un comportement bizarre et imprévisible, scaffold n'a pas été prévu pour être utilisé en version finale, il n'est là que pour aider le développeur.



Je vous rappelle que CakePHP est un *framework*, pas un CMS (système de gestion de contenu). Ça veut dire que les pages n'arriveront pas toutes seules par magie, il va falloir les coder.

Je vous conseille de garder les scaffolds au maximum, puis de passer sur vos vraies pages d'administration quand vous en aurez vraiment besoin. Il faut savoir que scaffold s'adapte à vos tables. Si vous virez ou ajoutez un champ il va automatiquement apparaître dans les formulaires et les listes, c'est un

énorme gain de temps !



Si un truc ne marche pas sur vos contrôleurs/vues scaffoldés, c'est probablement qu'il y a un problème soit dans vos modèles (les relations *hasOne* et *belongsTo*) soit dans les noms de vos tables (un pluriel mal placé ? une majuscule ?).

Et voilà, j'espère que vous ne m'en voulez pas de ne vous avoir montré cette fonctionnalité que maintenant. J'estimais qu'il est important de savoir le faire soi-même, à la main, avant d'utiliser les outils d'automatisation.

Et puis si vous aviez été voir la documentation officielle vous l'auriez peut-être découvert par vous-mêmes. 😊

Ne vous en faites pas, je vous cache encore beaucoup d'autres trucs, notamment la console cake qu'on va décortiquer dans quelques chapitres. 🧐

Partie 4 : Annexes

Conventions de nommage

Les règles de noms sont très strictes dans CakePHP, au début on s'y perd un peu alors voici un tableau récapitulatif qui pourra vous aider. Vous pouvez l'imprimer pour le garder sous la main, mais vous allez vite vous y habituer car il y a une logique derrière tout ça.

Règles de base

Les noms des fichiers sont en minuscules et les mots sont séparés par des underscores :
`livre.php`, `livres_controller.php`, `categories_controller.php`.

Les noms des classes commencent par une majuscule et les mots sont séparés par des majuscules :
`Livre`, `LivresController`, `CategoriesController` (cette règle est souvent appliquée, pas seulement dans Cake PHP).

Les noms des fonctions (les actions) commencent avec une minuscule et sont séparées par des majuscules :
`effacer()`, `afficherTout()`, `effacerEtRemplacer()` (ici aussi c'est une convention qui est respectée partout, même en dehors de CakePHP, même dans les autres langages).

Les modèles sont au singulier :

- `Livre` => `livre.php` ;
- `Utilisateur` => `utilisateur.php` .

Les contrôleurs sont au pluriel :

- `LivresController` => `livres_controller.php` ;
- `UtilisateursController` = `utilisateurs_controller.php` .

Les vues sont dans un répertoire portant le nom du contrôleur et portent le nom de l'action, en minuscules avec des underscores :

- `Livres::ajouter()` => `views/livres/ajouter.ctp` ;
- `GrosLivres::afficherTout()` => `views/gros_livres/afficher_tout.ctp` .

Le petit problème

Oui il y a un petit problème : CakePHP a été développé en anglais, donc pour la gestion du pluriel c'est... anglais. 😊

Vous allez sûrement tomber dessus quand vous voudrez faire des catégories.

`CategoriesController` pour le contrôleur et `Categorie` pour le modèle... sauf que le singulier de `Categories` c'est... `Category` !!
Vous avez aussi : `BigPerson` qui devient : `BigPeople`... `person`=>`people`...

Il y a aussi les adjectifs qui ne sont pas accordés en anglais.

- `BlueBook` => `BlueBooks` ;
- `BleuLivre` => `BleuLivres` ;

- LivreBleu => LivreBleus.

Les deux dernières lignes ne sont pas correctes en français, mais ça fonctionne sur CakePHP.

Eh oui c'est dur la vie. 😊

Rien de bien méchant, de toute façon si vous vous trompez vous aurez un joli carré rouge qui vous dira "controller xxxxxxx is missing !", donc en gros il vous donnera le fichier avec le nom qu'il veut, qu'il suffira de renommer.

