ABSTRACT

Rapid application development is quickly growing in popularity. The draw for most programmers is the ability to reuse defined components that are commonly used in the development of a Web application. Frameworks make use of a set of libraries that contain such modules – bits of reusable code. Using these libraries means developers do not have to “reinvent the wheel” for each project, thus saving time on common features and allowing more time to focus on the unique aspects of a project. This project examined two rapid development frameworks: CakePHP and Symfony. The research portion of the project studied the structure, differences, and similarities of each framework and used that knowledge to choose a framework to begin development of an online language placement exam template. Once a framework was chosen, research continued on that framework in the form of additional reading and tutorials. A basic template prototype for the backend was developed, which includes the ability to create an exam and add questions with the corresponding answer selection.
# TABLE OF CONTENTS

ABSTRACT ............................................................................................................................................. ii  

TABLE OF CONTENTS ......................................................................................................................... iii  

LIST OF FIGURES ............................................................................................................................... vi  

1. INTRODUCTION AND BACKGROUND ............................................................................... 1  
   1.1. PHP Frameworks .................................................................................................................. 1  
   1.1.1. CakePHP ....................................................................................................................... 2  
   1.1.2. Symfony ........................................................................................................................ 2  
   1.2. Sites Using Frameworks ..................................................................................................... 3  
   1.2.1. Sites Built with CakePHP ............................................................................................. 3  
   1.2.2. Sites Using Symfony ...................................................................................................... 4  
   1.3. How Can Frameworks Help Achieve the Goal................................................................. 4  

2. NARRATIVE ................................................................................................................................... 5  
   2.1. The Frameworks: A Brief Overall Comparison ............................................................... 5  
   2.1.1. Similarities ..................................................................................................................... 5  
   2.1.2. Differences .................................................................................................................... 7  
   2.2. Components of the Placement Exam Template ............................................................. 8  

3. RESEARCH ................................................................................................................................... 9  
   3.1. Equipment ............................................................................................................................ 9  
   3.2. Framework Tutorials .......................................................................................................... 9  
   3.3. CakePHP’s Blog Research ................................................................................................... 10  
   3.3.1. Beginning the Blog Tutorial ......................................................................................... 10  
   3.3.2. Naming Conventions .................................................................................................... 11
3.3.3. Working with the Database ................................................................. 11
3.3.4. Creating a Post Model, Controller, and View .................................... 12
3.3.5. Adding, Editing, and Deleting Posts .................................................. 14
3.3.6. Routes .................................................................................................. 15
3.3.7. Other Features ..................................................................................... 16
3.3.8. Scaffolding and the Bake Console ...................................................... 17
3.3.9. CakePHP Conclusion ......................................................................... 19
3.4. Symfony Jobeet Research ........................................................................ 19
  3.4.1. Introduction to Symfony and the Jobeet Tutorial ............................ 19
  3.4.2. Doctrine ............................................................................................. 21
  3.4.3. YAML ................................................................................................ 22
  3.4.4. The Data Model ................................................................................ 23
  3.4.5. Controllers and Views ................................................................. 25
  3.4.6. Routing ............................................................................................... 27
  3.4.7. Unit Tests and Functional Tests ....................................................... 28
  3.4.8. Admin Generator ............................................................................ 31
3.5. Online Exams and Templates ................................................................. 32
4. EVALUATION AND RESULTS ................................................................. 34
  4.1. Evaluation of the Frameworks ....................................................... 34
    4.1.1. Documentation ............................................................................. 35
    4.1.2. MVC ............................................................................................... 35
    4.1.3. Routing .......................................................................................... 36
    4.1.4. Integrated Form Validation ......................................................... 36
4.1.5. ORM ......................................................................................................... 37
4.1.6. Integrated Testing ..................................................................................... 37
4.2. Framework Selection ........................................................................................ 37
4.3. Project Results .................................................................................................. 39
  4.3.1. Prototype Screenshots ............................................................................... 40
5. FUTURE WORK ...................................................................................................... 44
6. CONCLUSION ......................................................................................................... 45
BIBLIOGRAPHY AND REFERENCES ......................................................................... 46
LIST OF FIGURES

Figure 2.1 A Diagram Representation of MVC [Zheng 2008] ............................................... 7
Figure 2.2 A Visual Representation of MVC [Leff 2001] ......................................................... 7
Figure 3.1 Interactive Bake Shell [CakePHP 2009] ................................................................. 18
Figure 3.2 Jobeet Application Entity Relationship Diagram [Symfony 2009] ..................... 23
Figure 3.3 Templates and the Layout [Symfony 2009] ............................................................. 27
Figure 3.4 A Successful Unit Test [Symfony 2009] ................................................................. 29
Figure 3.5 A Failed Unit Test [Symfony 2009] ..................................................................... 30
Figure 4.1 Home Page ........................................................................................................ 41
Figure 4.2 Question List ...................................................................................................... 41
Figure 4.3 Add New Question ............................................................................................. 42
Figure 4.4 Edit Question ..................................................................................................... 43
1. INTRODUCTION AND BACKGROUND

The idea behind this project was the need for an online language placement exam template, specifically focusing on less commonly taught languages and languages comprised of non-Latin characters such as Chinese, Hebrew, and Russian. Web-based exams have been around for several years. What makes language exams special, however, is the fact that they are in need of more than just text or simple multiple-choice questions. This project sought to address these needs by researching PHP frameworks and using one to create a template.

There are commonly three major components to a language placement exam: grammar, listening comprehension, and reading comprehension. However, instructors would like the capability for students to be able to record audio and to be able to type in the subject language. This project looked at two open source Web development frameworks to determine the pros and cons of each and how they can be used to provide a platform as the basis of creating a template for the online language placement exam.

1.1. PHP Frameworks

“PHP is a widely used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML” [PHP 2009]. It has been around for several years and has grown in popularity as the Web has become more and more prevalent. In the past, developers wrote their own code and did not necessarily share it with others, thus creating the need for other developers to create similar code with the same or very similar functionality. Frameworks were created as a way to collect all this code and store it in a way that it could be available to many developers. The main
point is to reduce redundancy and thus time. Below are three of the more popular PHP frameworks, which will be described in more detail in section three.

1.1.1. CakePHP

CakePHP was founded by Larry Masters in 2005 [Cake PHP 2009]. Masters is currently also the president of Cake Development Corporation, which was formed in 2007 to provide services to businesses interested in using CakePHP. CakePHP is an open source project published under the MIT license. A wealth of information can be found on CakePHP’s Web site. The Bakery is a collection of news and articles related to CakePHP. The Cookbook is CakePHP’s online manual that starts at the beginning with basic principles of CakePHP then moves on to installation followed by common tasks, libraries, and console applications and finally tutorials and examples. In addition to the manual and a multitude of tutorials, there is also a journal, a GoogleGroup and even a podcast called The Show. “The Show is a live Internet radio broadcast where [developers] discuss CakePHP-related topics and answer questions live via IRC” [CakePHP2009]. Just about anything a developer wants to know can be found on the Web site or one of its many support components.

