EXTENDING AWESOME

Final Report for CS39440 Major Project

Author:
Benjamin Brooks
beb12@aber.ac.uk

Supervisor:
Dr. Hannah Dee
hmd1@aber.ac.uk

Friday 8th May, 2015

This report was submitted as partial fulfilment
of a BSc degree in Computer Science
(inc Integrated Industrial and Professional Training) **[G401]**

Prifysgol Aberystwyth University

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

Declaration of Originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the sections on unfair practice in the Students' Examinations Hand- book and the relevant sections of the current Student Handbook of the Department of Computer Science.
- I understand and agree to abide by the University's regulations governing these issues.

Signature _____ Date _____

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature _____ Date _____

Ethics Form Application Number

The Ethics Form Application Number for this project is: **1019**.

Student Number

Many thanks to my supervisor Hannah Dee for the guidance and support throughout this project; Sandy for providing a server, and support when it went wrong; and the rest of the staff at Aberystwyth University for the countless hours they put into making it an outstanding place to study.

Thanks to Gareth Williams for providing better Welsh translations than Google Translate ever could, and thanks Keiron O'Shea for the AWESOME prototype upon which this project is based.

Thank you to my fellow classmates, flatmates, friends, and family for the continued motivation, ideas, and encouragement; not only over the past few months, but also the past four years through my university course.

But most of all thank you to Bethany Foskett, for being there with sugary snacks, caffeinated beverages, and occasionally being my rubber duck [1].

Abstract

The Aberystwyth Web Evaluation Of Module Experiences (AWESOME) is an open-source, web-based tool that enables departments to generate personalised questionnaires that are sent to students to gather feedback about modules. The aim of this project is to rewrite the AWESOME prototype and have it replace existing methods of module evaluation in the Computer Science department at Aberystwyth University (AU) by improving upon, and fixing problems with the existing module evaluation procedures.

Contents

List of Tables

List of Figures

List of Acronyms

ASTRA	Aberystwyth Student Records and Admissions
AU	Aberystwyth University
AWESOME	The Aberystwyth Web Evaluation Of Module Experiences
CI	Continous Integration
CSV	Comma-Separated Values
i18n	Internationalisation
IS	Information Services
MEQ	Module Evaluation Questionnaires
MVC	Model-view-controller
OOP	Object-oriented programming
PDO	PHP Data Objects
RDBMS	Relational Database Management System
SME	Student Module Evaluation
SQL	Structured Query Language
SSCC	Staff Student Consultative Committees
UML	Unified Modeling Language
VPN	Virtual Private Network

1. Background & Objectives

1.1. Introduction

The Aberystwyth Web Evaluation Of Module Experiences (AWESOME) is a web-based tool that enables departments to generate personalised questionnaires that get sent out to students to gather feedback about modules, lecturers, or even events such as BCS Show & Tell[1] and other talks.

By tailoring questionnaires to individual students, only asking relevant questions, and being able to target students who have yet to complete a survey to send out reminder emails, we can increase the response rate and quality of answers when gathering student module feedback. AWESOME aims to solve the problems with existing module evaluation methods, and to improve student response rates to gather accurate module evaluation data.

1.2. Background

Departments at Aberystwyth currently have no formalised process of gathering student feedback unlike many other universities. The University of Sussex requires courses to be evaluated through their Module Evaluation Questionnaires (MEQ) [2] which contains seven core quantitative questions and up to ten additional questions at the school-level, module-level or a mixture of both. The University of Westminster have Student Module Evaluation (SME) [3], which is an online questionnaire containing ten questions per module sent via e-mail about modules. There are many other similar examples of module evaluation systems across UK universities, with their main features being personalised, anonymised, and incentivising completion.

At Aberystwyth, some departments such as Geography & Earth Science, and English choose to hand out paper-based questionnaires in lectures for each module a student does. Having a student take multiple questionnaires to provide essential feedback can lead to many issues due to survey fatigue. This can significantly reduce response rates and also have an impact on the answers students provide.

[1] BCS Mid Wales Show & Tell: http://midwales.bcs.org/show-and-tell-events

The Computer Science department has tried many methods of collecting module feedback with varying successes which will be elaborated upon in subsection 1.2.1.

AWESOME was originally proposed and developed under the Learning and Teaching Enhancement Fund by Dr. Hannah Dee, and work on the prototype was undertaken by Keiron O'Shea. The project was selected as my dissertation project to extend and implement.

Several meetings by the Learning and Teaching Enhancement Committee and Pro-Vice-Chancellor Professor J. Grattan took place to discuss the future of AWESOME and the need for a centralised module evaluation system used university-wide, and talks are still ongoing to investigate this.

1.2.1. Module Evaluation Method Analysis

There are several important factors to consider when collecting module evaluation feedback. First and foremost, responses must be anonymised, secondly being able to provide incentives for completion drastically increases response rates. Being able to have students complete the survey in their own time, and also send targeted reminders to complete the survey are also important. If reminders are constantly being sent to students who have already completed the survey, they are likely to get frustrated or annoyed and are less likely to notice another survey e-mail.

Consolidated surveys help with fatigue, as students only have to answer one survey. There have been several studies on how survey fatigue affects response rates and poor answer quality [4]. This consolidation only works if the surveys are personalised, as seen from Google Forms response rates in Table 1.1, students are lazy, and they will not skip over modules if asked to and will significantly drop response rates.

Table 1.1 shows a feature comparison of current methods of module evaluation at Aberystwyth University and corresponding response rates. AWESOME aims to solve the problems with existing module evaluation methods and to improve student response rates to gather accurate module evaluation data.

Paper based module feedback during lecture time can have effective response rates because lecture-time is set aside to complete questionnaires. However students usually have to complete one questionnaire per module which can lead to fatigue very quickly.

Qwizdom[2] is a hardware-based voting system, with 'clickers' handed to students in a lecture who can then cast their votes through a powerpoint style questionnaire. Response rates for Qwizdom module evaluations are high for the same reasons as paper based forms. Students are stuck in a lecture for an hour with nothing else to do. One problem Qwizdom does give, is that answers can only be quantitative and not qualitative. Students can't easily input textual comments through Qwizdom so some of the most valuable information is not gathered from students.

Google Forms has been the standard way of running module evaluation questionnaires for the past few years in CompSci. Google Forms provides anonymous answering in the student's own time and can also provide a way of gathering valuable textual comments. One disadvantage of using Google Forms is that there is no way of knowing which modules a student is enrolled for, so students have to skip over modules that aren't applicable. This makes the survey confusing and error prone at times and response rates suffer as a result.

AWESOME has been created from the ground-up to address all of these problems. Responses are anonymised, while retaining the ability to see who has, and has not completed the questionnaire yet. This allows for targeted reminders and incentives for completing the survey. Additionally, AWESOME imports data directly from Aberystwyth Student Records and Admissions (ASTRA) which allows all student, staff, and module data to be easily used without lots of manual data entry. This also allows for personalised surveys, asking questions only relevant to modules a particular student is enrolled for. By collecting both Quantitive and Qualitative data, AWESOME can run advanced analytics can be run on the data gathered. The questionnaires sent out are also fully responsive, working on phones to tablets, and to desktop computers by utilising Bootstrap and can be completed at any time by the student.

[2]Qwizdom Homepage: http://qwizdom.com/higher-education/home
[2]Bootstrap Homepage: http://getbootstrap.com

Method	Tailored Questions	Anonymous	Qualitative	Quantitative	Incentives for completion	Completion on own time	Targeted reminders	Responsive	Consolidated	Response Rate
Paper	✗	†	✓	‡	✓	✗	-	-	✗	75%[5]
Qwizdom	✗	✓	✗	✓	✓	✗	-	✓	✗	50%[5]
Google Forms	✗	✓	✓	✓	✗	✓	✗	✓	✓	20%
AWESOME	✓	✓	✓	✓	✓	✓	✓	✓	✓	TBC

† Anonymity may be compromised when completing paper-based form.
‡ Manual processing is required in order to analyse the data.

Table 1.1.: Module Evaluation methods at Aberystwyth University

1.3. Objectives

The overall objective of the project is to implement AWESOME on the AU network and collect module evaluation feedback for the Computer Science department in both short mid-term, pre-Staff Student Consultative Committees (SSCC) questionnaires and full end-of-semester questionnaires.

This can be broken down into four main aims to tackle:

- **Security Audit the AWESOME prototype.** This was a large issue, as there were known security flaws with the prototype and it needed to be looked at immediately before any other work took place.
- **Bring the prototype up to modern development standards.** The program was known to be written in a procedural style, and the security audit brought up the poor i18n implementation too.
- **Finish any incomplete functionality.** Many areas of the prototype were half-implemented, but not fully completed. These had to be done before it was useable by students and staff.
- **Run AWESOME on a departmental server.** The main objective was to get a survey sent out to students and collect real-world data. This is the final step in that process.

2. Requirements

2.1. Features

The following feature list is the one proposed for the AWESOME prototype. The project follows this feature set as a baseline set of requirements.

- **Automatic questionnaire generation per-student** - Generate unique questionnaires per-student, depending on their modules and lecturers.
- **The ability to generate quick mid-term questionnaires** - A one question per module survey with a one to five scale from 'This module is going well' to 'This module has problems'.
- **No need to type in registration details** - Import of module registration data via ASTRA Comma-Separated Values (CSV) export.
- **Targeted follow-up reminder emails** - Only send reminder emails to respondents who have yet to complete their questionnaire.
- **Anonymous responses** - Ability to know which particular student has, or has not completed the questionnaire, but not who has said what.
- **Visually appealing analytics** - Reports available to staff on a by-module, by-department, and by-scheme basis, with graphs and textual responses laid out nicely.

