CakePHP 3.x

Table des matières

1. Documentation

Cookbook 3.x en Français

2. Installation/ Création de projet

Avec **Composer**

1. Depuis une invite de commande, se déplacer vers le répertoire parent qui contiendra le projet (exemple « C:\wamp\www\ »)
2. Créer le projet : remplacer « my_app_name » par le nom de projet désiré

```
composer self-update && composer create-project --prefer-dist cakephp/app my_app_name
```

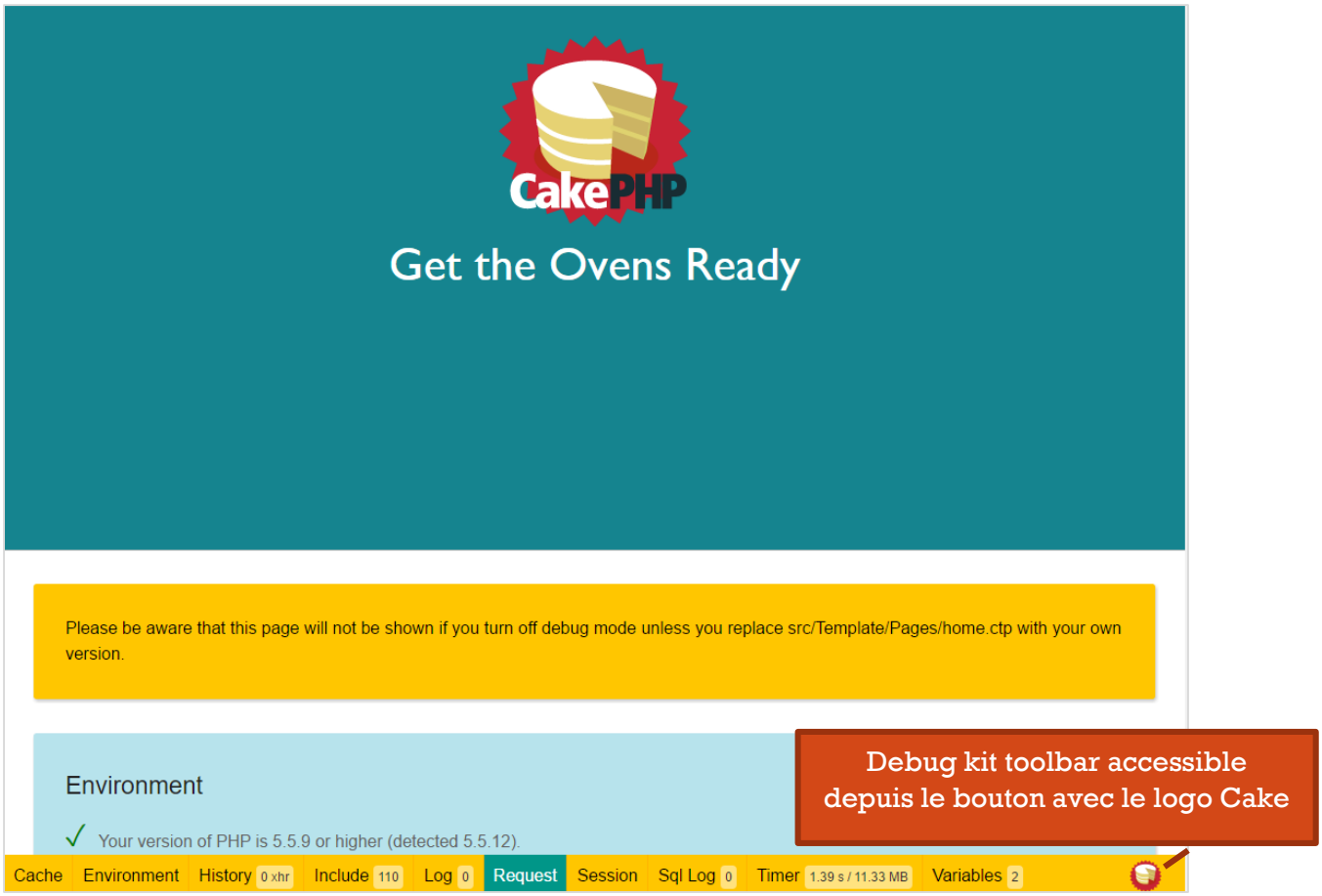
On peut aussi faire l'inverse, commencer par un « composer init » dans le dossier du projet puis ajouter la dépendance

```
"require": {
    "cakephp/cakephp": "dev-master"
}
```

… et enfin lancer un « composer install »

On peut déjà se rendre « http://localhost/cakedemo/ » (le projet nommé ici « cakedemo ») pour afficher la page.

