Classroom Quiz System

Final Report for CS39440 Major Project

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

7th May 2017
Version: 1.9 (Release)

This report was submitted as partial fulfilment of a BSc degree in Computer Science (Inc Integrated Industrial And Professional Training) (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 .....Alexander Michael Levi Taylor.....

Date .....07/05/2017.....

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 .....Alexander Michael Levi Taylor.....

Date .....07/05/2017.....

Acknowledgements

I'd like to thank my supervisor Chris Loftus for his continues support and advice throughout the project, and my second marker Laurence Tyler for his feedback from the mid project demonstration. In addition, I would like to thank Alun Jones, Stephen James and Max Atkins for helping set me up a server to use for hosting the project.

I am also grateful to Carly Jackson and Kerey Taylor for taking time out to proof read my dissertation.

Abstract

The focus of this project was to create an application that can be used by lecturers to allow students to answer questions and display these results live in lectures. The project was based on a pre-existing application, called Qwizdom.

The system is a web based application built in Laravel, a PHP based framework. The system provides two main functions, the running of questions on their own, and streaming 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. It also stops students changing the HTML to submit their own answers to be displayed by the lecturer, a security problem prevalent with Qwizdom.

Development followed an Extreme Programming approach, utilising a number of practices such as Refactoring. Stories were created and used as the functional requirements for the system, and stories were worked on individually within weekly iterations. Analysis, design, implementation and testing were done for each story rather than having an upfront design stage or end testing stage.

User testing was carried out within a lecture in which it was used to gather some module feedback from students in a workshop. The students and lecturer gave positive feedback about the system.

The project was an overall success, with some potential future changes and additions but was received well by users and the requirements of the system were met.

CONTENTS

LIST OF FIGURES

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 [1] 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

There has been a demand for a replacement to Qwizdom to fix some key problems it 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 per question
- Students can submit their own answers by changing the HTML on the page which are then displayed when the lecturer displays the results
- Creating quizzes is tied into Microsoft PowerPoint which not all lecturers use

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 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 way 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

PHP was the language selected due to two 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 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, and has been chosen for the web server part of this project [2]. There are a number of reasons for choosing Laravel in this project over other available frameworks. The 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 [3], 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 WebSockets, 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 a Microsoft PowerPoint extension much like Qwizdom to create slides embedded with questions [4]. 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.

1.3 Process

The methodology chosen for this project was an Extreme Programming [5] (XP) 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 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

XP does not enforce the use of any single practice and as such 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 adapted to be done by one person at the start of the project, the list of stories 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.

On the flip side, a number of practices were less appropriate and had to be dropped, such as Pair Programming and Collective Code Ownership, due to there only being one developer.

1.3.2 Stories - functional requirements

Before any development could start, an initial 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. See appendix 5.1 for the intial list of user stories.

1.4 Planning Game

A plan was created early in development:

- 10 iterations beginning on Thursdays after 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 complete the majority of coding 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

Iterations

This chapter chronicles the development in a diary format. Iterations were used during the development of this project, with each iteration typically consisting of at least one story and often included non story based work. However, due to the iteration process, each story consisted of analysis, design, implementation and testing. Therefore, each of these pieces of work are given within the iteration they were completed in, negating the need for an initial design section.

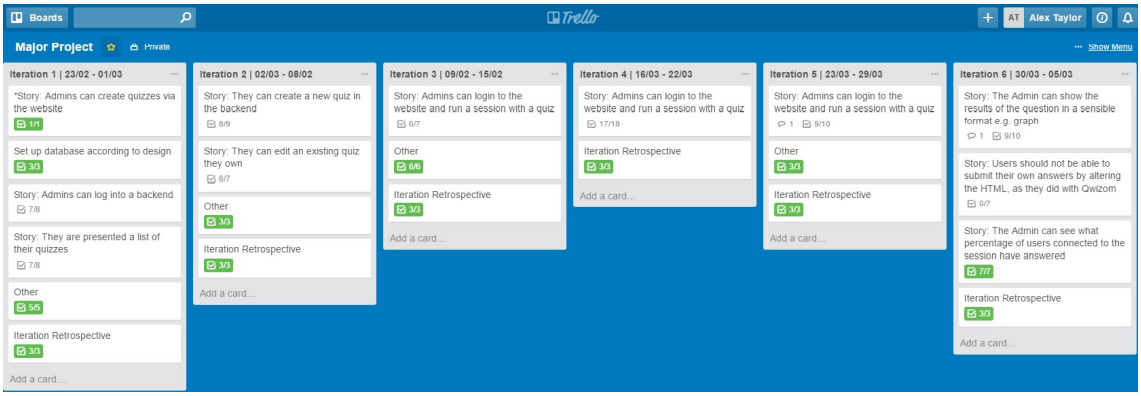
2.1 Outline

1. Iteration 0 mostly involved setting up of the tools to be used in the project, some research and the creation of an initial database design.
2. Iteration 1 was concerned with the implementation of the database design and setting up the initial admin area with a list of quizzes owned by the admin who was logged in.
3. Iterations 2 focussed on the creation of quizzes by admin users.
4. Iterations 3, 4 and 5 involved the implementation of running sessions.
5. Iteration 6 focussed on the submission of answers and displaying results. It also dealt with the security issue from Qwizdom.
6. Iteration 7 involved making the application mobile responsive, the part two redesign which led to implementation of the the way slideshows could be run on the application.
7. Iteration 8 is only concerned with report writing.
8. Iteration 9 contained security testing, more report work, the server setup , and bug fixing.
9. Iteration 10 was the last iteration and followed the final bits of report writing, the user testing, and the implementation of downloading results and various bug fixes.

2.2 Tools used

There were a number of tools used throughout development that do not fit under a single iteration. The main three were a Wordpress diary, a Trello board and Git.

Figure 2.1: Part of the Trello board used for this project, showing a few iterations and the high level tasks within them.



The diary was used to keep track of the major pieces of development done each week and any questions/ issues that needed to be raised during supervisor meetings. This was a simple Wordpress site that was set up several years ago and repurposed for the use in this project.

Trello is a project management tool that allows users to create a ”board” for a project that then allows the creation of cards that tasks can be added to [6]. Feature tracking is a key part of the system as tasks and cards can be marked as complete, Trello suits XP based approaches as cards can be created and edited on the fly for each iteration. For this project, a card was created for each iteration, with a list of stories and tasks to be completed within that iteration. Once a task was completed, it was checked as such giving an easy to read percentage completion for the story. See 2.1 below for an example of the board used in this project.

Git was used extensively during development, primarily for version control but also a form of backup and as an easy way to deploy the system. Version control is very important as it allows tracking changes to the project and if a mistake is made, can be used to undo these mistakes. It also allows branching out bits of functionality, so developing features can be done separate from the rest of the project and once working, merged back in. Github was used as the repository service due to it being free and straightforward to use [7].

Laravel comes with a custom Command Line Interface, CLI, called Artisan [8]. Artisan is used to run the majorty of commands needed in Laravel, from the migrations to the creation of controllers. An example would be *php artisan migrate* which migrates the database. It is always prefixed with ”php”.

The last main tool used was Microsoft Visio [9] which was used to create the majority of diagrams in the design sections of the report.

As well as tools, three online sources were used extensively. The first is Laravel Recipes, which provided a number of examples of how to use helpers and facades [10]. The second, Laracasts, an online Laravel tutorial site, especially the Intro to Laravel series [11]. And finally, the documentation pages [12].

2.3 Iteration 0 16/02 - 22/02

2.3.1 Initial work

This iteration did not consist of any development. It consisted mainly of research and design work. But most importantly it included writing the stories for the project.

Other items of work involved setting up a diary, a Trello board for documenting all the work and setting up the initial Git repository along with linking it to Github.

2.3.1.1 Research

Research was focussed on several areas, and some documents were produced from this research. The first area of research was on the frameworks available, after which Laravel was selected. Hosting options were also explored and a document discussing both of these was produced, this document will form part of the Background chapter when it comes to writing the report.

Some additional research was also completed on the Microsoft PowerPoint Add-In for the second part of the project. This was mainly to get a sense of the amount of work involved, and what language/ environments would be needed. This research did not go into much depth as further research will be undertaken when that part of the work is reached.

2.3.1.2 Design

The design work completed was a database design. This would be needed throughout the whole project so made sense to write initially, though it may be subject to significant changes. Figure 2.2 is the ER diagram.

2.3.1.3 Stories

A list of stories was produced and written up in a separate document. These stories will be the basis of the project and act as the functional requirements for the evaluation at the end of development (appendix 5.1).

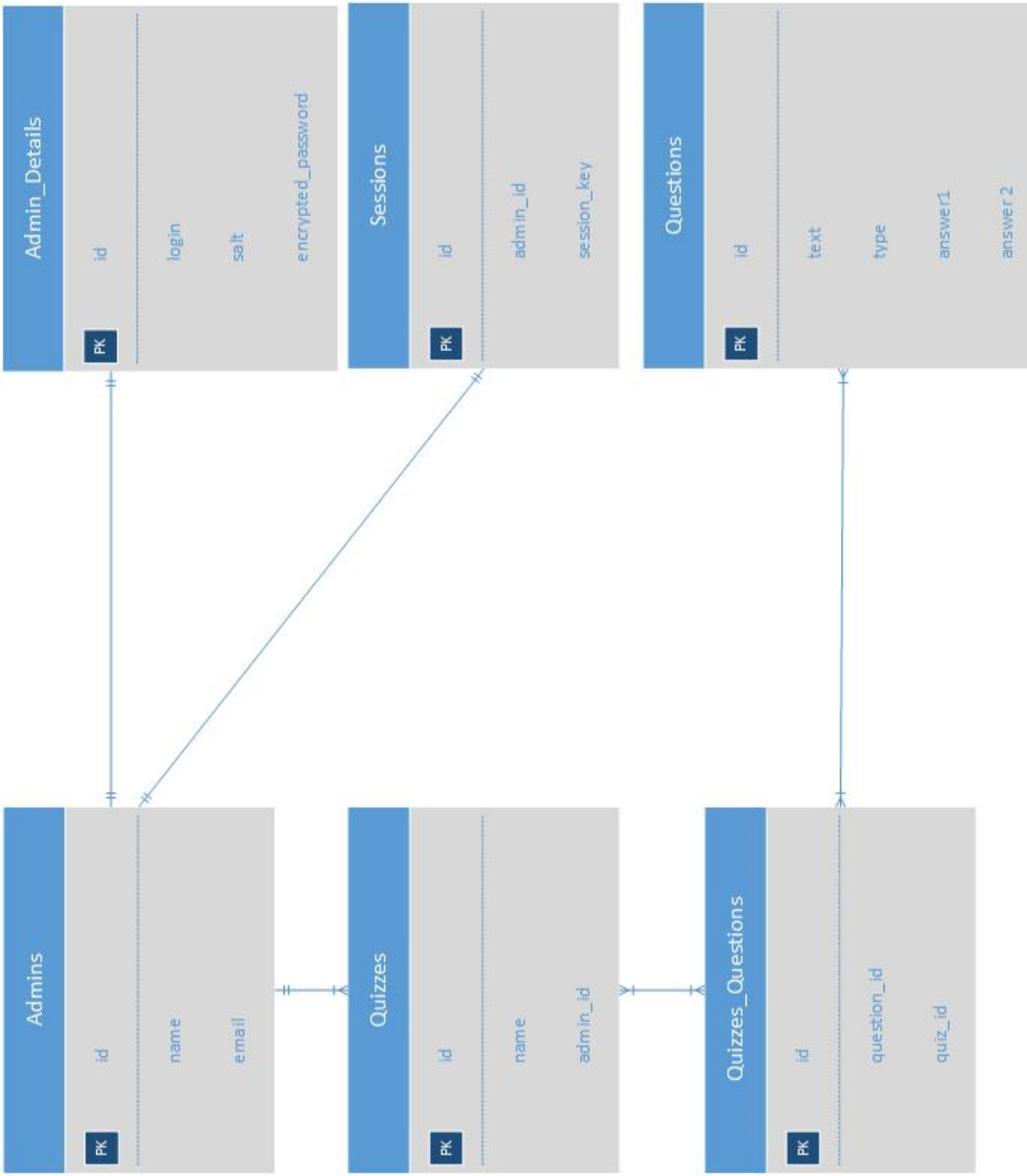


Figure 2.2: Entity relationship digram for the initial database

2.4 Iteration 1 23/02 - 01/03

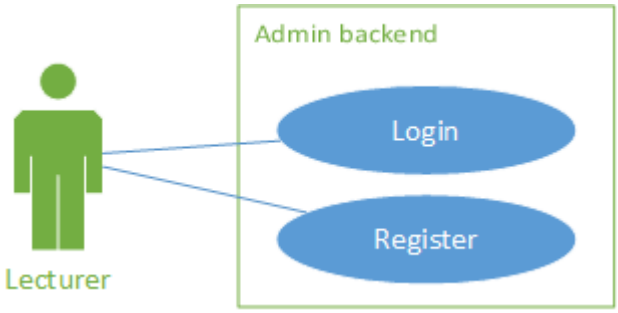
2.4.1 Story: Admins can create quizzes via the website

2.4.1.1 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 with a list of their quizzes
- (5) They can create a new quiz in the backend
- (3) They can edit an existing quiz they own

Figure 2.3: Use case diagram for the admin backend for login



2.4.2 Story: Admins can log into the backend

2.4.2.1 Analysis - breakdown of tasks

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

2.4.2.2 Design

A use case diagram was produced to show the actions a lecturer has for this stage2.3.

