

## Chapter 5 -- Untwhirling The Spaghetti

### Fat Models, Skinny Controllers

One of the main reasons to refactor this legacy PHP application into CakePHP was that the structure was hopelessly interdependent. To make changes to one part of the system risked wrecking something else inside the same file. But if I was to do it right, and make this tool more viable going forward then applying a framework like CakePHP to this application was a needed step.

One of the design philosophies when it comes to MVC-style frameworks is that I follow is “Fat models, skinny controllers”. The idea is that you put as much of the business logic in your models as you can, reducing your controllers to something that does no more than move code from one model to another and then passing it on to the view.

Now sometimes this isn’t possible, and that’s okay. There is no perfect way to do it. Often it is not clear where some particular functionality should go, and there is a danger in over-abstracting things. So keep that in mind when looking at the code examples.

### Hopelessly Twisted

Here is the code for the page that deals with making a trade.

```
<html>
<head>
<title>WebReg -- Make A Trade</title>
</head>
<body>
<h3 align="Center">WebReg -- Make A Trade</h3>
<?php
// make_a_trade.php
// Interface to make trades between two teams
require_once 'DB.php';
require_once 'db_config.php';
$task="";
$db =& DB::connect(DSN);
if (isset($_POST["task"])) $task=$_POST["task"];
if ($task=="show_rosters") {
    // Make sure they didn't pick the same team for both parties
    $team1=$_POST["team1"];
    $team2=$_POST["team2"];
    if ($team1==$team2) {
        ?>
        <div align=center>
            <font color=red>You must pick two different teams!</font>
        </div>
        <?php
            $task="";
    } else {
        // Okay, let's show the rosters so we can do a trade
        $sql="
            SELECT tig_name
            FROM teams
            WHERE ibl_team='$team1'
            ORDER BY tig_name";
```

```

$result=$db->query($sql);
while ($result->fetchInto($row)) {
    $team1_list[]=trim($row[0]);
}
$sql="
    SELECT tig_name
    FROM teams
    WHERE ibl_team='$team2'
    ORDER BY tig_name";
$result=$db->query($sql);
while ($result->fetchInto($row)) {
    $team2_list[]=trim($row[0]);
}
$t1_size=count($team1_list);
$t2_size=count($team2_list);
if ($t1_size>$t2_size) {
    $drop-down_size=$t1_size;
} else {
    $drop-down_size=$t2_size;
}
$team1_drop-down="<select multiple name=team1_trade[] size=$drop-
down_size><br>";
foreach ($team1_list as $player) {
    $team1_drop-down.="<option value='$player'>$player</option><br>";
}
$team1_drop-down.="</select>";
$team2_drop-down="<select multiple name=team2_trade[] size=$drop-
down_size><br>";
foreach ($team2_list as $player) {
    $team2_drop-down.="<option value='$player'>$player</option><br>";
}
$team2_drop-down.="</select>";
// Let's display the form to do the trade
?>
<div align=center>
<form action=<?php print $_SERVER["PHP_SELF"];?> method=POST>
<input type="hidden" name="task" value="do_trade">
<input type="hidden" name="team1" value="<?php print $team1;?>">
<input type="hidden" name="team2" value="<?php print $team2;?>">
<table>
<tr>
<td align=center><b><?php print $team1;?></td>
<td align=center><b><?php print $team2;?></td>
</tr>
<tr>
<td><?php print $team1_drop-down;?></td>
<td><?php print $team2_drop-down;?></td>
</tr>
<tr>
<td align=center colspan=2><input type="submit" value="Make Trade"></td>
</tr>
</table>
</form>
</div>
<?php
}
}
if ($task=="do_trade") {