La page permet de savoir ce qui est correctement configuré



3. Configuration (« config\app.php »)

a. Base de données

Modifier les informations de base de données

```
'Datasources' => [
    'default' => [
        'className' => 'Cake\Database\Connection',
        'driver' => 'Cake\Database\Driver\Mysql',
        'persistent' => false,
        'host' => 'localhost',
        /**
         * CakePHP will use the default DB port based on the driver selected
         * MySQL on MAMP uses port 8889, MAMP users will want to uncomment
         * the following line and set the port accordingly
         */
        //'port' => 'non_standard_port_number',
        'username' => 'root',
        'password' => '',
        'database' => 'cakedemodb',
        'encoding' => 'utf8',
        'timezone' => 'UTC',
        'flags' => [],
        'cacheMetadata' => true,
        'log' => false,
```

b. Email

Il est possible de configurer l'envoi de mails

```
'Email' => [
    'default' => [
        'transport' => 'default',
        'from' => 'you@localhost',
        //'charset' => 'utf-8',
        //'headerCharset' => 'utf-8',
    ],
],
```

4. Console « cake »

Documentation

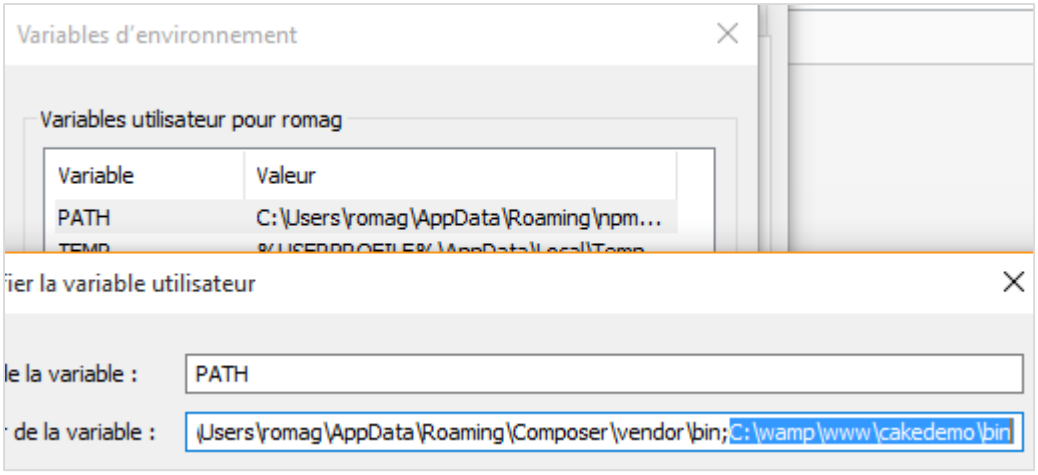
On va pouvoir générer rapidement (RAD) les tables, entités, contrôleurs, templates avec Bake … mettre à jour la base avec des migrations… en ligne de commande.

a. Installation avec Composer

```
composer require --dev cakephp/bake:~1.0
```

Astuce

Il est possible d'ajouter à la variable d'environnement **PATH** le chemin vers « cake » afin de saisir directement « cake » au lieu de « bin\cake » dans la console



On peut obtenir la liste des commandes disponibles avec « **bin\cake** » :

```
Available Shells:

[Bake] bake

[DebugKit] benchmark, whitespace

[Migrations] migrations

[CORE] i18n, orm_cache, plugin, routes, server

[app] console
```

… **Bake** (commande « **bin\cake bake** »)

```
Available bake commands:

- all
- behavior
- cell
- component
- controller
- fixture
- form
- helper
- mailer
- migration
- migration_snapshot
- model
- plugin
- seed
- shell
- shell_helper
- task
- template
- test

By using `cake bake [name]` you can invoke a specific bake task.
```

Bake va nous permettre de générer le code rapidement (commande « bin\cake bake [name] »)

b. Bake

all

Crée « tout d'un coup » …model (table + entity), contrôleurs, Templates pour les tables de la base configurée

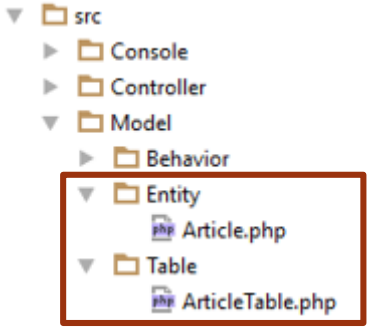
```
bin\cake bake all Article
```

model

Création de « model » (table + entity)

```
bin\cake bake model Article
```

La table et l'entité sont créées



debug depuis le controller correspondant

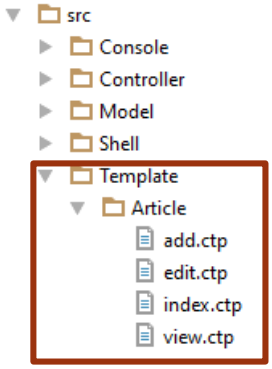
```
debug($this->Article);
```

controller

```
bin\cake bake controller Article
```

template

```
bin\cake bake template Article
```



Les vues sont générées

5. Planification

a. Problème

- Soit on a une **base de données existante**
- Soit on **crée sa base** en SQL, etc.
- Soit on génère des **migrations à partir** d'une **base existante**
- Soit on utilise la console « cake » pour créer des fichiers de **migrations**, remplir/modifier les tables, **mettre à jour la base**. Ce choix sera sans doute le plus long et laborieux, donc plutôt à éviter.