1.1.2. Symfony

Symfony is the product of a French Web agency, Sensio Labs, which was founded by Gregory Pascal and Fabien Potencier in 1998 [Symfony 2009]. Currently Potencier is the Project Lead for Symfony. According to the Web site, Symfony “aims to speed up the creation and maintenance of Web applications, and to replace the repetitive coding tasks by power, control and pleasure” [Symfony 2009]. The framework is free
and is published under the MIT license. As with CakePHP, much information can be found on the Symfony project Web site. Materials include a real-world app step by step tutorial, screen casts, and plenty of documentation. There is a blog, a forum, and a development timeline is posted to let users know what is going on with issue tickets and changes. A user can also find a list of upcoming training sessions and where they will be hosted.

1.2. Sites Using Frameworks

All three frameworks being studied are popular and in use in many different industries. The fact that the frameworks are open source makes them an attractive platform for a number of industries to utilize in their applications. On the one hand, companies can save time and money by using open source software and not being locked into a vendor contract. On the other hand, a company needs technical staff on site to develop applications using a framework. It is possible to contract with a framework’s support company, but using open source always provides the ability to have full access to the underlying code. This openness gives companies an advantage over a proprietary commercial product.

1.2.1. Sites Built with CakePHP

CakePHP’s popularity has seemed to explode recently. Some of the notable sites using CakePHP include Mozilla Add-ons, The Onion Store, and the new Miss Universe Web site [CakePHP 2009]. Each of these sites has a different purpose and a different audience, but all have made use of the CakePHP framework. A look at the Mozilla Add-ons site highlights the ability to have a repository of a variety of downloads for the
popular Firefox Web browser. A quick view of The Onion Store shows the ability to build e-Commerce components into a site using CakePHP. The Miss Universe site is a more graphics intensive site with content including video and photos and focuses more on an entertainment aspect.

1.2.2. Sites Using Symfony

It has been a challenge to find sites using Symfony simply because the Symfony project site does not provide a list of sites using the framework, nor do sites necessarily display a credit of what framework was used or if one was used at all. Of course the Symfony Web site and its partner Web sites are using Symfony, but a definitive list of other users has yet to be found. This is research in progress.

1.3. How Can Frameworks Help Achieve the Goal

In a world filled with deadlines, frameworks are important tools to help users get things done. However, they are flexible enough to accommodate unforeseen changes. This flexibility is one of the core components drawing Web developers to use frameworks. Using a framework allows a developer to spend more time focusing on how each project is unique and what needs to be done to take the project to completion. Once these factors have been determined, a framework can be used to create custom content or features in a rapid fashion.
2. NARRATIVE

The major aim of this project was to compare two popular PHP frameworks.
What follows is a brief comparison based on a broader, less technical viewpoint. Section 3 provides a more in depth view of the architecture and code of each framework and how to go about creating an application.

2.1. The Frameworks: A Brief Overall Comparison

The two frameworks in the study have a number of things in common, but there are also some distinguishing characteristics. The differences helped determine which framework was most suitable to the task at hand.

2.1.1. Similarities

A major similarity among the frameworks is that they are all open source. The open source movement is growing at a rapid pace. As more tools such as frameworks become available and wide spread, more people are getting involved with doing things on the Web. Being open source means that anyone can download and install the frameworks without paying a licensing fee, as would be the case with commercial products. Open source also means that if a person wanted to have a more detailed look at the underlying code driving an application he or she has access to that code without worrying about proprietary roadblocks. In addition to viewing the code a person can modify the code to introduce new functionality or customize something as needed.

A common element of each of the frameworks, which is often the case with open source software, is the support of a for-profit company that clients can utilize for training, consulting or complete application creation. In CakePHP’s case, this entity is the Cake
Software Foundation, which encompasses the Cake Development Corporation. CakeDC, as it’s also known, offers application development, consulting, database design and application architecture, and server deployment and optimization. Symfony’s support company is its founding company Sensio Labs, whose services include consulting, training, application development or integration and technical assistance. It may be confusing for developers to distinguish between what is open source and what is for-profit. The basic idea is that the framework itself can be obtained and used at no cost, however, if a client does not have the technical support staff or if a developer needs help, there is an option for paid support. The existence of paid support services indicates that although a framework may have started as a small open source idea, the project has matured enough to warrant the creation of a profitable company.

Another main point in a lot of frameworks, including those not mentioned here, is use of the model-view-controller (MVC) pattern. MVC is a concept that was “initially introduced for user interfaces in applications implemented with the programming language Smalltalk” [Selfa 2006]. The most distinguishing element about MVC is the idea of separating the presentation layer and the business logic. “MVC was designed for applications that provide multiple views of the same data. Now it is virtually the central feature of modern interactive systems. MVC describes a GUI application in terms of three fundamental abstractions: models, views, and controllers” [Tao 2002]. Figures 2.1 and 2.2 show two different interpretations of the MVC architecture. “The model contains the application data and manages the core functionality. The view manages the visual display of the model and the feedback to the user. The controller interprets the mouse
and keyboard inputs from the user, commanding the model and the view to change appropriately” [Selfa 2006].

![Diagram Representation of MVC](image1)

**Figure 2.1 A Diagram Representation of MVC [Zheng 2008]**

![Visual Representation of MVC](image2)

**Figure 2.2 A Visual Representation of MVC [Leff 2001]**

### 2.1.2. Differences

Though there exist many similarities, of course there exist differences as well. Some of the things to look at when choosing a framework do not necessarily have to do with the code itself. One of the things to consider is the community that is supporting the framework. An active community indicates a strong framework that is continually being worked on and improved. It also provides a place for users to ask questions and report
bugs. Community support can take the form of tutorials, blogs, forums and some type of issue tracking system.

2.2. Components of the Placement Exam Template

As stated in the abstract, at the end of the project an online language placement exam template is created using one of the three frameworks. There are certain components that need to be present in the template. The following are important aspects that need to be considered when researching each framework and the subsequent creation of the template:

- Security
- Database interaction
- Ability to easily add audio and video
- Ability to record audio
- Intuitive interface and navigation

The above list applies to creating the exam. There are also some additional administrative tasks that need to be taken into account. These include user management and results management. In addition, there needs to be a way for the exam data to be extracted for further analysis from the standpoint of an instructor researching second language acquisition.
3. RESEARCH

To gain a better idea of the frameworks, the researcher worked through a series of tutorials and various documents including books, blogs, and forums. This portion of the project discusses the hardware and software involved in the study and subsequent creation of the online placement exam template.

