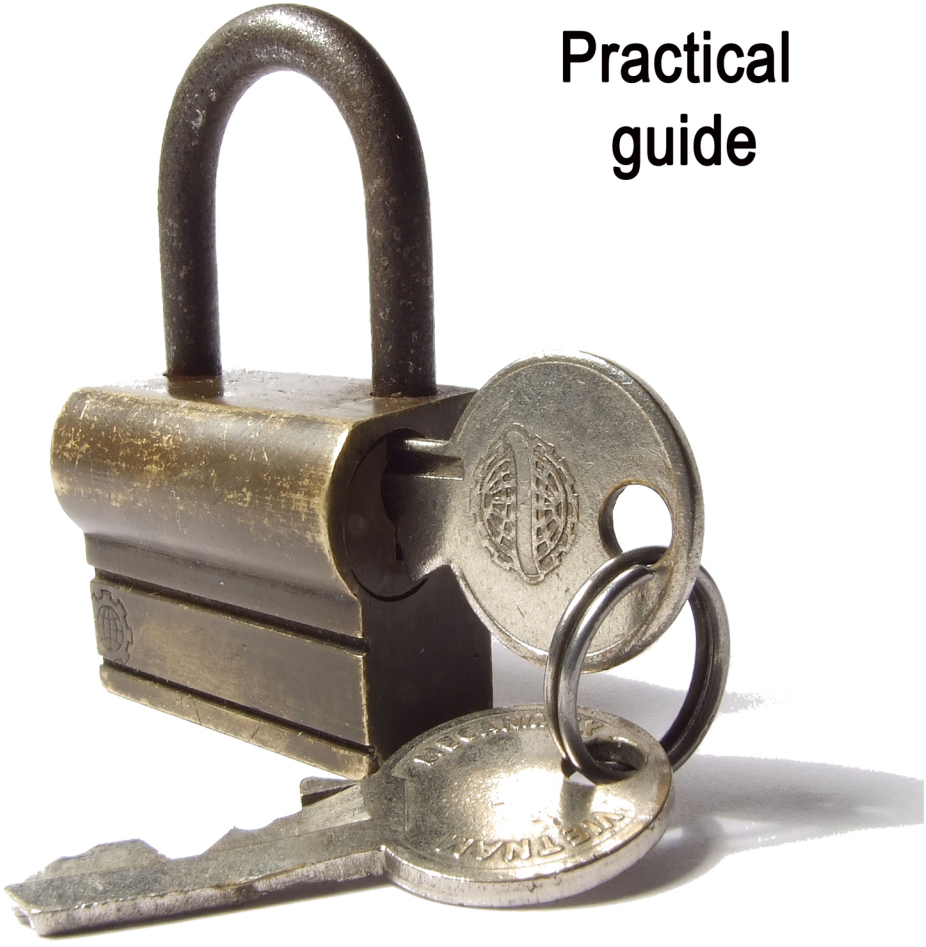
Practical guide



CakePHP User Authentication

- rrd -

CakePHP User Authentication

rrd

This book is for sale at
http://leanpub.com/CakePHPUserAuthentication

This version was published on 2014-12-20

Tweet This Book!

Please help rrd by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just bought CakePHP User Authentication e-book. #CakePHP
https://leanpub.com/CakePHPUserAuthentication

The suggested hashtag for this book is #cakephp.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#cakephp

Contents

Introduction

Nearly all web application needs some kind of user handling functionality. User handling could be simple like a rod, or a really sophisticated and complicated system. Sometimes we will have only one user, who could add new things, edit and delete them in our application. Sometimes we have to handle many users, different permissions, roles and groups and do it as comfortable as possible. Luckily CakePHP offers a great help in our work by its Auth component.

About the Book

I wrote this book for CakePHP developers who want to know how to handle users and user related functions. For people who do not want to reinvent the wheel but wants flexibility and like to write their own code using best practices.

The examples in this book use CakePHP, but the presented ideas themselves are not framework or language specific.

Why I Wrote This Book

I started to use CakePHP at version 1.1. The framework is wonderful, and improving nicely, with an open and helpful staff and community. It helped me to became a better programmer.