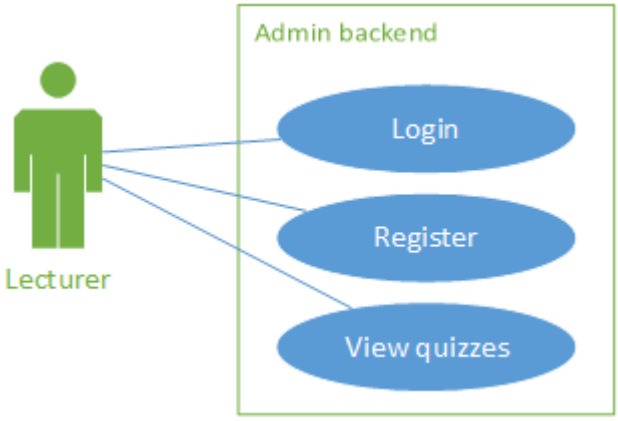
2.4.2.3 Implementation

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

2.4.2.4 Testing

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

Figure 2.4: Use case diagram for the admin backend including show quizzes



2.4.3 Story: They are presented a list of their quizzes

2.4.3.1 Analysis - breakdown of tasks

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

2.4.3.2 Design

The HomeController to be removed in this period of work and the QuizController used in its place.

See figure 2.4 for the use case.

2.4.3.3 Implementation

Changing the controller was a small change to the routes and the quiz view file so that it used the same layout as the original home views. However, the current user was not known. However, a helper function is provided: *auth()->user()* which gets the user object. Using this objects id inside an Eloquent, Laravels Object-Relational Mapping tool, call it is easy to find all the quizzes owned by that user.

2.4.3.4 Testing

The tests written here highlighted a problem with the testing framework. 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.) Chrome is used as the Remote Web Driver for running these tests and 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 incorrect due to the test trying to log in even though it

is already logged in. The solution to this was to create each new user in the next id record, whilst this was somewhat convoluted, it worked.

2.4.4 Non-story work

2.4.4.1 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. Setting up the tables and their relationships was undertaken at the start because it was perceived that this would not take long. Building these manually with a Database Management System would take a while, however the database in Laravel is built using migrations, speeding up the process a lot.

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*

2.4.4.2 Seed data

Because of the amount of changes to the database that were being made, the tables were repeatedly wiped with each migration data was needed for testing it all worked. 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. The objects are then saved to the database and the seeders can be run when a migration is called to generate this data.

2.4.4.3 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 [13] and is aesthetically pleasing so the basic styling has been kept. This layout was modified somewhat to add some menu options that persist across pages. This layout was then used by all the backend pages by extending it.

2.4.5 Review and restrospective

2.4.5.1 Review

Completed stories with their associated difficulties in parenthesis:

1. (6) Admins can create quizzes via the website
2. (3) Admins can login to a backend
3. (2) They are presented a list of their quizzes

The first story was quite large so was split up, thus making it quite an easy task with no problems.

The second and third were not that challenging either, two in particular was made much easier thanks to the built in auth builder from Laravel. This means that the rating of three is wrong and it should be lower at a one. Story three fits a difficulty of two as it provided no major problems, but it did give ample opportunity to learn Laravel much better. For example learning the structure and how to use helper functions such as the one to get the currently authenticated user.

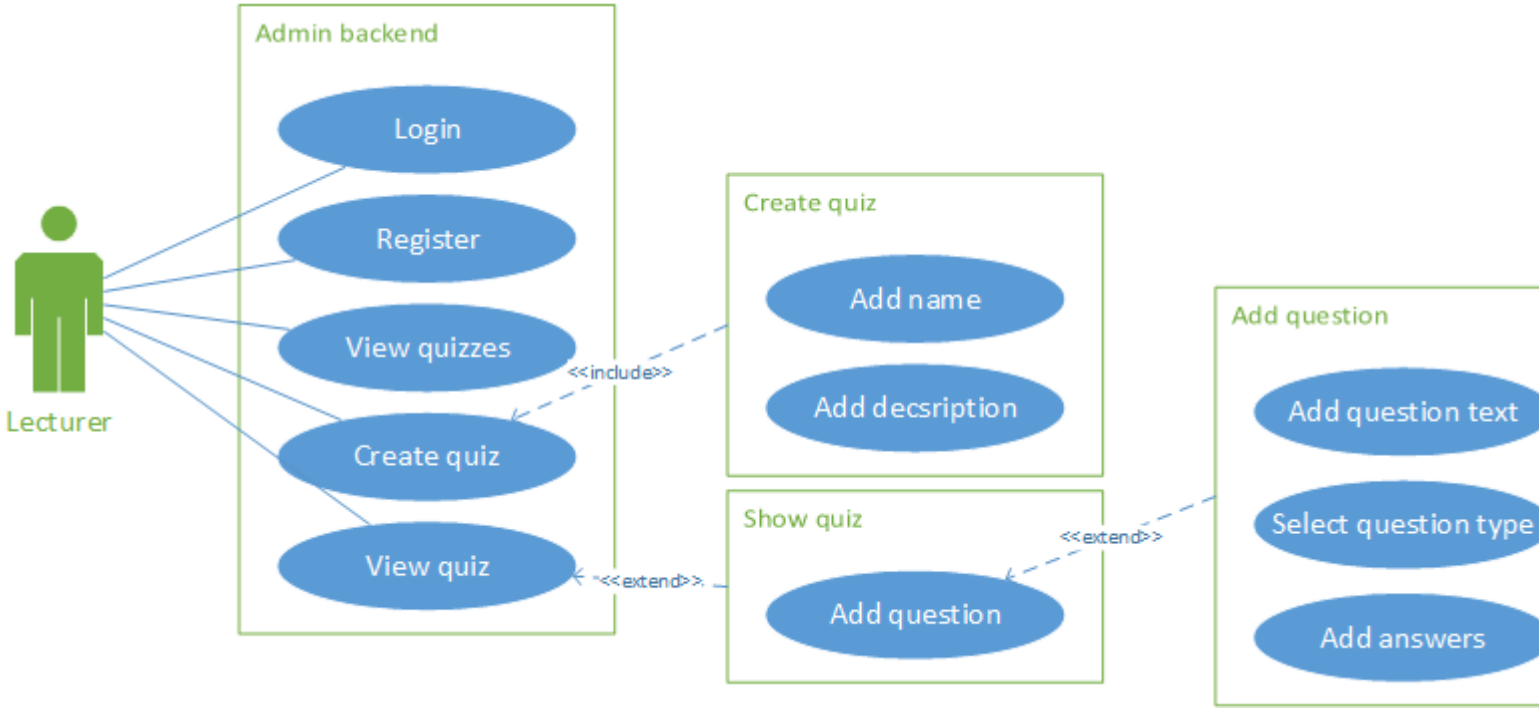
All implementation tasks were completed, but some documentation for story two and three was missed. That is to be completed at the start of the next iteration.

2.4.5.2 Retrospective

Breaking the stories up into substories and then those stories into subtasks is very useful for the project. It splits the functionality well allowing individual components to be built and showcased on the development site.

However, the documentation did not go well. A majority of the design, testing and implementation documentation was missed during the iteration. This was mostly due to overestimating the amount of time available to write this. In further iterations, more emphasis should be placed the documentation section of work.

Figure 2.5: Use case diagram for the admin backend with create quiz functionality



2.5 Iteration 2 02/03 - 08/03

2.5.1 Story: They can create a new quiz in the backend

2.5.1.1 Analysis - breakdown of tasks

- Need quiz/new form to add name
- Add validation
- Redirect to quiz.show for new quiz
- Add a button to quiz.show for adding a question

2.5.1.2 Design

There are a number of function that lecturers will have in this story, see the use case in figure 2.5 for these.

2.5.1.3 Implementation

A create page was added for quizzes, which was linked to from the home page. This was mapped to /quizzes/create address. A simple form was added to this page with the name and description of the quiz. Questions would be added on an individual quiz page. The form sends the form as an HTTP POST request to the /quizzes page which then maps onto the 'Store' action in the Quiz

Controller. This action is used for server side validation and saving the data to the database using an Eloquent query. The user_id is assigned to the quiz by getting the users id. This Eloquent query also returns the id of the new quiz. Using this id, this Store function redirects to the newly created quiz.

A quiz page displays the name and description and has a button to add any questions to the quiz. This button links to the questions/create page that functions in the same way as the quiz creation, albeit with the relevant fields in the form. The questions are assigned to the quiz by passing the quiz id to the question creation page in a GET variables via the url. This allows the question to be assigned to the quiz during its creation in the database and to be redirected back to the quiz page.

For validation there are two parts, one on the HTML form and the second on the server side. The HTML validation (client side) uses simple HTML 5 validation like the ”required” attribute in the form inputs. Further frontend validation could have been added utilising JavaScript but the HTML solution was far easier to implement, being the addition of one word and not several lines of code that have to be built and added to the correct pages. The server side complements this as client side validation can be circumvented by determined users editing the HTML or JavaScript.

The server side validation in Laravel is well supported and made as easy as possible to implement. The Store function used to save data from the POST request simply calls *$this->validate()* on the request data. Inside this validation function, various rules can be imposed on individual pieces of the request data such as simply requiring that is is filled or that it has to have a minimum length [14].

Something that Laravel enforces is the use of CSRF tokens to prevent cross site request forgery attacks on the site. A token is generated for the project during its creation, and this token must be placed in a hidden field within every form so that it can verify that the authenticated user is the one actually making the requests to the application [15].

2.5.1.4 Testing

A problem encountered in this set of tests is the speed of testing. Running six tests takes about a minute. The reason for this is that the tests use DatabaseMigrations rather than DatabaseTransactions, meaning that the database is migrated for every test rather than just rolling back any additions made in a test before running the next test. Using transactions would reduce the time taken by only doing a migration at the start and then using a transaction for each test. Alternatively having a pre migrated test database on which you run transactions would work too, but this would mean any time you change your database, you would have to remember to migrate the test database.

After attempting to use the transactions it appears as though they are not usable within Dusk. This is because Dusk is running in another process and migrations are the only choice [16].