```

```

if (isset($_POST['team1_trade'])) $team1_trade=$_POST["team1_trade"];
if (isset($_POST['team2_trade'])) $team2_trade=$_POST["team2_trade"];
if (isset($_POST['team1'])) $team1=$_POST["team1"];
if (isset($_POST['team2'])) $team2=$_POST["team2"];
if (isset($team1_trade)) {
    foreach ($team1_trade as $player) {
        $team1_trade_players[]=$player;
        $trade_date = date("m/y");
        $comments = "Trade {$team1} {$trade_date}";
        $sql="UPDATE teams SET ibl_team='{$team2}',
            comments = '{$comments}',
            status = 2 WHERE tig_name='{$player}'";
        $db->query($sql);
    }
}
if (isset($team2_trade)) {
    foreach ($team2_trade as $player) {
        $team2_trade_players[]=$player;
        $trade_date = date("m/y");
        $comments = "Trade {$team2} {$trade_date}";
        $sql="UPDATE teams SET ibl_team='{$team1}',
            comments = '{$comments}',
            status = 2 WHERE tig_name='{$player}'";
        $db->query($sql);
    }
}
$team1_trade_report = "";
$team2_trade_report = "";
if (isset($team1_trade_players)) $team1_trade_report=implode(" ",
$team1_trade_players);
if (isset($team2_trade_players)) $team2_trade_report=implode(" ",
$team2_trade_players);
$team1_transaction="Trades {$team1_trade_report} to {$team2} for
{$team2_trade_report}";
$team2_transaction="Trades {$team2_trade_report} to {$team1} for
{$team1_trade_report}";
require_once 'transaction_log.php';
transaction_log($team1,$team1_transaction);
transaction_log($team2,$team2_transaction);
print "
    <div align=center>
    <b>$team1</b> trades $team1_trade_report to <b>$team2</b> for
$team2_trade_report<br>
    </div>";
}
if ($task=="") {
    ?>
    <div align=center>Please select two teams for the trade</div>
    <?php
    $sql="SELECT DISTINCT(ibl_team) FROM teams";
    $result=$db->query($sql);
    if ($result!=FALSE) {
        while ($result->fetchInto($row)) {
            $ibl_team[]=$row[0];
        }
    }
    $team_option="";
    foreach ($ibl_team as $team) {

```

```

        $team_option.="<option value='$team'$team</option>\n";
    }
    ?>
    <div align=center>
    <form action=<?php print $_SERVER["PHP_SELF"];?> method="POST">
    <input name="task" type="hidden" value="show_rosters">
    <select name="team1">
    <?php print $team_option;?>
    </select>
    <select name="team2">
    <?php print $team_option;?>
    </select>
    <br>
    <input type="submit" value="Use These Teams">
    </form>
    </div>
    <?php
}
?>
<hr>
<div align=center>Return to <a href=roster_management.php>Roster Management</a></div>
</body>
</html>

```

So our task here is to figure what here needs to go into the models and what needs to go into the controller. First up, we're going to take a look at how to make the first screen people will see when they try to make a trade.

## Easy Drop-Down Lists

When approaching refactoring I like to break down the functionality that I need. For the "make a trade" starting page, I need to do the following:

- get a list of teams
- create a form where you pick two teams
- after picking two different teams, send them to another page where they can choose players

```

class TradeController extends AppController {
    ...
    public function index() {
        $this->pageTitle = 'WebReg -- Make A Trade';
        $teams = $this->Franchise->find('list', array(
            'fields' => 'Franchise.nickname',
            'order' => 'Franchise.nickname'
        ));
        $this->set('teams', $teams);
    }
}

```

Those familiar with CakePHP will recognize the code as being pretty standard, but there is one little twist. The `find('list')` command is a way to generate an array that can be passed into a `<select>` form element. Now, by default `find('list')` tries to create a hash of primary key and the 'name' field in your model, if you

have one. But the existing page is using the nicknames of the team, a 3-character string. So we instead need the hash to be made up of primary key and nickname. Easy to do by simply passing the field you want to the `find('list')` command.

Next, we create the view:

```
<h3 align="center">WebReg -- Make A Trade</h3>
<div align="center">Please select two teams for the trade</div>V
<div align="center">
<div align="center"><font color="red"><?php $session->flash() ?></font></div>
<?=$form->create(array('action' => '/trade/choose_players')) ?>
<?=$form->select('Franchise.team1', $teams, null, null, false) ?>
<?=$form->select('Franchise.team2', $teams, null, null, false) ?>
<br />
<?=$form->submit('Use These Teams') ?>
<?=$form->end() ?>
</div>
<hr>
<div align="center">Return to <a href="/rosters">Roster Management</a></div>
```

Now that we have the list, the next thing we need to do is process the data coming in and then either spit out an error message that you cannot choose the same teams or show a view that contains the players from both teams.