Ça peut être déranger, sauf si vous nommez vos classes et fichiers en anglais. 😊

Tableau récapitulatif

Table MySQL	Modèle	Contrôleur	Action	URL Vue
En minuscules/underscore, au pluriel.	Majuscules, au singulier.	Majuscules, pluriel + "Controller".	Minuscules mais les mots sont séparés par des majuscules.	views/[contrôleur en minuscules]/[action en minuscules].
livres	Livre <i>app/models/livre.php</i>	LivresController <i>app/controllers/livres_controller.php</i>	LivresController::index()	<i>app/views/livres/index.ctp</i>
			LivresController::enregistrer()	<i>app/views/livres/enregistrer.ctp</i>
			LivresController::nouveau()	<i>app/views/livres/nouveau.ctp</i>
dernier_messages	DernierMessage <i>app/models/dernier_message.php</i>	DernierMessagesController <i>app/controllers/dernier_messages_controller.php</i>	DernierMessagesController::index()	<i>app/views/dernier_messages/index.ctp</i>
			DernierMessagesController::toutEffacer()	<i>app/views/dernier_messages/tout_effacer.ctp</i>
categories	Category <i>app/models/category.php</i>	CategoriesController <i>app/controllers/categories_controller.php</i>	CategoriesController::readAll()	<i>app/views/categories/read_all.ctp</i>
			CategoriesController::renommerEtDeplacer()	<i>app/views/categories/renommer_et_deplacer.ctp</i>
people	Person <i>app/models/person.php</i>	PeopleController <i>app/controllers/people_controller.php</i>	PeopleController::rename()	<i>app/views/people/rename.ctp</i>

Introduction à la POO

On va absolument avoir besoin de savoir utiliser les objets pour utiliser CakePHP.

Heureusement, c'est très simple à utiliser.

Je vais tout vous expliquer, vous ne deviendrez pas des experts en POO, mais au moins vous pourrez les manipuler en sachant ce que vous faites. Je veux juste vous apprendre ce dont vous avez besoin pour utiliser CakePHP.

Cette annexe est une simple "remise à niveau". Je vous invite à approfondir le sujet sur la toile ou sur le site du zéro, la POO n'est pas simplement utilisée dans PHP, mais dans de nombreux autres langages de programmation.

Cependant, si vous voulez juste connaître le minimum, vous pouvez lire la suite.

Les exemples simples du début

On va commencer très très très simple. On va commencer par faire une classe "Animal".

Les Attributs

Code : PHP - [Sélectionner](#)

```
<?php
// Les noms des classes commencent généralement par une majuscule (et pas seulement en
PHP, c'est une convention)
class Animal {
    var $nom = 'Popy';
    var $faim = true;
}

// On essaie d'y accéder mais on ne peut pas.
echo '$nom = ' . $nom . '<br />';
echo '$faim = ' . $faim . '<br />';
?>
```

Voilà, notre classe contient deux variables : `$nom` et `$faim`. Quand des variables se trouvent dans une classe on dit que ce sont des **attributs de Animal** (Animal étant la classe).

Vous notez qu'elles sont dans les accolades donc totalement séparées du reste du code, on ne peut pas y accéder depuis l'extérieur.



Bah, à quoi ça sert si on peut pas y accéder ?

Si si, on peut y accéder, mais il faut le faire spécialement. En fait on a pas du tout déclaré de variables, on a juste expliqué à PHP "ce qu'est un Animal". De base PHP sait qu'un *Integer* c'est un entier, un *Float* c'est un nombre flottant (avec des virgules), un *String* c'est une chaîne de caractères, etc.

Ben là, on lui a dit qu'un Animal, c'est un *string* `$nom` et un booléen `$faim`.

On va faire « mumuse » avec :

Code : PHP - [Sélectionner](#)

```
<?php
class Animal {
    var $nom = 'Popy';
    var $faim = true;
}

// On crée un animal, qui s'appellera $a
// Pour cela on utilise le mot-clef new, suivi du nom de la classe, suivi de deux
parenthèses vides.
$a = new Animal();

// On accède à $nom en utilisant une flèche (un tiret + un "plus grand que")
echo $a->nom . '<br />';

// Ici pareil, on accède à $faim.
if ($a->faim == true) {
    echo $a->nom . ' a faim !<br />';
}

// On peut très bien faire un deuxième animal.
$b = new Animal();

// On change le nom du deuxième animal pour pas se mélanger les pinceaux
$b->nom = 'Toto';
echo $b->nom . '<br />';

// Notez $a a toujours gardé son nom, on a seulement changé celui de $b
echo $a->nom . '<br />';
?>
```



Pour accéder à un attribut on serait tenté d'utiliser la syntaxe suivante : `$a->$nom`. Ce n'est pas bon ! Il ne faut pas mettre de `$` après la flèche, un simple `$a->nom` suffit.