2.2. Development Practices

AWESOME should be developed using Object-oriented programming (OOP) practices if it is to be maintainable by other developers after the project is over. Internationalisation should be easily extensible and translation strings should be easy to add. Additionally, unit testing is vital if the application is going to be extended further and refactored at a later date for any reason.

2.3. User Roles

The user roles in AWESOME are quite simple, there is only an admin and a respondent and each one can only do a few tasks.

- **Admin** - The person who creates the surveys, adds questions, and sends them. They can also view results and see who has not yet completed the questionnaire.
- **Respondent** - The recipient of a questionnaire email who fills it in and submits answers. They can also submit feedback about AWESOME if debug mode is enabled.

2.3.1. Use Cases

Admin User Role

As seen in Figure 2.1, admins can view surveys, create surveys, add questions to surveys, view more detail about survey such as respondents, response rate, modules, and results. They can also send an email to respondents to remind them to fill in the questionnaire.

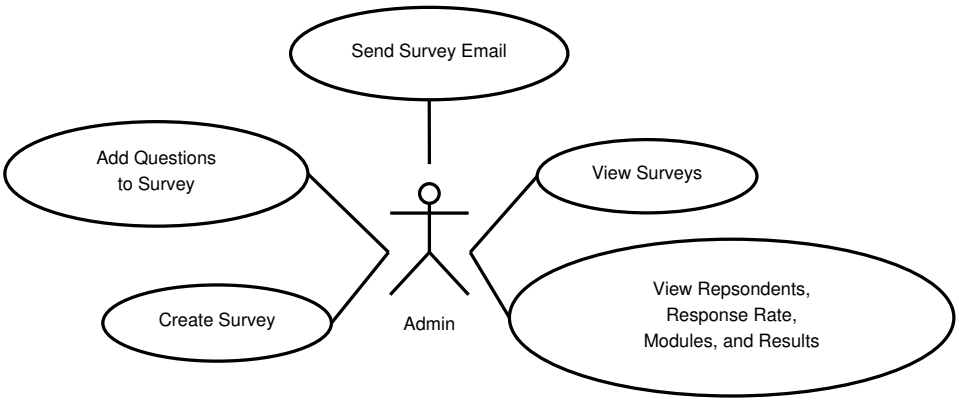


Figure 2.1.: Admin use-case diagram.

Respondent User Role

Figure 2.2 shows that a respondent only has one responsibility in the system, and that is to respond to a survey through a link sent via email. They can only respond once, and submitted answers cannot be seen as it would break the irreversible anonymity of the system. If debug mode is enabled, respondents can also fill in a feedback form about AWESOME which gets sent to developers.

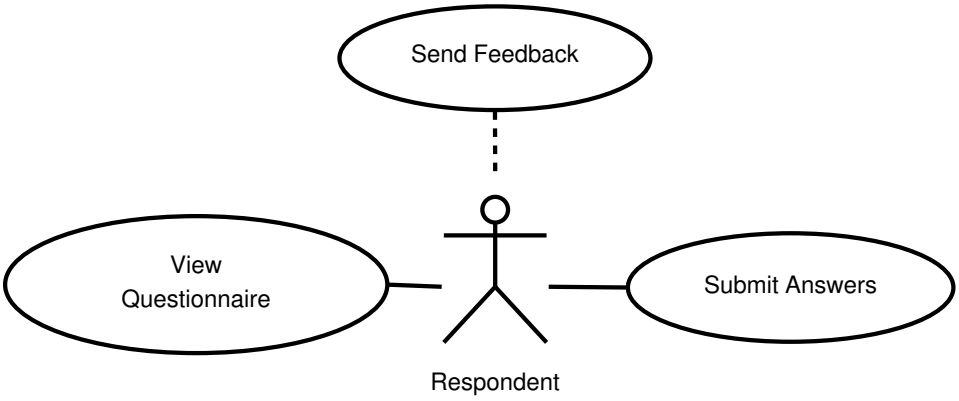


Figure 2.2.: Respondent use-case diagram.

3. Design

After the security audit, it was apparent that the majority of the codebase needed to be refactored in order to make it object-oriented. The decision to rewrite the program was not a light one, by rewriting the program, the scope of extensions to AWESOME became much more limited via time constraints.

With the rewrite came the opportunity to utilise both a framework and a design pattern. Frameworks were ruled out under the stance that it had to be deployed on a server without shell access. However later research discovered that this is achievable in Laravel[1], which would have been the framework of choice for several reasons.

Laravel would have provided a complete, and mature Model-view-controller (MVC) framework, as well as an i18n framework and a solid basis for AWESOME. Ultimately, poor research into Laravel on a shell-less server resulted in an attempt to produce a bespoke MVC framework, which, unsurprisingly is harder than it first seems.

3.1. Programming Language

The programming language choice was fairly fixed, as it had to be easily runnable on an AU server. This limited the choice to PHP, but which minor version of PHP was only found out later in the project which is discussed further in section 4.5.

If this requirement wasn't an issue, using Ruby with the Rails framework would have been a great candidate for this project, as rails already provides a mature MVC framework, many third-party gems for i18n, and many other language-inherent features, such as better type safety, and integrated testing tools.

Ruby on Rails can also perform background tasks, unlike PHP, which can be really useful when handing large volumes of mail, such as those sent out to students with unique questionnaire links.

[1] Laravel Homepage: http://laravel.com

3.2. Model-view-controller Framework

Since using a ready-made framework was ruled out, a custom-made MVC framework was written to support this project. The design of which is laid out in this section and is very heavily taken from the 'Write your own PHP MVC Framework' series of tutorials[6].

3.2.1. Routing

URL Routing is a way to access specific controllers and views from a URL. This works via an Apache rewrite rule, which appends a $_GET variable which contains the current requested URL.

The URL routing is handled as such: *awesome.url/controller/view[/query]*

By using routing in the URLs, we can eliminate the 'messy' looking get parameters in the URL, and have nice clean looking links.

3.2.2. Directory Structure

Figure 3.1 shows the directory structure of the MVC framework.

'*src*' is the source code to the program. Inside it contains everything needed to run AWESOME.

Inside '*tests*' are the unit tests to be run automatically by TravisCI and PHPUnit.

'*src/app*' has the MVC classes (controller, models and views) which are used when the routing engine rewrites URLs.

'*src/config*' contains the config file for the program. In this file, database credentials are set, as well as SMTP mail server settings, some i18n settings, and the debug flag which displays errors and shows a feedback form/notice (See Figure 3.11).

'*src/db*' is a directory for Structured Query Language (SQL) dumps for the database schema.

'*src/i18n*' contains translation files for the i18n framework, but more detail is in section 3.3

'*src/lib*' is the 'glue' that holds the MVC framework together. It contains the autoloader class, the bootstrap script, a PHP Data Objects (PDO) database class, and the third-party, open source PHPMailer[7] classes used for sending mail via SMTP.

'*src/logs*' is where any PHP errors get logged when debug mode is off. This is in order to hide any potential security issues with displaying errors.

'*src/public*' is the main entry point to the program and where the Apache vhost will be set.

'*src/public/assets*' contains the Javascript, SASS, CSS, and images used in the HTML. Users visiting the public directory with a valid token will be taken to their corresponding questionnaire.

'*src/public/admin*' is protected by HTTP authentication to prevent anybody from creating surveys and sending them out, or reading the results of previous surveys.

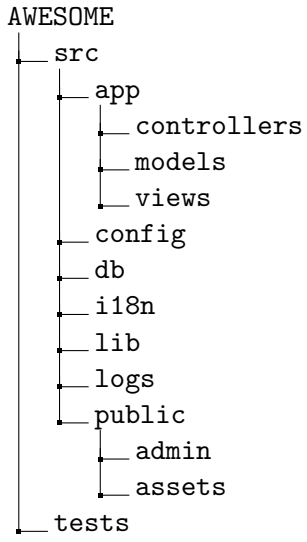


Figure 3.1.: Directory structure of the MVC Framework

3.3. Internationalisation Framework

The i18n framework is a tiny standalone OOP module which utilises JSON formatted files with strings for translation. JSON was chosen as it is easily human-readable, is easy to manually create, and not verbose enough to put off a non-programmer from modifying it. This means that even people not familiar with JSON can still easily provide translation files for AWESOME.

The JSON files are structured like the one in Figure 3.2. The first two entries are i18n metadata, for ISO639-1 code[2], and language. Each string has an ID, and a translation string to be returned. Every language uses the

[2]ISO639-1: http://en.wikipedia.org/wiki/ISO_639-1

same ID, and what gets returned depends on the language set via a global variable.

The i18n class has a global function, __($string_id) which will then return the appropriate translated string depending on the $lang global variable.

```
{
    "@ISO639-1" : "en",
    "@lang" : "English",
    "invalid-url" : "The URL provided is incorrect.",
    "send-responses" : "Send Responses"
}
```

Figure 3.2.: JSON Format for i18n language files.

3.4. Database Schema

The database schema (see Figure 3.3) used is very similar to the prototype version. Documentation needed to be written for the existing schema, so the opportunity was taken to make some changes in the structure of the database before this happened. The old database schema did not use any foreign key constraints, nor was the use of primary keys ideal.

