A WEB APPLICATION USING MVC FRAMEWORK

Marissa Principe¹ and David Yoon²
¹,² CIS Department, University of Michigan - Dearborn

Abstract - With the rapid development of the web, the facets of software have been changing rapidly. The web software is no longer static and should be able to cope with constant interactions with the users who are communicating with the system by clicking the mouse. This generates huge amount of data. The web-based software should be able to update the persistent data accordingly. In this paper, we will demonstrate the need for the MVC framework for the design of the web-based software to address these issues and present a web-based application software using CakePHP framework.

Keywords: MVC, Three-Tier Architecture, CakePHP Framework

1 Introduction

Unlike a static website, a web application requires user interaction and persistent data storage capabilities. This introduces complexities to the software that can be alleviated with the help of a framework. Without a framework, a significant part of a programmer’s time is spent on coding standard services such as caching, persistence and security. Since these are most likely application specific, there is no interoperability between different applications. This result is a lot of duplicate effort for each new project. A framework is a foundation of code that provides functionality that is common to a specific type of application. The programmer can build upon this foundation which can be used in multiple projects. Having lower-level functions, such as database connection, already created will cut down on development time and allow the programmer to focus on the main requirements or business logic; which is the project’s objective. Some features of a web framework are the ability to easily store, access and update the application’s persistent data; data integrity; session management; user authentication; security; and caching. The framework typically controls the flow of execution and can assist in programming to design patterns such as Model-View-Controller (MVC).

2 MVC and Three-Tier Architecture

In the MVC approach the application is divided in three components: model that represents the application data, view that present the user interface, and control that handles user input [2]. The MVC architecture is ideal for building websites that require user interaction. It makes applications easier to code, maintain and expand. This is due to the separation of the program components into different areas of logic; presentation (view), business or logic (control) and data access (model). Using the traditional description of MVC, the components are described as follows:

Model – Represents persistent application data and notifies views and controllers of changes to its state or data.

View – Presents model data to the user and observes the model for updates.

Controller – Handles any type of input and sends commands to update the model based on that input if necessary.

Three-tier architecture is based on the same principles of separation; with the idea that the components can be physically separated to reside on different machines. In addition to the advantages of code separation already discussed, three-tier architecture offers the advantage of being able to upgrade or replace any of the tiers without affecting the others.

2.1 CakePHP Framework

CakePHP is a rapid development framework for PHP that provides an extensible architecture for developing, maintaining, and deploying applications. Using commonly known design patterns like MVC and ORM, within the convention over configuration paradigm, CakePHP reduces development costs and helps developers write less code [7]. This is just one of many frameworks available for developing PHP applications. Some other popular frameworks are Laravel, Phalcon, Zend, Symfony and CodeIgniter. Some of the reasons for choosing CakePHP are that it is open source, has a lot of built-in security tools, has built-in ORM and has a
large community of active developers. In CakePHP, the MVC pattern is slightly modified to fully separate the view from the model. The view only receives information from the controller and has no direct communication with the model. The framework is laid out as follows:

2.2 Object-Relational Mapping

CakePHP also provides object relational mapping. Once you define what type of database you want to connect to, the appropriate syntax is automatically used. Since Cakephp employs the convention over configuration paradigm, if you follow the naming conventions provided in the CakePHP documentation, relationships between models are easy to create and are recognized throughout the application. For example, a simple call to get data from a table with related tables; will retrieve all related data as well.

A relationship is defined between the User and Comment objects.

```php
class User extends AppModel {
    public $hasMany = 'Comment';
}
```

A simple call to retrieve users, results in the users’ comments being retrieved as well; without any extra programming effort.

```php
//Sample results from a $this->User->find() call.
Array
([User] => Array
([id] => 121
[name] => Gwoo the Kungwoo
)
[Comment] => Array
([0] => Array
([id] => 123
[user_id] => 121
[title] => On Gwoo the Kungwoo
[body] => The Kungwooness is not so Gwooish
[created] => 2006-05-01 10:31:01
)
([1] => Array
([id] => 124
[user_id] => 121
[title] => More on Gwoo
[body] => But what of the Nut?
[created] => 2006-05-01 10:41:01
))
)
Both of the above arrays are made available to the corresponding view. The framework is flexible as well and this scenario is easily overwritten for an instance when you do not want to retrieve the related data.