2.5.2 Story: They can edit an existing quiz they own

2.5.2.1 Analysis - breakdown of tasks

- There is an edit button on quizzes that links to quiz/edit
- Name of the quiz can be edited
- Questions scan be added or deleted
- Questions content can be edited
- A quiz can be deleted

2.5.2.2 Design

There was not much design for this stage as it only adds an edit page which should look the same as the create pages in the above story except that the input fields are pre-filled with data from the database. Additionally a delete button could be added for quizzes and questions. Figure 2.6 for the use case that now includes editing and deleting quizzes and questions.

2.5.2.3 Implementation

Pre filling the edit page was done by getting the record in the database by the question id and setting the input fields to the content from the question being edited. The main difference between the creation stage was that the form had to use the HTTP PATCH method rather than POST. This PATCH method was automatically routed to the update function in the controller [17]. Within this function, server side validation was added and Eloquent used to update the row in the database. It then redirects back to the individual quiz page.

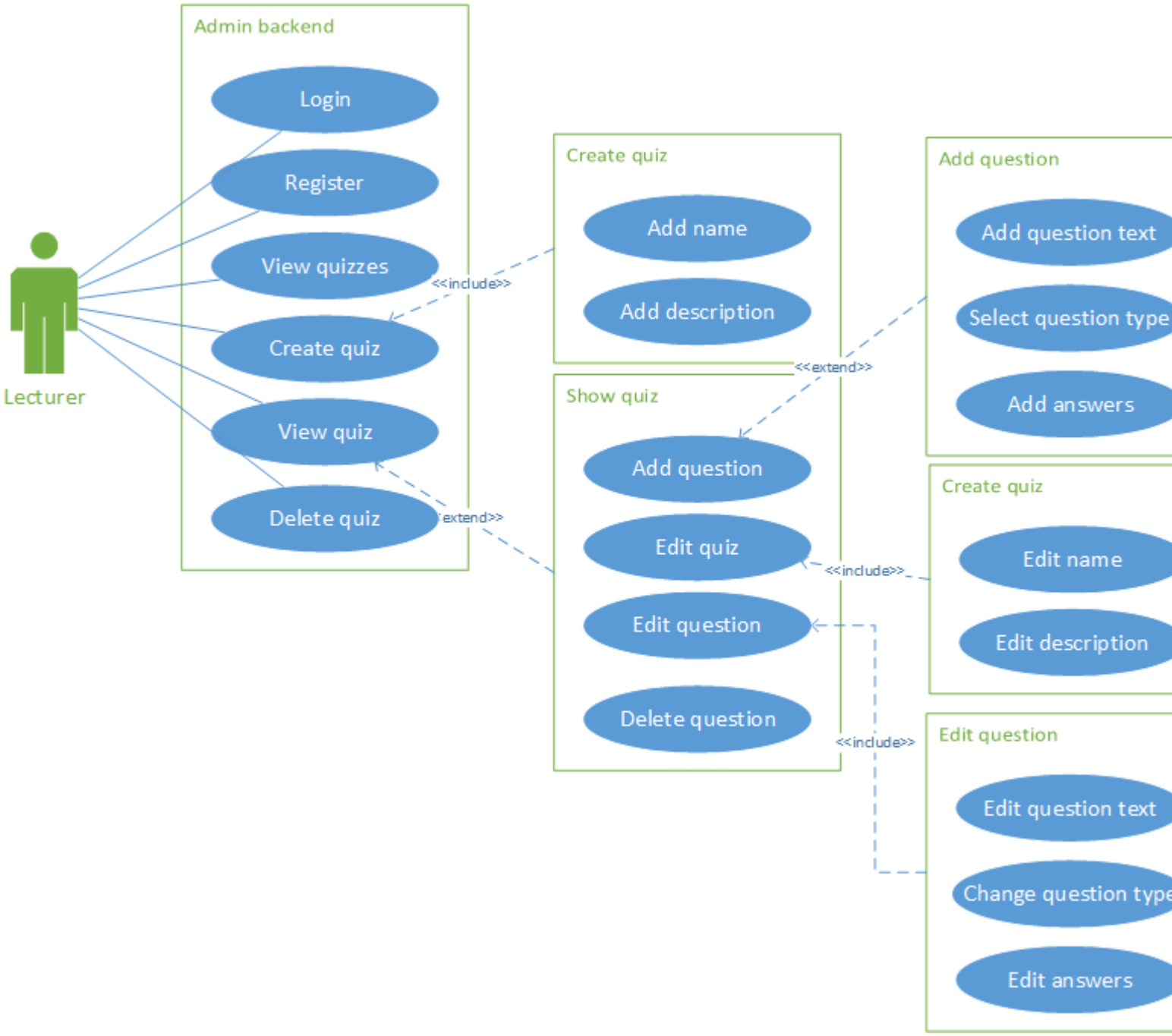
Delete buttons were added to the quizzes and questions. These were not a simple <a>tag that linked to a delete page as with the create and edit pages. It required an HTTP DELETE request instead of a normal GET so had to be a form that specified the DELETE method and an id of the quiz. This was mapped to the destroy functions in the respective controllers which simply called an Eloquent function that removed the record.

A problem was that when a quiz or question was deleted, its associated quiz_question row was not. To fix this the foreign key reference columns in the quizzes and questions tables were changed to include onDelete cascade. This meant that they would delete the rows in the quiz_question table that referenced the row being deleted. For a quiz deletion, this had to go further and also find the questions associated with the quiz and delete all of those, which used a Eloquent function to find them using the quiz_question table before the rows in that table were deleted and then delete the questions using those ids.

2.5.2.4 Testing

Some simple tests written for editing and deleting quizzes and questions, no issues encountered.

Figure 2.6: Use case diagram for the admin backend with edit quiz functionality



2.5.3 Non-story work

2.5.3.1 Seeding

Some more data needed to be seeded for this subsection of work, namely the question data. This involved creating some new Model Factories and using them in the seeders correctly. One issue was trying to create many questions for individual quizzes, but this was overcome using some very basic looping and calling the model factories in the right places.

2.5.3.2 Quiz description

It was decided that quizzes should probably have descriptions attached to them, in case the lecturer needs reminding of what it was in 6 months time. This involved creating a new migration and adding a column to the table.

2.5.3.3 Travis setup

Travis is a CI tool that can be used to run tests automatically on git pushes. Setting it up involved doing some spike work with another git project. It was straightforward to set up, needing the addition of a travis.yml file which specifies what Travis will do after a push. The associated Github project also had to be enabled on Travis to be run.

2.5.4 Review and restrospective

2.5.4.1 Review

1. (5) Admins can create a new quiz in the backend
2. (3) They can edit an existing quiz they own

The difficulty ratings were accurate, with the work taking about the expected amount. Some of the extra work here involved trying to set up Travis for continuous integration which could be used for running tests automatically was somewhat problematic but did not take up too much time. These stories were almost fully completed with the exception of some testing as described below.

2.5.4.2 Retrospective

The documentation stage improved this iteration compared to the last, even with the added responsibility of finishing documentation from the week before.

Unfortunately, testing took a hit, this is mainly due to the attempted use of test driven development. This practice was originally done to try and improve the code quality. However the nature of a web application and the use of a large framework is problematic for TDD. Trying to write tests for all the various parts of the web application is hard, and the framework also does a lot of work for you, so writing things like unit tests beforehand is hard. Application tests using Laravel Dusk are more appropriate but they test the story itself rather than underlying logic, which is less applicable to TDD.

To mitigate the lack of tests written in future, TDD will be dropped due to the above reasons.

2.6 Iteration 3 09/03 - 15/03

2.6.1 Story: Admins can login to the website and run a session with a quiz

2.6.1.1 Analysis - breakdown of tasks

This story was large and whilst it will probably be split into a number of tasks, the first task was some spike work on the way this system would work. One route is with WebSockets, a relatively new technology that allows the server to push data to the page quickly and easily [18]. This would be a nice solution as it is relatively future proof and a better alternative to other solutions that involve writing a lot of JavaScript and forcing page changes on the users.

2.6.1.2 Design

WebSockets were introduced into Laravel 5 and have become one of the defacto ways to update the front end in real time. Unlike other web frameworks such as Ruby on Rails the web sockets in Laravel requires some extra set up. In Rails the WebSockets can be run on the main web server that is used to run the site [19]. In Laravel however another server has to be set up to run these. Laravel offers several different drivers for running the WebSockets, including a Redis server and a third party application called Pusher [20]. Pusher handles most of the work for the developer and requires little set up other than creating a free account [21].

2.6.1.3 Implementation

Pusher was chosen due to its ease of use and due to its high recommendation rate within the Laravel community. A problem with Pusher is that due to it being a third party service, it is not free indefinitely (also it has a number of users limitation). However, you could host your own Redis server to mitigate this cost. This means that the system had to be designed in a way that the driver for WebSockets could be changed with ease.

After configuring the spike work application to use Pusher, a simple Laravel Event was created to send a message to the Pusher. To test this event a couple of methods were implemented. The first method was to register a route that triggers this event when the page is visited.

The other method a custom made artisan command that can trigger the event: php artisan quiz:send message. A command is useful for testing the event as it can be used without building a button on the front end to trigger it.

Two online guides were used to help write this section of the system due to WebSockets being a new technology [22] [23]. Though they were used to get the basic concepts working, the final code used within the project is tailored to the system and is therefore different overall.

2.6.1.4 Testing

There was no testing as it was spike work.

2.6.2 Non-story work

2.6.2.1 Refactor controllers

A problem was that the Eloquent functions for modifying and reading from the database were within the controllers. In a strict MVC system, this functionality should be inside the models. To fix this, the Eloquent functions were refactored into the respective quiz and question models. This would make it easier for future development if anything ever needed to change within the logic concerning any database interactions.

2.6.2.2 Changing the database structure

The original design was changed, and the quiz_questions table for linking quizzes and questions together was removed. The original reason for this table was that questions could be reused between quizzes. However, after thinking about the potential for that to happen, and the issues that the structure was causing in the model logic, it was decided that the quiz_question table was more of a hindrance than a help.

The questions table now has a quiz_id column that references the quiz to which it belongs to. This change means that the relationships between the two tables are much easier to define in the models, having belongsTo and hasMany functions in both that automatically return the objects they own or which they belong to. Thanks to the previous refactoring of model logic, changing this functionality was straightforward. Deleting rows in the database was also simpler now that there was no quiz_questions table.

However, when a quiz was deleted, the question associated with it were not automatically deleted. To fix this, the onDelete cascade option was added to the quiz, that means all objects belonging to the quiz being deleted would also be deleted.

2.6.2.3 Frontend setup

This was the first time that any custom CSS was written and Laravel uses SASS to generate its CSS. To build this SASS into CSS, and also to build any future JavaScript, Laravel Mix was needed to run builds for this code. For this, npm and node had to be installed so that they could run their Webpack build scripts. There were some issues trying to get the build scripts to run, even though it worked on fresh installs of Laravel, but eventually a Github issue was discovered that had some solutions [24].

2.6.2.4 Dusk and Travis

The aim was to try and set up Laravel Dusk on Travis. Dusk can use a few different browser drivers for running the tests in, by default this is Chrome but Travis comes preconfigured with PhantomJS which is also supported by Dusk. It should be easy to swap the drivers such that Travis can use PhantomJS. Unfortunately this did not work, and no reason could be found. A project that was supposed to run its Dusk tests on Travis was even copied and that does not work, therefore it had to be concluded it is currently not working.

2.6.3 Review and restrospective

2.6.3.1 Review

No actual story work was completed, however a large amount of non story work was completed. Additionally a large amount of spike work was completed for the story: Admins can login to the website and run a session with a quiz

There was a significant amount of refactoring in this iteration which was not too challenging. The most problematic bit was trying to get the database migrations done properly.