Foreign keys are very useful to have in a system like this, as orphaned rows can be cleaned up nicely with a cascade delete. This prevents any data remaining in the table incase it is missed by an SQL query that hasn't been updated in the code.

Some changes were made to the way things were named, mostly to prevent confusion. In the new schema, a survey is a set of questionnaires, each of which is tailored to a specific student. In the old schema, everything was called a questionnaire, which led to confusion between a group of questionnaires and a single questionnaire.

Additional changes need to be made in order for AWESOME to support multiple departments across the university, but this is out of the scope of the dissertation, although this feature is on the roadmap if AWESOME is to be used university-wide.

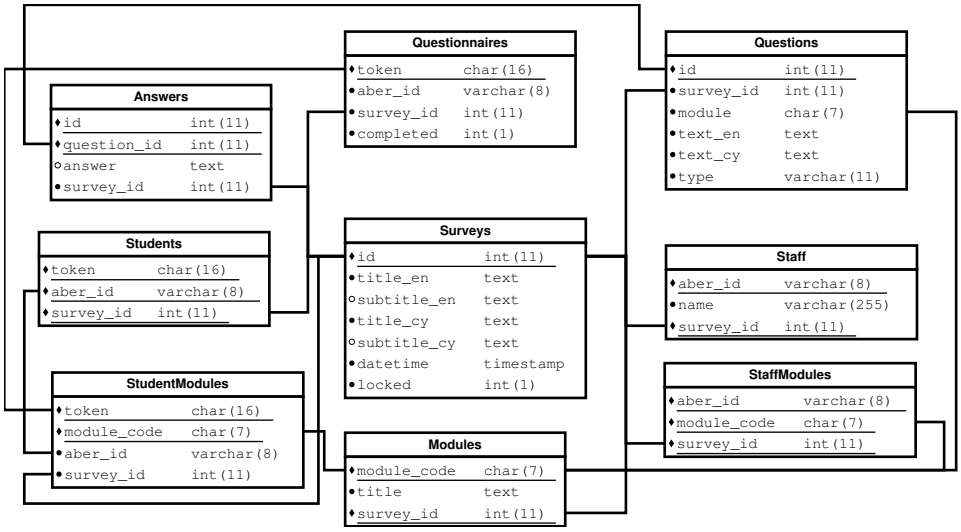


Figure 3.3.: AWESOME database schema design.

3.5. User Interface

The user interface was fairly polished on the respondent-facing sections in the prototype, so a lot of elements were re-used in that, with a few minor changes to visual aspects to improve readability, as can be seen in Figure 3.4.

The admin dashboard of the prototype wasn't very user friendly, creation of a survey was split over several pages which didn't need to be separated. This resulted in the creating of surveys, adding questions, modules, students, and staff being quite convoluted and not straight-forward. Results weren't clear to look at either, and took up a lot of space per-question by pie charts, which were unnecessary given the type of data provided.

Figure 3.5 shows the main dashboard screen, which contains a list of all surveys. In this list, the survey name, description, response rate, date and locked status are displayed. By displaying the response rates in here, you can easily see progress of the surveys, and quickly gauge the response rate of students for a particular survey.

Figure 3.6 shows the difference in results pages. Information density is a lot greater in the new version and it makes it much easier to see the results of Likert scale questions at a glance. Textual comments have a large presence for good reason in the results page, as often they provide the most valuable information about a module. If comments were hidden behind a scrollable frame, comments may be overlooked or missed.

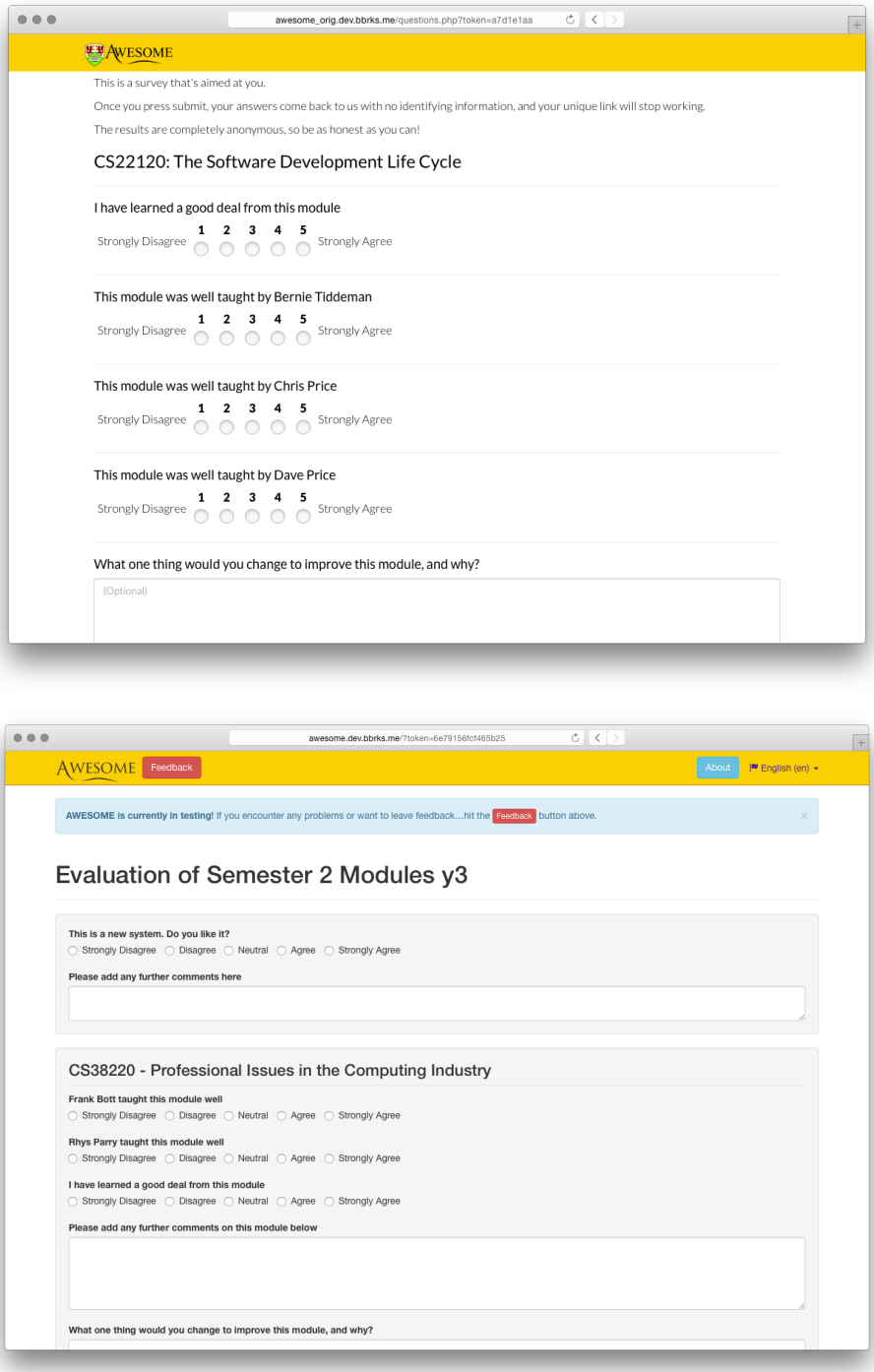
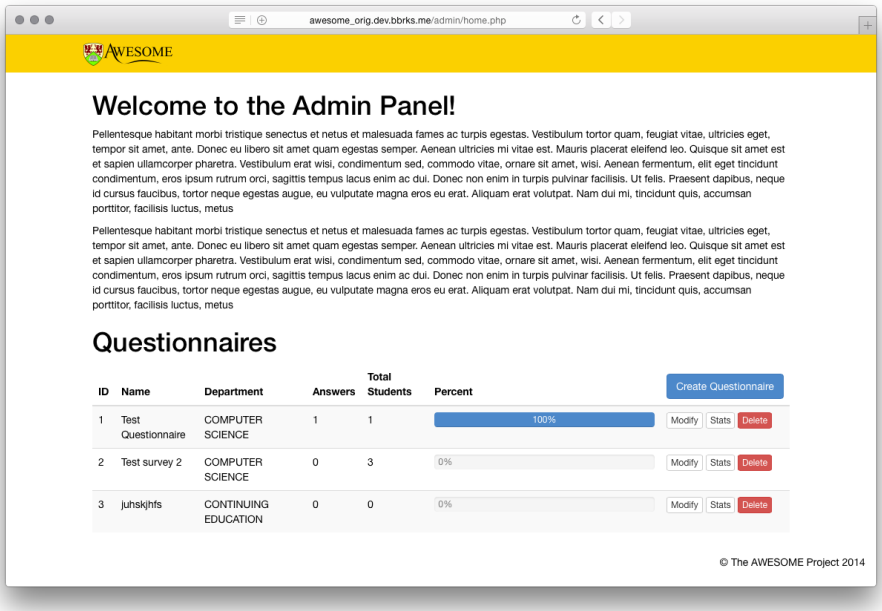


Figure 3.4.: A comparison of questionnaire pages between the prototype version and the submitted version of AWESOME