3.1. Equipment

Before beginning to work with a framework, it needs to be housed somewhere. This project utilizes the popular LAMP server. LAMP stands for Linux, Apache, MySQL, and PHP, although sometimes Perl is also used. The Linux flavor is Debian 5.0 Lenny. Apache is the Web server. The database is MySQL 5.0 and the scripting language installed is PHP5. SQLyog, a MySQL GUI program, is used to interact with the database. Putty, an FTP program, is used to communicate with the Web server via the command line. As with the frameworks, all these pieces of software are open source and can be downloaded at no cost.

3.2. Framework Tutorials

Each project has a vast amount of resources available via the Web. Working through some basic tutorials provided a better understanding of the framework and how it is helpful in the creation of a useful Web application. As the tutorials were worked through, specific attention was paid to the areas listed in section 2.2. How does each framework handle security? How easy is it to integrate Web 2.0 content into the
application? How does the application interact with the database? These questions and more are explored in the following sections.

3.3. CakePHP’s Blog Research

Consistent with the cake theme, CakePHP’s manual has been christened The Cookbook. The Cookbook begins with basic principles of CakePHP and a brief introduction to the MVC architecture. There is also a short discussion of naming conventions in CakePHP, which are important if a developer wants everything to work together automatically. Then there are some sections discussing the installation of CakePHP on a Web server. Towards the end of the manual is a Tutorials and Examples section. The researcher spent time going through a blog tutorial to get a better understanding of CakePHP’s structure and how the naming conventions allow different pieces to work together. This section included creating a model, a controller, and a view. Next, the events for adding, deleting, and editing posts were explored.

3.3.1. Beginning the Blog Tutorial

The focus of CakePHP’s tutorial is the creation of a simple blog application. The assumptions made by the tutorial include use of an Apache Web server, a database server, in this case MySQL, and basic knowledge of PHP and MVC. The latest version of CakePHP can be downloaded from the Web site at www.cakephp.org. At the time of this project the latest version was 1.2. The file should be saved and decompressed in the DocumentRoot folder. For this research, this meant creating a caketest folder inside /var/www off the root directory and installing CakePHP there. CakePHP’s installation creates the basic file structure, which consists of the folders: app, cake, docs, and
Vendors. Two files are also created: .htaccess and index.php. These folders house all the files needed to create a CakePHP application.

3.3.2. Naming Conventions

CakePHP has a rather strict set of naming conventions. According to the Web site, using Cake’s conventions “you save time in the long run, […] get free functionality, and you free yourself from the maintenance nightmare of tracking config files. Convention also makes for a very uniform system development, allowing other developers to jump in and help more easily” [CakePHP 2009]. Naming conventions apply to the database as well as the models, controllers and views. File names are underscored and class name are CamelCased, meaning the first letter of each word is capitalized. Model class names are singular and CamelCased. Database table names are plural and underscored. For example, in the case of the blog tutorial: the table is posts, the model is post.php, the controller is posts_controller.php, and the views are in the posts folder. The model class name in post.php is Post. The documentation specifies that in special cases, these conventions can be overridden, but generally the conventions should be followed to allow all the different components to work together seamlessly. CakePHP is able to perform some automation based on what names it expects to find for the different pieces of the application.

3.3.3. Working with the Database

After installation, the next step is to set up the database where all application data will be stored. This process involves the standard SQL commands CREATE DATABASE and CREATE TABLE. Gaining access to the database is achieved via SQLyog or Putty
where these commands can be executed. Some default values are inserted into the database as shown in the code below taken from the CakePHP Web site.

```sql
/* First, create posts table: */
CREATE TABLE posts (  
id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,  
title VARCHAR(50),  
body TEXT,  
created DATETIME DEFAULT NULL,  
modified DATETIME DEFAULT NULL
);

/* Then insert some posts for testing: */
INSERT INTO posts (title,body,created)  
VALUES ('The title', 'This is the post body.', NOW());
INSERT INTO posts (title,body,created)  
VALUES ('A title once again', 'And the post body follows.', NOW());
INSERT INTO posts (title,body,created)  
VALUES ('Title strikes back', 'This is really exciting! Not.', NOW());
```

After creating the database, CakePHP must be configured to work with it. This task is handled in a configuration file in `app/config/database.php`. The appropriate connection values are filled into an array that is read later when the application attempts to connect to the database.

### 3.3.4. Creating a Post Model, Controller, and View

The first step to write some PHP code comes when creating the model. The model is important because it is what the application uses to communicate with the database. All models are stored in the `app/models` directory. Because this model is related to blog posts it is named `post.php`. The following code shows the basic Post model.

```php
<?php

class Post extends AppModel {  
    var $name = 'Post';
}
```
Next up is creating the Posts controller. The file `posts_controller.php` and all controller files are stored in the `/app/controllers` directory. The controller is the location of functions, called “actions,” that are used to communicate between the model and the view. Here is the basic code for the Posts controller with an `index()` action.

```php
<?php
class PostsController extends AppController {

    var $name = 'Posts';

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

The third part is creating the view. The view is the part of the application that generates the display the user will see. Views are stored in the `/app/views` directory inside a folder named after the corresponding controller, in this example `/posts`. This view is related to the `index()` action in the controller so it is named `index.ctp`. The file extension `.ctp` indicates the file is a view file. The following is the view example taken from the blog tutorial.

```html
<!-- File: /app/views/posts/index.ctp -->

<h1>Blog posts</h1>
<table>
    <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Created</th>
    </tr>
    <!-- Here is where we loop through our $posts array, printing out post info -->
    <?php foreach ($posts as $post): ?>
    <tr>
        <td><?php echo $post['Post']['id']; ?></td>
        <td><?php echo $post['Post']['title']; ?></td>
        <td><?php echo $post['Post']['created']; ?></td>
    </tr>
    <?php endforeach; ?>
</table>
```
The view file is a combination of HTML and PHP code. Clicking on any of the links may display an error indicating that the action associated with the link has not been defined yet. As the application is built, the model, controller, and views need to be updated to correspond to each new piece of functionality.

3.3.5. Adding, Editing, and Deleting Posts

Adding, editing, or deleting posts can be done by modifying the controller and adding functions to correspond to each action. For example, function `add()` would contain the code needed to add a post to the database.