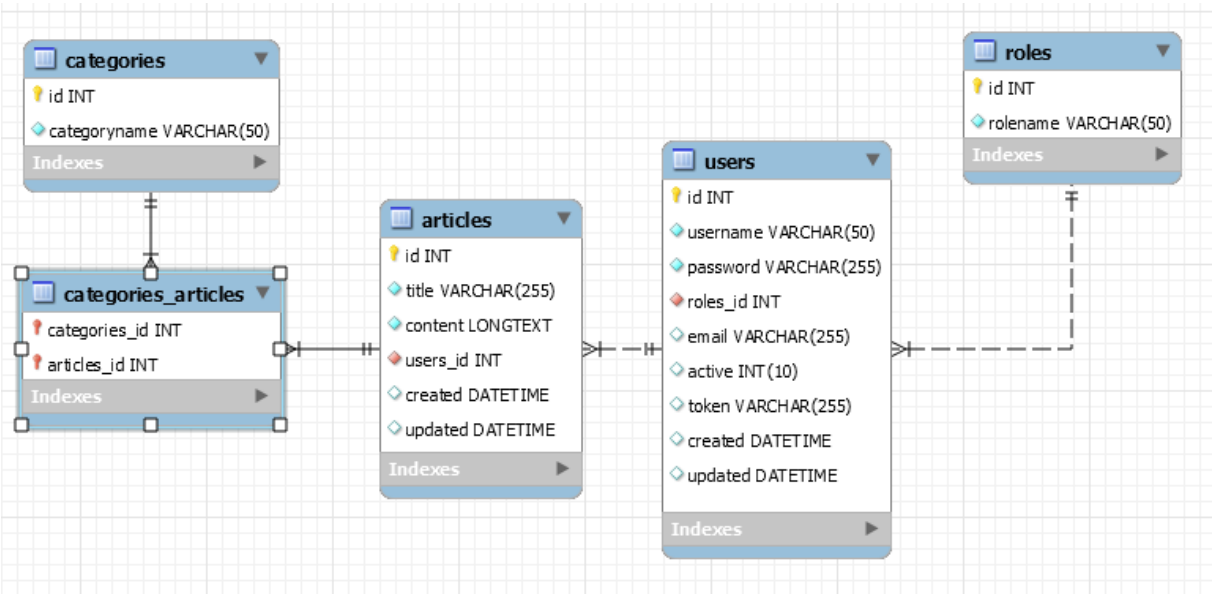
Note :

Les tables doivent avoir en premier un champ **id** de type **INT** et **auto incrémenté**, des champs **created** et **updated** de type **DATETIME** qui seront « automatiquement » remplis

b. Première solution : Création du schéma de la base avec MySQL Workbench

Installer MySQL Workbench

Menu « Database » … « Reverse Engineer »



Puis :

- Soit on génère un script SQL .. Menu « File>.. »Export » … « Forward engineer SQL Create Script… ». Il suffit ensuite de mettre à jour la base depuis phpMyAdmin.
- Soit on met à jour directement la base avec menu « Database » … « Forward Engineer »

c. Seconde solution : Migrations

On peut définir le nom du fichier de migration à créer

Ligne de commandes

```
bin\cake bake migration NomFichier colonnes
```

…puis les colonnes

Exemple

```
bin\cake bake migration CreateUsers username:string[50] password:string
email:string active:integer[10] token:string roles_id:integer created updated
```

Nom de fichier des Migrations doivent respecter les règles suivantes

- **(/^(Create)(.*)/)** Ajout de table
- **(/^(Drop)(.*)/)** Migration supprimant une table
- **(/^(Add).*(?:To)(.*)/)** Ajout de champs à la table spécifiée.
- **(/^(Remove).*(?:From)(.*)/)** Suppression de champs de la table spécifiée.
- **(/^(Alter)(.*)/)** Modifier la table spécifiée.

Pour les **colonnes** :

- On n'indique pas la colonne id
- On peut indiquer simplement created updated (seront automatiquement de type DATETIME)
- Pour les autres colonnes le format

```
fieldName:fieldType[length]:indexType:indexName
```

Exemples :

- username :string[50]
- password :string
- active :integer[10]
- email:string:unique
- email:string:unique:EMAIL_INDEX
- email:string[120]:unique:EMAIL_INDEX

Types possibles :

- string
- text
- integer
- biginteger
- float
- decimal
- datetime
- timestamp
- time
- date
- binary
- boolean
- uuid

Clé primaires et étrangères : On ne peut pas les définir en ligne de commande, il faut modifier le fichier de migration généré

Attention les colonnes sont toutes définies à **n'acceptant pas NULL par défaut**, il faudra repasser dans les fichiers de migrations générés