Les Méthodes

Bon, vous n'espériez pas que ça soit si simple. Notre classe contient des variables : les attributs. Mais elle peut également contenir des FONCTIONS ! Ces fonctions sont appelées méthodes. Depuis l'extérieur on y accède de la même façon que les attributs, avec la flèche :

Code : PHP - [Sélectionner](#)

```
<?php
class Animal {
    var $nom = 'Popy';
    var $faim = true;

    // On utilise function pour déclarer la méthode, comme pour une fonction classique
    function parler() {
        echo 'Cet animal hurle<br />';
    }

    // De même, on peut lui envoyer un ou plusieurs paramètres entre parenthèses.
    function manger($nourriture) {
        // La méthode peut également renvoyer une valeur avec return
        return 'Cet animal mange '.$nourriture.'<br />';
    }
}

$a = new Animal();

// On appelle la fonction/méthode parler().
$a->parler();

// On appelle la méthode manger(). On met un echo devant pour afficher le return.
echo $a->manger('viande');
```

Points particuliers

Le problème

Imaginons qu'on veuille faire un animal qui peut se présenter.

Code : PHP - [Sélectionner](#)

```
<?php
class Animal {
    var $nom;

    function presentation() {
        echo $nom;
    }
}

$a = new Animal();
$a->nom = 'popy';
$a->presentation();
?>
```

Et toc ça ne marche pas. 😊

Pourquoi ? Eh bien PHP ne peut pas savoir quel `$nom` afficher !



Mais il n'y en a qu'un seul de `$nom` ?

Code : PHP - [Sélectionner](#)

```
<?php
class Animal {
    var $nom;

    function presentation() {
        echo $nom;
    }
}

$a = new Animal();
$a->nom = 'poppy';
$b = new Animal();
$b->nom = 'toto';

$a->presentation();
?>
```

Et là il y en a deux de `$nom`... Bon on pourrait dire que comme on appelle `$a->presentation`, il serait logique qu'on utilise le nom de `$a`. Mais il y a aussi une autre subtilité, s'il y a un paramètre `$nom`.

Code : PHP - [Sélectionner](#)

```
<?php
class Animal {
    var $nom;

    function presentation($nom) {
        echo $nom;
    }
}
?>
```

Il se passe quoi ? Pour PHP, c'est très simple, il va utiliser le `$nom` envoyé en paramètre. Il nous faut un truc pour accéder aux attributs depuis l'intérieur des méthodes...

La solution : `$this`

Pour gagner en clarté on utilise `$this` !

Code : PHP - [Sélectionner](#)

```
<?php
class Animal {
    var $nom;

    function presentation() {
        echo $this->nom.'  
';
    }
}

$a = new Animal();
$a->nom = 'poppy';
$b = new Animal();
$b->nom = 'toto';

$a->presentation();
$b->presentation();
?>
```

Voilà, ça fonctionne ! Dans le premier appel, on utilise le nom de `$a`, et dans le deuxième le nom de `$b`. `$this` est très pratique ! On peut même faire pareil avec des méthodes.

Code : PHP - [Sélectionner](#)

```

<?php
class Animal {
    var $nom;

    function presentation() {
        $this->parler();
    }

    function parler() {
        echo $this->nom . ' hurle';
    }
}

$a = new Animal();
$a->nom = 'poppy';
$a->presentation();

?>

```

Là ça se passe en plusieurs étapes :

1. on appelle `$a->presentation()` ;
2. dans la méthode `presentation()`, on appelle la méthode `parler()` ;
3. dans la méthode `parler()`, on affiche le texte.

Grâce à `$this` on peut également faire des choses de ce genre-là :

Code : PHP - [Sélectionner](#)

```

<?php
class Animal {
    var $nom;

    function bonjour($autre_animal) {
        echo $this->nom . ' dit bonjour à ' . $autre_animal->nom . '<br />';
    }
}

$a = new Animal();
$a->nom = 'poppy';

$b = new Animal();
$b->nom = 'toto';

$a->bonjour($b);
$b->bonjour($a);

?>

```

Étudions `$a->bonjour($b)` : on envoie `$b` à la méthode `bonjour()`. À l'intérieur de `bonjour`, qu'est-ce qui se passe ?

- `$this->nom` c'est le nom de `$a` puisqu'on a fait appel à la méthode `bonjour()` de `$a` ;
- `$autre_animal->nom` c'est le nom de `$b`, c'est l'animal qu'on a envoyé en paramètre. Ici on peut accéder à l'attribut sans `$this` parce que cet objet a été reçu en paramètre ! `$this` sert seulement à récupérer les attributs de l'objet avec lequel on a appelé la méthode.

L'inverse est aussi possible, puis `$b` possède également cette méthode.