The spike work was particularly challenging this week because it was attempting to work with a new technology. It also involved the use of various parts of the Laravel framework not encountered before, like events and writing custom artisan commands.

2.6.3.2 Retrospective

The spike work was very useful for the overrall devlopment of the application. Additionally the refactoring and restructing of the database and testing has helped a lot with the future outlook of the application.

A negative was the continuous integration on Travis. Laravel Dusk is a relatively new testing framework and Travis does not seem to work well with it. This means that the tests written cannot be run on Travis. It therefore makes CI somewhat pointless if the tests cannot be run and this will be left for now.

2.7 Iteration 4 16/03 - 22/03

2.7.1 Story: Admins can log into the website and run a session with a quiz

2.7.1.1 Analysis - breakdown of tasks

After doing some spike work in the previous iteration, this task can now be approached and broken into several sub tasks:

- Set up config for Pusher
- Add event for broadcasting
- Write a command to trigger this event
- Add a button to quizzes to trigger this event
- Display the response on the front end using the JavaScript which listens for Pusher events
- Add a session key box to front page
- Add session_id to admin users and allow them to change it
- When admin clicks run, this session_id can be entered into the key box to join a channel with the name of the id
- The user will be presented with the initial quiz page which will be filled with the name and description of the quiz by default
- The admin will see the same page but with an ”admin panel”
- This admin panel has a next and previous button for questions
- When these buttons are pressed, the question is sent to pusher
- These question are rendered as a form on the user end and admin end
- The user can submit the form

2.7.1.2 Design

Mockups of the student end of the system:

Use case of the new system 2.9

When the event is triggered, quiz data will be sent to Pusher, which will then send back this data to the channel specified. The session page will be listening on that channel, and this data can be appended to the page using JavaScript.

Figure 2.7: Design for a quiz on a desktop

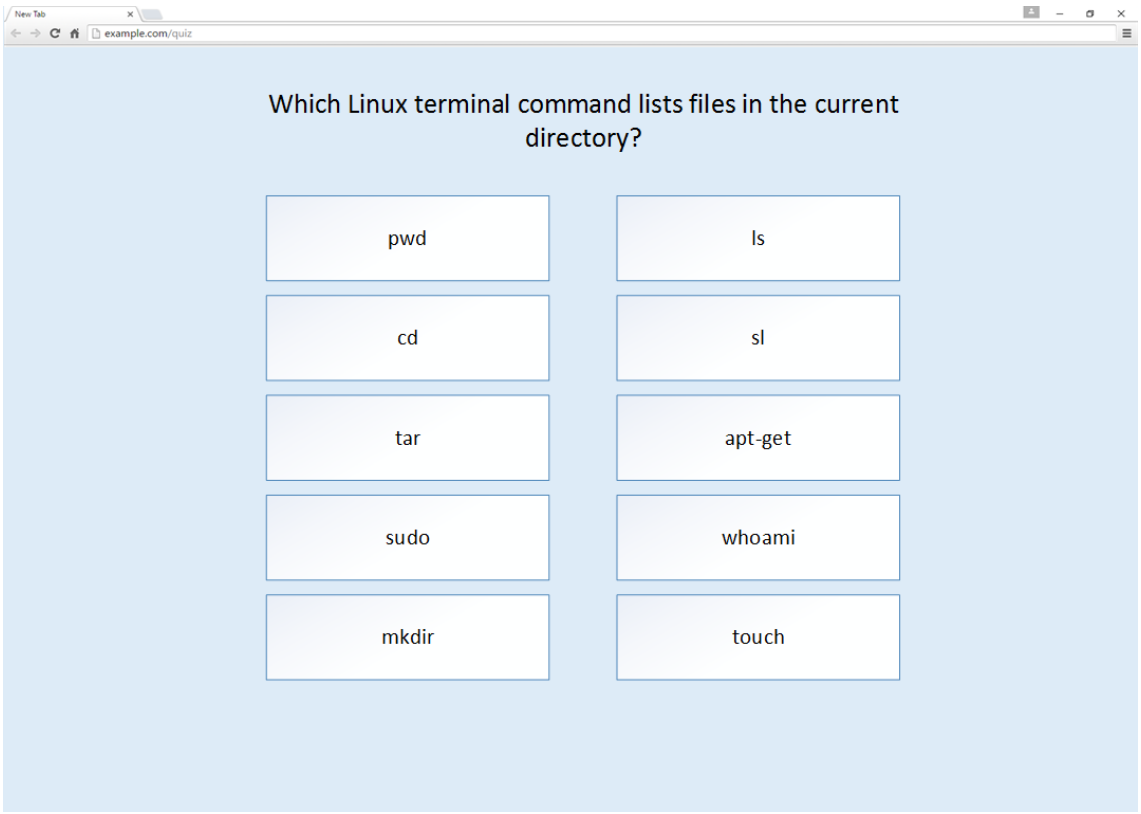


Figure 2.8: Design for a quiz on a mobile device

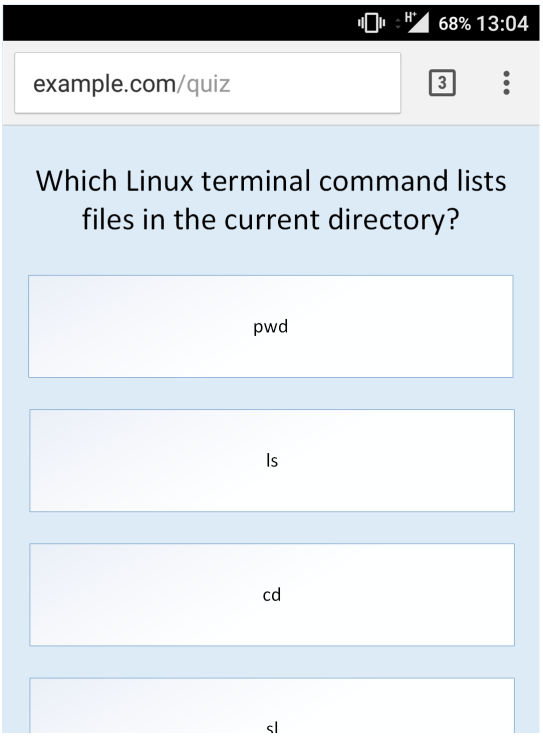
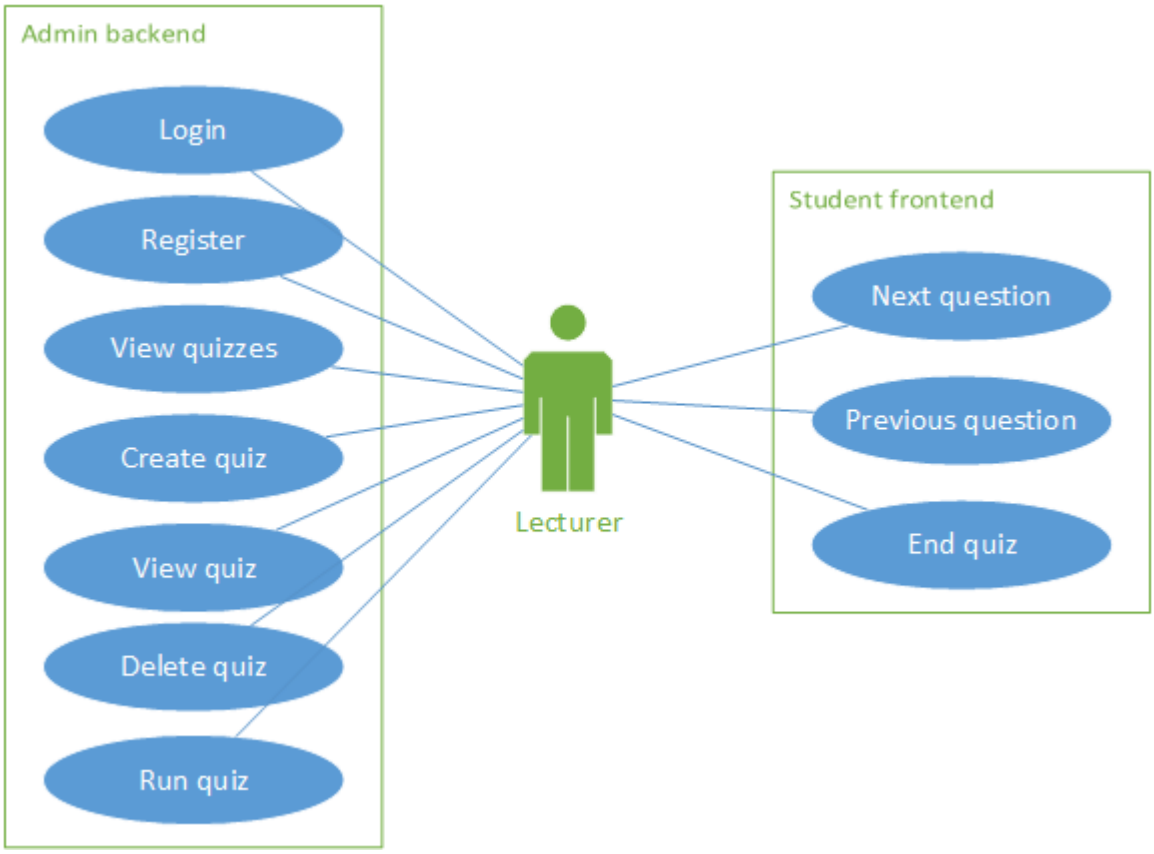


Figure 2.9: Use case diagram with the quiz control functionality. Omitted backend extends and includes to make more readable. See previous iterations for more complete backend use case.



2.7.1.3 Implementation

Work began by configuring Pusher, creating an event and writing some basic JavaScript to append the data to the front page. Work was also done on the admin panel, making it only visible to logged in users (the lecturer running the quiz), and adding the functionality for previous and next question buttons. These are buttons that send AJAX requests to quiz controller actions which then trigger the event for WebSockets with the appropriate question data.

Early into the iteration a major flaw with using the WebSockets was discovered, whilst new content was easy to add to the page, if a new user joined the session late, they would see a blank page or the original content of the page. To remedy this, the current position in the quiz was kept track of and the page renders the question specified at that position. At the same time, the WebSockets would be running and updating the page for the user if the lecturer was changing questions. It was decided that the best place for keeping track was within the session table, and so columns for the position, quiz_id and if it was running were added.

Rendering the questions calls an action in the question controller, which takes the type of question, the quiz_id and position. Using this data one of several available views can be rendered, one for each type of question for example multiple choice or boolean. The views render a form with the question text, the possible answers as buttons and a submit button.

For a question to be rendered by default on page load as described above, the page includes the question by rendering the appropriate view. However, the type of question to render is not known, so the types are looped over and checked if they are equal to the ones defined earlier in the config file and rendered to the view if they are the same type. One problem with this is accessing the custom config file. The function to do this was inside the question controller and not the quiz controller.

Whilst the function could be copied, it was better to create one single helper function that is global to the application. This involved creating a helpers file and adding the function there and then registering the helper function in the composer.json file [25].

2.7.1.4 Testing

No testing as the story is not yet complete.