Ordre de création

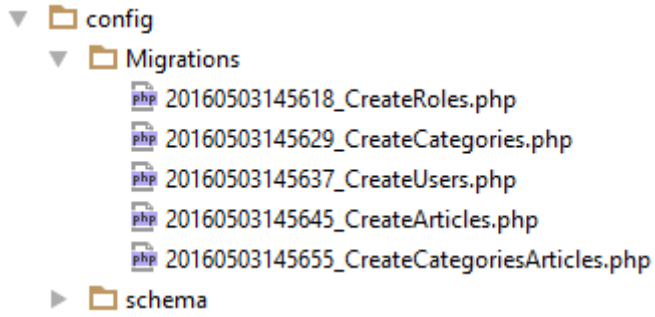
Pour éviter les conflits de clés étrangères mieux vaut respecter un ordre de création :

Table sans clé étrangère -> tables avec clés étrangères, ce qui donnerait :

- categories et roles en premier
- Puis articles et users
- Et categories_articles en dernier

```
bin\cake bake migration CreateRoles rolename:string[50]
bin\cake bake migration CreateCategories categoryname:string
bin\cake bake migration CreateUsers username:string[50] password:string email:string active:integer[10]
token:string roles_id:integer created updated
bin\cake bake migration CreateArticles title:string content:text users_id:integer created updated
bin\cake bake migration CreateCategoriesArticles categories_id:integer articles_id:integer
```

Les fichiers de Migrations sont créés dans « config\Migrations »



Contraintes/Relations

Ajout d'une contrainte de clé étrangère

Il faut la définir dans le fichier de migration généré

Exemple : un utilisateur a un rôle

```php
<?php
use Migrations\AbstractMigration;
class CreateUsers extends AbstractMigration
{
    public function change()
    {
        $table = $this->table('users');
        $table->addColumn('username', 'string', [
            'default' => null,
            'limit' => 50,
            'null' => false,
        ]);
        $table->addColumn('password', 'string', [
            'default' => null,
            'limit' => 255,
            'null' => false,
        ]);
        $table->addColumn('email', 'string', [
            'default' => null,
            'limit' => 255,
            'null' => false,
        ]);
        $table->addColumn('active', 'integer', [
            'default' => null,
            'limit' => 10,
            'null' => false,
        ]);
        $table->addColumn('token', 'string', [
            'default' => null,
            'limit' => 255,
            'null' => false,
        ]);
        $table->addColumn('roles_id', 'integer', [
            'default' => null,
            'limit' => 11,
            'null' => false,
        ]);
        $table->addColumn('created', 'datetime', [
            'default' => null,
            'null' => false,
        ]);
        $table->addColumn('updated', 'datetime', [
            'default' => null,
            'null' => false,
        ]);
        $table->create();

        $this->table('users')
            ->addForeignKey(
                'roles_id',
                'roles',
                'id',
                [
                    'update' => 'NO_ACTION',
```

Placé à la fin, après la création de la table

```
                    'delete' => 'NO_ACTION'
                ]
            )
            ->update();
    }
}
```

On peut faire la même chose pour la table articles : chaque article est écrit par un utilisateur

```
<?php
use Migrations\AbstractMigration;

class CreateArticles extends AbstractMigration
{
    public function change()
    {
        $table = $this->table('articles');
        $table->addColumn('title', 'string', [
            'default' => null,
            'limit' => 255,
            'null' => false,
        ]);
        $table->addColumn('content', 'text', [
            'default' => null,
            'null' => false,
        ]);
        $table->addColumn('users_id', 'integer', [
            'default' => null,
            'limit' => 11,
            'null' => false,
        ]);
        $table->addColumn('created', 'datetime', [
            'default' => null,
            'null' => false,
        ]);
        $table->addColumn('updated', 'datetime', [
            'default' => null,
            'null' => false,
        ]);
        $table->create();

        $this->table('articles')
            ->addForeignKey(
                'users_id',
                'users',
                'id',
                [
                    'update' => 'NO_ACTION',
                    'delete' => 'NO_ACTION'
                ]
            )
            ->update();
    }
}
```

Table pour la relation n:n

Id à false (ne pas ajouter la colonne id à la table) et les deux colonnes de clés primaires, retirer également auto increment

```php
public function change()
{
    $table = $this->table('categories_articles', ['id' => false,
'primary_key' => ['categories_id', 'articles_id']]);
    $table->addColumn('categories_id', 'integer', [
        'default' => null,
        'limit' => 11,
        'null' => false,
    ]);
    $table->addColumn('articles_id', 'integer', [
        'default' => null,
        'limit' => 11,
        'null' => false,
    ]);
    $table->create();

    $this->table('categories_articles')
        ->addForeignKey(
            'articles_id',
            'articles',
            'id',
            [
                'update' => 'NO_ACTION',
                'delete' => 'NO_ACTION'
            ]
        )
        ->addForeignKey(
            'categories_id',
            'categories',
            'id',
            [
                'update' => 'NO_ACTION',
                'delete' => 'NO_ACTION'
            ]
        )
        ->update();
}
```

