Classroom Quiz System

Final Report for CS39440 Major Project

Author: Alexander Taylor (amt22@aber.ac.uk)
Supervisor: Mr. Chris Loftus (cwl@aber.ac.uk)

13th April 2017
Version: 1.1 (Draft)

This report was submitted as partial fulfilment of a BSc degree in
TODO: Computer Science With A Year In Industry (G401)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name .....

Date .....

Consent to share this work

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name .....

Date .....

Acknowledgements

I am grateful to...

I’d like to thank...

Abstract

This project converns creating an application that can be used within lectures by lecturers to allow students to answer questions and for the lecturer to display live results of these questions. This project is based on a preexisting application, called Qwizdom (TODO: cite) but this project aims to improve upon Qwizdom in several ways.

The system is a web based application built in Laravel, a PHP based framework. The system provides two main functions, firstly the running of questions on their own, and the second is to stream lecture slides with questions embedded within the slides to students. This system improves upon Qwizdom in several ways, it runs via a website rather than via a PowerPoint extension, meaning more lecturers can use it, and it provides more functionality such as more possible answers and students cannot submit their own answers as they can with Qwizdom.

Development followed an Extreme Programming approach, utilising a number of practices to help development.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Background & Objectives

1.1 Background

1.1.1 Qwizdom

Currently lecturers at Aberystwyth University have the option to use a service called Qwizdom (TODO cite) which allows lecturers to embed quiz questions into their slideshow presentations. During a lecture, students can join a session through an online portal and have the slides and questions streamed through to their devices. Quiz questions can then be answered by the students and their results can be displayed live by the lecturer, who can also save these results for later analysis.

1.1.2 Replacement

A replacement system was wanted to fix some of the problems that Qwizdom has. Primarily that the University only has one session key, which means only one session can be used by all lecturers at any one time, leading to some clashes. With a new system, there would be no limit to the number of sessions that can be run. Other problems with Qwizdom include:

- Maximum of 6 answers (TODO: was it six? ask Chris)
- Students can submit their own answers by changing the HTML on the page
- Creating quizzes is tied into Microsoft PowerPoint which not all lecturers use
- Aging interface on the student end

1.2 Analysis

1.2.1 Two parts

It was decided that the project would be split into two main parts, the first was to create an application that lecturers can run a simple quiz from without any slides. The second part involves

developing an application or extension to the first part that allows the lecturer to create the quiz as part of a set of slides. The slide the lecturer is viewing would then be displayed in the quiz session on the web where students can access it. Once a relevant quiz slide appears, the students will be given the quiz options to select an answer in the same way as the first part. The lecturer can then display the results in the same way, though it would not be via the web application as before. This part of the project behaves in much the same was as Qwizdom. The reason for having two parts is that the first would be mostly built as part of the development for the second part but if built as a standalone part it allows lecturers to create quizzes rather than a set of slides with quizzes within it. Features like the session joining, front-end view for students and the way answers are submitted would be used in the second part, which means only creating quizzes would need to be added for the first part for it to be stand alone. Additionally, the second part had the potential to be much larger in scope than originally anticipated and as such having the first part to extend in a different direction should the second part become unviable would be provide a safety net.

1.2.2 Part 1

A web based approach was chosen for the first part. This application would allow lecturers to log in to an admin panel and from there create and run quizzes for the students. This would also lay the groundwork for the second part, setting up the front end for students, how sessions are run and how answering questions worked.

1.2.2.1 Framework

The PHP language was selected due to two primary reasons. The first is that the developer was familiar with PHP after using it in an industrial job. The second is that a large number of University projects are PHP based which will make integrating this with the University easier in the future. Whilst Ruby or Javascript might able be applicable, the familiarity and ability to extend PHP made it the best choice.

1.2.2.2 Laravel

Laravel is a web framework written in PHP(TODO: cite laravel), and has been chosen for the web server part of this project. There are a number of reasons for choosing Laravel in this project over other available frameworks. The first and most important is familiarity with Laravel as the developer has used it in the past, and the main framework used in their year in industry was based on Laravel.

Laravel is also the most popular PHP framework available(todo cite this), which means there is an enormous amount of support available in the form of user forums, video guides and its own tag on Stack Overflow.

Laravel 5 also natively supports web sockets, a technology that was being considered for the first part of the project. Other features of Laravel is its conformance to PSR-2, a PHP coding standard, meaning it would be easier to follow good coding standards more easily.