```php
function add() {
    if (!empty($this->data)) {
        if ($this->Post->save($this->data)) {
            $this->Session->setFlash('Your post has been saved.);
            $this->redirect(array('action' => 'index'));
        }
    }
}
```

Saving information to the database brings up the need for validation. CakePHP sets up validation rules inside an array named `$validate`. This array is defined in the model and is referenced when the `save()` method is called from within the `add()` function. The following is a simple validation example indicating that `title` and `description` are required fields.
The action of deleting a post consists of creating a `delete()` function in the Post controller and updating the view to add a *Delete* link to pass the Post id to the `delete()` function.

Editing is similar in that the view passes the Post id to an `edit()` function, so the application knows which post to retrieve from the database. Everything is very systematic: first update the model, then add actions to the controller, and last edit the view to use the newly created functions.

### 3.3.6. Routes

“Routing is a feature that maps URLs to controller actions” [CakePHP 2009]. Routes are useful to provide more meaningful information via a URL or to make a URL more readable by the user. The routing configuration is found in `/app/config/routes.php`. The default pattern is: `http://example.com/controller/action/param1/param2`. Using the Post example, the pattern would look like `http://example.com/posts/add` for the `view()` action. In the routes file, `routes.php`, the syntax to produce this URL is:

```php
Router::connect ('/', array('controller'=>'posts', 'action'=>'view'));
```

Routing is not a necessity to use with an application. It is simply a way to make URLs look cleaner and easier to read.
3.3.7. Other Features

CakePHP contains a lot of features to assist with the Web development process. While not all these features were utilized in the blog tutorial, many of them were referenced and the tutorial provided a link for more details. It would be beneficial for a programmer learning CakePHP to read the entire manual to gain a better understanding of CakePHP’s structure and features. A few of the well-known features are listed below.

Components

Components “provide out of the box functionality for several commonly used tasks” [CakePHP 2009]. Component files contain actions that can be used across all the controllers. Components are stored in the /app/controllers/components directory. Some examples of components include: Cookie, Email, Security, and Session.

Elements

“An element contains presentation output that can be pulled into multiple views” [Golding 2008]. The view can call the element in its markup to display the contents of the element. A menu is a good example of code that could be contained in an element, because it will be the same in all pages of the site. Other code that would go well in an element are a header and footer that could also be the same across many different pages. Variables can be sent to elements in the same way they are sent to views, which adds a bit of flexibility to an element’s content. Elements are stored in the /app/views/element directory.

Helpers
Similar to elements, helpers are classes that provide functionality that can be used across more than one view. They are stored in the /app/views/helpers directory. One of the most commonly used helpers is $html->link(). An example of this helper in action is

```php
<?php echo $html->link('Read Post', '/posts/view/55') ?>
```

This line would replace the usual HTML code for producing a link:

```html
<a href="/blog/posts/view/55">Read Post</a>
```

What would be the advantage of using the helper over the HTML version? Links created using the $html->link() helper will not break if things get moved around. CakePHP will check and route the paths for these links, so that they will remain valid despite changes made in the code or directory structure. There are a number of helpers provided in CakePHP’s feature set. Some of these include: HTML (such as the link helper), Form, AJAX, and Javascript.

### 3.3.8. Scaffolding and the Bake Console

“Application scaffolding is a technique that allows a developer to define and create a basic application that can create, retrieve, update and delete objects” [CakePHP 2009]. These actions are often referred to as CRUD operations. In short, scaffolding is a quick way to get a new application up and running. Only a model and its corresponding controller are needed to create a scaffold. It allows a developer to test database connections and logic without having to worry about the view part of the application. To create a scaffold for an application, add a $scaffold variable to the controller. Scaffolding is not intended to be used for an application in a production environment.

The Bake console is CakePHP’s command line tool. Bake is actually just a script, `bake.php`. Using the scaffold generated the basic CRUD actions and database
connections. Now using Bake, CakePHP can generate some generic views that are ready to be customized for a new application. Bake must be run in a bash shell. In this project, Putty is used to communicate with the server, or sometimes work is done in a terminal window directly on the server. To start Bake, run the command `$cake bake`. This command brings up the Interactive Bake Shell.

```
App : app
Path: /path-to/project/app
Interactive Bake Shell

[D]atabase Configuration
[M]odel
[V]iew
[C]ontroller
[P]roject
[Q]uit
What would you like to Bake? [D/M/V/C/P/Q]
```

**Figure 3.1 Interactive Bake Shell [CakePHP 2009]**

The shell provides a list of options the developer can choose from. Once a choice is made, the script asks a series of questions such as whether or not to use route, components, or helpers. According to Golding [2008], the controllers should be baked first. Controllers generated by the script will overwrite any previous controllers used with the scaffold. Next bake the views. At this time the views are generated and are editable by the developer. It should be noted that direct commands can be entered as an alternative to using the interactive shell. For example, to create a controller the command is `$cake bake controller`. To create a specific controller, add the controller name to the end of the command: `$cake bake controller posts`. Using Bake is another tool to get things set up quickly.
3.3.9. CakePHP Conclusion

This research was a general overview of CakePHP. To really learn the framework would take a good deal of time and study. There is a vast amount of information in the manual as well as via the community. CakePHP has a lot of useful features and a number of tools to help developers create applications that are feature-rich and flexible. Due to its popularity and amount of resources, CakePHP would be a good framework for developers who are interested in getting their feet wet in regards to MVC architecture or creating applications in a PHP framework.

3.4. Symfony Jobeet Research

Symfony’s flagship tutorial is the Jobeet Web site. According to Symfony’s project page “The Jobeet Web site is a real-world Web app, developed entirely in Symfony” [Symfony 2009]. The product of the tutorial is a job board. The tutorial is setup in such a way that a section can be completed each day for a total of 24 days. At the conclusion of the tutorial there should be a functional application, which will allow users to post a job or search job listings. There is also an RSS feed so a job seeker can see the latest jobs being posted. A demo Jobeet site can be found at http://www.jobeet.org.

3.4.1. Introduction to Symfony and the Jobeet Tutorial

Similar to CakePHP’s blog tutorial, the environment required by Symfony’s tutorial includes use of a Web server, a database engine, and knowledge of PHP, MVC, and object oriented programming. This project uses the Apache Web server, MySQL 5, and PHP 5.2.6. At the time of this project the latest version of Symfony was 1.4. The
.tar file can be downloaded from the Symfony Web site at http://symfony-project.org. It should be noted Symfony uses the command line quite a bit. This project used the SSH client, Putty, to interact with the Web server.

The first step is to create a directory to house the new project and all its files. This directory was created off the root directory in /home/user/sfprojects/jobeet. Next is to create a subdirectory, /lib/vendor, beneath the base project directory. It is in this subdirectory where the downloaded tarball should be unpacked. The resulting folder should be renamed symfony. The following command, ran from the SSH client, is used to verify the installation:

```bash
$ php lib/vendor/symfony/data/bin/symfony -V
```

This command should display the version number of Symfony that was just installed and the directory location where it was installed.

Different commands issued to Symfony via the command line are referred to as “tasks.” The first task is to generate the file structure and basic files needed to begin the new project. The syntax to create the Jobeet project is:

```bash
$ php lib/vendor/symfony/data/bin/symfony generate:project jobeet
```

Looking at the base directory, several new directories have been created. These directories will be referenced in more detail in later sections. In addition to generating many new files and folders, running this task has created a shortcut to the symfony executable. Now tasks can be issued using `php symfony` instead of using the full path name. The next task is to create the Jobeet application front end. The command to do this is:

```bash
$ php symfony generate:app frontend
```
This command generates a *frontend* directory in the *apps* directory. A large chunk of the work for symfony projects is done with files inside this *frontend* directory. Later sections will deal with these files.

Before going any further, it should be noted that Symfony makes extensive use of other technologies. Two of these technologies are Doctrine and YAML. These tools are discussed in the following two sections.

### 3.4.2. Doctrine

According to the Doctrine Web site, “Doctrine is an object relational mapper (ORM) for PHP 5.2.3+ that sits on top of a powerful database abstraction layer (DBAL). One of its key features is the option to write database queries in a proprietary object oriented SQL dialect called Doctrine Query Language (DQL)” [Doctrine 2010]. What does object relational mapping mean? “Mainstream application software, that interfaces with databases, needs to deal with both object oriented and relational data models. [...] Mediating between [these] data models is an important problem [and] object relational maps are the most common ways to do this mediation” [Mehra 2007]. Symfony uses Doctrine as a means to interact with the database. Because Symfony employs object oriented programming, “instead of writing SQL statements to retrieve records from the database, [Symfony] would rather prefer to use objects” [Symfony 2009]. Symfony makes use of DQL. Below is an example of a query using raw SQL.

```sql
SELECT a.id AS a_id, a.name AS a_name FROM account a;
```

To produce the same query the DQL would be:

```php
$q = Doctrine_Query::create()
  ->select('a.name')
  ->from('Account a');
```
The advantage of using Doctrine over raw SQL statements is database independence. If for some reason the project needed to switch database engines from MySQL to Oracle, for example, the data models would not need to change their code. Doctrine ensures the correct syntax is generated based on the database specified in the database configuration file. Symfony can also be used with another ORM, Propel. However, because the Jobeet tutorial uses Doctrine this project opted to focus solely on Doctrine.

3.4.3. YAML

Both Symfony and Doctrine make use of YAML. YAML stands for YAML Ain’t Markup Language. YAML “is a data serialization language designed to be human-friendly and work well with modern programming languages for common everyday tasks” [YAML 2010]. In fact, according to the YAML Web site the most important design goal for YAML is to be easily readable by humans with a secondary goal of data portability between programming languages. The YAML syntax consists of sets of key:value pairs. Multiple values are comma separated. Multi-line values are preceded with a vertical line “|”. Comments are indicated with a hash mark “#”. An important thing to keep in mind about the YAML format is “indentation must be done with one or more spaces, but never with tabulations” [Symfony 2009].

The Symfony development team has been working on a YAML component, which incorporates base YAML. “Symfony YAML is a PHP library that parses YAML strings and converts them to PHP arrays. It can also convert PHP arrays to YAML strings” [Symfony 2009]. In the Jobeet project, YAML is used both to describe the
schema of the database tables and to customize configuration files for different parts of
the application. Below is the schema of the JobeetJob table in YAML format.

```yaml
JobeetJob:
  actAs: { Timestampable: ~ }
  columns:
    category_id: { type: integer, notnull: true }
    type: { type: string(255) }
    company: { type: string(255), notnull: true }
    logo: { type: string(255) }
    url: { type: string(255) }
    position: { type: string(255), notnull: true }
    location: { type: string(255), notnull: true }
    description: { type: string(4000), notnull: true }
    how_to_apply: { type: string(4000), notnull: true }
    token: { type: string(255), notnull: true, unique: true }
    is_public: { type: boolean, notnull: true, default: 1 }
    is_activated: { type: boolean, notnull: true, default: 0 }
    email: { type: string(255), notnull: true }
    expires_at: { type: timestamp, notnull: true }
  relations:
    JobeetCategory: { onDelete: CASCADE, local: category_id, foreign: id, foreignAlias: JobeetJobs }
```

3.4.4. The Data Model

This section reviews the relational model of the Jobeet application. Here is the
diagram:

![Entity Relationship Diagram](image.png)

*Figure 3.2 Jobeet Application Entity Relationship Diagram [Symfony 2009]*
The previous section showed the JobeetJob table in YAML format. The JobeetCategory, JobeetCategoryAffiliate, and JobeetAffiliate tables are defined in a similar fashion. Defining the database tables is done in `config/doctrine/schema.yml` configuration file. Before creating the tables, the database must be manually created either via the MySQL command line or using SQLyog. Once the database is created, Symfony can be configured to use it with the `configure:database` task. The following is an example to use the Jobeet database:

```
$ php symfony configure:database
"mysql:host=localhost;dbname=jobeet" user password
```

The first value tells Symfony what kind of database engine is being used. In this case, the database is MySQL. The next argument is the name of the database. Last is the user name and password of a user who has been given access to the database. Of course “user” and “password” would be replaced with the actual values. These values can also be set manually by editing the `config/databases.yml` file.

Now that the prerequisite configuration files have been set up, Doctrine tasks can be used to generate the SQL commands to create the tables. To build models and SQL from the schema files:

```
$ php symfony doctrine:build --model
$ php symfony doctrine:build --sql
```

These two commands generate the models and SQL commands. To actually create the tables in the database a third command is run:

```
$ php symfony help doctrine:insert-sql
```

Running the `doctrine:build --model` task creates PHP files in the `lib/model` directory. These files are the classes that are used to interact with the database. Doctrine generates
three classes per table. The first class is a base class that is recreated every time the
`doctrine:build -model` task is executed. The second class is used for objects that
represent a single record in the table. Customizations are done in this second class. The
third class returns collections of objects created using the second class. For the Jobeet
application these classes are `BaseJobeetJob`, `JobeetJob`, and `JobeetJobTable` respectively.
Throughout the tutorial, changes are made to the last two files to introduce new
functionality into the application.

### 3.4.5. Controllers and Views

Once the database and default models are set up, work can begin with the front
end of the application. The first step is to generate a module. This is done with the
`doctrine:generate-module` task.

```
$ php symfony doctrine:generate-module --with-show --non-verbose-templates frontend job JobeetJob
```

This command creates the `job` module and some initial files with default functions inside.
The module controllers are in the `apps/frontend/modules/job/actions/` directory. The
view portion of the module is created in the `apps/frontend/modules/job/templates/`
directory. Another, more general part of the view can be found in the
`apps/frontend/templates/` directory. These three directories and their contents are
discussed in a bit more detail below.

**Actions**

Inside the `actions/` directory there is a single file, `actions.class.php`. This file
contains the functions that will feed content to the view. The default actions include
index, show, new, create, edit, update, and delete. Without doing much, the application is
ready to perform these functions and interact with the database. Throughout the tutorial changes are made to the actions to make them better suited to this particular application.

Templates

The module templates will render whatever the module actions have produced. The index page, for example, displays a list of all the jobs in the jobeet_job table. These results could be output via an HTML table. The code to render the table and the job data is contained within the indexSuccess.php template file. Each action has a corresponding template file. The executeEdit() function is viewed in the editSuccess.php template. The executeNew() function corresponds to the newSuccess.php template, and so on.

Layouts

The third directory, apps/frontend/templates/, contains the file layout.php. This file defines the overall look of the application. It is a global template that will apply to all pages in the application. This is where css and javascript files are linked. This is also the place to enter static portions of the view. This includes the main header and logo, the footer, and the navigational menu. Within the layout there is a portion that will output content sent from the module actions and template files.

