CE903 Group 1

Final Report

The Faculty Cooperative – An Interactive Website for “Crowd Sourced” Academic Entrepreneurial Activity

**Team Members:**

ALSHUAIBI, ENAAM ABDULMONEM O

CHEN, XIAOFENG

HAMID, BAHIT BIN

HERASCU, IRINA

THOMAS, SARAH ANNE

**Supervisor:**

GARDNER, MICHAEL

**Date:** 19 March 2014

Table of Contents

1 Introduction 3

1.1 Background 3

1.2 Server Side Technology 3

1.3 PHP Frameworks 3

2 Software Tools 4

2.1 Laravel 4

2.1.1 Laravel Migrations and Seeds 4

2.1.2 Laravel Eloquent ORM 5

2.2 Bootstrap 5

2.3 LAMP/MAMP 5

2.4 GitHub 5

3 System Design 5

3.1 Use Cases, 5

3.2 Class Definitions 5

3.3 Overview of Code Listings 6

4 Testing 6

5 Conclusions 6

5.1 Successes 6

5.2 Shortcomings 6

5.3 Possible Extensions 6

5.4 Project management 6

6 Appendices 7

6.1 User Documentation 7

6.2  B. Minutes of All Group Meetings + (possibly) others (details of design, tests, etc.) 7

6.3 C. Full Code Listing 7

7 Glossary 7

8 References 7

# Introduction

## Background

description of problem domain and project objectives

## Server Side Technology

There are a number of web development server side languages in common usage. The team considered these in relation to the teams skills, available jobs and popularity. Figures for available jobs (Feb 2014, UK) were obtained from [www.itjobswatch.co.uk](http://www.itjobswatch.co.uk). Figures for popularity were obtained by searching Google for the relevant file extension using for example: filetype:php. Whilst this method is rather crude it does support the assertion that PHP is the most popular server language on the Internet.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Team members have skills | Matching job adverts | Google File Results/Millions |
| Ruby | No | 2310 | 14 |
| ASP.NET(MVC) | Little | 4300 |  |
| ASP.NET(Web Forms) | Some | 1020 | 593 |
| PHP | Yes | 6318 | 2620 |
| JSP | Some | 1286 | 88 |
| Python | Little | 2940 | 1.2 |

Table 1 - Popular Server Side Technologies Compared

The team members are currently all following the Advanced Web Engineering [1] MSc program, which includes compulsory modules covering some ASP.NET, and JSP. It was felt that adding some additional PHP experience to this would be beneficial to our education and future career prospects.

## PHP Frameworks

Once the decision has been made to use PHP for the server side of our system, we faced the choice of a number of PHP frameworks. Whilst it was not entirely necessary to use a framework, we felt that it would speed up development. Frameworks can take much of the repetition out of such routine web site tasks such as authentication and validation.

We considered a number of frameworks in terms of licencing, ease of use, testing and documentation. One team member (Bahit) had some experience with CodeIgnitor and another (Sarah) experimented with CakePHP in the initial stages of the project. Based on experiences and research from various web forums we drew up the comparison in Table 2 - Framework Comparison [2][3]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Laravel** | **CodeIgniter** | **Zend** | **CakePHP** | **Symfony** |
| **Licence** | MIT | MIT | BSD | MIT | MIT |
| **Documentation** | Good | Good | Good | OK | OK |
| **Ease of Use** | OK | OK | Harder | Easy | OK |
| **Testing** | PHPUnit | No Library | PHPUnit | PHPUnit | PHPUnit |
| **Problems** | Installation quite involved | May not continue to be supported | Not Free | Not flexible enough to fulfil requirement |  |
| **Advantages** | Very popular and well rated | Good community support |  | Easy to use and install |  |

Table 2 - Framework Comparison

Table 3 shows recent trends in PHP framework popularity. The raw data was obtained from Google Trends Laravel has shown a steady increase in popularity since its release in June 2011 [4].

Table 3 - Trends in PHP Framework Popularity

Based on the previous findings the most appropriate choice for a framework seemed to be either CodeIgniter or Laravel. As there was some doubt that CodeIgniter would continue to be developed we chose to use Laravel.