When I started to code, I did everything from scratch. I did not know about programming patterns, utilities, or libraries. I was able to build up middle size systems this way.

After a while, I decided that mixing application logic and presentation logic has more cons than pros, so I started to use Smarty[1]. Smarty was a great help. In time it helped me

[1] http://smarty.net/

to see that my best practices were really my worst practices. I knew I needed something more.

I realized there are a few features I need in most of my web applications. I started to think about how I code and after some time came up with an extremely simple and dull framework, without really knowing that frameworks exist.

That was when I heard about MVC pattern. At first it seemed like unnecessary complications in the code, but I wanted to give it a try anyway. When I tried to understand MVC, I found some information about frameworks. I tried CodeIgniter[2], and then CakePHP[3].

The first bite of CakePHP was awful. Especially because I was (and am) a big fan of `bake` auto code generation. I thought that the whole framework just saved so much time and produced a much clearer and more maintainable code.

CakePHP has a lot of built-in functionality, including authentication. It is clear, well written, easy to use. Still we tend to forget what we did last time, how we solved different problems. I think many of you have had similar experiences. Let's try to shorten our learning curve.

I hope this book will help you and that my suggestions can save you some time.

[2] http://ellislab.com/codeigniter
[3] http://cakephp.org

My Development Environment

I tried to use code examples that are independent from the environment but, as we all know, this is impossible. With that in mind, this is my system and the software I'm using.

- Ubuntu 14.04
- PHP 5.5.9-1ubuntu4.4
- CakePHP 2.5
- MySQL 5.5.38
- Komodo Edit 8.5
- phpMyAdmin 4.0.10

Who This Book Is For

This book is for novice and intermediate programmers.

It assumes that you have a general understanding of PHP and object-oriented programming (OOP).

It's good if you are already familiar with CakePHP. But even if you're not, you will probably still be able to understand most of the principles and codes.

Prerequisites

I assume that you have already successfully installed CakePHP[4], created a database for your application and set debug level to `1` or `2`.

[4] http://cakephp.org

About the Author

rrd started to code for the web in plain HTML in 1998. As the web evolved, he turned to PHP, then to Javascript. As mobile technologies arrived, he started to play with Java for android development.

He's a big fan of CakePHP, jQuery, Prototype, and Scriptaculous frameworks, open source, and pizza. All of them are there in his web development.

Thanks

I would like to thank you for spending your time reading this book.

I also should say thank you to the core CakePHP team and everybody else who's contributed anything, small or big, to this wonderful framework.

Finally, special thanks go to Italy for *Pizza* and to Hungary for *Turo Rudi* - both are essential for web development.

My E-books on CakePHP

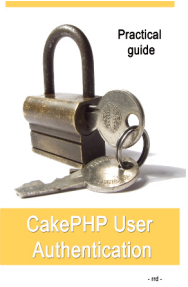
CakePHP Unit Testing

CakePHP Unit Testing

Download from Leanpub[5]

[5] https://leanpub.com/cakephpunittesting

CakePHP User Authentication



CakePHP User Authentication

Download from Leanpub[6]

[6] https://leanpub.com/CakePHPUserAuthentication

My CakePHP Plugins

Protection against brute force attacks rBruteFore[7]

[7] https://github.com/rrd108/rBruteForce

Model Preparation

Creating User Table

We will need a user table in our database. Let's create it.

```
CREATE TABLE `users` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `email` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `email` (`email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREM\
ENT=1;
```

As a start we have only 3 columns in our table, `id`, `email` and `password`. The password will be stored encrypted, so the field should be enough big.

Baking User

Our next step is to create a basic skeleton for our user handling functionality. For this we should create our user model, view and controller. As always the easiest way to

use CakePHP's automatic code generation tool and `bake` them.

Open a terminal and step into your application's root directory. Step into `app/Console` folder and type `./cake bake`". This will start CakePHP's interactive `bake` console. `bake` console is a smart and handy tool and it is self explanatory. If you do not familiar yet, we will walk through the whole process, what normally takes 20 seconds.