1.2.3 Part 2

The second part required more extensive research to be done when it was reached in development. But the main suggestion was to create an Microsoft PowerPoint extension much like Qwizdom to create slides embedded with questions. Due to the size of such an undertaking, another potential solution was to create the web application in such a way that an extension could be worked on in the future, i.e. set up the system for future extensions. TODO: cite microsoft extension stuff

1.3 Process

The methodlolgy chosen for this project was an Extreme Programming based approach. One of the core advantages of using an XP based approach is the ability to adapt to change. The second part of this project, implementing a method to stream slides to students, was much more open to change than the first part where the scope was far more understood. If another process such as Feature Driven Development was used, it would work for the first part of the project as all the requirements and features are known but most likely would be a struggle to use in the second part. XP gives the flexibility needed to complete the second part of the project whilst also allowing the first part to be done with ease.

1.3.1 Practices

A number of practices were selected for the project that work in a single developer project:

- Test Driven Development - Writing tests before coding any of the application logic helps to enhance both the design and also mean tests should actually be written rather than left until the end and ”hacked” in.
- Coding Standards - Keeping to strict coding standards helps ensure the code is good quality and easy to extend in the future as this project may well be.
- Small Releases - Small releases help enforce releases bit of working code regularly and results in a better overall project if the sub parts are all working.
- On Site Customer - This practice is somewhat applicable, the developer can act as a customer, and the project supervisor can also somewhat act as a customer, due to them being the originator of the idea and also a lecturer, one of the main users of this application.
- Merciless Refactoring - Refactoring is already encouraged by using TDD, but it can also be used at other points in the project to ensure the code is structured sensibly.
- Planning Game - This can be adapated to be done by one person at the start of the project, the list of stories was written and then iterations and releases planned out.
- Continuous Integration - Some online tools can be used to provide an CI workflow, to run tests continually with the constant small releases.

1.3.2 Stories - functional requirements

Before any development could start, a list of functional requirements was needed. For an XP based project, these would be in the form of ”stories”. The stories have a difficulty ranking associated with each item in relation to the other stories. 1 would the easiest and 10 the hardest. Stories for the first part:

- (6) Admins can create quizzes via the website
- (4) Quizzes contain a variety of questions
- (7) Admins can login to the website and run a session with a quiz
- (8) Multiple different sessions can be run simultaneously
- (3) The Admin specifies which question is being run by clicking next/prev question etc
- (5) Sessions can be joined by users via the website
- (3) Up to 300 users should be able to join and answer questions
- (1) Users answer the question being displayed by the quiz
- (2) The Admin can see what percentage of users connected to the session have answered
- (4) The Admin can show the results of the question in a sensible format e.g. graph
- (2) Admin can then save results as CSV or XML
- (2) The Admin can load from saved file to display again
- (4) Users should not be able to submit their own answers by altering the HTML, as they did with Qwizom

These ones fit the second part, though there are some stories from the first part that are applicable within this part.

- (1) The Admin creates slides in a slideshow editor (Most likely Microsoft Office Power Point)
- (2) The Admin can place question slides within the slides during creation
- (10) The Admin can stream these slides to a session
- (5) Users can join the session and follow the slides as they are used
- (4) The specified question slides will act as questions for the users connected to the session
- (2) Results are handled in the same way as the first web only part
- (3) The Admin can embed HTML content in quiz slides during creation

1.4 Planning Game

A plan was created early in development:

- 10 iterations beginning on Thursdays remaining from when the original plan was made (22/02/2017)
- Leave 2 iterations for Report Writing and emergency bug fixing at the end, from iterations beginning 20/04/2017
- 8 iterations between planning and iteration 8, these to be devoted to majority of coding
- 3 iterations before mid-project demonstration from initial planning

It was decided that it would be beneficial to try and get the majority of coding done within the 8 iterations specified, this would give the final two iterations breathing room to put together the final document more formally and give some bug fixing time and cleaning up time. In terms of releases, the aim was to have sets of features that would be releasable at the end of every iteration, though no planning of which features being released when was made.

Chapter 2

Design

2.1 Overall Architecture

2.2 Database Design

2.3 System Interaction Diagram

This could possible be part of the overall architecture section

Chapter 3

Iterations

3.1 Iteration 0 16/02 - 22/02

3.2 Iteration 1 23/02 - 01/03

3.2.1 Story: Admins can create quizzes via the website

3.2.1.1 Analysis - Breakdown of Tasks

This story is quite large and should be broken down into several substories:

- (3) Admins can log into a backend
- (2) They are presented a list of their quizzes
- (5) They can create a new quiz in the backend
- (3) They can edit an existing quiz they own

These have been added to the story list document.