# Software Tools

The following sections describes the software tools utilised

## Laravel

### Laravel Background

Laravel is a free open source PHP framework based on the MVC paradigm [5]

When PHP version 5.3 was released in 2009, it introduced several new features that improved the functionality of Object-Orientated applications. Laravel’s creator, Taylor Otwell felt that the PHP frameworks available at the time were not leveraging the new features in PHP. So he created Laravel “*simply to solve the growing pains of using CodeIgniter PHP framework*” [6].

The first version of Laravel was released in June 2011 and it is now up to version 4.1

### Laravel and Composer

Laravel uses *Composer* for dependency management and installation of packages [7]. It works on a per project level to keep track of which library the project requires, and which libraries those libraries in turn require.

### Laravel Migrations and Seeds

Laravel includes a versioning system for databases, to allow for any structural changes to the database to be shared and tracked. This is accomplished by creating and running a class, that extends Laravel Migration. Any changes go into a new class and these are executed in order, or can be rolled back. For example if a developer changed a table name this would go into a new class saved as a file with an auto generated name reflecting creation time. If another developer then changes the table name again, another class is created, and added to the list of classes to be run to update the database. The problem with this method is that the structure of the database will eventually be contained within many different class files, obscuring the structure of the database and making maintenance harder.

As the project already has the version control capabilities provided by GitHub, we decided not to stack Laravel version control on top of GitHub. Instead we kept only a single file of class Migration and used this to reflect all changes to the database. If we needed to roll back any changes we could do this by reverting the single file on GitHub.

Migrations are made to the database with the following steps:

1. Empty the migrations table in the database. This will allow us to run a migration that has been run previously
2. Use *Laravel’s* command line interface: php artisan migrate

*Laravel* also has a method for seeding the database. A test set of data is included within a class that extends Seeder and run with the following command line: php artisan db:seed

### Laravel Eloquent ORM

*Eloquent* is *Laravel’s* (Object Relational Model) ORM implementation [5]. Each table in the database has a corresponding model that can interact with it. *Laravel* uses certain naming conventions to facilitate the mapping. A class name becomes a singular version of the table name, in Pascal case. For example the table name skills will map to the class Skill. The following code shows the Skill class

<? php
class Skill extends Eloquent
{
}

This is all that is required to map a row from the database to an object in the application.

### Laravel Validation

Enaam might like to do this bit?

### Laravel Blade Templating System

### Laravel Package Management - Using Imagine

*Imagine* is a PHP image manipulation library. We added this library to *Laravel* as a package via *Composer.* *Laravel* uses facades as a static interface to classes within the inversion of control (IoC) container. The IoC container is used to manage class dependencies.

?????

## Bootstrap

IRENA

## LAMP/MAMP

BAHIT

## GitHub

FENG

## Trello

FENG

# System Design

brief recap of requirements  - system architecture  - main components and their relationships (interface, dbase, servers and clients, etc.)  -

Implementation

- programming language issues

- details on implementation of key components of the system (including hardware if applicable)

- client / server model realisation (if applicable) - overview of code listings (e.g., which files correspond to which classes or functions)

- tools used to produce the code, some indication of which parts of the code have been generated automatically

# Use Cases

This section summarises how each of the use cases set out in the SRS document cite has been achieved.

Probably repeat diagram from SRS here

We could indicate level of implementation from 0-100%. Most require validation, error checking and the authentication to be implemented.

### Use Case Description Implementations

**New User Registration**

BAHIT

**User login**

BAHIT

**Search Public profile/ Search Ventures**

The search page allows any user to search public profiles by name and skills offered and to search for ventures by their titles or the skills offered.

This could easily be extended to allow search of bio details and venture description – any volunteers???

**View Public profile**

As a result of a search or by seeing a users name in the team of a venture any user can view another users public profile. A URL can also be provided or shared to allow direct access to a public profile.

**View ventures details**

As a result of a search any user can view a venture page. A URL can also be provided or shared to allow direct access to a venture page.

**Edit private profile**

A registered user can edit their profile

**Contact another user**

Clicking the contact link in a public profile allows the user to send a message

**Create a new venture**

Registered user from the University eco-system a user can create a new venture by entering a title and then proceeding to the edit venture page to full in the remaining details.