2.7.2 Review and retrospective

2.7.2.1 Review

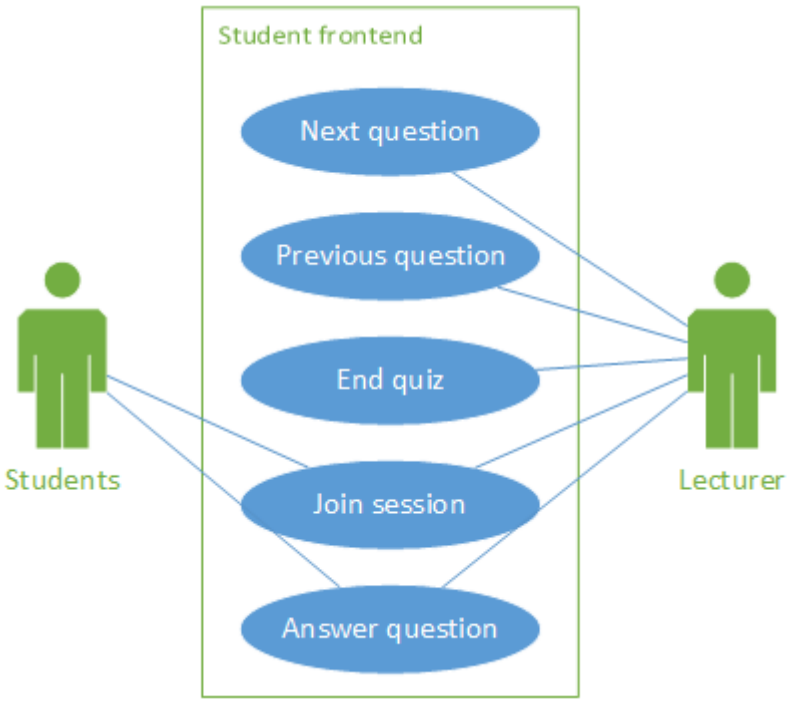
Once again this week no actual story has been completed. This is due to the size of the story and also that some stories are a struggle to work on individually. It seems that the current work covers about six stories rather than just one. Something to not then is that once this piece of work is complete, the velocity will shoot up.

2.7.2.2 Retrospective

The amount of work completed was significant, even if project velocity is 0. But as mentioned this is due to many stories being worked on simultaneously.

However, finishing stories and breaking them up into smaller bits of work that can be finished to give a velocity.

Figure 2.10: Use case for front end now including what students can do



2.8 Iteration 5 23/03 - 29/04

2.8.1 Story: Admins can login to the backend and run a session with a quiz

2.8.1.1 Analysis - breakdown of tasks

Tasks left over from the previous week:

- Form on front page redirects to quiz
- Questions rendered as a form
- User can submit the form with their answers
- Questions need limiting on going above/ below max and min

2.8.1.2 Design

Students will be able to connect to the sessions so the frontend use case was extended to reflect this, see figure 2.10.

2.8.1.3 Implementation

Users still had to connect to the quizzes from the welcome page. A problem with this was that the sessions were being run under the url /quiz/session_name. This meant that submitting a GET

or POST form would not work as the url could not be specified, as the user specifies the session key. A solution was to have an input field that took the session key and then used JavaScript to redirect to that session. An alternative solution would have been to use a form that redirected to a controller action that then itself redirected to the correct session page. Whilst this method would work, having multiple redirects in one request is not good practice or acceptable for the users' browser.

The JavaScript lacked the ability to properly validate inputs, and so there also needed to be server side checks. A custom Middleware class was written to handle the checking of valid session keys. Middleware provides the functionality to filter incoming HTTP requests within controllers [26]. When the form was submitted, the Middleware function would check the database, and if the requested key was not in the database, would redirect back to the welcome page with an error. Else, it would allow the redirection to the quiz session.

Once a student was on the session page, the page still had to update when the lecturer pressed next or prev. When this was pressed the page receives a message through the WebSockets with the new questions content. It first removes the content of the page with JQuery. It then checked the position of the quiz, if it is zero, it then appends the quiz start data on to the page. This start data includes information about the quiz, such as the name and description. If the position is not zero, then it will render the question at that position in the quiz.

To render a question, an Ajax request is made to a question page that will be populated with the appropriate data using the information from the Web Sockets as the variables for finding the correct question. The Ajax'ed page content is then appended on to the current page using more JQuery.

For the user to submit an answer, the form again had to use JavaScript. This was primarily to make the form user friendly and for different questions types. JS was used to add a class to the options and highlight the ones that were chosen, be it one option in a multiple choice question or multiple answers on a multiple selection question. When the student hits submit, the answers are selected by the options with the classes that were added when an option is selected. A POST request is then sent with Ajax to the quiz session page. The POST'ed data consisted of the answers given, in the form "answer*", where the * was the answer number, as well as the question number. This data is not yet handled by the server.

A bug with multiple selection questions was with the answers selection. To get the answer value, the array of elements was iterated over and a new array with the answers was produced. However, due to using a JQuery array function to iterate over these, it created a JQuery collection, rather than a simple array. This collection was not accepted by the Ajax request and this collection had to be transformed into an array with a vanilla JS toArray() function.

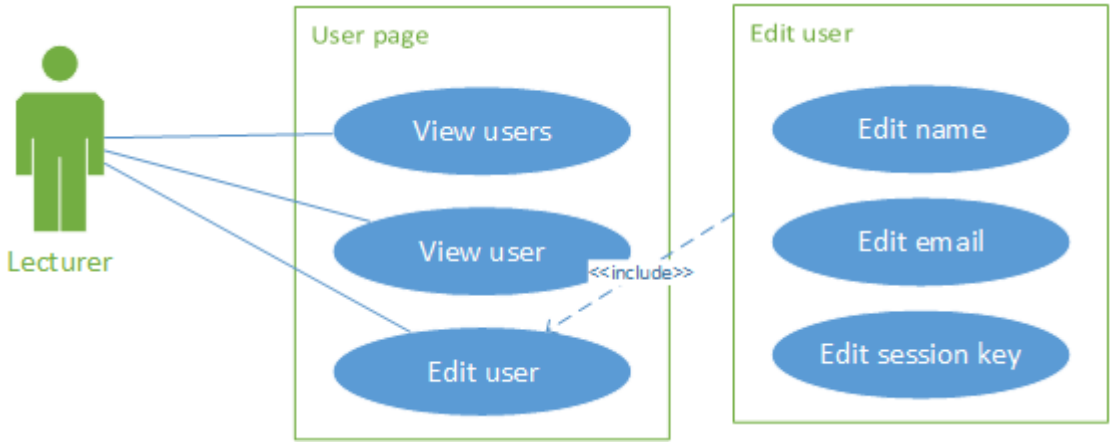
The last part of development for this iteration involved adding some simple checks to the next and previous position functions to ensure they did not go below 0 or above the number of questions in the quiz.

2.8.1.4 Testing

A number of tests were written for this large part of the system split into three main areas: admins running quizzes, students joining quizzes, and multiple sessions being run simulataneously. Some of tests used Dusks ability to create two browsers and imitate two users of the system, this was

used to check WebSocket functionality, see appendix C for the function.

Figure 2.11: Use case for the users page



2.8.2 Non-story work

2.8.2.1 User page

Significant work was done on the user pages to ensure that an admin could view and edit their details including changing their session key. This work just involved creating the user blade pages, adding the various functions to the controller and model and adding functionality to the authentication actions for registering a new user. Similar work had taken several hours when the initial quiz pages were created, however with the experience from that work this only took an hour. Figure 2.11

2.8.2.2 JavaScript clean up

A lot of JavaScript that was being rendered on the page was not used and proved useless to the project. In fact the libraries are quite large and take up a significant amount of space. Removing these unused libraries, VueJs being the largest, freed up the JavaScript output file considerably, which increased load times significantly.

2.8.3 Review and retrospective

2.8.3.1 Review

This week seven stories have been completed:

1. (7) Admins can login to the website and run a session with a quiz
2. (8) Multiple different sessions can be run simultaneously
3. (3) The Admin specifies which question is being run by clicking next/prev question etc
4. (5) Sessions can be joined by users via the website
5. (3) Up to 300 users should be able to join and answer questions
6. (1) Users answer the question being displayed by the quiz
7. (2) Admins should be able to change their session key

The first six were completed at the end of three iterations of work, though the work was listed under the first story in that list, which means they some were undoubtedly finished before this iteration but not counted as being done. This was primarily because of the large amount of cross over between the stories but also that some stories are not easily made into tasks and act more as functional requirements, such as the story about 300 users.

In terms of difficulty, because of the way this worked on, breaking down difficulty is hard. However, the first one is relatively accurate, the second story is not however. Once one channel was established, creating a second was trivial. Story 3 also really stands out as this is where a lot of work went during the last iteration, rendering the question was a large amount of work and also involved changing the database structure alongside the html, PHP and JavaScript work.

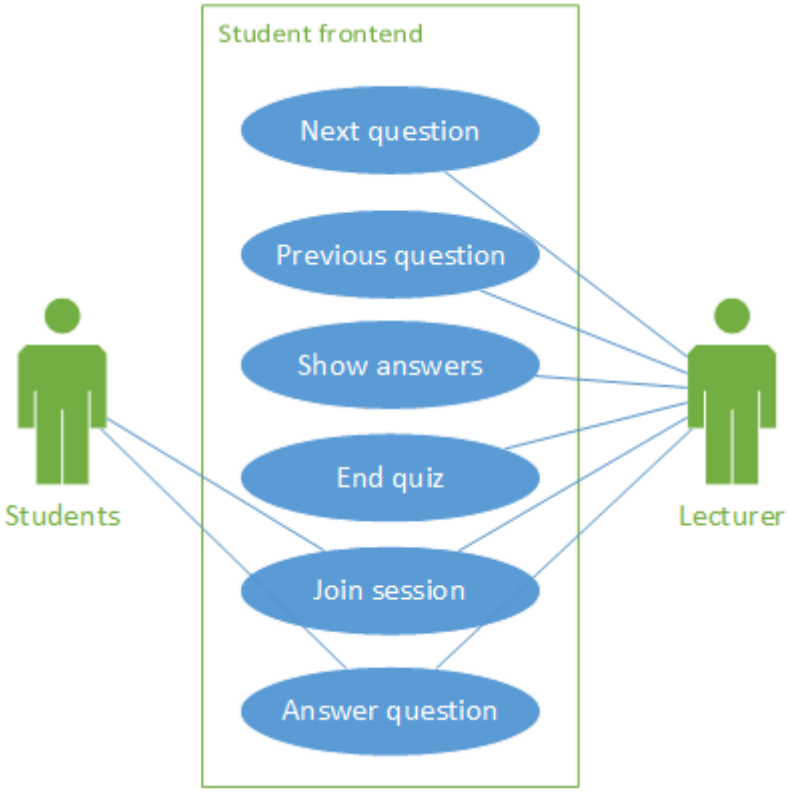
Story 4 is also overestimated, joining a channel was relatively easy, so should probably be lowered to a difficulty of around 3. Number 5 is subjective, as Pusher the WebSocket service provider says it can support up to 100 users, though there has been no large scale test as of this moment. Story 6 should probably be increased as it was a bit more work than simply rendering a form, there is some JavaScript for submitting answers. The last story is quite accurate, even though it was a large amount of work, it was easy after having done a lot of similar work earlier in the project.

2.8.3.2 Retrospective

The work progressed well, with a lot of completed stories the project is in a much better state now.

Unfortunately breaking all the work into their respective stories was not done, and this could have been used to track progress much better. Also it seems as though a lot of the predicted difficulties for the stories were off, both above and below estimate.

Figure 2.12: Use case for front end now including show results



2.9 Iteration 6 30/03 - 05/04

2.9.1 Story: The Admin can show the results of the question in a sensible format e.g. graph

2.9.1.1 Analysis - breakdown of tasks

- Save the data
- Ensure no users submit answers twice
- Display that data in the admin panel
- Prettify the results tab

2.9.1.2 Design

The new use case for the front end of the system: 2.12.

2.9.1.3 Implementation

There was significant work in this story, with the first item containing the majority of work. At first it was decided that a simple database could be used to store all the responses however it was soon

realised that users could probably submit more than once if users were not associates responses. The best way to deal with users that do not login is to use the Laravel session cookie to identify them anonymously.