Baking User Model

After starting `bake` console in your terminal window you will see this.

```
Welcome to CakePHP v2.4.0 Console
---------------------------------------------------
App : app
Path: /home/rrd/public_html/blog/app/
---------------------------------------------------
Interactive bake Shell
---------------------------------------------------
[D]atabase Configuration
[M]odel
[V]iew
[C]ontroller
[P]roject
[F]ixture
[T]est case
[Q]uit
```

First we `bake` our user model. Press `m` after the question "What would you like to `bake`?" appears. By this we started to `bake` our user model.

```
What would you like to bake? (D/M/V/C/P/F/T/Q)
> m
-----------------------------------------------
`bake` Model
Path: /home/rrd/public_html/blog/app/Model/
-----------------------------------------------
```

Next we should choose which database schema should be used. Just press enter and it will select default.

```
Use Database Config: (default/test)
[default] >
```

After this we should tell to the `bake` script what model we want to create. Use the number in front of `User`. As in `core.php` I choose to use database for my sessions I should select `2`. If you did not changed this and left as default your User option will be `1`. If you have more database tables than choose the number in the beginning of the User row.

```
Possible Models based on your current database:
1. CakeSession
2. User
Enter a number from the list above,
type in the name of another model, or 'q' to exi\
t
[q] > 2
```

You should set your display field. CakePHP will use it in select boxes. We do not need it in this phase so you could press `n` and let the `bake` script know that we will go without any display filed.

```
A displayField could not be automatically detect\
ed
would you like to choose one? (y/n)
> n
```

Perhaps we do not want to have useless data anywhere, so we will use validation.

> Validation is not a security check. Never forget to sanitize your input.

```
Would you like to supply validation criteria
for the fields in your model? (y/n)
[y] >
```

After this the `bake` script will ask validation criteria for each fields.

For the field `id`, we do not need any validation criteria, as we will just let the database to auto increment its value. So you could just press enter and accept what the `bake` script offers as default.

```
Field: id
Type: integer
-----------------------------------------------
Please select one of the following validation op\
tions:
-----------------------------------------------
 1. alphanumeric              18. maxlength
 2. between                   19. mimetype
 3. blank                     20. minlength
 4. boolean                   21. money
 5. cc                        22. multiple
 6. comparison                23. naturalnumber
 7. custom                    24. notempty
 8. date                      25. numeric
 9. datetime                  26. phone
10. decimal                   27. postal
11. email                     28. range
12. equalto                   29. ssn
```

```
13. extension                 30. time
14. filesize                  31. uploaderror
15. inlist                    32. url
16. ip                        33. userdefined
17. luhn                      34. uuid

35 - Do not do any validation on this field.
-------------------------------------------------
... or enter in a valid regex validation string.

[35] >
```

Next is the field `email`, what should looks like an email address. Validation do not check if it is a real email address.Email addresses has their own built-in validation criteria so we choose this.

```
Field: email
Type: string
-------------------------------------------------
Please select one of the following validation op\
tions:
-------------------------------------------------
 1. alphanumeric              18. maxlength
 2. between                   19. mimetype
 3. blank                     20. minlength
 4. boolean                   21. money
 5. cc                        22. multiple
 6. comparison                23. naturalnumber
```

```
 7. custom                    24. notempty
 8. date                      25. numeric
 9. datetime                  26. phone
10. decimal                   27. postal
11. email                     28. range
12. equalto                   29. ssn
13. extension                 30. time
14. filesize                  31. uploaderror
15. inlist                    32. url
16. ip                        33. userdefined
17. luhn                      34. uuid

35 - Do not do any validation on this field.
-------------------------------------------------
... or enter in a valid regex validation string.

[11] >
```

As we could have multiple validation criteria the `bake` script asks if we would like to add an other. This time just press enter at accept the default answer, no.

```
Would you like to add another validation rule? (\
y/n)
[n] >
```