**Enter private team area**

Not implemented – any volunteers?

**Build team**

Sort of started but not finished

**Site administrator**

Not implemented

## Class Definitions

Any volunteers?

## Overview of Code Listings

#### app/routes.php

This file contains all the routing instructions for the application. Most of the routes direct requests at a named controller to separate application logic from routes. For example the URL: /publicProfile/1 follows this route:

Route::get('publicProfile/{id}',

'ProfileController@showPublicProfile');

This routes a request with the parameter *id=1* to the *showPublicProfile* function of the *ProfileController*.

Routes can go straight to views instead of controllers, for example the route to the search page view:

Route::get('search', function()

{

return View::make('search');

});

#### app/models/

This folder contains classes that correspond to tables in the database. These classes extend the Laravel Eloquent class and form the foundations for Laravel’s object relational mapping (ORM). By convention each class is named as a singular of the plural table name, with a capitalised first letter. Thus the class *Message* maps to the table *messages*. Column names in the database correspond to properties in the class, so the *body* column of a row in the corresponding *messages* table can be set by:

//create a new object of Message Class

$message = new Message;

//body property now available and maps to database column

$message->body = ‘This is some text in the message body’

Validation can also be added to models

Enaam ?

#### views/createVenture.blade.php

This view allows a logged in user ??user type?? to create a new venture and give it a title. This creates a new *ventures* record in the database and assigns the logged in user as a team member BUT IT DOESN’T AS NO AUTHENTICATION DONE YET. The user is then presented with a link that will take them to the *editVenture* view with the *id* corresponding to the newly created venture.

#### views/editVenture.blade.php

This view edits any venture in the database EDITOR AUTHENTICATION.

If details already exist in the database these will appear in the editable form areas for the user to adjust as they see fit.

In the skills the venture seeks area the checkboxes reflect the state of the *skill\_wanteds* table in the database. Any changes here are reflected back to the database.

A loop has been used in this view to avoid repeating the skill category for every skill in the list, but just print out the skill category as a header. The *Laravel Blade* template does not allow for variable assignment, as it is not recommended to assign variables in views. However, as this is more a view issue than a control issue, we felt using a little embedded PHP here to be an acceptable solution.

<?php $category=$skill["category"]?>

This just keeps track of the current category. Interestingly, the Blade mark up below does work for variable assignment, but as it also performs a PHP echo back to the page it could not be used in the case

{{$category=$skill["category"]}}

Due to the nature of HTML checkboxes, to maintain state of the checkboxes an array of arrays has been returned to the view from the controller instead of an array of objects.

#### views/showProfile.blade.php

This page displays the fields from the *users* table that are available for any other user to view. Offered skills are obtained fro the *skill\_offers* table.

It also provides a link to send a message to this user.

#### views/editProfile.blade.php

This page allows a logged in user to edit the information that is displayed in the public profile with *showProfile.blade.php.* The user can upload a new photo and edit their biography text as well as change the list of skills they offer.

#### views/readMessage.blade.php

This allows a user to view their internal messages. This page uses jQuery to allow the message body to slide down when the message subject button is clicked. We felt that a simple sliding animation here helped users to navigate and understand the page. An AJAX request sent via jQuery marks messages that have been read with a request via the *messageController* . This updates the *read\_flag* in the *messages* table. Clicking on the message subject also reveals a link to respond to the sender of the message.

#### views/sendMessage.blade.php

This allows for a message to be sent to a user using the internal messaging system. The message is sent from the logged in user

#### views/search.blade.php

This page allows for various searches to be performed. This has been split into several different searches, as the logic required for each search is very different. For example it would not be meaningful to search the entire site/database for ‘php’. This phrase only occurs once in the database in the *skills* table. It is however, more meaningful to search for *all users who offer php as a skill* or *all ventures seeking php skills*.

The searches are all based on SQL LIKE statements so empty search fields list all possible results. We only have a small dataset so this does not present a problem but in a real system some sort of pagination or limit on the number of results returned would be required.