To store all these responses in a database would require a lot of storage space. Rather than having say six rows for a question, each with six answers that just totals up the number of times an answer was selected, the user session identifier had to be stored with the answer given so it could be associated with specific students. This meant that now there would be a row per answer, equalling at worst case 300 rows per question. This was thought to be a bit drastic so an alternative was suggested: to use a CSV file with each row having the user and an answer, with each question being a separate CSV file. These would be stored locally under a session folder in the public directory. These files would be created and destroyed during quiz cycle. This method would also be good for the downloading of answers from another story.

Ultimately, trying to write all this data to a csv proved too troublesome to do. If a user changed their answer then the entire CSV had to be copied with one line changed. Editing a single line is not possible with PHP. Therefore it was decided that going back to a database based method would be better, with the idea that at the end of a quiz, all the data would be deleted.

An answers table was created that specified the session, question, user and the answer given. The first three of those were used in a unique composite key to prevent multiple answers from the same users on the same questions in the same session. The model was then written to save this data when data was POST'ed to the /results url. Other functions included those to delete the data when a quiz was ended and various DB association functions.

To read this data on the front end, an Ajax request was set up from a button on the admin panel that GETs the same results url. This calls an action to render the results data in a JSON format. The results are in a key-value format; the answer as the key, with the value as the total number of times it was selected.

This JSON data could then be used in the ChartJS library which was selected to render the results bar chart on the page [27]. ChartJS allows the creation of an object that can be attached to an HTML canvas tag. Within this object, the options and variables that make it up can be specified using JSON. The JSON data is split into two arrays of equal length, one of the keys and one of the values, they are then passed to the chart object and the results are rendered as a bar chart. This chart is then deleted if it already exists and rendered when the results button is pressed. This means every time it is pressed, the most up to date results will be displayed.

2.9.1.4 Testing

Two areas were tested here, the first was concerned with users submitting their answers and that they were saved correctly in the database. Thus it mostly involved standard unit tests using database records rather than using Dusk.

The second area was testing lecturers showing results on their view. Testing this was difficult as the results box is rendered using an iframe and canvas HTML tags. It seemed that anything within these cannot be seen by the assertions within Laravel Dusk and therefore cannot be reliably tested. A single test was written to check whether or not the iframe was made visible when the results button was clicked.

2.9.2 Story: Users should not be able to submit their own answers by altering the HTML, as they did with Qwizom

2.9.2.1 Analysis - breakdown of tasks

Very few tasks as will be explained in the implementation subsection:

- Add check for empty arrays on frontend
- Add check for empty arrays on backend

2.9.2.2 Implementation

Thanks to the design of the previous subsection, changing the data meant an empty key was sent if the user tried to change the data submitted to the server. This could be checked in either the JavaScript or PHP, but it was decided both would be the best for maximum security.

2.9.2.3 Testing

This was hard to test, as Dusk does not allow the changing of HTML on the page or the possibility to submit false data. Additionally, even if the data could be submitted, the tests in the previous story indicate that the results cannot be read and therefore this story could not be tested fully. User testing might be a good replacement for this.

2.9.3 Story: The Admin can see what percentage of users connected to the session have answered

2.9.3.1 Analysis - breakdown of tasks

- Get total number of responses
- Post this number on the page as part of the results form

2.9.3.2 Design

This should appear above the results table, possibly as part of the title.

2.9.3.3 Implementation

After attempting to get a number of users connected to the channel it became clear that this number is not readily accessible for public channels, reserved more for use with private channels which require authentication. The number can be obtained via the Pusher API but implementing that would have meant locking some functionality down with Pusher and the long term plan is likely to involve changing to a Redis based WebSocket system. Therefore this percentage has been changed to the total number of responses. This gives a good idea to the lecturer of what percentage of users have responded. This number was simply added to the title attribute of the chart.

2.9.3.4 Testing

As with the above stories this iteration, this could not be tested because the information is contained within the iframe that cannot be seen in Dusk tests.

2.9.4 Review and restrospective

2.9.4.1 Review

Three stories have been completed this week:

- (4) The Admin can show the results of the question in a sensible format e.g. graph
- (4) Users should not be able to submit their own answers by altering the HTML, as they did with Qwizom
- (2) The Admin can see what percentage of users connected to the session have answered

These difficulty estimates for these stories were all too high. Using a third party library made it easy to create the graphs and thanks to the way the system had been previously implemented, users submitting their own answers was a few lines of code to implement. The last story was changes somewhat as described above, however adding a simple number of the number of submitters was simple.

2.9.4.2 Retrospective

An XP approach really helped with the design stage of the results, as it changed a few times during this iteration. If this was being done in a more planned way, those changes might have been harder to implement.

2.10 Iteration 7 06/04 - 12/04

2.10.1 Story: Site should be mobile responsive

2.10.1.1 Analysis - breakdown of tasks

- Create standard media query sizes
- Create media query for quiz questions
- Media query for welcome page

2.10.1.2 Design

A quiz page should look like the design specified in iteration 4, figures (2.9).

2.10.1.3 Implementation

For the media query sizes, Bootstrap recommends some default sizes for phones, tablets etc [28]. The Chrome developer tools were used to view the page in mobile view and change the various sizes of buttons and titles in the CSS editor to be more mobile friendly.

2.10.1.4 Testing

This was not really possible to test with Dusk or unit tests, but should be easy to test with users later in the project. Additionally once live the system could be tested using the Google Responsive Test that gives a score for the mobile usability.

2.10.2 Part two re-design

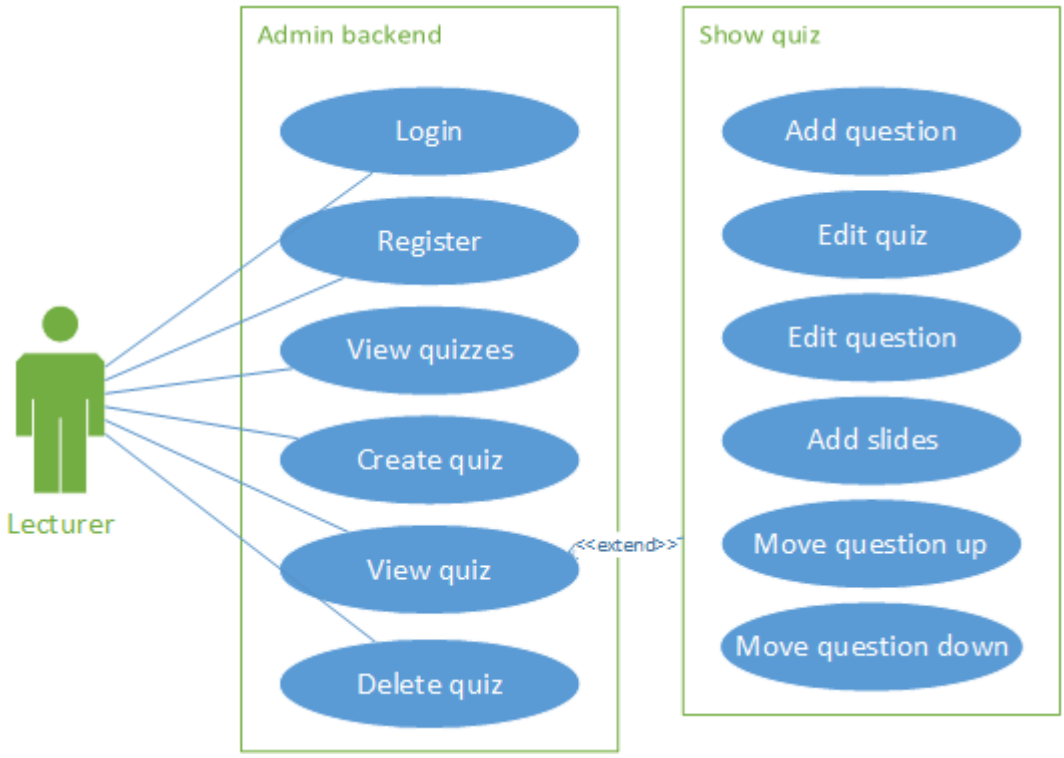
Part two involved streaming slides to students with embedded question and was originally planned to be an extension to Microsoft PowerPoint or a similar technology, such as Libre Office. However, after some further research this work was determined to be too large. This meant an alternative solution had to be devised or some extra requirements had to be added to the system. An alternative solution was found rather than abandon the idea of having slides within the quizzes. Slideshow programs usually have the ability to render their slides as PDF, a format which is more heavily supported compared to a propriety format such as .pptx provided by Microsoft. If a PDF is uploaded to the application, PHP could be used to turn these slides into images, which can then be placed on the quiz pages.

This new approach meant some changes to the original stories for the second part, here are the revised stories:

- (1) The admin creates slides in their preferred editor and exports them as a PDF
- (4) The admin can upload these slides to a quiz they have created in the past
- (3) The admin can reorder the questions within the quiz to move them around the slides
- (2) When this quiz is run, it should render the slides as well as questions in the order specified

There are some disadvantages to this however. The main issues are that slide animations are not rendered as separate slides on the PDF. There are extensions for Microsoft PowerPoint that let the slides be rendered with animations occupying separate slides so as to provide a ”fake” animation. Another problem is that the slides would have to be uploaded before a lecture as it can take a few minutes to render PDF slides as images. This could be argued to be an advantage however, if lecturers upload their slides before a lecture they only need to log in to the application when they want to run them. They need not carry the slides on a memory stick or save them to their University storage.

Figure 2.13: Use case for quiz creation featuring moving questions and the add slides function. See Iteration 2 for more detailed use case diagram of the rest of the backend.



2.10.3 Story: The admin can upload these slides to a quiz they have created in the past

2.10.3.1 Analysis - breakdown of tasks

- Upload pdf slides
- Convert these slides to images
- Save references to these slides in the database
- Show the slides on the Quiz.show page

2.10.3.2 Design

See 2.13

2.10.3.3 Implementation

To upload the slides, it was done with a standard HTML file input but the backend used techniques native to Laravel [29]. These images were then saved in the /storage/app/public/slides folder. Within this, they were organised into folders named after the session id and then quiz id within those.

These slides were then converted using a library that uses Imagick [30]. A problem is that the details of these images are not kept track of, so after saving the images, references were also saved to a new table in the database. This is so that the images can be referenced quickly later on when displaying a slide. The new table was called slides which stores the file name, quiz it was associated with and a position.

The last task involved showing the slides and questions on the Quiz.show page. Seeing as one of the other new stories involved reordering questions, merely rendering the slides after the questions would not do. Both the slides and questions needed to be rendered in order of their positions. Unfortunately, there exists no simple SQL query that can select two sets of results and then order them by a common column. The easier solution was to create a PHP function to create an array of all the items in the order required, see appendix C for the function.

2.10.3.4 Testing

There does not appear to be the capacity to upload files in Dusk tests, making this impossible to test unfortunately.

2.10.4 Story: The admin can reorder the questions within the quiz to move them around the slides

2.10.4.1 Analysis - breakdown of tasks

- Add buttons to reorder questions
- This button increases or decreases position of question
- The button swaps the positions of two questions or a slide and question if required
- Add some limits to positioning such as not going below 1

2.10.4.2 Design

As a minor change to the system, no major design choices were made. Simple Bootstrap icons can be used for the position changing arrows, which was the only front end change. For the newest use case see figure 2.13.

2.10.4.3 Implementation

The main functionality to add was the changing of positions. This involved adding a new route and buttons that triggered the actions associated with this new route. It was decided that the Route would be a POST request rather than GET as the button is submitting some data, the direction and id of the question. The action that was called updates the position in the database after doing some checks of the position it wants to move to. The problem here is that there might be a slide or question occupying that position already, therefore it needs to be checked and swapped if true.

2.10.4.4 Testing

Again as there are no slides it is not fully testable. However, it was possible to test that the questions positions changed in the database when the buttons were clicked without any slides on the page.