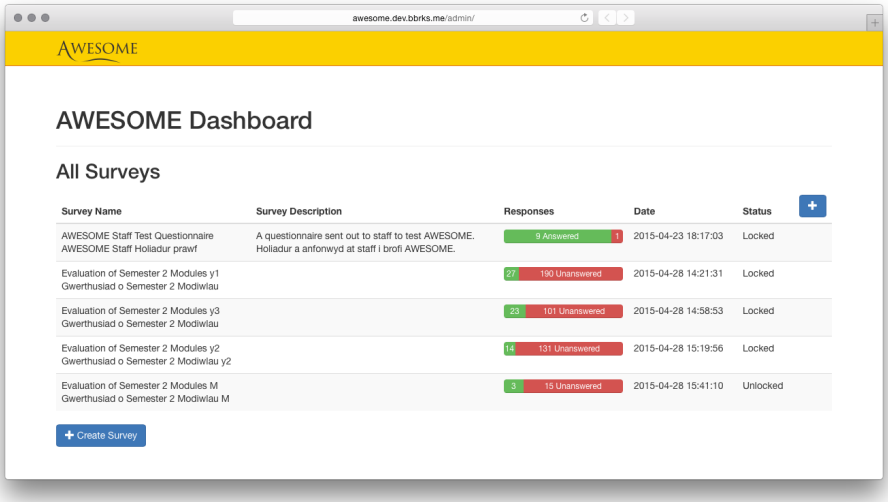


Figure 3.5.: A comparison of surveys view between the prototype version and the submitted version of AWESOME

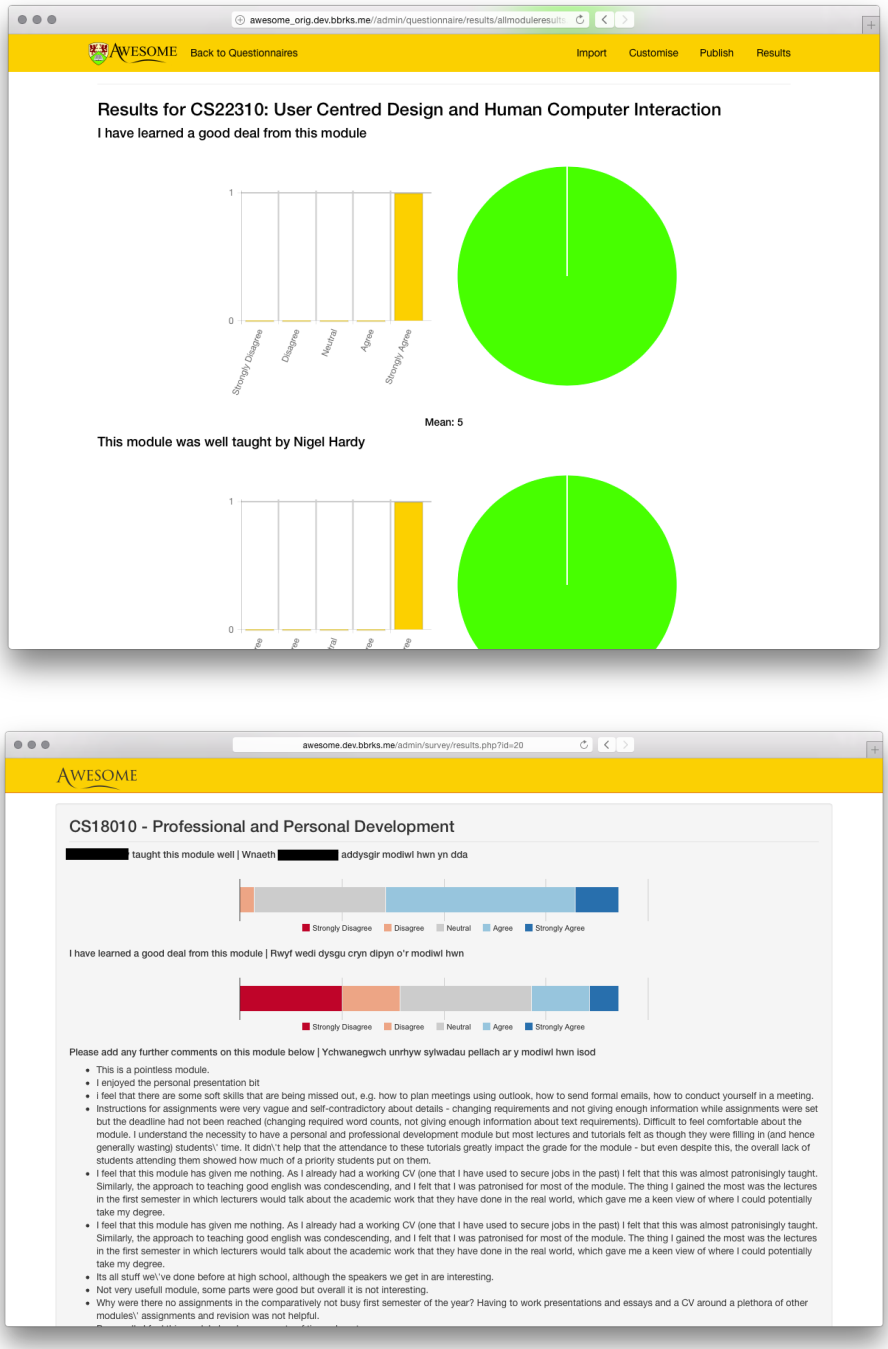


Figure 3.6.: A comparison of results between the prototype version and the submitted version of AWESOME

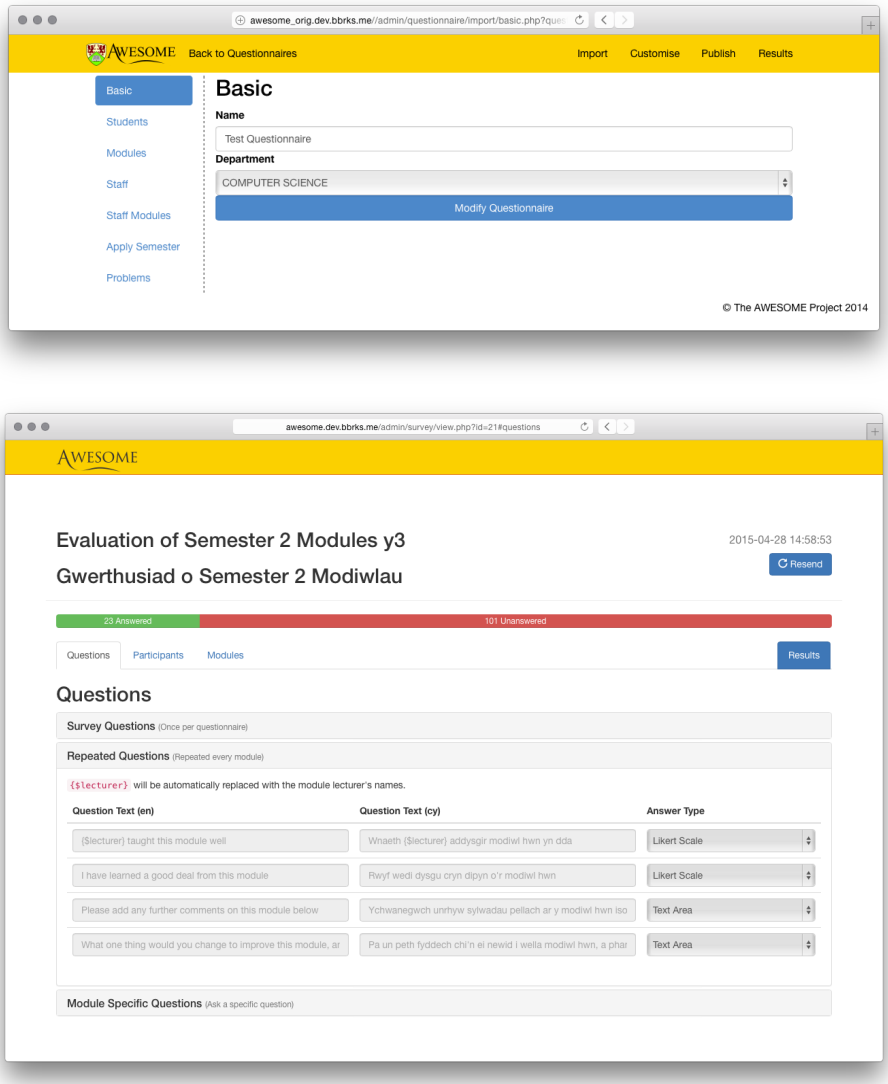


Figure 3.7.: A comparison of survey view between the prototype version and the submitted version of AWESOME

