How To Use CakePHP to Create a Small Web Application

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: *http://faq.asphosthelpdesk.com/article.php?id=250*

CakePHP is a powerful and robust PHP framework built around the Model-View-Controller (MVC) programming paradigm. In addition to the flexible way you can use it to build your application, it provides a basic structure for organizing files and database table names - keeping everything consistent and logical.

In the last tutorial, we started to create a small application that would perform some basic CRUD (create, read, update, delete) operations on our database. So far we managed to read the data (the posts) using a Model (**Post**), request it using a Controller (**PostsController**) and created a couple of Views to display the information. In this tutorial we will finish the application by implementing the other CRUD operations, namely create, update, and delete. For this, we will mainly work inside our existing Controller and use the Model we created to access our data.

Adding Data

After seeing how we can read the posts in the table, let"s see how we can add new posts. Before creating our Controller method, let"s first make sure the Controller has all the components we need. In the last tutorial we"ve included only the **Form** helper:

```
public $helpers = array('Form');
```

Let"s add now the HTML and Session helpers as well by adding to the array:

```
public $helpers = array('Form', 'Html', 'Session');
```

Additionally, let"s include the Session component. Below the line we just edited, add the following:

```
public $components = array('Session');
```

Now let"s go ahead and create a View that will house our form to add a new post. This will use the Form helper we just included to make things much easier. So create a file in the*app/View/Posts/*folder called **add.ctp**. Inside, paste the following code:

```
<h1>Add Post</h1>
<?php
echo $this->Form->create('Post');
echo $this->Form->input('title');
echo $this->Form->input('body', array('rows' => '3'));
echo $this->Form->end('Save');
?>
```

As you can see, we can access the Form helper straight from the View and it allows us to quickly draw our Form. One important thing here is that if the **create()** method is not passed any parameters, it will assume that the form submits to itself (the Controller method loading this View - that we will create in a second). The

input() method as you can see is quite self-explanatory but one cool thing is that it will generate the form elements that match our data in the table. So let"s save the file and create our method in the **PostsController** to take care of this.

Add the following code right below the **view()** method we created earlier in the PostsController:

```
public function add() {
        if ($this->request->is('post')) {
            $this->Post->create();
            $post_data = $this->request->data;
            if ($this->Post->save($post_data)
 ) {
                $this->Session->setFlash(__('
 New post saved successfully to the database'));
                return $this->redirect(array('action' => 'index'));
            }
            $this->Session->setFlash(__('
 Unable to save the post to the database.'));
        }
}
```

This function (named as the View it corresponds to) first checks if the there is a request of the type POST sent to it and tries to insert the new data into the table using the **Post** model if it is. If successful, it will set a flashdata to the current user session and redirect to the **index()** method (the one displaying all the posts) which will then display a positive confirmation message as well. If not, it will set an error message instead. You can go ahead and test it out a t **www.example.com/project/posts/ add**. Fill in the form and it should save the new post, redirect you to the posts index page and display a confirmation message.

Since **PostsController** was created by extending the default CakePHP Controller, we have access to a lot of goodies, such as the request object. Using that, we can check what kind of HTTP request is being made and get access to the POST data as well. Additionally, we get access to the **redirect()** method by which we can quickly redirect the user to another method or Controller.

Validating Data

As I am sure you don"t want posts being submitted without any information, let"s set up a quick rule in our **Post** Model to make sure it forces the user to set a title when submitting a new post. Inside the **Post** Model, add the following property:

```
public $validate = array('title' => array('rule' => 'notEmpty'));
```

This will make sure that the title field cannot be empty. Save the Model file and try adding a new post without filling up the title field. You"ll notice that now you are required to fill the title field but not necessarily the body field.

HTML Helper?

The reason we included the **HTML helper** into the **PostsController** is to show you how to put a link on a

page in the CakePHP way. So let"s open the**index.ctp**View located in the*app/View/Posts/*folder and let"s add the following code after the H1 tag:

```
<?php echo $this->Html->link(
    'Add Post',
    array('controller' => 'posts', 'action' => 'add')
); ?>
```

This will output a link with the anchor **Add Post** that will go to the **add()**method of the **PostsController**. If you want, you can also further edit the file and using the same technique, turn the post titles on this page into links to their respective pages. So instead of:

```
<?php echo $post['Post']['title']; ?>
```

You can put:

```
<?php echo $this->Html->link($post['Post'
 ]['title'], array('controller' => 'posts', 'action' => 'view',
$post['Post']['id'])); ?>
```

More information about the HTML helper you can find here.