Clés étrangères

Mise à jour de la base

```
bin\cake migrations migrate
```

Rollback

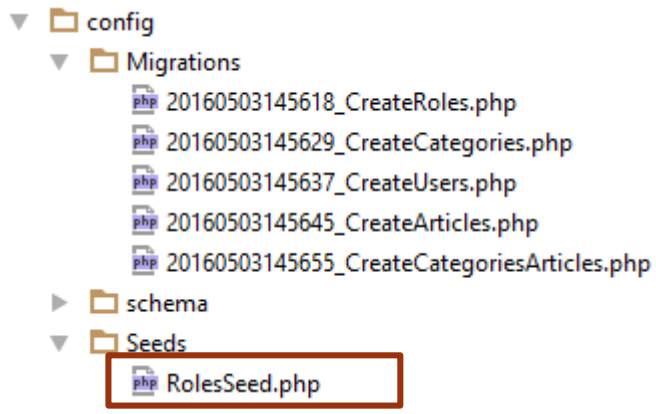
```
bin\cake migrations rollback
```

d. Seed

Création d'un fichier pour remplir une table.

Exemple avec roles

```
bin\cake bake seed roles
```



Exemple

```php
<?php
use Phinx\Seed\AbstractSeed;

class RolesSeed extends AbstractSeed
{
    public function run()
    {
        $data = [['rolename'=> 'admin'],['rolename'=> 'user'],['rolename'=> 'writer']];

        $table = $this->table('roles');
        $table->insert($data)->save();
    }
}
```

Ajout de données

Remplir la base de données

```
bin\cake migrations seed
```

e. Créer un fichier de Migration à partir d'une base existante

```
bin\cake bake migration_snapshot Initial
```

6. Routes

Avec CakePHP la plupart du temps on n'aura pas à définir les routes.

Il suffira de respecter « /controleur/action/ » …

Exemple

```
http://localhost/cakedemo/article/index
http://localhost/cakedemo/article/add
```

a. Redéfinir la route par défaut

Exemple dans « /config/routes.php »

```php
//$routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);

$routes->connect('/', ['controller' => 'Articles', 'action' => 'index', 'home']);
```

b. Ajouter un préfixe

Documentation

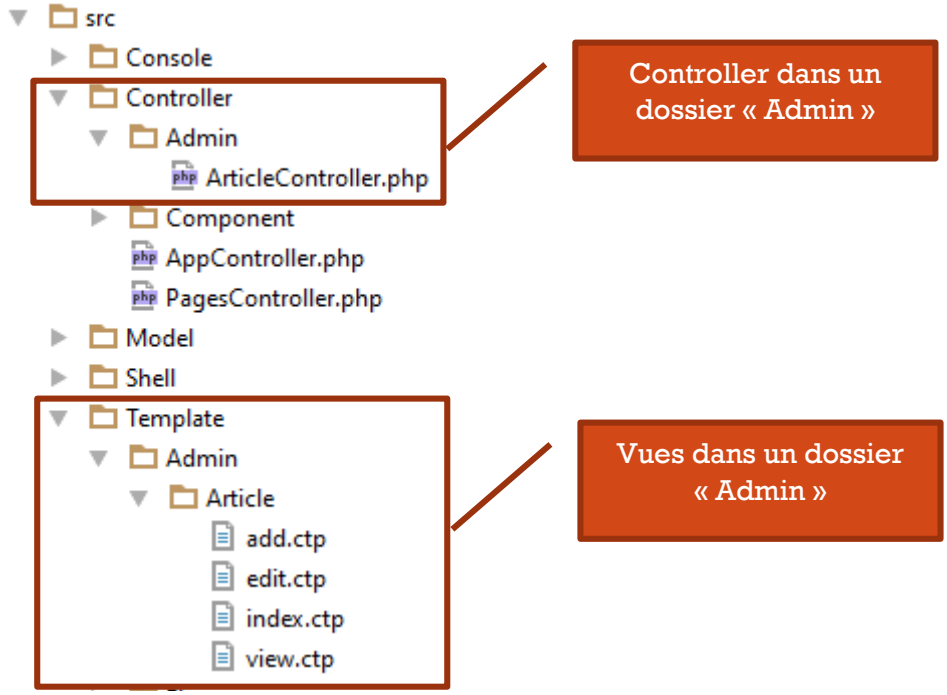
Création d'une section « Admin »

Dans « config\app.php », ajouter

```
Router::prefix('Admin', function ($routes) {
    $routes->fallbacks('InflectedRoute');
});
```

Déplacer le contrôleur et les vues. Changer le namespace du contrôleur :

```
namespace App\Controller\Admin;
```