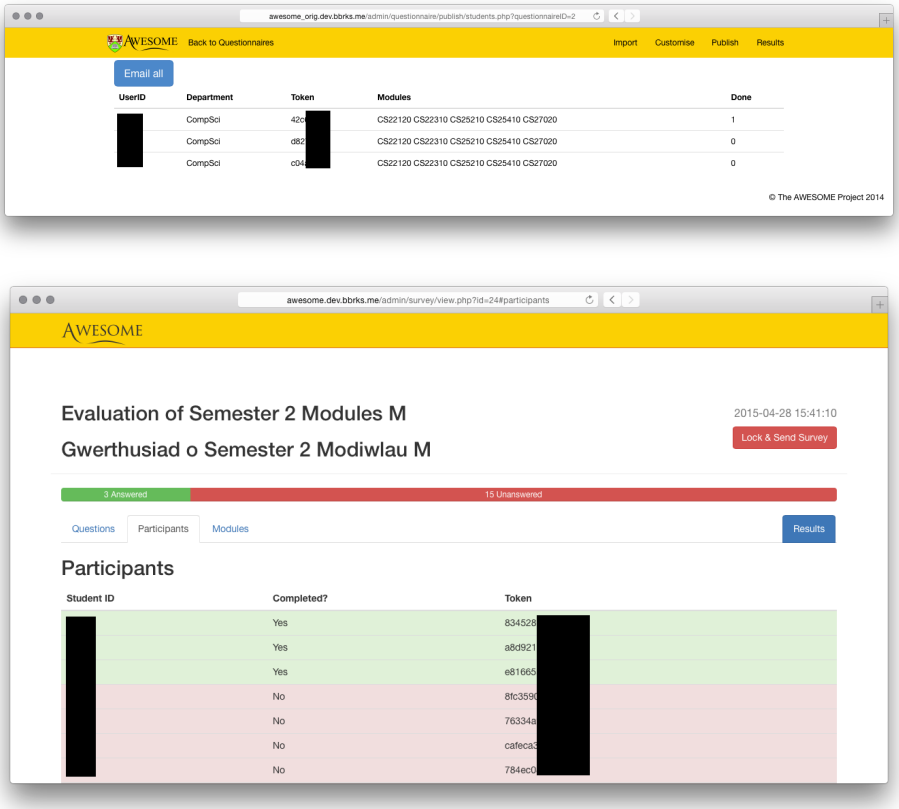


Figure 3.8.: A comparison of respondents between the prototype version and the submitted version of AWESOME

Figure 3.7 shows the great difference between the two survey page designs. The newer one has all of the information contained in four tabs in the page, as well as being able to easily see response rate again. Questions are in collapsible panes to easily separate the three different types of questions available and provide additional information if required.

Figure 3.8 shows the Respondents tab in the new version, compared to the old respondents page. As you can see, the colour coding of completed/incomplete makes a large difference in the availability of data at a quick glance, and also matches the response rate bar's colour scheme. This is the page that can be used to chase up students who fail to answer the survey as well as reward students for completing it. Targeted reminder emails can be sent from here to only those who have not yet completed their questionnaire.

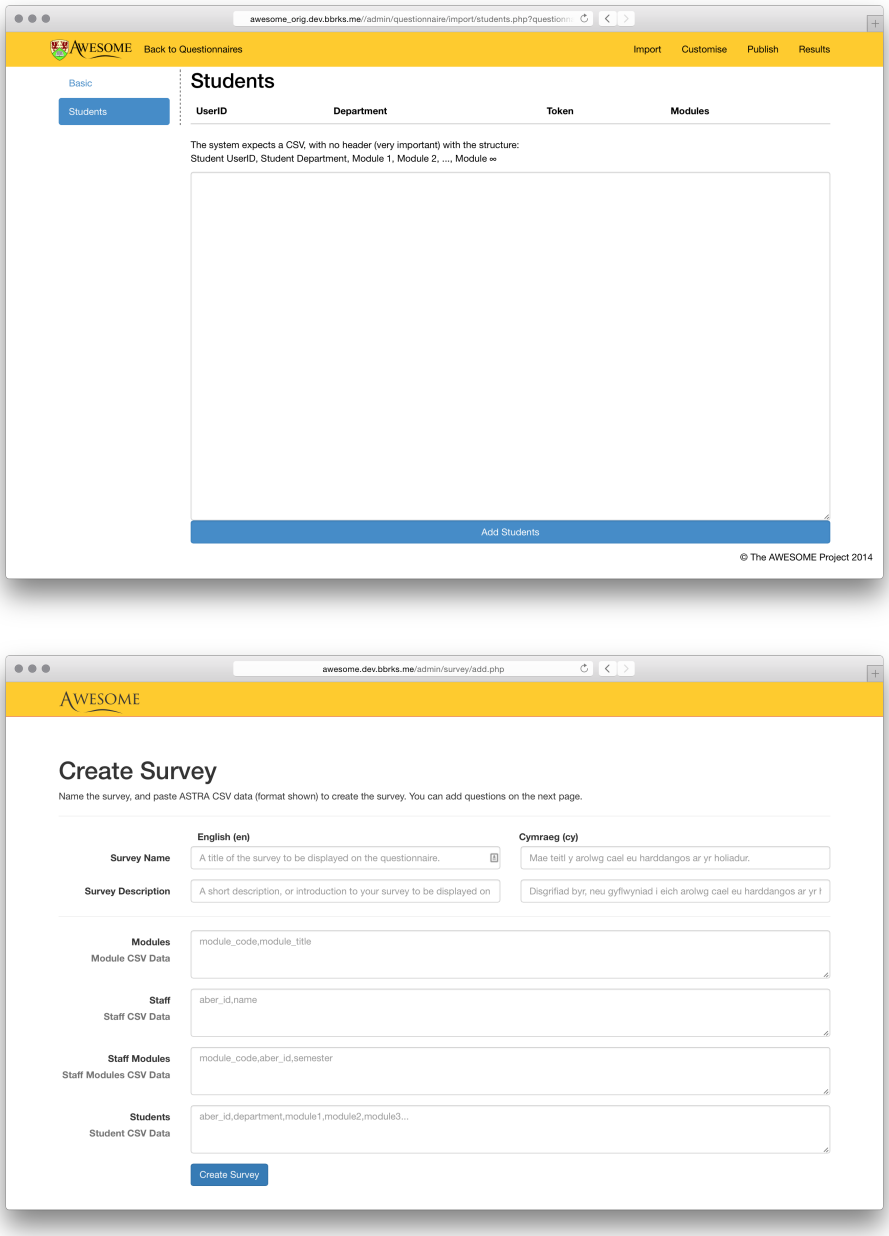


Figure 3.9.: A comparison of CSV input between the prototype version and the submitted version of AWESOME

Figure 3.9 shows the drastic difference between the prototype CSV import and the new one. The prototype version has one page per CSV file. This is a slow process to go through to get data imported, what would be much

nicer is if you could copy and paste directly into fields on the same page and submit them all at once. This is what the submitted version does, and it does it quite well too. It's very fast at getting data into the system.

Additional features include a feedback form (Figure 3.11) when AWESOME is in debug mode. This allows users to send an email to developers, containing any text they wish, as well as automatically including user-agent and other metadata to help identify the problem.

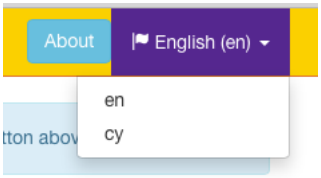


Figure 3.10.: A screenshot of the i18n selector in AWESOME

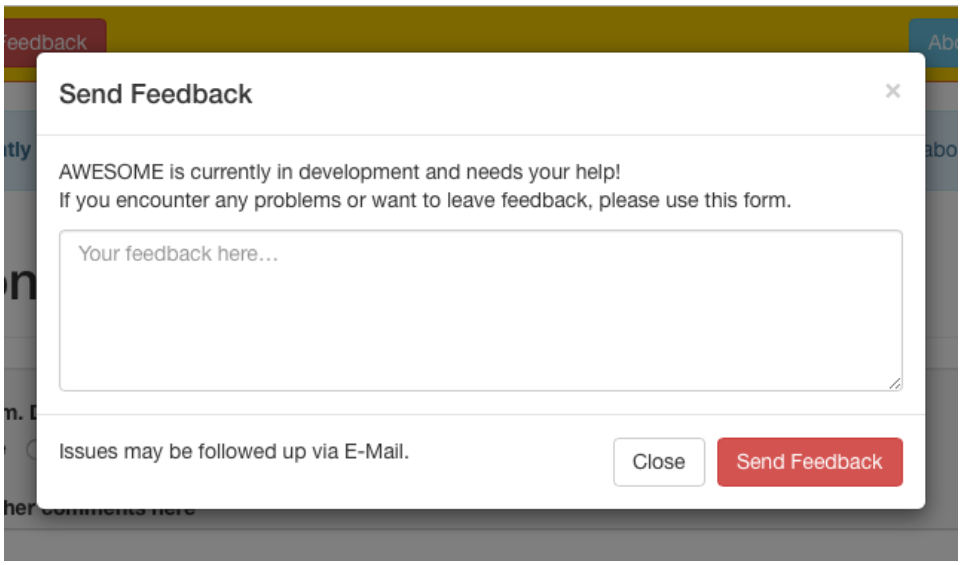


Figure 3.11.: A screenshot of the feedback form in AWESOME

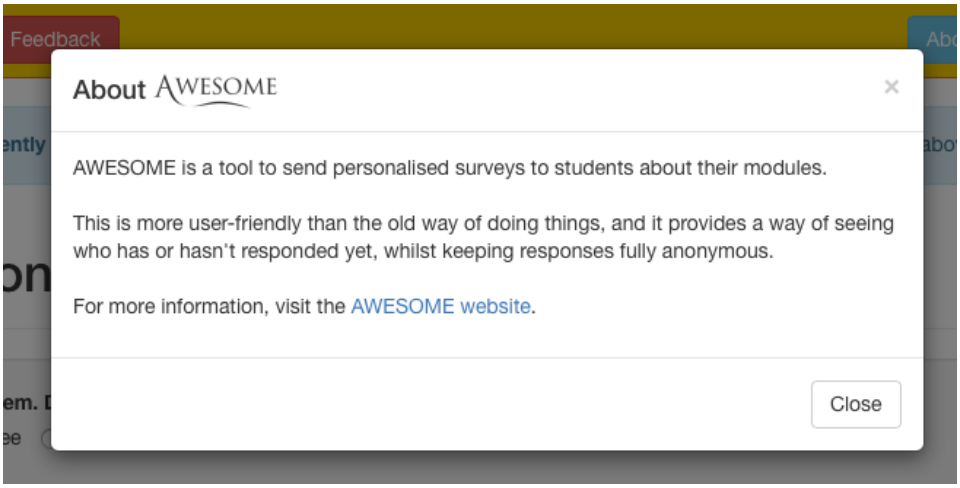


Figure 3.12.: A screenshot of the about dialog in AWESOME

4. Implementation

4.1. Prerequisites

4.1.1. Third-party services

AWESOME's development relies on the use of a few third-party services in order to make development a little easier. By setting up these at the very start, valuable time and effort can be saved further down the project's lifecycle.