For the password field we will have more criteria like it should be more than 5 characters, etc. At this point we do not have to worry about this, we will add extra validation criteria later. So just accept default answer.

```
Field: password
Type: string
-------------------------------------------------
Please select one of the following validation op\
tions:
-------------------------------------------------
 1. alphanumeric              18. maxlength
 2. between                   19. mimetype
 3. blank                     20. minlength
 4. boolean                   21. money
 5. cc                        22. multiple
 6. comparison                23. naturalnumber
 7. custom                    24. notempty
 8. date                      25. numeric
 9. datetime                  26. phone
10. decimal                   27. postal
11. email                     28. range
12. equalto                   29. ssn
13. extension                 30. time
14. filesize                  31. uploaderror
15. inlist                    32. url
16. ip                        33. userdefined
17. luhn                      34. uuid

35 - Do not do any validation on this field.
-------------------------------------------------
... or enter in a valid regex validation string.

[24] >
```

```
Would you like to add another validation rule? (\
y/n)
[n] >
```

We do not have more fields, so `bake` moves forward and asks about model relationships. In this example app we do not have any model relations so we could press `n`.

```
Would you like to define model associations
(hasMany, hasOne, belongsTo, etc.)? (y/n)
[y] > n
```

Finally `bake` gives us a summary. Press enter.

```
---------------------------------------------------
The following Model will be created:
---------------------------------------------------
Name:       User
DB Table:   `blog`.`users`
Validation: Array
(
    [email] => Array
        (
            [email] => email
        )

    [password] => Array
        (
```

```
            [notempty] => notempty
        )

)

-------------------------------------------------
Look okay? (y/n)
[y] >
```

After this CakePHP try to create our User Model. If it is successful you will see something like this.

```
Baking model class for User...

Creating file /home/rrd/public_html/blog/app/Mod\
el/User.php
Wrote `/home/rrd/public_html/blog/app/Model/User\
.php`
```

Last question is about unit testing. This time we do not need unit tests, so press `n`.

Unit testing is an extremely useful tool in software development. Please read my book CakePHP Unit Testing if you are interested.

```
PHPUnit is not installed. Do you want to bake un\
it test files anyway? (y/n)
[y] > n
```

Now you could open your automatically generated /app/-Model/User.php file.

```
<?php
App::uses('AppModel', 'Model');
/**
 * User Model
 *
 */
class User extends AppModel {

/**
 * Validation rules
 *
 * @var array
 */
  public $validate = array(
    'email' => array(
      'email' => array(
        'rule' => array('email'),
        //'message' => 'Your custom message here\
',
        //'allowEmpty' => false,
        //'required' => false,
        //'last' => false, // Stop validation af\
```

```
ter this rule
        //'on' => 'create', // Limit validation \
to 'create' or 'update' operations
      ),
    ),
    'password' => array(
      'notempty' => array(
        'rule' => array('notempty'),
        //'message' => 'Your custom message here\
',
        //'allowEmpty' => false,
        //'required' => false,
        //'last' => false, // Stop validation af\
ter this rule
        //'on' => 'create', // Limit validation \
to 'create' or 'update' operations
      ),
    ),
  );
}
?>
```

Controller Preparation

Create a basic controller with CRUD functions is also a really simple and straightforward process. It will take an other 20 seconds.

If you did not closed your `bake` terminal you could just continue. If you did please start it again. This time you should press `c` at the first step. By that the `bake` script will auto generate your controller code.

```
1  ---------------------------------------------------
2  Interactive Bake Shell
3  ---------------------------------------------------
4  [D]atabase Configuration
5  [M]odel
6  [V]iew
7  [C]ontroller
8  [P]roject
9  [F]ixture
10 [T]est case
11 [Q]uit
12 What would you like to Bake? (D/M/V/C/P/F/T/Q)
13 > c
```

As we did for our model we should choose which database

to use, and what controller we want to build. Chose defaut database and Users controller.

```
1  -----------------------------------------------
2  Bake Controller
3  Path: /home/rrd/public_html/blog/app/Controller/
4  -----------------------------------------------
5  Use Database Config: (default/test)
6  [default] >
7  Possible Controllers based on your current datab\
8  ase:
9  -----------------------------------------------
10  1. CakeSessions
11  2. Users
12 Enter a number from the list above,
13 type in the name of another controller, or 'q' t\
14 o exit
15 [q] > 2
```