On peut même faire `$a->bonjour($a)`, si vous avez l'esprit tordu. 😊

Constructeur

Vous pouvez écrire plein de méthodes dans vos classes, mais il y en a une qui est un peu particulière : le constructeur. Elle ressemble à une méthode normale, la seule façon de la différencier des autres c'est par son nom, c'est exactement le même que celui de la classe (avec la majuscule). Cette fonction peut aussi accepter des paramètres, bien que cela ne soit pas obligatoire.

La particularité de cette méthode, c'est qu'elle est appelée toute seule dès qu'on crée un objet de cette classe. On va s'en servir pour remplir automatiquement le nom de l'animal.

Code : PHP - [Sélectionner](#)

```
<?php
class Animal {
    var $nom;

    function Animal($nom) {
        echo 'Naissance de ' . $nom . '<br />';
        $this->nom = $nom;
    }
}

// On envoie des paramètres au constructeur quand on déclare les objets.
$a = new Animal('poppy');
$b = new Animal('toto');

// C'est cool, on a plus besoin de faire des $a->nom='poppy';
?>
```

Vous savez maintenant pourquoi on mettait des parenthèses après le `new Animal()`, il n'y avait pas de constructeur alors on n'envoyait pas de paramètres !

Un objet en attribut

Une petite chose que j'aimerais vous montrer, ce n'est pas un nouveau truc, on va revenir sur les attributs. Pour l'instant on n'a utilisé que des attributs qui sont des chaînes de caractères, mais ça peut être n'importe quoi, un entier, un flottant et même... un autre objet ! On va faire un exemple, disons que notre animal a un fils.

Code : PHP - [Sélectionner](#)

```
<?php
class Animal {
    var $nom;
    var $fils;

    function Animal($nom) {
        echo 'Naissance de ' . $nom . '<br />';
        $this->nom = $nom;
    }

    function presentation() {
        echo "Salut, moi c'est " . $this->nom . "<br />";
    }
}

$a = new Animal('poppy');
// On crée le deuxième animal mais on le met dans l'attribut $fils de l'objet $a !
$a->fils = new Animal('toto');

// On peut accéder au nom du fils avec deux flèches.
echo $a->nom . '<br />';
echo $a->fils->nom . '<br />';

// Pareil pour les méthodes
$a->presentation();
$a->fils->presentation();
?>
```

Encore plus fort !

Allez, je vais me faire plaisir, je vais améliorer l'exemple !

On va essayer de faire un truc un peu tordu, mais pas trop, juste ce qu'il faut. 🐱

Code : PHP - [Sélectionner](#)

```
<?php
class Animal {
    var $nom;
    var $fils;

    function Animal($nom) {
        $this->nom = $nom;
        $this->fils = false;
    }

    function presentation() {
        echo "Salut, moi c'est ". $this->nom. "<br />";
        if($this->fils) {
            echo "J'ai un fils(" . $this->fils->nom. "), laissons-le se présenter : <br />";
            $this->fils->presentation();
        }
    }
}

$a = new Animal('poppy');
$a->fils = new Animal('toto');
$b = new Animal('boulga');

$a->presentation();
echo '<hr />';
$b->presentation();
?>
```

Maintenant, quand un animal se présente on regarde s'il a un fils.

Si oui, son fils se présente ! Donc on n'appelle pas la même méthode, mais celle du fils !

Amusez-vous avec, essayez de faire un fils au fils, et encore un fils au fils...

Faites une petite pause, la prochaine partie sera la dernière.

Héritage

Bon je vous préviens tout de suite, l'héritage, je vous le présente parce que vous allez l'utiliser. Et seulement l'utiliser. Vous allez utiliser des classes qui héritent d'autres classes, mais vous n'aurez pas besoin de réfléchir à fabriquer des classes qui devront être héritées par d'autres (et là, je prends 1 minute pour me relire, histoire d'être sûr que ma phrase est correcte 😊).

Si vous vous dites que vous ne serez jamais capable de faire des imbrications pareilles, ça tombe bien parce que vous n'aurez pas à le faire ! En gros vous devez juste savoir que ça existe. 😊

Plusieurs types d'animaux

On a fait joujou avec des animaux. Ils ont des attributs, des méthodes, tout est bon. Maintenant, on veut faire des chiens. Seulement un chien, c'est pareil qu'un animal, ça possède un nom, ça parle (aboie) et ça mange. Ça peut même avoir un fils. Est-ce qu'on va devoir retaper tout le code ? Non, il nous faut une solution plus simple !



Le premier qui me dit que le copier/coller est une solution convenable, je l'envoie dans la cave de M@teo21.

On va simplement demander à notre classe Chien de copier la classe Animal. La classe Chien va hériter de la classe Animal.