3.2.2 Story: Admins can log into the backend

3.2.2.1 Analysis - Breakdown of Tasks

- Create users table
- Add login page
- Add register page

3.2.2.2 Design

Design was limited due to the automated builder. However it added some view files and a default HomeController for a basic homepage.

3.2.2.3 Implementation

Implementing this was far easier than originally anticipated, Laravel comes with the needed tables out of the box and has a command to run that sets up simple auth for users: *php artisan make:auth*

3.2.2.4 Testing

Because the login and auth is handled by Laravel by default, testing it was deemed to not be a priority. However, three simple tests were written to ensure it never breaks due to future changes. Dusk Tests:

1. Test to ensure the application redirects to /login is the user is not already logged in
2. Test for logging in with a user in the database
3. Test for registering a new user

3.2.3 Story: They are presented a list of their quizzes

3.2.3.1 Analysis - Breakdown of Tasks

- Make the homepage the QuizController rather than the HomeController
- Check and get the user who is logged in
- Display a list of quizzes for that user

3.2.3.2 Design

The HomeController was removed in this period of work and the QuizController used in its place. (TODO: Add UI mock maybe)

3.2.3.3 Implementation

Quite an easy amount of work, changing the controller was a simple change to the routes and then updating the quiz view file to use the same layout as the original home views. To get the user, a helper function is provided: *auth()->user()* which gets the user object. Obtaining the id from this is simple and then using an Eloquent ORM call it is easy to find all the quizzes owned by that user.

3.2.3.4 Testing

Dusk tests for this story:

1. Test to see if the /home page lists the quizzes as it would on the /quizzes page
2. Test to see if a quiz that belongs to a user is present on the page
3. Test to see if a quiz that belongs to another user is not present on your page whilst your own is

These tests highlighted a problem with the testing framework however. Chrome is used as the Remote Web Driver for running these application tests in. For each test a new migration is made within the test database, thereby wiping the data created within each test. An unintended side effect however is that every new user created starts at id=1 in the users table (a user has to be created for all these tests.) The Chrome driver seems to remember that the user of id=1 logged in, in the previous test and therefore skips the auth step. This means the test order is messed up due to the test trying to log in even though it is already logged in. The solution to this is to create each new user in the next id record, whilst this is somewhat convoluted, it seems to work.

Potential future tests: Use sessions have two users log in and see/ not see the relevant quizzes.

3.2.4 Non-Story Work

3.2.4.1 Database Work

Some initial work that is needed for almost all the stories is having a working database set up to store the users, quizzes, questions etc. Seeing as the amount of work to setup all the tables and their relationships would not take long, it was deicded that this could be done all at once at the start of the iteration.

While creating these tables it was possible to create the controllers needed within the application at the same time using: *php artisan make:model *name* -mc*

3.2.4.2 Seed Data

Because of the amount of changes to the database that were being made, the tables were repeatedly wiped and seeding some data was needed. To do this some seeders were generated with *php artisan make: seed *name**. These were created under database/seeds/ and simply required creating new objects of the desired Model and adding the various fields as parameters of the objects. These objects are then saved to the database. These seeders can be run when a migration is called such that the data is replaced as soon as its lost.

3.2.4.3 Layout Changes

The initial *make:auth* command created a default home page with a menu bar and some basic styling. This styling was created with Bootstrap and looks quite nice so the basic styling has been kept. This layout was modified somewhat to add some menu options that persist across pages. This layout is then used by all the backend pages created by extending it.

3.3 Iteration 2 02/03 - 08/03

3.4 Iteration 3 09/03 - 15/03

3.5 Iteration 4 16/03 - 22/03

3.6 Iteration 5 23/03 - 29/04

3.7 Iteration 6 30/03 - 05/04

3.8 Iteration 7 06/04 - 12/04

3.9 Iteration 8 13/04 - 19/04

3.10 Iteration 9 20/04 - 26/04

3.11 Iteration 10 27/04 - 03/05

Chapter 4

Testing

4.1 Overview

At the start of the project, testing was to use two practices from Agile. These were Test Driven Development and Continuous Integration. Whilst these two practices both failed due to a mixture of reasons as described in the Iterations chapter, testing was still performed within each iteration. Rather than go over the tests made each week within the previous chapter, this section will summarise the tests. Overall there were x tests (TODO: fill in) that test the overall functionality of the system. Unit tests were dropped for a number of reasons in favour of application tests (TODO: why? take from iter docs).