2.10.5 Story: When the quiz is run, it should render the slides as well as the questions in the order specified

2.10.5.1 Analysis - breakdown of tasks

- Find the position in the database and decide whether its a slide or question
- If it is a question, render it like before
- If a slide, render a simple img tag and populate with the image name from the WebSockets
- Resize image to fit the page, including for mobile responsive
- Render question by default for users joining late

2.10.5.2 Design

This should work in much the same way as the questions, in that it will send an Ajax request to a page with the image name and then copy the content of the Ajax page into the live page. The content page to Ajax should be a simple container with an img tag. The other tasks are extensions to previously created functions.

2.10.5.3 Implementation

The first task was to determine what is at the position. This involved changing the current function to include a check for if the question at the requested position was null. If it was then it should find a slide at the specified position instead of a question. As well as the data about the slide or question, it would return a type so that when it was called the system knew what type of content was expected. The data was then sent to the WebSockets and another case was added to the JavaScript receiving end for triggering an Ajax call to a slide page. This page renders a simple img tag with the slide specified which was then added to the current page, similar to the way questions are rendered.

A problem with rendering the images was that files are stored within the /storage folder which was not publicly accessible. Usually only files within the /public folder are. However, there is an artisan command, *php artisan storage:link*, that creates a symbolic link from the public folder to the storage/app/public folder. This meant that the images could be accessed via /storage/ when specifying the image path.

2.10.5.4 Testing

Given that slides cannot be uploaded, testing for them is not really practical and would involve leaving some test data on the system permanently.

2.10.6 Review and restrospective

2.10.6.1 Review

- (5) Site should be mobile responsive
- (4) The admin can upload these slides to a quiz they have created in the past
- (3) The admin can reorder the questions within the quiz to move them around the slides

The difficulties associated with the stories are relatively accurate, and due to more time being spent on this iteration than others before means they could all be completed, even though some earlier iterations had less stories with lesser difficulties completed. Most of the difficulties for these stories came from the first one, which involved a lot of CSS, something with which the developer is not experienced in writing. Story two also had some difficulty associated with saving files, however this was well documented online and in the end rather simple.

After doing this work, including the incomplete story, a design decision has been called into question. Instead of saving the position of slides and questions within their prospective tables, it might make sense to have a table for the position that then links to a relevant question or slide. It would make selecting all the data easier. This could be a potential future change.

Only one story was not fully completed: When the quiz is run, it should render the slides as well as question in the order specified. This is because only a tiny part is missing, which is that the images of slides it renders are somewhat out of position on the page, they just need some css and possible resizing before the story as a whole is finished.

2.10.6.2 Retrospective

This iteration really showcased the advantages of an iteration and story based workflow. The second part of the system needed a redesign due to the amount of time left and amount of work still required if the original idea was followed. Seeing as only some research had been done on the original idea, not much time has been lost moving over to the new design. This new design was also a lot smaller in scope, meaning that it was almost finished completed within the same iteration it was designed.

Something of note that is good is the previously written documentation. Due to a few of the stories in this iteration requiring changes to existing functions that were written a while ago, knowledge of how the system worked was lacking. Thanks to the PHPDoc and comments, it was easy to piece together how the system worked again.

Unfortunately, testing did not go that well, some was missed due to the rushing of development on the new design of part two.

2.11 Iteration 8 13/04 - 19/04

2.11.1 Report writing

The majority of this iteration was devoted to report writing. Whilst there is still some development to do, the report accounts for 30% of the final grade and needed to be worked on. There were some small changes to phpdoc and comments added to the code, but no actual functionality was changed. In terms of the report, the first chapter concerning the background had been added. All the iterations documents were also added into the final report. There are a lot of TODOs scattered throughout, mostly for citations but also some to check with the project supervisor about certain details. Finally, some content was fleshed out for the testing chapter, giving an overview and descriptions of the technologies.

2.11.2 Review and retrospective

Not applicable as no development took place.

2.12 Iteration 9 20/04 - 26/04

2.12.1 Report work

A lot of report work was completed during this iteration. This primarily focussed on trying to get through the TODOs scattered throughout the report. It also included adding most of the citations for the system that were noted down during development. The testing and design chapters were also fleshed out far more, and once all this was completed a draft was sent to the project supervisor for some feedback.

2.12.2 Security testing

Five tests were written to test the security of the system. These tested SQL Injection and Cross Site Scripting across the site. Whilst Laravel is supposed to handle much of this it is still good practice to test it and ensure the security of the system. There are only two main places where these attacks could take place, the session search field on the front page of the application, and the question and quiz creation pages in the admin area. The tests proved that Laravel does indeed stop XSS and SQL injection by escaping the inputs. For further information see section 4.3.

2.12.3 Server setup

There was some time spent on setting up the server for use in the user tests in the final iteration. The server was provided by the Department of Computer Science. The setup included moving the project onto the server via Git, setting up the MySQL server, and making the site visible to the internet with an htaccess. With this task, help was obtained from two fellow students Stephen James and Max Atkins, and also from the Computer Officer in IMPACS, Alun Jones. This was due to insufficient experience in the area of server setup.

2.12.4 Bug fixing

There was a significant amount of bug fixing in this iteration. The main two bugs were fixing the CSS of the slides in quizzes and multiple selection questions not submitting correctly.

For the first, a lot of time was spent on trying to write custom CSS for the image tag that would centre it and fill the page. However, there proved to be no need to write any CSS, instead the solution was to add a simple bootstrap container around the image. Whilst the image might be too high for the page, scrolling up and down is not much of a usability problem assuming most slides have blank space at the bottom. This Bootstrap container fixes the image sizing problems that were encountered beforehand, both on a standard desktop and on mobile and tablet devices automatically.

The second major bug fix concerned how multiple selection question answers were saved. The difference between this and a boolean or multiple choice question is that multiple answers are submitted, and an array was chosen to do this. However, this array was not handled correctly when saving these results. The solution was to create entries of the string versions of the answers joined together, so an answer row would contain the answer of ”answer1, answer3” for someone who picked both answer1 and answer3. Displaying these results did not require much changing

except needing to split this string and loop over the items and replacing the ”answer1” fields with the actual answer given in the database, see appendix C for the code.

2.12.5 Review and retrospective

2.12.5.1 Review

No story development this iteration

2.12.5.2 Retrospective

The bug fixing resulted in the major features of the system being ready for testing with users. However, there was some time wasted on trying to fix the image CSS when the solution was just to use a standard bootstrap container.

2.13 Iteration 10 27/04 - 03/05

Note: This iteration is slightly longer as it includes the half week between where iteration 10 would have ended and the hand in.

2.13.1 Story: Admins can then save the results as csv or xml

2.13.1.1 Analysis - breakdown of tasks

The most convenient way to save these results would be to convert the data to a csv format, as that can be read by many Spreadsheet programs. After some researching, an extension for Laravel was found that could easily create spreadsheets within a Laravel application [31].

There the list of tasks for this story:

- Add route, button and an action for this
- Install Laravel spreadsheet converter extension
- Pass the formatted answer data into the csv creator

2.13.1.2 Design

Due to the way the original answer handling was designed, results are only saved for a session and not for quizzes. This means when a new quiz is run, the results are wiped. This means a good place for the download action would be the quiz admin panel, the one containing the next and previous buttons. This would be an ideal place as it means lecturers can press download before they end the quiz.

The action itself does not seem to fit onto any of the other controllers, and a new one should be created to handle this function. A SessionController would be the best fit as this action falls under a session and not quizzes so would not be sensible in the QuizController.

The CSV format would be in the form of a single sheet with many rows. The first row would be the question, the second row the answers and the third the number of times the answer was chosen. These three rows would then repeat for each question, so a three question quiz would contain nine rows.

2.13.1.3 Implementation

Setting up the new route, button and controller was a quick task. A new controller was generated, a new route added to the web routing file and a new button added to the admin panel to call this route.

The extension was downloaded and installed and it was then added as a Facade [32], which allow the quick use and access to the functions they contain. The extension took an array of data and turned that into a spreadsheet of the type specified, in this case a CSV. The array is looped over and each item in the array is a row in the sheet. If an item of the array is an arrays itself, then

Figure 2.14: Example of a csv generated

	A	B	C	D
1	What is 1 + 1?			
2	3	2	4	
3	2	2	1	
4	What operating system do you use?			
5	Windows, Mac	Linux, Windows, Mac	Linux	Mac
6	1	2	1	1
7	Did you enjoy this quiz?			
8	FALSE	TRUE		
9	2	3		

each item within these sub arrays would be a cell within the row. This means the sheet is built from an array of arrays.

Formatting the arrays was the primary task here. First all the questions from the session were selected, and then looped over. Within this loop the question was added as the first row. Then all the answers associated with that question needed to be added. Getting them from the database used an Eloquent database query. Two arrays needed to be created from this data, an array of the answer text and an array of the number of times these options were submitted. The total times these were submitted had not been calculated so this was done first, looping over the answers and counting the times an answer was given by creating a new array of the answers and times clicked in a key=¿value pairing. This array of answer=¿answeredNumberTimes could then be looped over and the each key added to the first array, and the values to the second array for the two rows.

The problem here was that the answer text in the database is stored as ”answer1” rather than what the student actually clicked. This meant that these answers had to be changed, this was accomplished by using the answer value, like ”answer1”, as the key in the current question variable and adding this value to a new ”keys” array. Another problem was the way multi selection questions were stored, such as ”answer1, answer2”. This meant adding another extra step to break the string into an array and replacing them with the actual answer values and recombining them into a single string, see appendix C for the function.

See figure 2.14 for an example of a generated CSV.

2.13.1.4 Testing

Not possible to test with Dusk as it is downloads a separate CSV file, but in future a normal PHPUnit test could be written to inspect the array produced and compare it to what is expected.

2.13.2 Extra implementation based on feedback from admin user tester

An admin user tested the application whilst using the system to build a quiz that would be used in the user testing. During this, some feedback was received about the application which could be worked on to improve it in small ways.

2.13.2.1 Quiz control buttons should be greyed out

The next and previous buttons on the quiz were changed so that they should be greyed out if the quiz was at the start or end of the quiz. To disable them the ”disabled” HTML attribute was added to the buttons. The position and total number of items in the quiz were passed to the page from the controller and could be used to determine whether or not the buttons should be disabled. For these to be disabled when the page loads, some embedded PHP was used to render the buttons with disabled if needed. When the quiz was running, JavaScript was used to add and remove the disabled attribute whenever next or previous was pressed.

2.13.2.2 Question creation true/ false changes

Some JavaScript was used to change the question creation and question edit pages. It was needed for true/ false questions which should be submitted with answer1 as ’True’, and answer2 as ’False’, with the other fields left blank. The other fields would be ignored when the question was rendered, but allowing them to be filled in could be confusing. So JavaScript was used to hide the other fields, fill the first two with ’True’ and ’False’, and disable the fields so they could not be edited. This was actually part of the original design for the question creator but was left due to more pressing tasks such as the running of quizzes.

2.13.2.3 Cancel button on question edit pages

Finally, there was a request for a cancel button on edit pages. This was because pressing the back button would work but having a button would probably be better for the users. This button just called a *back()* function.

2.13.3 Bugs

Various other bugs were reported during this testing. One was a bug concerning the upload time of slides, which took long enough for the PHP execution time to be exceeded. Whilst normally this would require a change to the php.ini file, this was not easily accessible on the server, so a call to the php function, set_time_limit() was made to increase this past the default for the function in question. This was a better solution due to it only increasing the limit for the one function, whereas changing the times in the ini file would change the whole system and possibly lead to unintentional side effects.

Another small bug was a ”LogicException” found by going to an incorrect url, which is not the standard 404 thrown by an incorrect url. This was caused by a route which had no defined action in the router, something left over from an earlier iteration by mistake.

2.13.4 User testing

Used the system in a first year workshop to gather some module feedback for the lecturer and to test the system with real users.