The `bake` script could build your controller interactively or silently. I prefer to use the interactive shell, as there is a good chance that we want to change at least one thing from default to something else. So press enter and accept the default answer.

```
-----------------------------------------------
Baking UsersController
-----------------------------------------------
Would you like to build your controller interact\
ively? (y/n)
[y] >
```

Scaffolding is an extremely good way to have a working basic web application in a few minutes. As we will change the controller code soon chose `n`. By this the `bake` script will generate the code and will not use scaffolds.

```
Would you like to use dynamic scaffolding? (y/n)
[n] >
```

Next `bake` asks about creating basic CRUD functions. Choose `y`. This will generate controller functions for adding, viewing, editing and listing users.

```
Would you like to create some basic class method\
s
(index(), add(), view(), edit())? (y/n)
[n] > y
```

Admin routing helps you differentiate between regular users and admins. It means you will have different CRUD functions for admins and for regular users in your controller. It is not a must have, you could code the same thing without admin routes, but this is a more clear and maintainable way. So press `y` for admin routing.

```
Would you like to create the basic class methods\
 for admin routing? (y/n)
[n] > y
```

The `bake` script will add basic helpers to your controller. Press enter and accept default answer. If we will need more helpers we would add them in our controller code later.

```
Would you like this controller to use other help\
ers
besides HtmlHelper and FormHelper? (y/n)
[n] >
```

Next question is about components. At this step we could just accept the default.

```
Would you like this controller to use other comp\
onents
besides PaginatorComponent? (y/n)
[n] >
```

We need to show flash messages, so choose yes for the next question.

```
Would you like to use Session flash messages? (y\
/n)
[y] >
```

As we choose to use admin routes we should edit `/app/Config/core.php` and switch it on. You could change the routing prefix from admin to anything you like.

```
-------------------------------------------------
You need to enable Configure::write('Routing.pre\
fixes', array('admin')) in /app/Config/core.php \
to use prefix routing.
What would you like the prefix route to be?
Example: www.example.com/admin/controller
Enter a routing prefix:
[admin] >
```

Finally bake gives us a summary. Press enter.

```
-------------------------------------------------
The following controller will be created:
-------------------------------------------------
Controller Name:
        Users
Components:
        Paginator
-------------------------------------------------
Look okay? (y/n)
[y] >

Baking controller class for Users...

Creating file /home/rrd/public_html/blog/app/Con\
troller/UsersController.php
```

Last question is about unit testing. This time we do not need unit tests, so press n.

Unit testing is an extremely useful tool in software development. Please read my book CakePHP Unit Testing if you are interested.

```
Wrote `/home/rrd/public_html/blog/app/Controller\
/UsersController.php`
PHPUnit is not installed. Do you want to bake un\
it test files anyway? (y/n)
[y] > n
```