4.2 Application testing

TODO: screenshot final output of tests, possible one running for appendices. Also could list all tests somewhere. To do application tests, a testing framework called Laravel Dusk (TODO cite https://mattstauffer.co/blog/introducing-laravel-dusk-new-in-laravel-5-4) was used, which is the default testing framework provided with Laravel 5.4. It runs the tests within a Chrome instance by running a standalone server which then calls the Chrome application installed on the machine that the tests are being run on. The advantages of running tests like this rather than say executing PHP ont he server and reading a virtual DOM is that this allows JavaScript on the page to be evaluated and executed. JavaScript is vital to many of the components of the application so if it was not usable within tests, many parts of the system would have to be ignored within the tests.

These tests test the application as whole, rather than individual bits of code. This allows the tests to be written with the original requirements, the stories, in mind and so makes these tests a good way to evaluate the application at the end of its development. Tests were organised by story, as these provided an area of the application that needed to be checked, which makes them more human readable for any future development.

Tests use a test database rather than the production one to ensure the integrity of data on the production system. This test database is left intentionally empty when, with each test performing a migration for the test. Within each test the database is populated with relevant data using a number of database factories. These factories allow the easy and quick creation of any data needed within

the test, though using a normal Model to input data would also still work. At the end of each test, the database is rolled back for the next test to run, leaving an empty db for a new migration and data.

Database transactions could have been used instead, which use a pre migrated database and then any data input during each test is deleted at the end of the test. Whilst the test db could be migrated before any tests were run to allow the use of transactions, this would mean than any future changes to the migration files would mean that the developer would have to remember to remigrate the test database as well. Doing a migration for every test might be somewhat resource intensive, but it allows the tests to be written and run completely standalone from each other with no need to tie into any earlier setup functions.

4.3 Security testing

One story specifically concerns security: Users should not be able to submit their own answers by altering the HTML, as they did with Qwizom. The tests for this however are not that easy to do within the confines of Dusk due to the nature of the framework. The framework is concerned with user facing application tests and has no ability to change the DOM of the page or send custom POST requests to the server. This means that this story is better tested in the User testing section.

Other aspects of security can be checked however, being a web application there are a number of well known attack types including SQL injection, cross site scripting and cross site request forgery attacks. These are tested in (TODO: add these)

4.4 User testing

TODO

4.5 Stress testing

Though there are a number of tools out there to do exactly that, there was no time to set up and any official stress testing. However, the user tests within the lecture helped a give a good idea of the stress the system could take. Like many sites though, the application was mostly restricted by its server setup rather than by its design. (TODO: prove after user testing)

4.6 Automated testing

TODO: remove this section? Whilst CI was abandoned early on into the development of the application, the Dusk tests are easy to run and automatically execute within a browser if the appropriate browser drivers are available. This means that it should not be too hard to integrate these tests into a CI tool if required in any future development.

Stuff to talk about: comprehensiveness of testing incl bdd and cucumber etc edge cases? how did we manage those (user and application)

Chapter 5

Evaluation

5.1 Methodology

Got rid a couple of practices Liked adaptability Liked stories Like iterations Overall happy with that

5.2 Tools and Technologies

Laravel, Dusk, Websockets, Javascripty stuff, Chrome? Vim? Better tools?

5.3 Extra Tools

Diary and Trello

5.4 Story Comparison

5.5 User Testing Evaluation

5.6 Summary

Appendices

The appendices are for additional content that is useful to support the discussion in the report. It is material that is not necessarily needed in the body of the report, but its inclusion in the appendices makes it easy to access.

For example, if you have developed a Design Specification document as part of a plan-driven approach for the project, then it would be appropriate to include that document as an appendix. In the body of your report you would highlight the most interesting aspects of the design, referring your reader to the full specification for further detail.

If you have taken an agile approach to developing the project, then you may be less likely to have developed a full requirements specification. Perhaps you use stories to keep track of the functionality and the 'future conversations'. It might not be relevant to include all of those in the body of your report. Instead, you might include those in an appendix.

There is a balance to be struck between what is relevant to include in the body of your report and whether additional supporting evidence is appropriate in the appendices. Speak to your supervisor or the module coordinator if you have questions about this.

