MVC in Trax

Typical PHP Code – Everything shoved into one file. ☹ Not Good!

```
<?
require_once(“config.inc.php");
require_once(“database.inc.php");

$dbh = dbConnect();
If($submit) {
	$sql  = “INSERT INTO my_table (name,address,city,state,zip) VALUES (”;
	$sql .= “’$name’,’$address’,’$city’,’$state’,’$zip’)”;
	$dbh->query($sql);
} else {
	$result = $dbh->query(“SELECT * FROM my_table”);
	$userArray = $dbh->fetchRow($result);
}
printHeader();
?>
<div>My HTML code blah blah</div>
<form method=“POST”>
	Name: <input type=“text” name=“name” value=“<?=$userArray[‘name’]?>”><br>
	…
</form>
…
<? printFooter(); ?>
```

Leads to frustrating code to maintain and update.

What are our options?

http://www.phpwact.org/php/mvc_frameworks

PHP on Trax - A True Ruby on Rails framework clone for PHP (PHP5).
PhpMVC (A Struts port)
SolarPHP - A PHP5 Web Framework. MVC-based.
Cake - A Ruby on Rails like framework for PHP. MIT licensed.
Biscuit - Similar to Cake only using much more procedural code (rather than OO). BSD licensed.
TaniPHP - PHP MVC Ruby on Rails like framework for PHP. LGPL licensed.
Aukyla PHP Framework - Nice ideas: Local URI's (stream wrappers) and OpenDocument file handling. Released under a dual GPL/Commercial license.
symfony - just another php5 framework ? Probably not. It takes the best of Mojavi, Propel and Rails, adds some more and packages it all into an integrated framework. MIT licensed.

Ruby on Rails

PHP on Trax

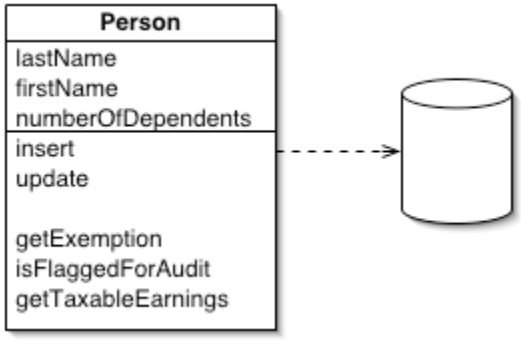
- Model (Active Record) - Connects business objects and database tables to create a persistable domain model where logic and data are presented in one wrapping.

- View (Action View) – Helper classes to display html elements in your views.

- Controller (Action Controller) - Is made up of one or more actions that performs its purpose and then either renders a template or redirects to another action. An action is defined as a public method in the controller, which will automatically be made accessible to the web-server through a mod_rewrite mapping.

Active Record

An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data.



An object carries both data and behavior. Much of this data is persistent and needs to be stored in a database. Active Record uses the most obvious approach, putting data access logic in the domain object. This way all people know how to read and write their data to and from the database.

Patterns of Enterprise Application Architecture **by Martin Fowler**

URL: http://www.martinfowler.com/eaaCatalog/activeRecord.html

Active Record Models

```
<?
class Person extends ActiveRecord {
}
?>
```

This Person Model/Object now knows all about the thing it represents, the database table “people”, and has the ability to insert, update, delete records from itself.

Active Record Models

```
<?
class Person extends ActiveRecord {
    public $has_many = “emails”;
    public $belongs_to = “family”;
}
?>
```

A Person has many emails which means that there is a table called “emails” that has a foreign key in it called “person_id”.

A Person belongs to a family which means that there is a foreign key called “family_id”, from a table called “families” in the table this model represents, “people”.

URL: http://www.mydomain.com/families/show_member/4

Controller (app/controllers/families_controller.php)

```
<?
class FamiliesController extends ApplicationController {
	function show_member() {
		$person = new Person();
		$this->person = $this->person->find($_REQUEST['id']);
	}
}
?>
```

View (app/views/families/show_member.phtml)

```
<h2> <?= $person->name ?></h2>
Address: <?= $person->address ?><br>
Family Name: <?= $person->family->name ?><br>
<h3>Notes:</h3>
<? if(count($person->emails)): ?>
	Emails:<br>
	<? foreach($person->emails as $email): ?>
		  -<?= $email->address ?><br>
	<? endforeach; ?>
<? else: ?>
	no emails found.
<? endif; ?>
```