Now `app/Controller/UsersController.php` is generated and you could open it with your favorite text editor.

```
<?php
App::uses('AppController', 'Controller');
/**
 * Users Controller
 *
 * @property User $User
 * @property PaginatorComponent $Paginator
 */
class UsersController extends AppController {

/**
 * Components
 *
 * @var array
 */
```

```
16    public $components = array('Paginator');
17
18  /**
19   * index method
20   *
21   * @return void
22   */
23    public function index() {
24      $this->User->recursive = 0;
25      $this->set('users', $this->Paginator->pagina\
26  te());
27    }
28
29  /**
30   * view method
31   *
32   * @throws NotFoundException
33   * @param string $id
34   * @return void
35   */
36    public function view($id = null) {
37      if (!$this->User->exists($id)) {
38        throw new NotFoundException(__('Invalid us\
39  er'));
40      }
41      $options = array('conditions' => array('User\
42  .' . $this->User->primaryKey => $id));
43      $this->set('user', $this->User->find('first'\
44  , $options));
```

```
45    }
46
47  /**
48   * add method
49   *
50   * @return void
51   */
52    public function add() {
53      if ($this->request->is('post')) {
54        $this->User->create();
55        if ($this->User->save($this->request->data\
56  )) {
57          $this->Session->setFlash(__('The user ha\
58  s been saved.'));
59          return $this->redirect(array('action' =>\
60   'index'));
61        } else {
62          $this->Session->setFlash(__('The user co\
63  uld not be saved. Please, try again.'));
64        }
65      }
66    }
67
68  /**
69   * edit method
70   *
71   * @throws NotFoundException
72   * @param string $id
73   * @return void
```

```
74   */
75    public function edit($id = null) {
76      if (!$this->User->exists($id)) {
77        throw new NotFoundException(__('Invalid us\
78  er'));
79      }
80      if ($this->request->is('post') || $this->req\
81  uest->is('put')) {
82        if ($this->User->save($this->request->data\
83  )) {
84          $this->Session->setFlash(__('The user ha\
85  s been saved.'));
86          return $this->redirect(array('action' =>\
87   'index'));
88        } else {
89          $this->Session->setFlash(__('The user co\
90  uld not be saved. Please, try again.'));
91        }
92      } else {
93        $options = array('conditions' => array('Us\
94  er.' . $this->User->primaryKey => $id));
95        $this->request->data = $this->User->find('\
96  first', $options);
97      }
98    }
99
100 /**
101  * delete method
102  *
```

```
103   * @throws NotFoundException
104   * @param string $id
105   * @return void
106   */
107    public function delete($id = null) {
108      $this->User->id = $id;
109      if (!$this->User->exists()) {
110        throw new NotFoundException(__('Invalid us\
111  er'));
112      }
113      $this->request->onlyAllow('post', 'delete');
114      if ($this->User->delete()) {
115        $this->Session->setFlash(__('The user has \
116  been deleted.'));
117      } else {
118        $this->Session->setFlash(__('The user coul\
119  d not be deleted. Please, try again.'));
120      }
121      return $this->redirect(array('action' => 'in\
122  dex'));
123    }
124
125  /**
126   * admin_index method
127   *
128   * @return void
129   */
130    public function admin_index() {
131      $this->User->recursive = 0;
```

```
132     $this->set('users', $this->Paginator->pagina\
133 te());
134   }
135 
136 /**
137  * admin_view method
138  *
139  * @throws NotFoundException
140  * @param string $id
141  * @return void
142  */
143   public function admin_view($id = null) {
144     if (!$this->User->exists($id)) {
145       throw new NotFoundException(__('Invalid us\
146 er'));
147     }
148     $options = array('conditions' => array('User\
149 .' . $this->User->primaryKey => $id));
150     $this->set('user', $this->User->find('first'\
151 , $options));
152   }
153 
154 /**
155  * admin_add method
156  *
157  * @return void
158  */
159   public function admin_add() {
160     if ($this->request->is('post')) {
```

```
161         $this->User->create();
162         if ($this->User->save($this->request->data\
163 )) {
164           $this->Session->setFlash(__('The user ha\
165 s been saved.'));
166           return $this->redirect(array('action' =>\
167  'index'));
168         } else {
169           $this->Session->setFlash(__('The user co\
170 uld not be saved. Please, try again.'));
171         }
172       }
173     }
174 
175 /**
176  * admin_edit method
177  *
178  * @throws NotFoundException
179  * @param string $id
180  * @return void
181  */
182     public function admin_edit($id = null) {
183       if (!$this->User->exists($id)) {
184         throw new NotFoundException(__('Invalid us\
185 er'));
186       }
187       if ($this->request->is('post') || $this->req\
188 uest->is('put')) {
189         if ($this->User->save($this->request->data\
```

```
190 )) {
191         $this->Session->setFlash(__('The user ha\
192 s been saved.'));
193         return $this->redirect(array('action' =>\
194  'index'));
195       } else {
196         $this->Session->setFlash(__('The user co\
197 uld not be saved. Please, try again.'));
198       }
199     } else {
200       $options = array('conditions' => array('Us\
201 er.' . $this->User->primaryKey => $id));
202       $this->request->data = $this->User->find('\
203 first', $options);
204     }
205   }
206 
207 /**
208  * admin_delete method
209  *
210  * @throws NotFoundException
211  * @param string $id
212  * @return void
213  */
214   public function admin_delete($id = null) {
215     $this->User->id = $id;
216     if (!$this->User->exists()) {
217       throw new NotFoundException(__('Invalid us\
218 er'));
```

```
219       }
220       $this->request->onlyAllow('post', 'delete');
221       if ($this->User->delete()) {
222         $this->Session->setFlash(__('The user has \
223 been deleted.'));
224       } else {
225         $this->Session->setFlash(__('The user coul\
226 d not be deleted. Please, try again.'));
227       }
228       return $this->redirect(array('action' => 'in\
229 dex'));
230     }
231 }
```

You will notice plain and admin CRUD functions are generated and the code are the same in them. Do not worry about this, we will get back to them in a short time.