Appendix A

Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. If third party code or libraries are used, your work will build on that to produce notable new work. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

As an example, you might include a definition such as:

Apache POI library - The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client's existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [**?**]. The library is released using the Apache License [**?**]. This library was used without modification.

Appendix B

Ethics Submission

This appendix includes a copy of the ethics submission for the project. After you have completed your Ethics submission, you will receive a PDF with a summary of the comments. That document should be embedded in this report, either as images, an embedded PDF or as copied text. The content should also include the Ethics Application Number that you receive.

Appendix C

Code Examples

For some projects, it might be relevant to include some code extracts in an appendix. You are not expected to put all of your code here - the correct place for all of your code is in the technical submission that is made in addition to the Final Report. However, if there are some notable aspects of the code that you discuss, including that in an appendix might be useful to make it easier for your readers to access.

As a general guide, if you are discussing short extracts of code then you are advised to include such code in the body of the report. If there is a longer extract that is relevant, then you might include it as shown in the following section.

Only include code in the appendix if that code is discussed and referred to in the body of the report.

3.1 Random Number Generator

The Bayes Durham Shuffle ensures that the psuedo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs [**?**].

```
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0 - EPS)
```

```
double ran2(long *idum)
{
  /*-------------------------------------------------*/
  /* Minimum Standard Random Number Generator        */
  /* Taken from Numerical recipies in C              */
  /* Based on Park and Miller with Bays Durham Shuffle */
  /* Coupled Schrage methods for extra periodicity   */
  /* Always call with negative number to initialise  */
  /*-------------------------------------------------*/

  int j;
  long k;
  static long idum2=123456789;
  static long iy=0;
  static long iv[NTAB];
  double temp;

  if (*idum <=0)
  {
    if (-(*idum) < 1)
    {
      *idum = 1;
    }else
    {
      *idum = -(*idum);
    }
    idum2=(*idum);
    for (j=NTAB+7;j>=0;j--)
    {
      k = (*idum)/IQ1;
      *idum = IA1 *(*idum-k*IQ1) - IR1*k;
      if (*idum < 0)
      {
        *idum += IM1;
      }
      if (j < NTAB)
      {
        iv[j] = *idum;
      }
    }
    iy = iv[0];
  }
  k = (*idum)/IQ1;
  *idum = IA1*(*idum-k*IQ1) - IR1*k;
  if (*idum < 0)
  {
    *idum += IM1;
  }
```

```
  k = (idum2)/IQ2;
  idum2 = IA2*(idum2-k*IQ2) - IR2*k;
  if (idum2 < 0)
  {
    idum2 += IM2;
  }
  j = iy/NDIV;
  iy=iv[j] - idum2;
  iv[j] = *idum;
  if (iy < 1)
  {
    iy += IMM1;
  }
  if ((temp=AM*iy) > RNMX)
  {
    return RNMX;
  }else
  {
    return temp;
  }
}
```

Annotated Bibliography

[1] Apache Software Foundation, "Apache POI - the Java API for Microsoft Documents," http://poi.apache.org, 2014.

This is my annotation. I should add in a description here.

[2] ——, "Apache License, Version 2.0," http://www.apache.org/licenses/LICENSE-2.0, 2004.

This is my annotation. I should add in a description here.

[3] W. Press *et al.*, *Numerical recipes in C*. Cambridge University Press Cambridge, 1992, pp. 349–361.

This is my annotation. I can add in comments that are in **bold** and *italics and then other content.*

[4] M. Neal, J. Feyereisl, R. Rascunà, and X. Wang, "Don't touch me, I'm fine: Robot autonomy using an artificial innate immune system," in *Proceedings of the 5th International Conference on Artificial Immune Systems*. Springer, 2006, pp. 349–361.

This paper...

[5] H. M. Dee and D. C. Hogg, "Navigational strategies in behaviour modelling," *Artificial Intelligence*, vol. 173(2), pp. 329–342, 2009.

This is my annotation. I should add in a description here.

[6] Various, "Fail blog," http://www.failblog.org/, Aug. 2011, accessed August 2011.

This is my annotation. I should add in a description here.

[7] S. Duckworth, "A picture of a kitten at Hellifield Peel," http://www.geograph.org.uk/photo/640959, 2007, copyright Sylvia Duckworth and licensed for reuse under a Creative Commons Attribution-Share Alike 2.0 Generic Licence. Accessed August 2011.

This is my annotation. I should add in a description here.