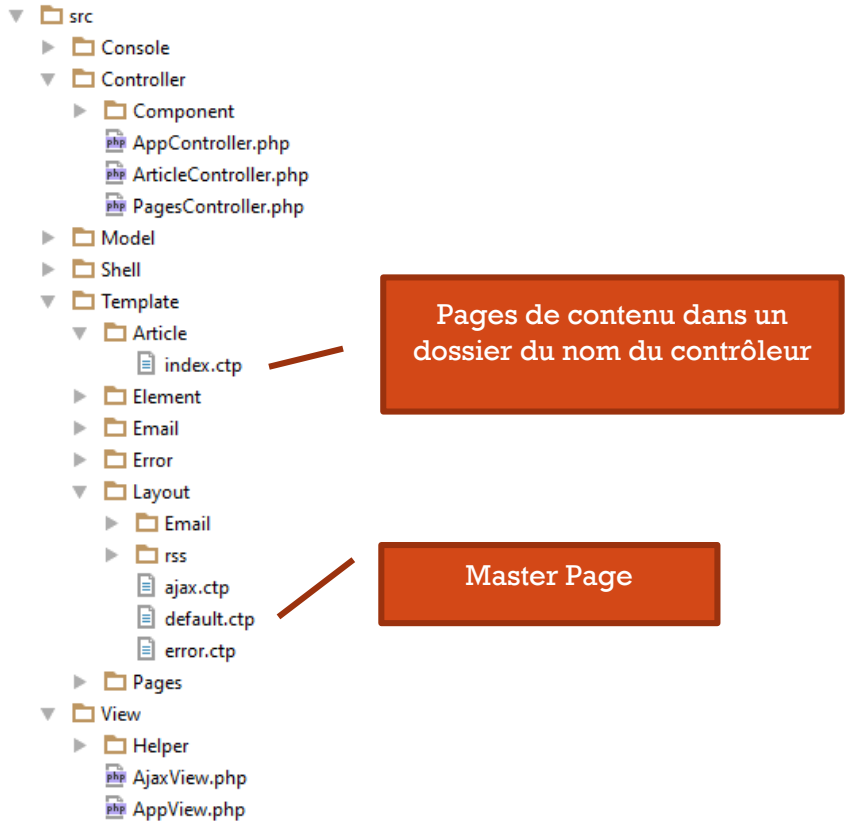
On accède désormais aux vues du contrôleur « Article » par

```
http://localhost/cakedemo/admin/article/index
http://localhost/cakedemo/admin/article/add
http://localhost/cakedemo/admin/article/edit
http://localhost/cakedemo/admin/article/view
```

7. Vues

Documentation

Création d'un dossier correspondant au contrôleur (exemple « Article ») dans « Template » puis ajout de vues *.ctp correspondant à l'action (exemple « index.ctp »)



a. Master page

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
     <?= $this->fetch('title') ?>
         <?= $this->fetch('meta') ?>
         <?= $this->fetch('css') ?>
         <?= $this->fetch('script') ?>

     <link rel="stylesheet" type="text/css" href="css/bootstrap.css">
     <link rel="stylesheet" type="text/css" href="css/style.css">

     <?php $this->Html->css('bootstrap.css'); ?>
     <?php $this->Html->css('style.css'); ?>


</head>
<body>

<section class="container">
    <?= $this->fetch('content') ?>
</section>
```

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.3/jquery.js"></scr
ipt>
<script src="js/bootstrap.js"></script>

<?php $this->Html-
>script('https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.3/jquery.js');
?>
<?php $this->Html->script('bootstrap.js'); ?>
</body>
</html>
```

Ajout de scripts

```
<?= $this->Html->script('https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.3/jquery.js') ?>
```

Blocs qui pourront être remplies par les pages de contenu

```
<?= $this->fetch('title') ?>
<?= $this->fetch('meta') ?>
<?= $this->fetch('css') ?>
<?= $this->fetch('script') ?>
```

Affichage des pages de contenu

```
<div class="container clearfix">
    <?= $this->fetch('content') ?>
</div>
```

b. Page de contenu

Exemple dans une page de contenu

```
<?= $this->assign('title','Liste des articles') ?>

<h1>Index!</h1>
<p>Lorem ipusm ..</p>
```

Bloc sur plusieurs lignes

```
<?php $this->start('sidebar'); ?>


<?php $this->end(); ?>
```

Inclure automatiquement dans les blocs depuis une page de contenu

```
<?php $this->Html->css('mystyles.css', array('inline'=>false); ?>

<?php $this->Html->script('myscript.js', array('inline'=>false); ?>
```

c. Helpers

Dans la master page

```
<?= debug($this->Html) ?>
```

Pour ajouter sa propre feuille de styles

```
<?= $this->Html->css('mystyles.css') ?>
```

Cela va chercher les fichiers dans « webroot »

8. Contrôleur

a. Afficher la vue pour l'action

Le contrôleur et l'action sans paramètres peuvent être vides, cela affichera automatiquement la vue lorsqu'on ira « ../article/index »

```
<?php

namespace App\Controller;


class ArticleController extends AppController
{

    public function index(){

    }
}
```