```php
<div class="content">
<?php echo $sf_content ?>
</div>
```

The $sf_content variable contains all the content sent from the module. This is the dynamic portion of the view. The difference between module templates and the layout can be seen in the following diagram.
3.4.6. Routing

Routing in Symfony is similar to routing in CakePHP. The usual PHP URL, which looks like this, /job.php?id=1 is turned into something nicer and more readable like this /job/show/id/1. The difference comes in how the routing is accomplished. In Symfony routing configuration is done in the routing.yml configuration file. “A route has a name (homepage), a pattern (/:module/:action/*), and some parameters (under the param key)” [Symfony 2009]. Below is the routing for the Jobeet application:

```yaml
homepage:
  url: /
  param: { module: default, action: index }

default_index:
  url: /:module
  param: { action: index }

default:
  url: /:module/:action/*
```

Routing is done via pattern matching. The order of the routes is important because the first pattern to match the URL request is the route that will be used. Words beginning with a colon are variables. In the above code, the default_index route will match a / followed by a module name. When a request comes in matching /:module, the action that will be executed is executeIndex(). The default route acts as a catch all, so it is placed last in the file.
Routing customization can be done, but it might take a little extra work. For a more descriptive URL, a more descriptive pattern can be used. For this example, the URL should indicate the company, location, and position in a job posting. A pattern to match this combination of information would look like this:

url: /job/:company/:location/:id/:position

The default routing will create a URL like the following:

http://www.jobeet.com.localhost/frontend_dev.php/job/Sensio+Labs/Paris%2C+France/1/Web+Developer. This URL is still a bit messy looking. Symfony will try to use a “slugify” method to clean it up. This method will use pattern matching to make the entire URL lowercase and replace out-of-place characters with a dash. The resulting URL will look like this: http://www.jobeet.com.localhost/frontend_dev.php/job/sensio-labs/paris-france/1/web-developer. The next section will look at testing the slugify() method.

3.4.7. Unit Tests and Functional Tests

Symfony has the ability to run tests to check actions, routes, and different functionality. There are two kinds of tests: unit and functional. “Unit tests verify that each method and function is working properly. […] On the other hand, functional tests verify that the resulting application behaves correctly as a whole” [Symfony 2009]. Test files are stored in the test/ directory. Unit tests and functional tests are divided in their own subdirectories.

Unit Testing

For unit testing, Symfony introduces the lime testing library. Lime tests are based on comparisons. Provided a given input, what are the results? How do the results compare to the expected output? The naming convention for unit tests is the name of the
class followed by Test. The tutorial begins with testing the Jobeet class, thus the test is named JobeetTest.php. The first line in a test file includes the unit.php bootstrap file to initialize things. The next line creates a new lime_test object. This line needs to contain the number of tests to be run, which will need to be updated as new tests are added.

```php
require_once dirname(__FILE__).'/../bootstrap/unit.php';
$t = new lime_test(1);
```

The test file is executed via the command line. This command and the results of a successful test are shown in Figure 3.4, in which case six tests were run.

![Figure 3.4 A Successful Unit Test [Symfony 2009]](image)

The actual test code corresponding to Figure 3.4 is:

```php
$t->is(Jobeet::slugify('Sensio'), 'sensio');
$t->is(Jobeet::slugify('sensio labs'), 'sensio-labs');
$t->is(Jobeet::slugify('sensio labs'), 'sensio-labs');
$t->is(Jobeet::slugify('paris,france'), 'paris-france');
$t->is(Jobeet::slugify(' sensio'), 'sensio');
$t->is(Jobeet::slugify(' sensio '), 'sensio');
```

In this test, the `slugify()` method used in the routing section is being tested. Looking at the first test, if “Sensio” is in the URL the `slugify()` method should change it to “sensio.” This particular unit test uses comparison to find out if the change is actually happening. This test passed indicating the `slugify()` method is working correctly. The other five tests
are checking slightly different scenarios. Figure 3.5 shows the result of a failed test.

![Figure 3.5 A Failed Unit Test [Symfony 2009]](image)

Notice that there are some descriptive phrases associated with this testing set. These descriptions are text that the developer can enter when creating the test. The tests are also numbered and the failing test is specified, which helps with debugging the offending code.

Tests can be added for the remaining classes such as category. Each test file can be run individually or all the test files can be run in one batch. To do a batch execution, run the test:unit task without specifying a class. Writing unit tests may be tedious, but it’s important to be able to find and correct problems early.

*Functional Testing*

Functional testing is concerned with the overall function of the application. What if a change in one part of the application breaks something in a different location? A functional test would be able to detect these breaks, which a developer may not find out about until he or she stumbles upon an error. “Functional tests are a great tool to test [an] application from end to end: from the request made by a browser to the response sent by
the server. They test all the layers of an application: the routing, the model, the actions, and the templates” [Symfony 2009]. In order to perform functional testing, Symfony employs a special class, sfBrowser. This class acts as a browser and uses functions, such as get(), post(), and reload(), to emulate browser actions. The purpose of using this class is to simulate a user navigating the application. Functional tests are written in code blocks. The following block demonstrates the basic test structure.

```php
$browser->
    with('request')->begin()->
        isParameter('module', 'category')-
            isParameter('action', 'index')-
        end();
```

Each test must begin with `with('request')->begin()` and end with `end()`. The lines in between are the tester functions. In the above example, `isParameter()` is the tester function. In simple terms, the test is requesting the index page of the category module.

Similar to unit tests, the command to execute a functional test is:

```
$ php symfony test:functional frontend categoryActions
```

There are a great number of other tester functions to check various scenarios.

Unit and functional tests can be quite extensive. This section has only touched on a very small piece of the testing framework. Testing is an important part of application development to catch errors and make sure the application behaves as expected.

### 3.4.8. Admin Generator

The admin generator creates a backend interface for the application. The command to create the backend is:

```
$ php symfony generate:app backend
```
Similar to creating the frontend, each module will need to be generated using the ```doctrine: generate:admin``` task. Also similar to the frontend, the backend has its own routing file, modules, templates, and layout all contained in the ```/apps/backend/``` directory. Automatically upon generating a backend module, the application is ready for CRUD operations. The backend also supports pagination, sorting, and filtering by default. A lot of customization can be accomplished via the configuration file ```generator.yml``` Each module, such as job and category, has its own ```generator.yml``` file. The tutorial provides a nice breakdown of the sections of the file [Symfony 2009]:

- **actions**: Default configuration for the actions found on the list and on the forms
- **fields**: Default configuration for the fields
- **list**: Configuration for the list
- **filter**: Configuration for filters
- **form**: Configuration for new and edit forms
- **edit**: Specific configuration for the edit page
- **new**: Specific configuration for the new page

### 3.5. Online Exams and Templates

The template resulting from this project is, in other words, an exam creator. A quick Google search results in a list of many offerings for creating exams. The problem with these applications is that they are often proprietary and must be purchased for what can be a high fee. Other problems include a lack of support for non-Latin alphabets, lack of customizability, limits on question types, or lack of support for audio and video components.

One company that does provide software with many of these features is Owl Testing Software [Owl 2009]. According to their Web site, Owl’s software is “the proven solution for creating, administering, and rating oral, aural, reading, and writing tests online.” The company works with K-12 and higher education, as well as corporate
and government entities. Owl’s product sounds great; however, a demo must be requested in order to see what the application looks like. This entails setting up an appointment with a service representative and only then does a potential customer get a glimpse of the software. A recent request revealed Owl’s pricing structure and it was quickly determined the cost would be much too high.

A popular language placement exam in use in many universities is WebCAPE, which was developed at Brigham Young University. WebCAPE is not an exam creator; rather it is a ready-made exam that universities can register to use. Unfortunately, as Ballester [2007] points out, “this placement exam is a valuable tool, […] however […] WebCAPE is not the appropriate tool because it only evaluates reading comprehension, vocabulary, and grammar by means of multiple-choice, matching, true-false statements, and fill in the blank.” Similar to many institutions, Ballester “subscribes to the philosophy that listening, writing, and speaking are essential components of Spanish proficiency” [Ballester 2007]. In addition to the lack of these components, a look at the pricing structure may be a large factor in determining if a department will be able to utilize this service.

Owl Testing Software and WebCAPE are just two of a multitude of products that can be used for language placement. However, these products and others have either functional or feasibility shortcomings that this project attempted to address.
4. EVALUATION AND RESULTS

A main goal of the project was to evaluate different PHP frameworks and then select one for the creation of an online language placement exam template. Based on the findings of the research that was conducted in section three, a framework was chosen to develop the template. This section will go over the findings that led to the framework selection as well as detail why that framework was chosen. There are some obvious differences between the technical report and the proposal. This section will also attempt to address those differences and explain the reasons why some of the points in the proposal were not realized.

4.1. Evaluation of the Frameworks

This project looked at two PHP frameworks and ultimately selected one to do more in depth research and begin creating an application. Below is a simple table identifying some key characteristics. The marks indicate whether that feature was present in the framework.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Documentation</th>
<th>MVC</th>
<th>Routing</th>
<th>Integrated Validation</th>
<th>ORM</th>
<th>Integrated Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>CakePHP</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symfony</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

Based on this evaluation, the frameworks look very similar. In fact, they share a lot of the same features. The difference comes in the details or how each framework implements those features.
4.1.1. Documentation

First and quite important for any project is documentation. Both frameworks have good Web sites with lots of information. A common complaint for CakePHP, however, is the lack of well-detailed and organized documentation. A look at the site leads one to the API and The Manual. These are good sources to find information about functions and helpers. The problem is mainly with the lack of good examples. There is documentation about a function and what it does, but not how to use it in an application. A large basis for this opinion is the Blog Tutorial. It may be the problem is with the tutorial rather than the framework. The tutorial seems to only scratch the surface when it comes to the power that CakePHP developers can use. On the other hand, Symfony’s Jobeet tutorial seems to try to use a number of features to show the user what he or she can do with the framework. The 24-day tutorial is in depth enough to feel like it could be used as a real world application. There are also books that have more advanced topics and additional examples covering more complex situations.

4.1.2. MVC

The Model-View-Controller, or MVC, architecture is a common feature the frameworks share, as do a lot of frameworks. MVC is not tied to frameworks; rather it is a way of coding that simplifies debugging and maintenance. By separating the different parts of code, it’s easier to pin point problems. CakePHP uses a strict naming convention. If a file, table, or class is not named consistently CakePHP may not be able to function as smoothly because it can’t find the parts the application needs. In this respect, Symfony may seem friendlier because with one command all the files are generated and named appropriately without developer intervention. The skeleton is
created from the file structure and file names down to the basic classes. All the developer has to do is fill in the application specific code.

4.1.3. Routing

Both frameworks employ some form of routing. For CakePHP, the routing is defined in the `routes.php` file. Routes are set up as arrays with key value pairs. The order of routes matters as the file is read in order and the search stops as soon as a matching route is found. Symfony uses YAML to set up routes in `routing.yml`. Symfony’s routes also use pattern matching, so order does matter. A page’s route is defined by a pattern, which is composed of a URL and parameters. Although similar, the implementation and syntax is very different.

4.1.4. Integrated Form Validation

Validation is a feature common to both frameworks. The difference is how the validation is defined. CakePHP uses the `$validate` array, which includes key value pairs indicating a field name and its corresponding requirement. Arrays can be broken down into sub-arrays to set up multiple validations on a single field. The `$validate` array can get quite messy, but it’s a relatively straightforward approach. Symfony uses a special `validator` class that is paired with a `widget`. Form elements such as an input textbox or textarea can be defined with a widget. Things can get confusing if the developer doesn’t know which validator to use with which widget, but usually the widgets and validators are named in such a way that one can deduce which should be used.
4.1.5. **ORM**

CakePHP does not use object relational modeling. The developer must set up the database by hand and be sure to name the tables according to convention. The related models must then also be created by hand and named correctly. Configuring the application to use the database is defined in the `database.php` file. Symfony’s use of YAML and Doctrine seems easier because once the table schema is defined in `schema.yml`, a Doctrine command can be used to generate the tables, the models, and other associated files such as basic forms and controllers. The use of `fixtures.yml` to hold initial testing data is also helpful when it comes to testing and building the application.

4.1.6. **Integrated Testing**

Unfortunately CakePHP’s tutorial did not talk about testing. However, a look at The Manual does turn up some information about testing an application. According to this documentation CakePHP does not have integrated testing, but it does have integrated support for testing. There is a testing framework, SimpleTest, which can be downloaded and installed to work with CakePHP. Symfony has integrated testing with the Lime testing framework. Symfony supports both unit and functional testing. New tests can be added to the testing file and executed as often as necessary to check everything after making a change or adding new code.