## Model Association Tweaking

After experimenting with some queries using the CakePHP testing console (well, I *did* write it) I discovered that I had one of my primary keys set up wrong. The relationship between Player and Franchise is via `Franchise.nick_name` and `Player.ibl_team`, so I decided to use `Franchise.ibl_team` as the primary key for my Franchise model.

```
class Franchise extends AppModel {
    public $name = 'Franchise';
    public $primaryKey = 'nickname';
    public $hasMany = array(
        'Player' => array(
            'className' => 'Player',
            'foreignKey' => 'ibl_team'
        )
    );
}
```

Now my queries will come up correctly.

## Multiple Select Drop-Downs

The `choose_players` method needs to do the following

- display the multiple-select form fields that let you pick players
- process the incoming form POST

- update the Player data to make the trade
- update the TransactionLog info with the results of the trade

```

class TradeController extends AppController {
    ...

    public function choose_players() {
        if (!$this->data) {
            $this->redirect('/trade');
        }

        if (!empty($this->data['Player'])) {
            $this->Player->set($this->data);
            $players = $this->Player->choosePlayers();

            if (!$this->Player->saveAll()) {
                $this->Session->setFlash('Unable to complete trade');
            } else {
                $this->Session->setFlash('Completed trade');
                $this->TransactionLog->set($this->data);
                $this->TransactionLog->saveEntries($players);
            }
        }

        $this->pageTitle = 'WebReg -- Make A Trade';

        // Make sure that the same two teams weren't selected
        if ($this->data['Franchise']['team1'] == $this->data['Franchise']['team2'])
        {
            $this->Session->setFlash('You must pick two different teams');
            $this->redirect('/trade/index');
        }

        $team1 = $this->data['Franchise']['team1'];
        $team2 = $this->data['Franchise']['team2'];
        $order = 'Player.tig_name';
        $conditions = array('Player.ibl_team' => $team1);
        $roster1 = $this->Player->find('list', array('fields' => array('Player.id',
'Player.tig_name'), 'conditions' => $conditions, 'order' => 'Player.tig_name'));
        $conditions = array('Player.ibl_team' => $team2);
        $roster2 = $this->Player->find('list', array('fields' => array('Player.id',
'Player.tig_name'), 'conditions' => $conditions, 'order' => 'Player.tig_name'));
        $this->set(compact('roster1', 'roster2', 'team1', 'team2'));
    }
}

```

Woah. Is that a serious reduction in the amount of code or what? In keeping with my “fat model, skinny controller” practices I moved a lot of functionality out into two methods in the Player and TransactionLog models respectively. Just remember to use your Model::set methods to pass the data posted to your action into your model. Otherwise you’ll be wasting time building your own parameters to pass into a method in your model. Less code is good code.

It would also be good at this time to mention how simple it is to tell CakePHP you want to use transactions if you are using an ACID compliant database. Postgres, which is the database we are using for this project, is capable of doing transactional saves with rollback when things go wrong.

To gain the use of transactions you can set and pass the 'atomic' variable to your model eg. `Model::saveAll($this->data, array('validate' => 'first', 'atomic' => true))`. That way, it validates your data first and then tries to save things only if the validation was okay.

Unfortunately we don't have the type of logic that would benefit from using transactions, so we can just leave things they way they are.

```
class Player extends AppModel {
    ...

    public function choosePlayers() {
        $data = array();
        $team1_players = array();
        $team2_players = array();

        foreach ($this->data['Player']['roster1'] as $player_id) {
            $data[] = array(
                'id' => $player_id,
                'ibl_team' => $this->data['Franchise']['team2'],
                'comments' => "Trade {$this->data['Franchise']['team1']} " .
date("m/y")
            );
            $team1_players[] = $player_id;
        }

        foreach ($this->data['Player']['roster2'] as $player_id) {
            $data[] = array(
                'id' => $player_id,
                'ibl_team' => $this->data['Franchise']['team1'],
                'comments' => "Trade {$this->data['Franchise']['team2']} " .
date("m/y")
            );
            $team2_players[] = $player_id;
        }

        $this->data = $data;

        return array(
            'team1_players' => $team1_players,
            'team2_players' => $team2_players
        );
    }
}
```

The tendency for some CakePHP developers is to save data in loops. The smart thing to do is to create an array that contains all the data you want to save, and then call your model's `saveAll()` method and CakePHP will do all the heavy SQL lifting for you. Again, let CakePHP do the work for you instead of doing it yourself.