View Preparation

We should bake our user views also. It is just as simple as for models and controllers. Go back to your `bake` console and choose `v` for generating views.

```
1  ---------------------------------------------------
2  Interactive Bake Shell
3  ---------------------------------------------------
4  [D]atabase Configuration
5  [M]odel
6  [V]iew
7  [C]ontroller
8  [P]roject
9  [F]ixture
10 [T]est case
11 [Q]uit
12 What would you like to Bake? (D/M/V/C/P/F/T/Q)
13 > v
```

As we did earlier we should choose the database and the controller for which we create the views.

```
-----------------------------------------------
Bake View
Path: /home/rrd/public_html/blog/app/View/
-----------------------------------------------
Use Database Config: (default/test)
[default] >
Possible Controllers based on your current datab\
ase:
-----------------------------------------------
 1. CakeSessions
 2. Users
Enter a number from the list above,
type in the name of another controller, or 'q' t\
o exit
[q] > 2
```

We choose interactive mode as we did for the controller.

```
Would you like bake to build your views interact\
ively?
Warning: Choosing no will overwrite Users views \
if it exist. (y/n)
[n] > y
```

As we already generated controller CRUD functions now we could generate their corresponding views.

```
Would you like to create some CRUD views
(index, add, view, edit) for this controller?
NOTE: Before doing so, you'll need to create you\
r controller
and model classes (including associated models).\
 (y/n)
[y] >
```

And we also generate views for admin functions.

```
Would you like to create the views for admin rou\
ting? (y/n)
[n] > y
```

The bake script now actually generate the files and writes them to our hard drive.

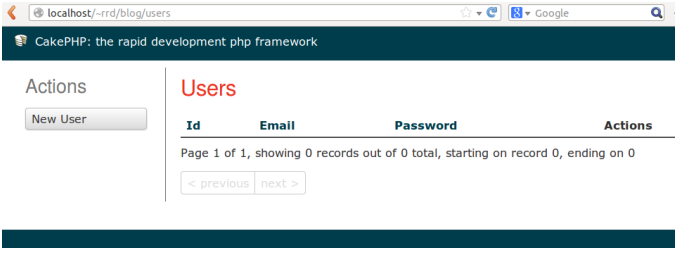
```
Baking `index` view file...

Creating file /home/rrd/public_html/blog/app/Vie\
w/Users/index.ctp
Wrote `/home/rrd/public_html/blog/app/View/Users\
/index.ctp`

Baking `view` view file...

Creating file /home/rrd/public_html/blog/app/Vie\
w/Users/view.ctp
```

```
Wrote `/home/rrd/public_html/blog/app/View/Users\
/view.ctp`

Baking `add` view file...

Creating file /home/rrd/public_html/blog/app/Vie\
w/Users/add.ctp
Wrote `/home/rrd/public_html/blog/app/View/Users\
/add.ctp`

Baking `edit` view file...

Creating file /home/rrd/public_html/blog/app/Vie\
w/Users/edit.ctp
Wrote `/home/rrd/public_html/blog/app/View/Users\
/edit.ctp`

Baking `admin_index` view file...

Creating file /home/rrd/public_html/blog/app/Vie\
w/Users/admin_index.ctp
Wrote `/home/rrd/public_html/blog/app/View/Users\
/admin_index.ctp`

Baking `admin_view` view file...

Creating file /home/rrd/public_html/blog/app/Vie\
w/Users/admin_view.ctp
Wrote `/home/rrd/public_html/blog/app/View/Users\
```

```
/admin_view.ctp`

Baking `admin_add` view file...

Creating file /home/rrd/public_html/blog/app/Vie\
w/Users/admin_add.ctp
Wrote `/home/rrd/public_html/blog/app/View/Users\
/admin_add.ctp`

Baking `admin_edit` view file...

Creating file /home/rrd/public_html/blog/app/Vie\
w/Users/admin_edit.ctp
Wrote `/home/rrd/public_html/blog/app/View/Users\
/admin_edit.ctp`
```

When it is ready you could open your view files and see the generated code. So we spent again 20 seconds with CakePHP automatic code generating tool `bake`.