4.2. **Framework Selection**

Both of the frameworks are good platforms for creating Web applications and both have a number of useful features. After researching each framework, Symfony was
chosen as the framework to use in the attempt to create a language placement exam prototype.

How and why was Symfony chosen when each framework’s feature set is so similar? This project was approached from the perspective of a developer who has no experience with frameworks. Symfony was chosen because it seemed to better encompass the rapid development idea. With a few terminal commands a basic application can be up and running complete with standard CRUD operations. The same is true of CakePHP, but Symfony’s implementation seemed more straightforward and simple.

Another factor that went into the choice was each framework’s community and support. This included the Web site, forums, tutorials and general documentation overall. CakePHP does have documentation, but it is not as complete and organized as that of Symfony. Symfony also has a lot of tutorials written by the developers and people involved with the project. That lends the tutorials a sense of validity that was missing in the CakePHP world. There are a lot of CakePHP tutorials, but they are scattered in personal developers’ blogs and tutorial sites. These examples are not necessarily supported or sanctioned by the CakePHP team.

Yet another factor of note was the fact that Symfony brought in a number of other technologies into its basic Jobeet tutorial. These other technologies include YAML, the Lime testing framework, AJAX and jQuery, and even pieces of the Zend framework. One might think bringing in these other tools would complicate things, but actually these are real world tools that a developer could very well need to use. It was nice to be able to see how to use these tools in a Symfony application. No need to look for hacks or plug-
ins to get things working. Symfony already provides a variety of plug-ins to support third party pieces of software. Being introduced to new technology was also a good experience for a developer who is interested in learning new things. It was an eye-opening look into not only Symfony, but also a few other good pieces of software.

In the end it may simply be a matter of personal choice. What seems easier or more logical to one person may not seem that way to someone else. It should also be mentioned whether a person prefers developing in a more GUI-oriented environment or if a person is comfortable with the command line. CakePHP has an option to use the command line more if so desired. However, using the command line is an integral part of Symfony’s development procedure. Producing the necessary files can be done without the command line, but using Symfony commands is much easier and faster. One command can generate all the files and directory structure needed to fill in an application-specific code.

4.3. Project Results

The results of the project varied quite a bit from the proposal. The main reason for all the differences was the limited time in which to complete the project. Initially the project was supposed to cover three PHP frameworks. Early on this was cut down to two: CakePHP and Symfony. These two frameworks were chosen over Zend because they are both very popular frameworks with good Web sites and large communities. The other large difference was the lack of testing of a prototype language placement exam template. The main reason for this was simply because of the amount of time needed to study and research the two PHP frameworks by the time the end of the project was reached. A prototype language placement exam does exist with some rudimentary
functionality. Progress with the prototype exam will be included in the project’s final documentation.

4.3.1. Prototype Screenshots

As was mentioned previously, a rough prototype was created at the end of the project. The following are screen captures of some of the main screens. These pages may look like simple HTML pages, but they are actually views generated by Symfony using a combination of pieces such as the layout, forms, and partials. The header, navigation menu, and footer are common to all pages because they are part of the layout. The question list is a template file of the PlacementQuestion module. The main content area of Figures 4.3 and 4.4 is composed of partials of the PlacementQuestion module. The partials are made of a form, which is an instance of the PlacementQuestionForm class, and an embedded form, which is an instance of the PlacementAnswerForm class. Other menu items such as Exams, Languages, and Categories are very similar to these figures and thus, are not included in these screenshots.
Language Placement Exam

Welcome to the language placement exam at the University of Kansas
This application is a template which you can fill with your own questions and answers.

Instructions
- Create an exam
- Add a new question
- Select a question type
- Insert some answers
- Save

Figure 4.1 Home Page

Language Placement Exam

Question List

<table>
<thead>
<tr>
<th>Number</th>
<th>Placement Exam</th>
<th>Placement language</th>
<th>Placement category</th>
<th>Type</th>
<th>Question</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hebrew Placement Exam</td>
<td>English</td>
<td>Grammar Comprehension</td>
<td>MC</td>
<td>Last night the dog ____ into the house.</td>
<td>Edit</td>
</tr>
<tr>
<td>2</td>
<td>Hebrew Placement Exam</td>
<td>Chinese</td>
<td>Grammar Comprehension</td>
<td>MC</td>
<td>Test question</td>
<td>Edit</td>
</tr>
<tr>
<td>3</td>
<td>Hebrew Placement Exam</td>
<td>Chinese</td>
<td>Grammar Comprehension</td>
<td>MC</td>
<td>Tomorrow we ______ spaghetti for dinner.</td>
<td>Edit</td>
</tr>
</tbody>
</table>

Choose an action ▼ or + New

Figure 4.2 Question List
Add New Question

Question

Exam
Hebrew Placement Exam

Language
Chinese

Category
Grammar Comprehension

Number

Type
- select -

Question

Answer Choices

Enter answer selection

Mark the correct solution

Save

Back to List

Figure 4.3 Add New Question
Editing Question

Question
- Exam: Hebrew Placement Exam
- Language: Russian
- Category: Grammar Comprehension
- Number: 1
- Type: Multiple Choice

Question

Last night the dog ___ into the house.

Answer Choices

Mark the correct solution

run
run
running
runs

Save

Back to list  Delete
5. **FUTURE WORK**

At the end of the project a functional Web-based placement exam template was supposed to be created; however, as often happens, things did not work out that way. The idea is still a platform from which to progress in the future. A lot of things from the proposal still hold true. The language instructor should be able to create the exam himself or herself without the need of a programmer or technical person. The instructor should be able to add audio or video content easily. He or she should be able to create the standard multiple choice, true/false, and fill-in-the-blank type of questions as needed and be able to arrange them as necessary.

Future work will be creation of the template. Even as the project comes to a close, there are new tutorials to work through and more documentation to read in the effort to accomplish this task. Since the selection of Symfony as the framework to be used, many additional tutorials have been worked through. Jobeet was a good place to start for the basics, but more complex designs call for more research. One of these designs for example, is how to use embedded forms. This topic was not covered in Jobeet, however it is discussed, with examples, in one of the Symfony published books. All the books are posted for free on the site, but a paper copy can be purchased as well.

Online language placement exams are a need not yet met on a wide scale. There are a few viable commercial products available; however, such products may not be suitable for use with non-Latin alphabets. Going with open source products allows developers to create new ways of handling exam content with the added benefit of community, instructor, and student feedback which can be taken into account and greatly improve the application.
6. CONCLUSION

This project focused on two PHP frameworks: CakePHP and Symfony. Both frameworks contain a number of features and are good points from which to develop robust Web applications. Using frameworks continues to grow in popularity as Web applications continue to become an ever-greater part of daily life. Being able to quickly create feature-rich applications using standard well-established practices is a plus for developers who engage in such work. Indeed, many companies are using Web applications to communicate with their customers and frameworks are one more way to accomplish this contact.

Based on the research in this project, Symfony was chosen as the framework for pursuing future work. This decision was based on many factors, but mainly on the experience of working through each framework’s tutorial. Unfortunately, the end of the project did not result in the creation of a functional language placement exam template, but there does exist a rough prototype. The next step for the project is to continue working with this prototype to eventually produce a functioning template.
BIBLIOGRAPHY AND REFERENCES