Second, discovering the little tricks you can use with `find('list')`. You can get `find('list')` to generate pretty much whatever output you want, so since I knew I needed a hash that was `"nickname" => "nickname"` *and* only players from the specific team I want. I passed in what fields I needed for the list and the condition that said I only want a list of players from a specific team.

It is possible to skip having to set the fields you want `Model::find('list')` to return if you don't plan on returning more than one type of list for a particular model. You can set `$displayField` to whatever you want the default value to be for lists generated on that model. In fact, there are many other values you can set as default for your model such as primary key for your lists and the default sort order. Again, the online documentation is the best place to check those things out.

```
class TransactionLog extends AppModel {
    ...

    public function saveEntries($players) {
        $this->Player = ClassRegistry::init(array('class' => 'Player'));

        $tradel_players = Set::extract('/Player/tig_name', $this->Player-
>find('all', array('conditions' => array('Player.id' => $players['team1_players']))));
        $trade2_players = Set::extract('/Player/tig_name', $this->Player-
>find('all', array('conditions' => array('Player.id' => $players['team2_players']))));

        $team1_transaction = "Trades " . implode(', ', $tradel_players) . " to
{$this->data['Franchise']['team2']} for " . implode(', ', $trade2_players);
        $team2_transaction = "Trades " . implode(', ', $trade2_players) . " to
{$this->data['Franchise']['team1']} for " . implode(', ', $tradel_players);
        $data = array(
            0 => array(
                'ibl_team' => $this->data['Franchise']['team1'],
                'log_entry' => $team1_transaction,
                'transaction_date' => 'NOW()'
            ),
            1 => array(
                'ibl_team' => $this->data['Franchise']['team2'],
                'log_entry' => $team2_transaction,
                'transaction_date' => 'NOW()'
            )
        );

        return $this->saveAll($data);
    }

    ...
}
```

The syntax for `Set::extract(...)` makes so much better sense now that it uses XPath 2.0 syntax instead of the old `{n}.Model.field` syntax. Of course, I'm slightly biased because I've dealt with XPath syntax quite a bit in my day job, which involves shoveling around a large amount of . So, I needed an easy way to get a comma-separated list of players to enter in the transaction log. `Set::extract` did the trick.

Here is the view that goes along with it:

```
<div align="center">
<?=$form->create('Player', array('url' => '/trade/choose_players')) ?>
<?=$form->hidden('Franchise.team1', array('value' => $team1)) ?>
<?=$form->hidden('Franchise.team2', array('value' => $team2)) ?>
<table>
<tr>
```

```
<td align="center"><?= $team1 ?></td>
<td align="center"><?= $team2 ?></td>
</tr>
<tr>
<td colspan=2><?php $session->flash() ?></td>
</tr>
<tr>
<td><?= $form->select('Player.roster1', $roster1, null, array('multiple' => true,
'size' => count($roster1)), false) ?></td>
<td><?= $form->select('Player.roster2', $roster2, null, array('multiple' =>
true, 'size' => count($roster2)), false) ?></td>
</tr>
</table>
<?= $form->submit('Make Trade') ?>
<?= $form->end() ?>
</div>
```

You might be wondering why I am not going with `$form->input()` for all the fields, instead preferring to use the actual field-specific methods. There's a simple reason, really. When you use `$form->input()` it often makes assumptions on how you want the data displayed. I mean, it is doing a lot of magic but in the end it is making decisions on how to display the data, decisions that might not match what you want to do.

Secondly, `$form->input()` also produces markup to go with the form field. Here's a sample, straight out of the Cookbook:

```
<?php echo $form->input('field', array('type' => 'file')); ?>
```

Output:

```
<div class="input">
  <label for="UserField">Field</label>
  <input type="file" name="data[User][field]" value="" id="UserField" />
</div>
```

Sometimes you might be okay with that. For this application I just need the form input code and nothing else. Experiment and play with `$form->input()` to determine what is right for your application.