TravisCI[1] was the first service I set up after receiving access to the GitHub repository. Travis allows for completely automated running of unit test suites whenever code is committed to Git. This is incredibly useful when working with modular libraries like the i18n framework. Travis also allows for testing across PHP versions, so I could simultaneously test PHP 5.3, 5.4, 5.5, 5.6 and hhvm (a custom PHP server).

Figure 4.1 shows when I made a change using a function that was new to PHP 5.6 that I wouldn't have noticed without Travis until a later date, as my development environment was running PHP 5.6.

Travis proved invaluable at times, although I could have made much better use of it by unit testing much more of AWESOME, especially the MVC framework.

Another third party service used was GitHub Pages. Pages allows you to publish a website from a GitHub repository branch, which is very ideal for this situation, as all code and host settings are tied to the AWESOME Git repository which will be easy to hand over to other developers.

The Pages website can be visited at http://bbrks.github.io/AWESOME. It is currently running Keiron's website which displays information about the prototype version of AWESOME.

[1] TravisCI Homepage: http://travis-ci.com

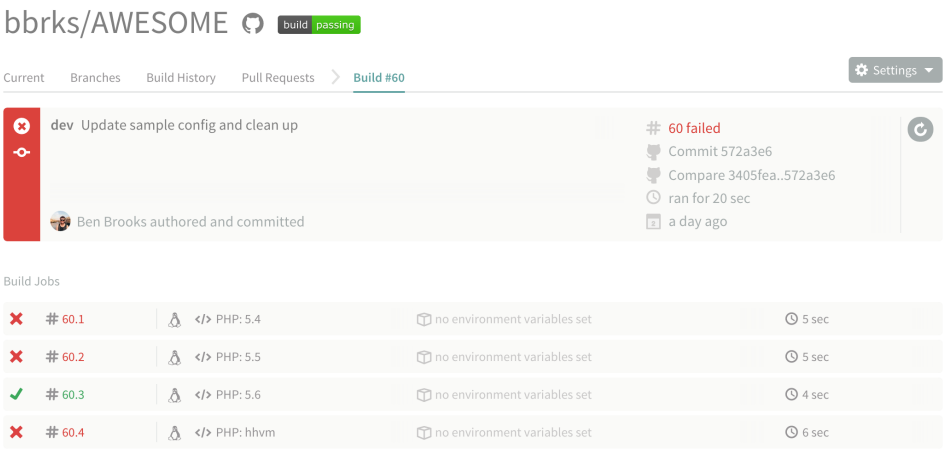


Figure 4.1.: A screenshot of a single build in TravisCI

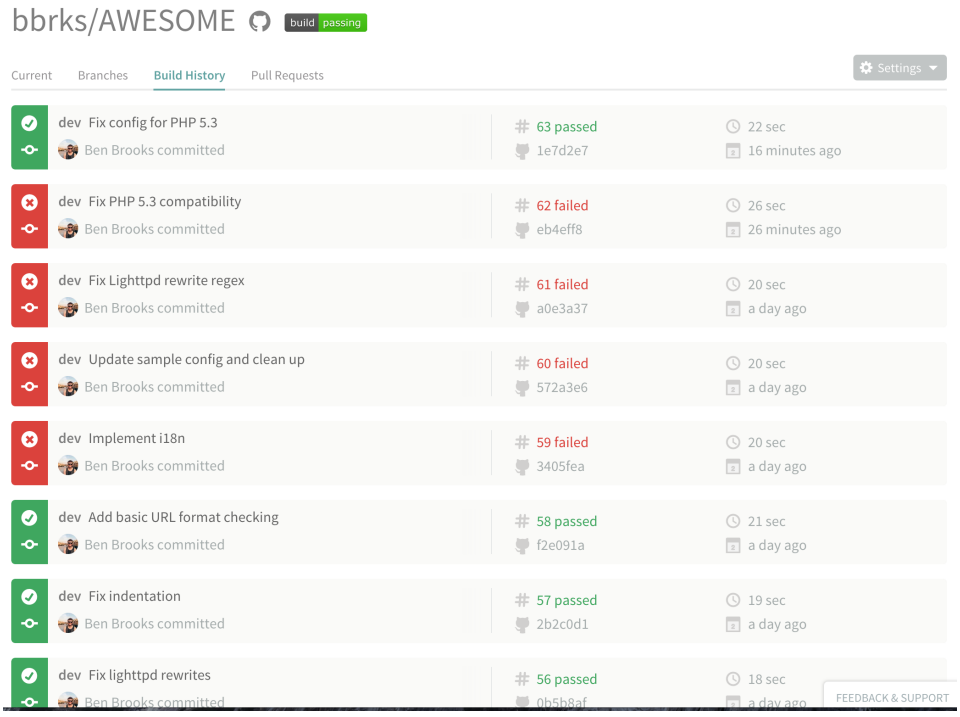


Figure 4.2.: A screenshot of build history in TravisCI

4.1.2. Open Source License

The prototype version of AWESOME was originally under the open-source MIT license[8] when branched and my work began. Some time after this point, the license on the original was changed to the GNU Affero General Public License, Version 3[9], however as I had branched the prototype version as it stood under the MIT license, the largely-rewritten version of AWESOME is also licensed under the MIT license and does not touch any code licensed under the GPL v3 Affero license and so I am not restricted to using a GPL-based license.

AWESOME uses an open source, third party library for handling mail via SMTP called PHPMailer[7] which is licensed under the LGPL v2 license. This license permits me to release AWESOME under the MIT license without any license impositions that GPL would.

Bootstrap, jQuery and jQuery StickyTabs plugin are used in the project, all of which are released under the MIT license, which again, has no restrictions on what license I have to use.

4.1.3. Development Environment

I developed locally on my personal laptop, a MacBook Pro Retina, which ran OS X 10.10.1 at the start of this project, and now 10.10.3 at the end. The web server is running from Apache with PHP 5.6 via Homebrew, although later downgraded to match the PHP environment on the AU server.

Code was written in SublimeText, and most of the browser testing took place under Google Chrome and on a Nexus 5 phone. Git was handled through the command line, as was the compilation of LaTeX. SequelPro[2] was used to connect to databases both locally and remotely.

Sketch[3] was used for the creation of logos and other graphics used in AWESOME, and Dia[4] was used to create the Unified Modeling Language (UML) diagrams and database schema design in this report.

4.2. Security Audit and Code Review

The security audit of AWESOME was one of my first tasks on this project and it uncovered some issues. First of all, the admin login accepted any credentials, and secondly, the mysqli functions used to connect to the database

[2]SequelPro Homepage: http://www.sequelpro.com
[3]Sketch Homepage: http://bohemiancoding.com/sketch
[4]Dia Homepage: http://dia-installer.de

are no longer recommended to be used. Instead, PDO should be used to connect to databases, as this provides much better security through prepared statements, preventing SQL injection attacks.

The prototype was also written in a completely procedural style, with if statements for each bit of Internationalisation spread throughout the codebase. Rewriting to use OOP practices as well as a design pattern would kill several birds with one stone, as I could also implement PDO and proper i18n whilst I was at it.

4.3. Model-view-controller Framework

The Model-view-controller framework was fairly straight forward to code at first, especially following the tutorial mentioned previously[6]. However I soon ran into limitations of the framework, and the time to code in additional features ate away at the time I needed to produce a working survey tool.

In the end, AU server limitations forced me to throw away most of the functionality in the MVC framework.

4.4. Internationalisation Framework

The i18n system is small but very useful, albeit not as flexible or powerful as a mature i18n framework. It has no ability to pluralise strings, nor does it offer any variable substitution like some frameworks might be expected to do. This lack of complexity, I think makes it better for non-technical people to provide language translations though, as everything is read in through simple JSON files.

4.5. Deploying to an AU server

Getting AWESOME running on a server at AU took much longer than anticipated for a few reasons. Firstly, the PHP version was 5.3, which was attempted to be upgraded to 5.6, but it didn't work out, so I had to change some code to ensure it was 5.3 compatible. Secondly, I didn't have any direct access to the server, so any change that was made had to be zipped, sent via email to Sandy, and then he would upload the files for me, provided it was between 9am and 5pm on a weekday. Another issue was that the database would not connect, even though it was configured correctly.

This was solved by Sandy adding a wildcard rule to the database hostname limit, which enabled connection from any machine. This poor access coupled with troubleshooting issues regarding PHP version and database connections issues were a serious hinderance.

Once these issues were resolved, AWESOME was working on the university server okay, with one condition. The server I was on could only be accessed through the AU Virtual Private Network (VPN), and so anybody who received a link to a questionnaire had to either be on the university network, or connected through the university VPN to be able to view it. Another issue is that HTTPS was not set up correctly, and so instead of either redirecting to HTTP, or failing a certificate check, it would not allow access to AWESOME. This caused problems for people using browser plugins that forced HTTPS on websites.