Ou

```
public function index()
{
    $this->render();
}
```

b. Afficher une vue spécifique

(exemple une vue nommée « custom.ctp » dans le dossier de Templates « Article »)

```
public function index()
{
    $this->render('custom');
}
```

c. Redirection

```
public function index()
{
    return $this->redirect(
        ['controller' => 'Orders', 'action' => 'confirm']
    );
}
```

Redirection vers la page appelante

```
return $this->redirect($this->referer());
```

Url

```
return $this->redirect('http://www.google.com');
```

Redirection vers une action du même contrôleur

```
$this->setAction('index');
```

d. Définir le layout de la vue

```
    public function index(){
        $this->viewBuilder()->layout('mylayout');
    }
```

e. Passage de paramètre à la vue

avec la méthode « set » depuis le contrôleur

```
<?php

namespace App\Controller;


class ArticleController extends AppController
{

    public function index($id){
        $this->set('id',$id);
    }
}
```

Dans la page de contenu

```
<p>id : <?= $id; ?></p>
```

9. Behavior

Exemple création du Behavior « Sluggable » permettant de définir le slug avant l'ajout d'un article.Dans le dossier « src\Model\Behavior »… ajout de « SluggableBehavior.php »

```php
<?php

namespace App\Model\Behavior;

use Cake\Datasource\EntityInterface;
use Cake\Event\Event;
use Cake\ORM\Behavior;
use Cake\ORM\Entity;
use Cake\Utility\Inflector;

class SluggableBehavior extends Behavior
{
    protected $_defaultConfig = [
        'field' => 'title',
        'slug' => 'slug',
        'replacement' => '-',
    ];

    public function slug(Entity $entity)
    {
        $config = $this->config();
        $value = $entity->get($config['field']);
        $entity->set($config['slug'], strtolower(Inflector::slug($value,
$config['replacement'])));
    }

    public function beforeSave(Event $event, EntityInterface $entity)
    {
        $this->slug($entity);
    }
}
```

Utilisation du **Behavior**

```php
<?php
namespace App\Model\Table;
use App\Model\Entity\Article;
use Cake\ORM\Query;
use Cake\ORM\RulesChecker;
use Cake\ORM\Table;
use Cake\Validation\Validator;

class ArticleTable extends Table
{
    public function initialize(array $config)
    {
        parent::initialize($config);

        $this->table('article');
        $this->displayField('title');
        $this->primaryKey('id');

        $this->addBehavior('Sluggable');
        $this->addBehavior('Timestamp');
    }
    // etc.
}
```

Le slug est créé automatiquement lors de l'ajout d'un article

10. Composants

Documentation

- Authentification
- CookieComponent
- Cross Site Request Forgery
- FlashComponent
- SecurityComponent (Sécurité)
- Pagination
- Request Handling (Gestion des requêtes)

a. Chargement

```
class PostsController extends AppController
{
    public function initialize()
    {
        parent::initialize();
        $this->loadComponent('Auth', [
            'authorize' => ['controller'],
            'loginAction' => ['controller' => 'Users', 'action' =>
'login']
        ]);
        $this->loadComponent('Cookie', ['expiry' => '1 day']);
    }

}
```

b. Configuration

```
public function beforeFilter(Event $event)
{
    $this->Auth->config('authorize', ['controller']);
    $this->Auth->config('loginAction', ['controller' => 'Users', 'action'
=> 'login']);

    $this->Cookie->config('name', 'CookieMonster');
}
```

c. Utilisation

```
class PostsController extends AppController
{
    public function initialize()
    {
        parent::initialize();
        $this->loadComponent('Flash');
        $this->loadComponent('Cookie');
    }

    public function delete()
    {
        if ($this->Post->delete($this->request->data('Post.id'))) {
        $this->Flash->success('Post deleted.');
        return $this->redirect(['action' => 'index']);
    }
}
```

11. Model

Documentation

Validation de données par défaut

```php
class ArticleTable extends Table
{

    /**
     * Initialize method
     *
     * @param array $config The configuration for the Table.
     * @return void
     */
    public function initialize(array $config)
    {
        parent::initialize($config);

        $this->table('article');
        $this->displayField('title');
        $this->primaryKey('id');

        $this->addBehavior('Timestamp');
    }

    /**
     * Default validation rules.
     *
     * @param \Cake\Validation\Validator $validator Validator instance.
     * @return \Cake\Validation\Validator
     */
    public function validationDefault(Validator $validator)
    {
        $validator
            ->integer('id')
            ->allowEmpty('id', 'create');

        $validator
            ->allowEmpty('slug');

        $validator
            ->requirePresence('title', 'create')
            ->notEmpty('title','Un titre est nécessaire.');

        $validator
            ->requirePresence('body', 'create')
            ->notEmpty('body');

        return $validator;
    }
}
```