```php
$this->User->unbindModel(
    array
    (hasMany => array
    ('Comment'))
);
```

This only affects the very next find function; subsequent calls revert back to the standard call unless they are also overwritten. Updating a model does not require any updating to the methods that manipulate and retrieve data from that model. The basic CRUD (Create Read Update Delete) logic remains the same. Therefore, the controller logic does not need to be changed.

### 2.3 CakePHP Web Application

This section presents an example of the CakePHP MVC framework in implementing an event sign up application. This can be a stand-alone application or just a part of a larger website. The objective of this application is to allow users to sign up for events online instead of filling out a form by hand and mailing it. This not only provides value for those signing up, but for the event organizers as well. Unlike with the current situation of having a static website, the organizers will not have to manually keep track of attendees from handwritten, sometimes illegible, forms; they will have a way to quickly retrieve past event attendees; and can easily add a new event or update existing events. Users can view all events; sign up for those whose registration deadline has not yet passed and get a quick view of what they have already signed up for and how much they still owe. Since the application was implemented using the MVC design pattern, it can easily be expanded and can also be adapted for use in other similar projects.

The flow of the application from a user’s perspective:

1. A user clicks on the events page of a website which sends a request to the controller.
2. The controller reads the request and sends the appropriate call to the model for a list of events.
3. The model connects to the database and performs the necessary logic to retrieve the requested data.
4. The data is then available to the view called by the controller and the list of events is displayed to the user. The event hyperlinks are disabled.
5. The page displays a message to log in to sign up for an event.
6. The user either logs in or creates an account and is routed back to the events page where the event hyperlinks are now enabled.
7. The user clicks on an event, signs up and is routed back to the event page where he can view a list of events for which he has signed up and how much he owes. If he has not already paid for an event, he can delete it from his list.

The flow of the application from an admin’s perspective:

1. The admin log in page is presented since no admin pages are available to non-admin users.
2. Once logged in, the controller routes the admin user to the admin events page where a different controller makes a call to the model and the model makes the list of events available to view.
3. The admin can edit an event or delete one if there are no paid attendees.
4. The admin clicks on an event to view the attendee list and can mark them as paid.
5. Once back at the events page, the admin can create a new event.

### 2.4 High-Level System Diagram

![High-level system diagram](image)

Fig. 3 High-level system diagram
2.5 Table Names

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Collation</th>
<th>Attributes</th>
<th>Null</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int(11)</td>
<td></td>
<td></td>
<td>No</td>
<td>AUTO_INCREMENT</td>
<td>Yes</td>
</tr>
<tr>
<td>2sens</td>
<td>varchar(80)</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3line</td>
<td>varchar(80)</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4line</td>
<td>varchar(80)</td>
<td>latin1_swedish_ci</td>
<td>Yes</td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5city</td>
<td>varchar(20)</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6state</td>
<td>varchar(10)</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7zipcode</td>
<td>varchar(10)</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8phone</td>
<td>varchar(12)</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For more screen shots, the readers are referred to the appendix at the end of the paper.

3 Conclusion

The benefits of using the MVC design method when programming a web application are abundantly clear. The fact that there are many existing frameworks available to help with the process is very encouraging. Taking the time to research, choose, and learn a framework is definitely worth the up-front effort. Even though it might take a lot of time away from the project in the beginning, the end result is that it makes the coding faster and less error prone. Since errors are a great source of frustration, this alone makes the effort worth it. Since many of the frameworks are open source, new components frequently become available. These plugins contain content that is common to many applications; such as shopping carts. This all contributes to the goal of making a programmer’s job less tedious.

4 References


[3] A Database and Web Application Based on MVC Architecture

http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

http://docforge.com/wiki/web_application_framework

http://en.wikipedia.org/wiki/Multitier_architecture


[8] CakePHP MVC Documentation

[9] CakePHP Documentation

[10] Murach, Joel and Harris, Ray. PHP and MySQL. Fresno: Mike Murach & Associates, Inc. 2010

APPENDIX

Application Screenshots

The user can look at events before logging in, but cannot sign up. Event hyperlinks are disabled.

After logging in, the user can sign up for events if registration is still open, see a list of events they have already signed up for, and edit their user information. Event hyperlinks are now enabled.

The user can log in or create an account.

User can view events and delete from their list if they have not yet paid.
Admin view of events page. Events can be deleted only if there are no paid attendees.

Try to delete an event with paid attendees.

Admin can create a new event.

Or edit an existing event.

Admin can view attendee list and mark them as paid.