Because of these issues, I feel that response rates through AWESOME are significantly lower than they would otherwise be, despite being around the same 20% figure that Google Forms had.

I feel that once server access is properly fixed, response rates could be 50% or even higher, which is a significant improvement upon previous online module evaluation methods.

5. Testing

5.1. Automated Testing

For this project, a Continous Integration (CI) platform was chosen to be used in order to facilitate automated unit tests. By using CI ensured that every time code was committed with unit tests, the suite was ran and results were instantly available. Any test failures resulted in a notification email being sent to identify when and where a problem had occurred.

After seeing fellow classmates use TravisCI on previous projects, and reviewing the features it provides, I decided to utilise it in this project. Travis offered testing across multiple PHP versions, which proved helpful when trying to provision a server on the university network.

This was indispensable mid-way through the project, when functionality in the i18n framework needed to be changed and tests started to fail. I wouldn't have noticed any errors when manually testing, but the unit tests brought up edge cases which were then dealt with.

5.2. User Testing

User testing was carried out by creating a survey and entering volunteer's AU usernames in the student CSV data along with sample modules. The first real test through the AU servers was sent to staff. This uncovered many issues with the setup of AWESOME on the AU server, which took some time to fix. After these issues were resolved, an end-of-semester questionnaire was sent out to the vast majority of students in the Computer Science department. From first years, to masters students.

User testing revealed some useful information via the feedback form, as well as through a question in the survey which was asked about how easy AWESOME was to use.

5.3. Acceptance Testing

Acceptance testing was carried out on the submitted version of AWESOME and results in test tables can be found in Appendix B with 16/18 (89%) of tests passing. More detail of the two failing tests can be found in the appendix entry.

6. Evaluation

6.1. Were the objectives and requirements met?

6.1.1. Objectives

The objectives listed in section 1.3 are listed below. All of which have been met, with the exception of the second one, for reasons listed below.

- ✓- **Security Audit the AWESOME prototype.** Prototype audited and code reviewed. Aggressive refactoring eliminated issues.
- ? - **Bring the prototype up to modern development standards.** An MVC and i18n framework were written and used, but to get it working on the AU server, the MVC framework had to be largely torn apart.
- ✓- **Finish any incomplete functionality.** The program is in a usable state, and has already been used to collect module evaluation data. Future extensions are listed in section 6.8.
- ✓- **Run AWESOME on a departmental server.** AWESOME was run on a departmental server and ran surveys for 504 Computer Science students.

6.1.2. Requirements

The requirements listed in section 2.1

- ✓- **Automatic questionnaire generation per-student**
- ✓- **The ability to generate quick mid-term questionnaires**
- ✓- **No need to type in registration details**
- ✓- **Targeted follow-up reminder emails**
- ✓- **Anonymous responses**
- ✗- **Visually appealing analytics** - Results are available with graphs on Likert Scale questions, and textual comments can be read, but advanced reports and analytics are not available.

6.2. Development Environment

Having a different development environment than the server it was being deployed on was a mistake that cost a lot of time, a lot of wasted effort, and even undid a lot of progress made with AWESOME.

It took a long time to get access to a university server, and even then it was sub-optimal, as I had to jump through hoops to get things updated.

Additionally, I was disappointed that the university server was only accessible either on the Aber network or through the Aber VPN. I think this drastically affected my response rates when sending out surveys, and although it matched the previous response rates of 20%, I feel it could have been higher, even up to 50% response rate, which would have been a huge success for AWESOME.

6.3. Choice of language and framework

I feel that PHP may have been the correct choice in language, given that Ruby is a lot harder to deploy on most servers. Especially if AWESOME is destined for wider use than a single department.

With regards to framework, I feel as though a large chunk of my time was spent writing my own framework, which turned out pale in comparison with a mature framework such as Laravel. Having said that though, I do question whether MVC was even the correct choice for this project. There is only ever going to be one view for a respondent, and that is a questionnaire with questions in it. For the admin dashboard, things do get a little more complicated, but in essence, you are creating, viewing, deleting or editing a survey. That is one item, and I don't think that MVC really suits this situation.

6.4. Blog

I used a development blog greatly to my advantage during this project. It was mainly used as a personal diary to help me remember things whilst writing this report, but it also served as a means of communication between my supervisor and me.

6.5. Degree

Working on AWESOME has been very relevant to my degree, G401, as it has made me tackle a significant amount of work, not only from scratch, but also taking over an existing one and making informed decisions on the best course of action.

6.6. Upper Management

Throughout the project, I have had meetings with the university management, and Professor J. Grattan to discuss the potential to use AWESOME on a university-wide scale. Everybody I talked to seemed to be impressed with the software, and what it could bring over existing methods of module evaluation.

About mid-way through the project, it came to light that Information Services (IS) had been working on a similar project too, through Blackboard. Theirs tied in to the university systems much better than AWESOME for obvious reasons, but focus on usability was certainly lacking in their prototype compared to AWESOME.

Discussions between the two systems are still ongoing, but I do believe that AWESOME is the superior system, especially after some of the improvements and extensions mentioned in section 6.8 have been carried out.

6.7. Time Management

I feel as though I wasted a lot of time writing the MVC framework for it to pretty much not be used. I think if the time I spent on writing the MVC framework was put elsewhere, I could have achieved a lot more a lot sooner and may have had the time to get a decent server set up to allow for access without the VPN. This would have improved response rates and I could have added some of the things listed in section 6.8.

If I were to take on this project again, I would either choose to go without a framework, just writing nice OOP code, or pick an existing and mature framework, such as Laravel.

6.8. Future Improvements

There is plenty of future improvements to be worked on for AWESOME. AWESOME's open source nature means that anybody can pick up the project and improve it, and the university upper-management have expressed keen interest in getting AWESOME implemented university-wide as a solution to student module evaluation.

Below are a small number of areas that could either be improved or added to AWESOME to make it a great piece of software that could be used university-wide to collect student module evaluation.

- **Complete Unit Test Suites** - Due to many time constraints in the project, unfortunately only the i18n framework is unit tested. By adding test suites to the MVC framework, as well as other areas of AWESOME, it allows for developers to easily refactor code without breaking stuff.
- **Re-implement MVC framework** - Due to the trouble faced with the AU server, most of the MVC framework had to be stripped back and little of it is currently used. Re-implementing this would greatly benefit codebase readability.
- **Add better CSV input validation** - Currently CSV imports only get rejected if the fields are empty. Some simple regular expressions could be made to check that the formatting of the CSV files are correct before sending any data.
- **Add more question types** - Currently, only two text-type questions exist, and one Likert Scale rating[1]
- **Add advanced results analytics** - Being able to select only certain responses can be a valuable tool to have. For example, return all textual comments for modules with a low rating, and have the word 'feedback' in the text.
- **Extend AWESOME to be multi-departmental** - Currently, AWESOME is only really suited to one department, however it could easily be extended to provide questionnaires for multiple departments.
- **University-management overview** - This was brought up in discussions with university management. They want to be able to see a list of all modules, with a rating of how well they are doing. This can be achieved by taking a mean of each module rating and displaying in a traffic light format.

[1] Likert Scale is a question-type which has the answers 'Strongly Disagree', 'Disagree', 'Neutral', 'Agree', 'Strongly Agree'

- **Pre-set Question Bank** - Having a set of pre-defined questions really speeds up the creation of questionnaires. This could be an addition which would be valuable, as it drastically reduces the amount of textual input when adding questions.

Bibliography

[1] Anonymous, "Rubber ducking." Portland Pattern Repository, Oct. 2014. (http://c2.com/cgi/wiki?RubberDucking) Accessed: 2015-04-30.

[2] Anonymous, "Module evaluation questionnaires : Student evaluation : Teaching and learning development unit." University of Sussex. (http://www.sussex.ac.uk/tldu/ideas/eval/ceq) Accessed: 2015-05-05.

[3] Anonymous, "Student module evaluation." University of Westminster, London. (http://www.westminster.ac.uk/study/current-students/your-studies/student-surveys/student-module-evaluation) Accessed: 2015-05-05.

[4] S. R. Porter, M. E. Whitcomb, and W. H. Weitzer, "Multiple surveys of students and survey fatigue," *New Directions for Institutional Research*, vol. 2004, pp. 63–73, Jan. 2004.

[5] N. McEwan, "Use of quizdom as a means of assessing student comprehension of lecture material," 2009. (http://cadair.aber.ac.uk/dspace/bitstream/handle/2160/7419/2009%20-%20McEWAN,%20N.%20-%20TC2%20-%20Use%20of%20Quizdom%20as%20a%20Means%20of%20Assessing%20Student%20Comprehension%20of%20Lecture%20Material.pdf?sequence=1) Accessed: 2015-05-05.

[6] A. Garg, "Write your own php mvc framework," Mar. 2009. (http://anantgarg.com/2009/03/13/write-your-own-php-mvc-framework-part-1/) Accessed: 2015-02-15.

[7] "Phpmailer github repository." (https://github.com/PHPMailer/PHPMailer) Accessed: 2015-04-12.

[8] M. I. of Technology, "Mit license." (http://opensource.org/licenses/MIT) Accessed: 2015-05-04.

[9] I. Free Software Foundation, "Gnu affero general public license, version 3," Nov. 2007. (https://www.gnu.org/licenses/agpl-3.0.html) Accessed: 2015-05-04.