The *search for user by name* selects all user names that contain the search string. The results are presented as links to that users profile page. The *search venture title* works in a similar fashion presenting links to ventures as results.

The *search for skills offered* does not include all skills in the system as possible results, but only searches the subset of skills actually being offered by users. So for example, as no users are offering the skill of Garden Design, a search for ‘garden’ yields no results. A search for ‘java’ will display all offered skills containing the string ‘java’ as links. Clicking on one of these links will display all users offering Java as a skill. Clicking on one of these users will take you to their profile page, and allow you to contact them with the messaging system,

The *search for skills wanted by ventures* works in a similar fashion to the previous search. This time it searches for skills that ventures require, and links to the relevant venture page.

#### views/footer.blade.php

Bahit

#### views/header.blade.php

#### views/marketing.blade.php

#### views/carousel.blade.php

Irena?

#### views/home.blade.php

Irina??

#### controllers/MessageController.php

This controller provides the database interactions for the *sendMessage* and *readMessage* views.

#### controllers/ProfileController.php

This controller provides the database interactions for the *showProfile* and *editProfile* views.

#### controllers/SearchController.php

This controller provides the database interactions for the *search* view.

#### controllers/TeamController.php

This controller provides the database interactions for the *editTeam* view.

#### controllers/VenureController.php

This controller provides the database interactions for the *viewVenture*, *editVenture* and *createVenture* views.

#### controllers/UsersController.php

Bahit?

# Testing

- strategy - unit level (black box / white box) - integration level - acceptance

- examples of important test results

(A complete set of tests may not be necessary, as long as it can be shown that your strategy would result in a fully tested system.)

# Conclusions

## Successes

## Shortcomings

## Possible Extensions

- system (successes, shortcomings, possible extensions) - methodology/language/tools (Were they effective? Were they appropriate?)

## Project management

(How was the project managed? How effectively was it managed? How were management problems handled? etc.)















































# Appendices

## User Documentation

## B. Minutes of All Group Meetings + (possibly) others (details of design, tests, etc.)

## C. Full Code Listing

(Hard copy is unnecessary. Please save your code in well structured folders in the CD)



# Glossary

**External eco-system** includes professionals and advisors from non-academic organizations and businesses

**University eco-system** includes students, and staff based in university

**Venture** a business idea or entrepreneurial activity requiring a team leader and team members

**HTTP** (hypertext transport protocol) is a communication protocol used by the Internet to communicate from one node to another.

**HTTPS** (hypertext transport protocol secure) is a communication protocol for secure communication over a computer network. Layering HTTP on top of SSL/TLS, adding it to standard HTTP communication.

**SSL** (Secure Socket Layer) a cryptographic protocol designed to provide secure communication over the Internet.

**UX** (User Experience) mainly deals with how users interaction and how they perceive the website design and layout.

**UI** (User Interface) mainly deals with the design of the layout of the website.

# References

[1] “University of Essex :: Module Directory :: Module details.” [Online]. Available: https://www.essex.ac.uk/modules/default.aspx?coursecode=CE881&level=7&period=SP. [Accessed: 07-Feb-2014].

[2] “Choose the right PHP framework | Web design | Creative Bloq.” [Online]. Available: http://www.creativebloq.com/design/choose-right-php-framework-12122774. [Accessed: 12-Mar-2014].

[3] Phpframework, “PHP Frameworks.” [Online]. Available: http://www.phpframeworks.com/. [Accessed: 12-Mar-2014].

[4] “Google Trends.” [Online]. Available: http://www.google.co.uk/trends/. [Accessed: 28-Jan-2014].

[5] “Laravel - The PHP framework for web artisans.” [Online]. Available: http://laravel.com/. [Accessed: 12-Mar-2014].

[6] M. Surguy, “History of Laravel PHP framework, Eloquence emerging - Maks Surguy’s blog on PHP and Laravel.” [Online]. Available: http://maxoffsky.com/code-blog/history-of-laravel-php-framework-eloquence-emerging. [Accessed: 12-Mar-2014].

[7] N. Adermann and J. Boggiano, “Composer.” [Online]. Available: https://getcomposer.org/. [Accessed: 12-Mar-2014].