Editing Data

Now that we saw how to create new posts, let"s see how to edit existing ones. Let"s create the View again next to where we placed the **add.ctp**View and call it **edit.ctp**. Inside, paste the following:

```
<h1>Edit Post</h1>
<?php
    echo $this->Form->create('Post');
    echo $this->Form->input('title');
    echo $this->Form->input('body', array('rows' => '3'));
    echo $this->Form->input('id', array('type' => 'hidden'));
    echo $this->Form->end('Save');
?>
```

The main difference between the **edit.ctp** and **add.ctp** Views is that in the former, we also included the ID of the post as a hidden input so that CakePHP knows you want to edit and not add a new post. Save the file and exit. Next, we create the **edit()**method in the PostsController:

```
public function edit($id = null) {
    if (!$id) {
        throw new NotFoundException(__('Post is not valid!'));
    }
    $post = $this->Post->findById($id);
    if (!$post) {
        throw new NotFoundException(__('Post is not valid!'));
    }
```

```
    if ($this->request->is('post') || $this->request->is('put')) {
        $this->Post->id = $id;
        $post_data = $this->request->data;
        if ($this->Post->save($post_data)
 ) {
            $this->Session->setFlash(__('
 Your post has been updated.'));
            return $this->redirect(array('action' => 'index'));
        }
        $this->Session->setFlash(__('
 Unable to update your post.'));
    }
    if (!$this->request->data) {
        $this->request->data = $post;
    }
}
```

This action first makes sure that the user is trying to access a valid post by checking the ID for validity and whether it exists in the database. Like in the **add()** method, it checks whether the request is POST and tries to update the post in the database if it is. If no data is present in the request object, it will fill the form elements with the data existing in the database. And as in the **add()** action, it then redirects the user to the **index()**method and displays a confirmation message. So go ahead and try it out.

You can also modify the **index.ctp** View and add a link to edit individual posts. Add the following code after the *Created* field:

```
<?php echo $this->Html->link('Edit', array('action' => 'edit',
$post['Post']['id'])); ?>
```

Deleting Data

The last thing we need to do is allow users to delete the posts. So let"s add the following action to the PostsController:

```
public function delete($id) {
   if ($this->request->is('post')) {
    if ($this->Post->delete($id)) {
      $this->Session->setFlash(__('
 The post number %s has been deleted.', h($id)));
      return $this->redirect(array('action' => 'index'));
    }
  }
}
```

This method first throws an exception if the request is of a GET type. Then it uses the **Post** Model like in the actions above but this time deletes the row in the table with the ID supplied in the request. Lastly, it sets a message to the user and redirects to the **index()** method where the message is displayed.

To trigger this **delete()** method, let"s edit the **index.ctp** View and use the **postLink()** function to output a small Form that will send the POST request to delete the table row. It will use javascript to add a confirmation

alert box and will then delete the post. Inside the **index.ctp** file after the edit link you can add the following:

```
<?php echo $this->Form->postLink(
                'Delete',
                array('action' => 'delete', $post['Post']['id']),
                array('confirm' => 'Are you sure you want to delete this
post?'));
?>
```

Save the file and try it out. Now you should be able to delete posts as well.

As a little recap if you followed along, your classes should look like this now:

PostsController.php - The Controller

```
class PostsController extends AppController {
  public $helpers = array('Form', 'Html', 'Session');
  public $components = array('Session');

  public function index() {
      $this->set('posts', $this->Post->find('all'));
  }

  public function view($id = null) {
        $post = $this->Post->findById($id);
        $this->set('post', $post);
  }

  public function add() {
        if ($this->request->is('post')) {
            $this->Post->create();
            $post_data = $this->request->data;
            if ($this->Post->save($post_data)
 ) {
                $this->Session->setFlash(__('
 New post saved successfully to the database'));
                return $this->redirect(array('action' => 'index'));
            }
            $this->Session->setFlash(__('
 Unable to save the post to the database.'));
        }
  }
  public function edit($id = null) {
    if (!$id) {
        throw new NotFoundException(__('Post is not valid!'));
    }
    $post = $this->Post->findById($id);
    if (!$post) {
        throw new NotFoundException(__('Post is not valid!'));
    }
    if ($this->request->is('post') || $this->request->is('put')) {
```

```
        $this->Post->id = $id;
        $post_data = $this->request->data;
        if ($this->Post->save($post_data)
 ) {
            $this->Session->setFlash(__('
 Your post has been updated.'));
            return $this->redirect(array('action' => 'index'));
        }
        $this->Session->setFlash(__('
 Unable to update your post.'));
    }
    if (!$this->request->data) {
        $this->request->data = $post;
    }
  }
 public function delete($id) {
   if ($this->request->is('post')) {
    if ($this->Post->delete($id)) {
      $this->Session->setFlash(__('
 The post number %s has been deleted.', h($id)));
      return $this->redirect(array('action' => 'index'));
    }
   }
  }
}
```

Post.php - The Model

```
class Post extends AppModel {
  public $validate = array('title' => array('rule' => 'notEmpty'));
}
```

Conclusion

We"ve seen in this short tutorial series how easy it is to work with CakePHP to perform CRUD operations on your data. We learned how to read and display information, how to edit and delete it, and how to add a new one. Furthermore, an important lesson to learn has been that following conventions set in place by CakePHP is highly recommended as it makes your life much easier. I encourage you to take this small application you created and play with it to build something bigger. To do this you should read more information about CakePHP components and helpers.