Validation HTML5 sur le formulaire + validation du modèle avant ajout

12. Authentification

a. Controllers

UsersController

```php
<?php
namespace App\Controller;

use App\Controller\AppController;

class UsersController extends AppController
{

    public function index()
    {
        $users = $this->paginate($this->Users);
        $this->set(compact('users'));
        $this->set('_serialize', ['users']);
    }

    public function view($id = null)
    {
        $user = $this->Users->get($id, [
            'contain' => ['Articles']
        ]);

        $this->set('user', $user);
        $this->set('_serialize', ['user']);
    }

    public function add()
    {
        $user = $this->Users->newEntity();
        if ($this->request->is('post')) {
            $user = $this->Users->patchEntity($user, $this->request->data);
            if ($this->Users->save($user)) {
                $this->Flash->success(__('The user has been saved.'));
                return $this->redirect(['action' => 'index']);
            } else {
                $this->Flash->error(__('The user could not be saved. Please, try
again.'));
            }
        }
        $this->set(compact('user'));
        $this->set('_serialize', ['user']);
    }
    public function login()
    {
        if ($this->request->is('post')) {
            $user = $this->Auth->identify();
            if ($user) {
                $this->Auth->setUser($user);
                return $this->redirect($this->Auth->redirectUrl());
            }
            $this->Flash->error(__('Invalid username or password, try again'));
        }
    }
    public function logout()
    {
        return $this->redirect($this->Auth->logout());
```

```
    }
}
```

AppController

```php
<?php

namespace App\Controller;

use Cake\Controller\Controller;
use Cake\Event\Event;


class AppController extends Controller
{

    public function initialize()
    {
        parent::initialize();

        $this->loadComponent('Flash');
        $this->loadComponent('Auth', [
            'authorize' => ['Controller'],
            'loginRedirect' => [
                'controller' => 'Articles',
                'action' => 'index'
            ],
//            'logoutRedirect' => [
//                'controller' => 'Pages',
//                'action' => 'display',
//                'home'
//            ]
        ]);
    }

    public function isAuthorized($user)
    {
        // Admin peuvent accéder à chaque action
        if (isset($user['role']) && $user['role'] === 'admin') {
            return true;
        }

        // Par défaut refuser
        return false;
    }

    public function beforeFilter(Event $event)
    {
        $this->Auth->allow(['index', 'view', 'display']);
    }
}
```

Chargement et configuration du component « Auth »

Autorisation ..par défaut tous les contrôleurs demandent à être authentifié

Dans « beforeFilter » des contrôleurs on définit les actions accessibles

Dans les autres contrôleurs

```php
<?php
namespace App\Controller;

use App\Controller\AppController;

class ArticlesController extends AppController
{
    // …
    public function isAuthorized($user)
    {
        // Tous les utilisateurs enregistrés peuvent ajouter des articles
        if ($this->request->action === 'add') {
            return true;
        }

        // Le propriétaire d'un article peut l'éditer et le supprimer
        if (in_array($this->request->action, ['edit', 'delete'])) {
            $articleId = (int)$this->request->params['pass'][0];
            if ($this->Articles->isOwnedBy($articleId, $user['id'])) {
                return true;
            }
        }
        return parent::isAuthorized($user);
    }
}
```

Override de isAuthorized

b. Table

```php
<?php
namespace App\Model\Table;

use App\Model\Entity\Article;
use Cake\ORM\Query;
use Cake\ORM\RulesChecker;
use Cake\ORM\Table;
use Cake\Validation\Validator;

class ArticlesTable extends Table
{
    // …

    public function isOwnedBy($articleId, $userId)
    {
        return $this->exists(['id' => $articleId, 'user_id' => $userId]);
    }
}
```

Dans la table on crée une fonction permettant de vérifier si l'utilisateur connecté est l'auteur de l'article

c. Templates

Connexion (« login.ctp »)

```
<div class="users form">
<?= $this->Flash->render('auth') ?>
<?= $this->Form->create() ?>
    <fieldset>
        <legend><?= __("Merci de rentrer vos nom d'utilisateur et mot de
passe") ?></legend>
        <?= $this->Form->input('username') ?>
        <?= $this->Form->input('password') ?>
    </fieldset>
<?= $this->Form->button(__('Se Connecter')); ?>
<?= $this->Form->end() ?>
</div>
```

Inscription (« add.ctp »)

```
<div class="users form">
<?= $this->Form->create($user) ?>
    <fieldset>
        <legend><?= __('Ajouter un utilisateur') ?></legend>
        <?= $this->Form->input('username') ?>
        <?= $this->Form->input('password') ?>
        <?= $this->Form->input('role', [
            'options' => ['admin' => 'Admin', 'author' => 'Author']
        ]) ?>
    </fieldset>
<?= $this->Form->button(__('Ajouter')); ?>
<?= $this->Form->end() ?>
</div>
```

13. Sécurity

Documentation

14. Plugins

Documentation