Now if you navigate your browser to your app (in my case it is at `http://localhost/~rrd/blog/`) and type `users` to the end of the url you should see the following screen.



Users Controller After Baking

You should be able to add, edit, delete and view users. As we generated validation rules your input will be validated against them. So if you try to add a new user with a non valid email address the app should reject it.

Login and Logout

So now we are able to create, edit, delete and list users. Better to say now anyone could do this things. So if anyone finds your application on the web will able to create new users, delete existing users and edit all users' data. Clearly this is not what we want.

We have to know who plays in our application. For this we should log the user in.

Login

Login - controller

Let's create our `login` method in our User controller. Open `/app/Controller/UsersController.php` and the following method.

```
public function login() {
  if ($this->request->is('post')) {
    if ($this->Auth->login()) {
      return $this->redirect($this->Auth->redire\
ctUrl());
    }
    else {
      $this->Session->setFlash(__('Username or p\
assword is incorrect'), 'default', array(), 'aut\
h');
    }
  }
}
```

As we would like to use Auth component in our controller we should add it to the controller's `$components` array. The best place for this is AppController.php. So replace the `$components` array with this.

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

CakePHP conventions are really handy but sometimes we want to use something else. The `Auth` component waiting for `username` and `password` fields. We use `email` and not `username` so we should set up the `Auth` component to handle this. Without this setting it will not log us in even with the right email password combination.

Open `UsersController.php` and at the beginning change `$components` array.

```
public $components = array(
  'Paginator',
  'Auth' => array(
    'authenticate' => array(
      'Form' => array(
        'fields' => array('username' => 'email')
       )
    )
  )
);
```

Login - view

We should create the login view with two input field for the e-mail address and the password.

```
<div class="users form">
<?php echo $this->Form->create('User', array('ac\
tion' => 'login')); ?>
        <fieldset>
                <legend><?php echo __('Login'); ?></legend>
        <?php
                echo $this->Form->input('email');
                echo $this->Form->input('password');
        ?>
        </fieldset>
<?php echo $this->Form->end(__('Login')); ?>
</div>
```

We would like to show error messages to the user. I prefer to have all error messages at the same place, because it improves usability. So open your `default.ctp` at `app/View/Layouts` and add the following line somewhere. I recommend to put to the top of the `content` div.

```
1  <?php echo $this->Session->flash('auth'); ?>
```

At this point we should able to log in. Thanks to `Auth` if we try to open any url it will automatically redirect us to the login page and on successful login we will be redirected to the original url.

From now without logging in nobody could access any part of our application except the login page.

Usability

Usability requires informing the user about successful login or about the fact he or she is already logged in. A best practice is to put a message to the top of the page.

Open `default.ctp` at `app/View/Layouts` and add this to the `header div`.

```
if($this->Session->read('Auth.User.id')){
  print 'Hi ' . $this->Session->read('Auth.User.\
email') . ' ' . $this->Html->link('Logout', arra\
y('controller' => 'users', 'action' => 'logout')\
);
  }
else{
  print $this->Html->link('Login', array('contro\
ller' => 'users', 'action' => 'login'));
}
```

Logout

Perhaps we need a logout method to `UsersController.php`.

```
function logout(){
  $this->redirect($this->Auth->logout());
}
```

Epilogue

I hope by reading this e-book your main questions about user authentication is answered. Perhaps I could write this book double or triple in size if I going to cover everything related to this topic. My purpose was to represent the idea itself, and I am pretty sure that you could easily find solutions to your further questions.

Happy coding.