Active Record Validations

```
<?
class Person extends ActiveRecord {

	public $has_many = “emails”;
	public $belongs_to = “family”;

	function validate() {
		if(strlen($this->name) < 4) {
		    $this->add_error(“Name must be at least 4 chars long”, “name”);
		}
	}

	function validate_name() {
		 return array(!empty($this->name), “Name can’t be empty”);
	}
}
?>
```

String: $model->get_errors_as_string(); “error a
error b
…”

Array: $model->get_errors(); array(“error a”,”error b”,…)

Action Controller

- **Layouts** – Templates that contains the look of your site.
- **Partials** – For convenience html that will be used in multiple places can be put into partials for inclusion into other parts of your code.
- **Routing** – Allows you to define specific urls and what they should link to.
- **Filters** – Specify one or more function that will be ran before or after every function in this controller class.
- **Flash** – Provides a way to pass temporary variables between actions, usually messages to be displayed out to the user.

Layouts

Location : app/views/layouts
Default layout: application.phtml

```
<html>
<head>
   <title>My Application</title>
   <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
   <link rel="stylesheet" href="/stylesheets/application.css" type="text/css" />
</head>
<body>
     <? if(Session::isset_flash('notice') or Session::isset_flash('error')): ?>
     <div class="messagebox">
       <? if(Session::isset_flash('notice')): ?>
             <p style="color: green"><?= Session::flash('notice') ?></p>
       <? endif; ?>
       <? if(Session::isset_flash('error')): ?>
             <p style="color: red"><?= Session::flash('error') ?></p>
       <? endif; ?>
     </div>
     <? endif; ?>
     <?= $content_for_layout ?>
</body>
</html>
```

Layouts

```
<?
class TestController extends ApplicationController {

        public $layout = “blue”;

        function some_method() {
            … some code
        }
}
?>
```

Loads layout app/views/layouts/blue.phtml

```
<?
class TestController extends ApplicationController {

        public $layout = “my_layout”;

        function my_layout() {
                    if(Session::get(‘user_level’) == ADMIN) {
                        return “admin”;
                    }  else {
                        return “user”;
                    }
        }
}
?>
```

If this is an admin then it will load the layout app/views/layouts/admin.phtml
Else it will load the layout app/views/layouts/user.phtml

Partials

Add (app/views/users/add.phtml)

```
<form action="/users/add">
<?= $this->render_partial("form") ?>
</form>
```

Edit (app/views/users/edit.phtml)

```
<form action="/users/edit/2">
<?= $this->render_partial("form") ?>
</form>
```

Form Partial (app/views/users/_form.phtml)

```
Name: <?= text_field("form", "name")?><br>
Age: <?= text_field("form", "age")?><br>
<?= submit_tag("Save") ?>
```

Routing

config/routes.php

```
<?
# Add your own custom routes here.
# The priority is based upon order of creation: first created -> highest priority.

# Here's a sample route:
$router->connect( "products/:id", array(":controller" => "catalog", ":action" => "view") );

# You can have the root of your site routed by hooking up "".
# Just remember to delete public_html/index.html.
$router->connect( "", array(":controller" => "search") );

# Install the default route as the lowest priority.
$router->connect( ":controller/:action/:id" );

?>
```

Filters

before_filters / after_filter – Making a protected area of your application.

```
<?
class TestController extends AdminAreaController {
    function delete_user() {
        … some code
    }
}
?>

<?
class AdminAreaController extends ApplicationController {
    public $before_filter = “authenticate”;
    function authenticate() {
        if(Session::get(‘logged_in’) == false) {
            Session::flash(‘error’, ‘You must be logged in to access this page.’);
            $this->redirect_to = “/login”;
        }
    }
}
?>
```

Flash

The flash provides a way to pass temporary variables between actions. Anything you place in the flash will be exposed to the very next action and then cleared out. This is a great way of doing notices and alerts, such as a create action that sets Session::flash(‘notice’,"Successfully created“) before redirecting to a display action that can then expose the flash to its template.

```
If($user->save()) {
        Session::flash(‘notice’,"Successfully created user“);
        $this->redirect_to = “/users”;
}
```

Lets see it in action!

Questions?

John Peterson
john@mytechsupport.com