A. Outline Project Specification

Aberystwyth Web Evaluation Surveys Of Module Experiences *(AWESOME)*

Report Name	Outline Project Specification
Author (User Id)	Benjamin Brooks (beb12)
Supervisor (User Id)	Hannah Dee (hmd1)
Module	CS39440
Degree Scheme	G401 (Computer Science)
Date	February 12, 2015
Revision	1.1
Status	Release

Outline Project Specification - 1.1 (Release) Benjamin Brooks (beb12)

1 Project description

The Aberystwyth Web Evaluation Surveys Of Module Experiences (AWESOME), is a prototype that enables departments to gather feedback by students about modules, lecturers, and other departmental issues. It is intended to replace and improve upon the current method of collecting feedback via Google Forms. This can be achieved by providing a personalised survey for each student to make questions personalised whilst also keeping results anonymous and confidential, and being able to chase up students for not completing their questionnaire.

AWESOME is a PHP web application developed by Keiron O'Shea during the summer of 2014 under the supervision of Dr. Hannah Dee. This project's goal is to bring the current prototype up to a well written, functioning, implementable, and extensible standard. Security of the system is critical, and so implementing a continuous integration system with unit tests and vulnerability scanning is vital to get up and running early on in the project. The system must be multilingual and accessible to adhere to the university's policies.

Advanced analytics and reports is also a feature which is highly requested by university management. For example, the system needs to be able to extract textual comments from the worst performing modules containing the word 'Feedback'. This can help upper management identify problematic areas in the university and look into the issue further.

The prototype has been demoed at a Learning and Teaching Enhancement Committee meeting to gather feedback about the current status of the project. The consensus from the committee is that the prototype is very impressive and would solve a need that is longed for by the university management. This confirmation by the committee further proves a need for this software to follow best software development practices to ensure that it can be used and extended in the future.

2 Proposed tasks

The prototype is currently written in procedural PHP, using the Twig [2] framework as a templating engine. In order to make the program more extensible and easier to maintain, it would make sense to refactor the current codebase to follow an object oriented (OOP) model-view-controller (MVC) [4] [5] architectural pattern.

The first task that needs to happen is a full security audit of the current version of the prototype. From there, the project can be refactored using best software practices, such as OOP, unit testing and MVC to separate the view and logic.

After the refactoring is done and the software is up to the same functional specification, additional functionality can then be introduced. The following task lists are what can be expected of each aspect of the project.

2.1 Refactoring tasks

Change procedural design – Design a new software architecture using MVC with OOP.

Unit Testing – Use Travis CI [1] to do automated unit testing and vulnerability scanning.

Secure admin dashboard – Use LDAP HTTP authentication via *.htaccess*.

PHP Data Objects (PDO) – Change the current *mysqli* and *tidy_sql* implementation to use PDO for greater security, flexibility and features when interacting with databases.

Accessibility (a11y) audit – Ensure all student-facing pages are accessible for disabled users.

2.2 Additional Functionality

Internationalisation (i18n) – Implement extensible i18n system to fully support Welsh, and additional languages.

Relational Database – Modify the database schema to be object oriented and relational.

Advanced Analytics – Create a system which can narrow down responses to criteria (e.g. Find modules with a low satisfaction score which mention 'feedback' in comments)

Traffic Light Dashboard – Have a dashboard showing traffic lights for all modules and departments to detect current issues.

3 Project deliverables

As this project has a fairly tight and fixed deadline on the temperature questionnaire testing, a lot of the refactoring work will be done as soon and as quickly as possible in order to get the project to a state that could be used.

After this initial sprint, the project can then be extended with the additional features described above. A rough outline of timeframes for deliverables is below.

Outline Project Specification – 2015-02-06 – This document.

OOP MVC Class Diagram – 2015-02-09 – UML Class diagram to describe MVC design.

OOP MVC Release – 2015-02-20 – Functional OOP MVC version of the prototype.

i18n and a11y – 2015-02-24 – Add internationalisation support and audit accessibility.

Temperature Test – Week 4-5 – Internal/closed functional testing.

Temperature Questionnaire – Week 6 – Questionnaire sent out to two departments.

Mid-Project Demonstration – 2015-03-09 – Start date of Mid-Project Demonstrations.

Analytics/Reports feature – 2015-04-17 – Have the analytics feature finished.

Final Report – 2015-05-07 – Final report hand-in.

Final Demonstrations – 2015-05-11 – Final project demonstrations.

Annotated Bibliography

[1] "Travis CI: Building a PHP project," http://docs.travis-ci.com/user/languages/php/, Feb. 2015, accessed Feb 2015.

Travis CI is a hosted continous integration system that connects to GitHub to help with automated unit testing and vulnerability scanning. I will be using this to provide automated unit testing and vulnerability scanning.

[2] "Twig - The flexible, fast, and secure template engine for PHP," http://twig.sensiolabs.org, Feb. 2015, accessed Feb 2015.

Twig is the templating engine used in the prototype by the previous author.

[3] K. T. Brinko, "The Practice of Giving Feedback to Improve Teaching: What Is Effective?" *The Journal of Higher Education*, vol. 64, no. 5, 1993. [Online]. Available: http://www.jstor.org/stable/2959994

An article describing effective practices in gathering feedback in an academic environment. This is general background reading to get a greater understanding of the project's aims and goals.

[4] C. Hopkins, "PHP Master — The MVC Pattern and PHP," http://www.sitepoint.com/the-mvc-pattern-and-php-1, Feb. 2015.

Another set of articles on OOP MVC in PHP which have been read in preparation for the refactor.

[5] J. Stump, "Understanding MVC in PHP," http://archive.oreilly.com/pub/a/php/archive/mvc-intro.html, Feb. 2015.

A series of articles on how to write MVC architecture in PHP which have been used as background reading ready for the OOP MVC design.

[6] H. K. Wachtel, "Student Evaluation of College Teaching Effectiveness: a brief review," *Assessment & Evaluation in Higher Education*, vol. 23, no. 2, pp. 191–212, Jan. 1998. [Online]. Available: http://dx.doi.org/10.1080/0260293980230207

An article describing effective practices in gathering feedback in an academic environment. This is general background reading to get a greater understanding of the project's aims and goals.

B. Test Tables

These tests were carried out as part of acceptance testing using the submitted version of AWESOME. Results are listed below.

- Test D8 fails as there is currently no safeguard in place for incorrect CSV format.
- Test D12 fails as questionnaire respondent tokens aren't created until the moment of sending.

Both of these issues a quick fixes, but are unable to be implemented in time for the dissertation hand-in. They will be fixed in a future version.

ID	Requirement	Input	Expected Output	Actual Output	Pass
D1	Admin Dashboard sits behind a login	Go to `awesome.url/admin`	Login form pops up	Login form pops up	✓
D2	Admin Dashboard login accepts correct username and password	Enter correct credentials into login form	Redirected to admin dashboard	Redirected to admin dashboard	✓
D3	Admin Dashboard login refuses incorrect username	Enter incorrect username and correct password	Login fails	Login Fails	✓
D4	Admin Dashboard login refuses incorrect password	Enter correct username and incorrect password	Login fails	Login Fails	✓
D5	Create Survey button starts the creation of survey wizard	Click 'Create Survey' button	Taken to screen asking for titles, description, and CSV data	Taken to screen asking for titles, description, and CSV data	✓
D6	Survey must have a title	Enter no title and press 'create' button	Message pops up disallowing action	Message pops up disallowing action	✓
D7	Survey must have CSV data	Enter no CSV data and press 'create' button	Message pops up disallowing action	Message pops up disallowing action	✓
D8	Must check CSV data for formatting issues	Enter incorrect CSV data and press 'create' button	Message pops up informing of CSV formatting error	Taken to survey page without any errors	✗
D9	Unlocked survey page allows entry of question text and type	Type a question title and select an answer type for a question	Able to enter text in question text and select an answer type	Able to enter text in question text and select an answer type	✓
D10	Able to add a new question	Click add question button	A new question text and answer type row appears	A new question text and answer type row appears	✓
D11	Able to delete an existing question	Click the delete question button next to a question	The question is removed and deleted on save	The question is removed and deleted on save	✓
D12	Be able view participants in a survey before sending	Click 'Participants' tab in a survey page	Respondents should be listed	Respondent list is empty	✗
D13	Be able to send a survey	Click the 'Send' button in a survey page	A message displaying how many people the survey was sent to	A message displaying how many people the survey was sent to	✓

Table B.1.: Acceptance testing table of AWESOME's Admin Dashboard

ID	Requirement	Input	Expected Output	Actual Output	Pass
Q1	Students receive an email with a link to personalised questionnaire	Check email inbox	See an email with unique link	See an email with unique link	✓
Q2	Questionnaire can be viewed correctly	Click link in email	See questionnaire displayed	See questionnaire displayed	✓
Q3	Answers can be selected or filled in	Click on a Likert rating or type in a text box	See answer selected or text typed in	See answer selected or text typed in	✓
Q4	Answers can be submitted	Select or type some answers and press the send button	Page redirected with a message informing of completion	Page redirected with a message informing of completion	✓
Q5	Questionnaire can't be completed twice	Re-visit the unique link and try to complete the survey again	Receive an error message informing that the questionnaire has already been completed	Receive an error message informing that the questionnaire has already been completed	✓

Table B.2.: Acceptance testing of AWESOME's Questionnaire