Code : PHP - [Sélectionner](#)


```
<?php
class Animal {
    var $nom;

    function Animal($nom) {
        $this->nom = $nom;
    }

    function parler() {
        echo $this->nom.' hurle !<br />';
    }
}

// On indique l'héritage avec le mot-clé "extends" (avec un S)
class Chien extends Animal {
    // N'oubliez pas de mettre des accolades, même si c'est vide à l'intérieur !!!
}

$b = new Animal('Popol');
$b->parler();

$a = new Chien('Médor');
$a->parler();
?>
```

Je pense que vous avez saisi, on a copié la classe en gros.



Euh, et ça sert à quoi en fait ?

Pour l'instant à rien, on aurait pu faire un Animal « Médor ».

Amélioration du Chien

L'intérêt de l'héritage, c'est qu'on peut ajouter des nouvelles méthodes au chien, des méthodes que l'Animal n'a pas. Par exemple un chien peut "faire le beau", ce n'est pas tous les animaux qui peuvent faire ça, c'est seulement une particularité du chien.

Code : PHP - [Sélectionner](#)

```
<?php
class Animal {
    var $nom;
    function Animal($nom) {
        $this->nom = $nom;
    }
    function parler() {
        echo $this->nom.' hurle !<br />';
    }
}

class Chien extends Animal {
    function faireLeBeau() {
        echo $this->nom.' fait le beau ! Brave chichien !';
    }
}

$b = new Animal('Popol');
$a = new Chien('Médor');

$a->faireleBeau();

$b->faireLeBeau(); // ERREUR ! un Animal ne peut pas faire le beau !
?>
```

N'oubliez pas que ça va dans un seul sens. Si Chien hérite de Animal, Chien récupère tous les attributs et méthodes de Animal, mais pas l'inverse. Dans notre exemple, « Popol » ne peut pas faire le beau, c'est un Animal, pas un Chien.



Oh, ça va devenir le bordel. Et si on ajoute une méthode qui a déjà été définie, comment ça va se passer ?

Allez, on va tester !

Code : PHP - [Sélectionner](#)

```
<?php
class Animal {
    var $nom;
    function Animal($nom) {
        $this->nom = $nom;
    }
    function parler() {
        echo $this->nom.' hurle !<br />';
    }
    function manger($nourriture) {
        echo $this->nom.' mange '.$nourriture.'<br />';
    }
}

class Chien extends Animal {
    // On va modifier la méthode parler, un chien ca ne hurle pas, ca ABOIE !
    function parler() {
        echo $this->nom.' aboie ! Ouaf ouaf !<br />';
    }
}

$a = new Chien('Médor');
$a->manger('viande');
$a->parler();
?>
```

Voilà il suffit de tester pour s'apercevoir du résultat. Quand on redéfinit la méthode, elle sera utilisée à la place de la méthode héritée. Ici on hérite de Animal, c'est une superbe classe, elle va nous être utile, sauf une : **parler()**. Elle ne correspond pas au chien car il aboie, il ne hurle pas. Alors on la redéfinit.

La méthode **manger()** est toujours valable par contre, on n'y touche pas : un chien ça mange aussi, surtout après avoir aboyé toute la nuit contre les ombres qui viennent cambrioler la maison !

À quoi ça va nous servir

C'est très simple. Avec le *framework* CakePHP on travaillera quasiment **uniquement** avec des classes héritées.

Le but du *framework* c'est d'éviter qu'on retape toujours les mêmes parties de code, il nous fournit du code déjà prêt. C'est par le biais de l'héritage qu'on le recevra. On aura des classes avec déjà plein de méthodes déjà prêtes. On aura juste à ajouter celles qui seront spécifiques à notre site.

Voilà c'est fini. Mais ce ne sont que des notions très basiques de POO. C'est juste ce dont vous avez besoin pour suivre la suite du tutoriel.

Juste pour information (ce n'est pas la peine de retenir ce que je vais dire, on verra ça en détails après) nos modèles seront des classes qui hériteront tous d'une même classe. Ils auront donc tous les mêmes méthodes, on aura juste à changer un ou deux attributs pour les "régler"... Pas de panique vous n'aurez pas à regarder le code super compliqué de la classe parente, je vous donnerai le nom de ces attributs.

Pareil pour les contrôleurs, il y aura une classe parente pour tous nos contrôleurs, et on va faire une méthode pour chaque page du site (pour chaque action).

Pour les vues, ça sera du code PHP/XHTML basique, (presque) pas de POO.

Ce tutoriel n'est pas fini.

J'avoue que je ne suis pas très ordonné : je ne sais même pas si je vais aborder tous les aspects de CakePHP, ni dans quel ordre.

Tout ce qui est sûr, c'est qu'actuellement ce n'est pas du tout fini...