2.13.5 Report writing

A lot of report work in this final iteration, including the restructuring of the iteration and design chapters, writing the design, evaluation, appendices, and generally editing the whole document based on feedback from proof readers.

2.13.6 Review and retrospective

2.13.6.1 Review

Just the one story concerning the downloading of slides done this week. The story was given a difficulty of 2, which is fairly accurate, even at the end of development. The use of a library made it easier, but there were still some minor issues as described above. The rest of the work this iteration also went well, as it was the last iteration and had little more than small bug fixes or corrections and report writing.

2.13.6.2 Retrospective

Everything seemed to go well, especially used testing which was a concern.

Chapter 3

Final design

This chapter will give an overview of the final design of the system reached by iteration 10. It will not go into detail for individual parts of the system as these were covered in the Iteration chapter but instead will offer a top level perspective of the system.

3.1 Overall architecture

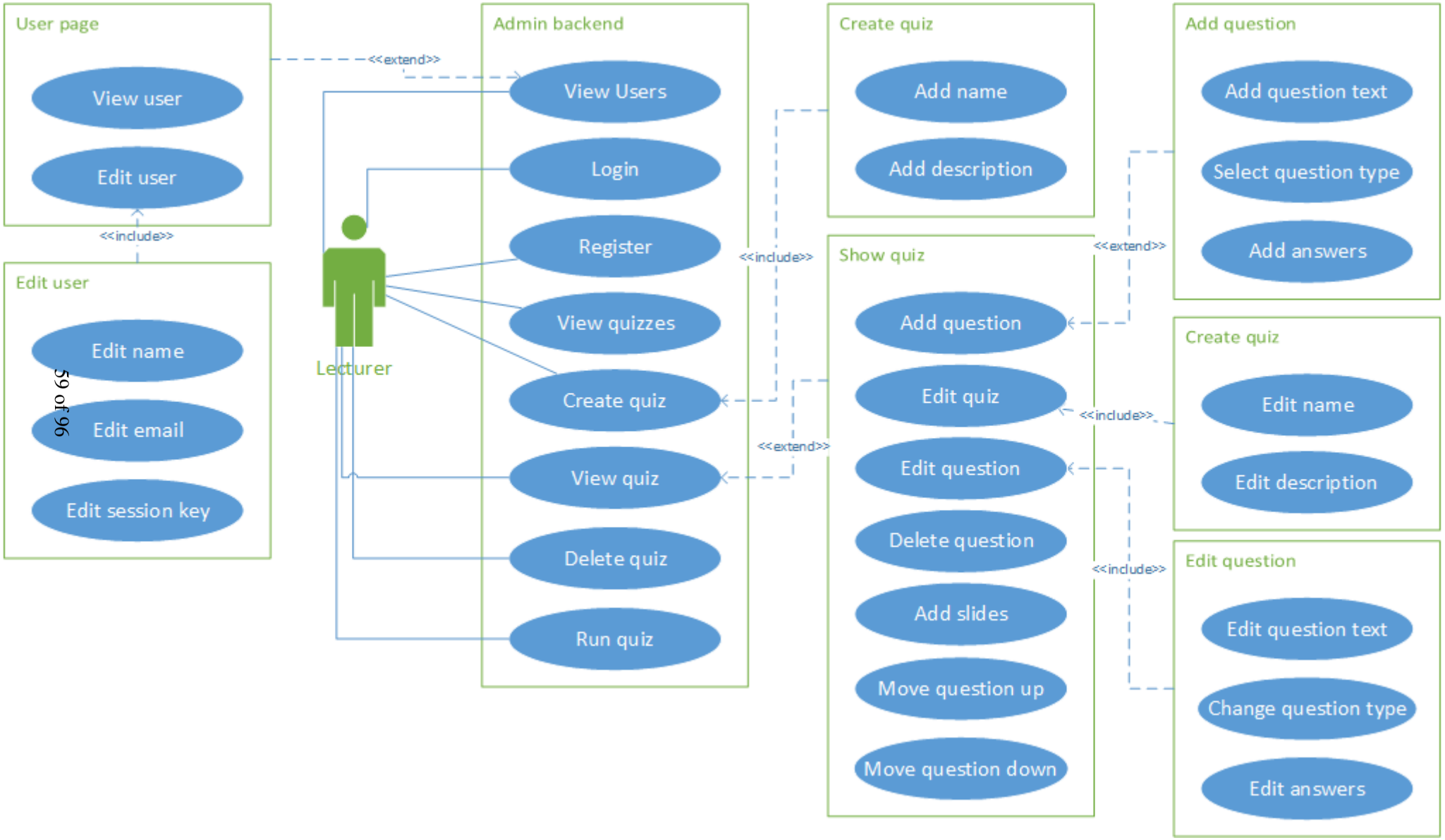
The Laravel framework uses a Model-View-Controller design pattern [33] to build applications. It works in the same way as a normal MVC application. Data is stored and manipulated within the models, controllers handle most of the interactions the user has with the system, and the views are the web pages presented to the users. Within Laravel, there is also a front controller, which is used to route incoming HTTP requests to the appropriate views and controllers [34].

There are two sections of the system, the student end, and the lecturer end. The lecturer end is an admin panel that they can login to, to create and manage their quizzes. The student end consists of the portal for connecting to a session, and the session pages themselves that display the slides and questions for users to answer.

Students only have two bits of functionality available to them, being able to join a quiz and then being able to answer questions. Lecturers however have more options on the site. On the admin backend they can create and edit new quizzes and questions, and add slides to the quizzes. They can also change their user details including their session key. The main operation they can perform is that they can run a quiz, this then takes them to the same view as the students, albeit with a small control panel for changing questions or slides, displaying the results of the quiz in real time and downloading the results. See 3.1 and 3.2 and 3.3 for the two use case diagrams and the overall class design.

3.2 Database design

Figure 3.1: Use case for backend of system



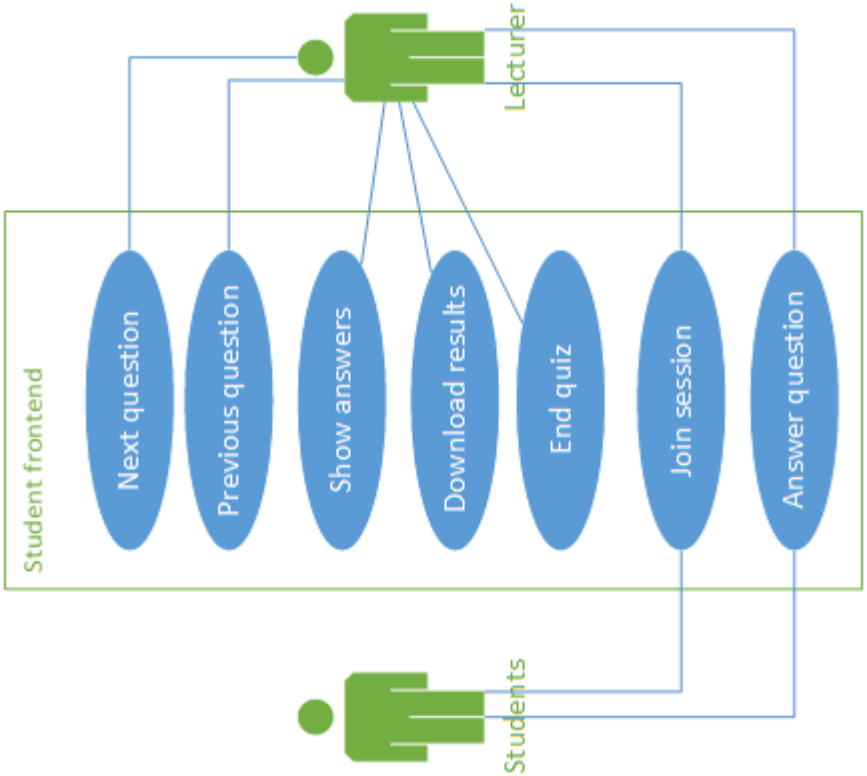
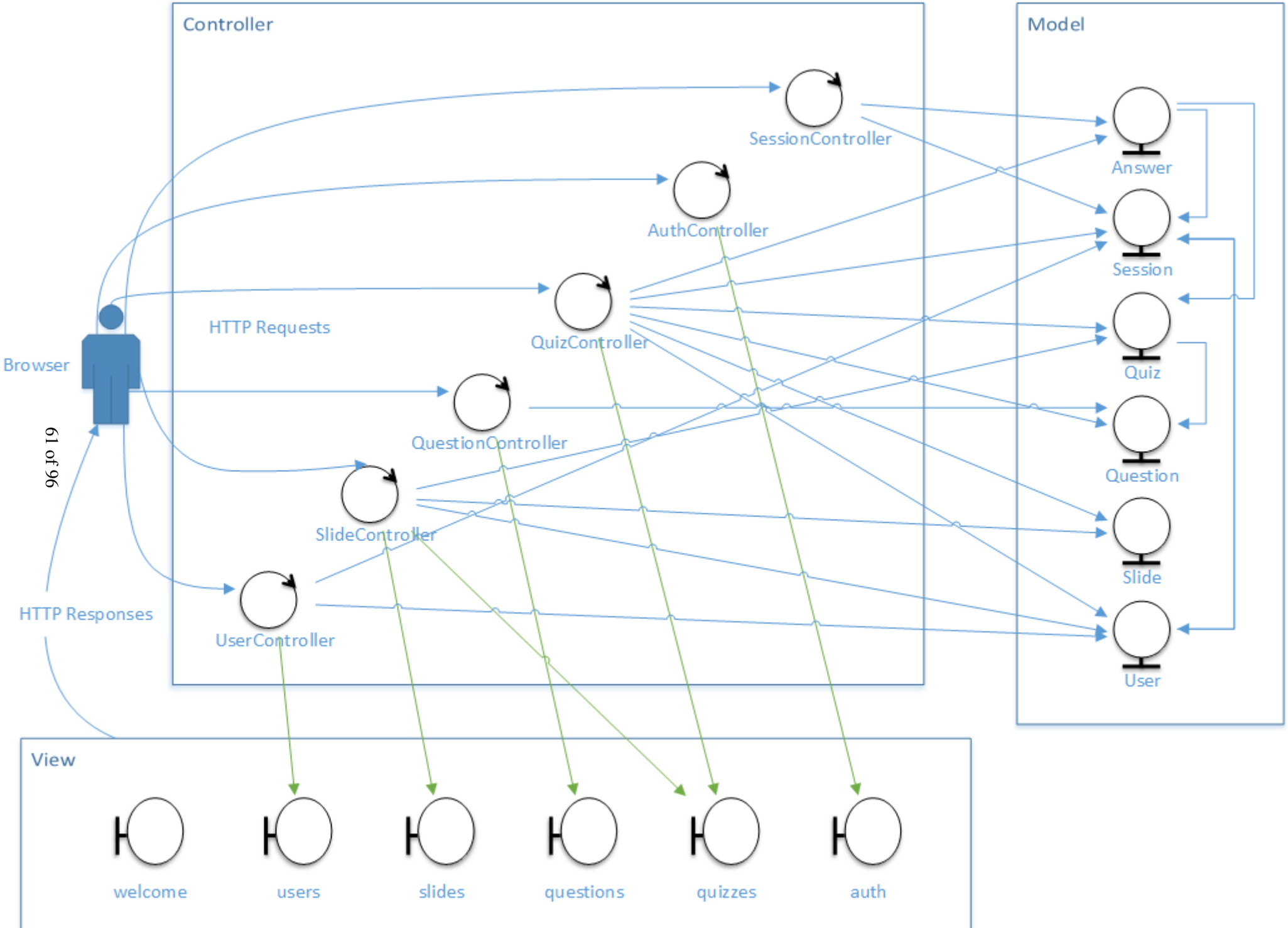


Figure 3.2: Use case for frontend of system

Figure 3.3: Class design for final system



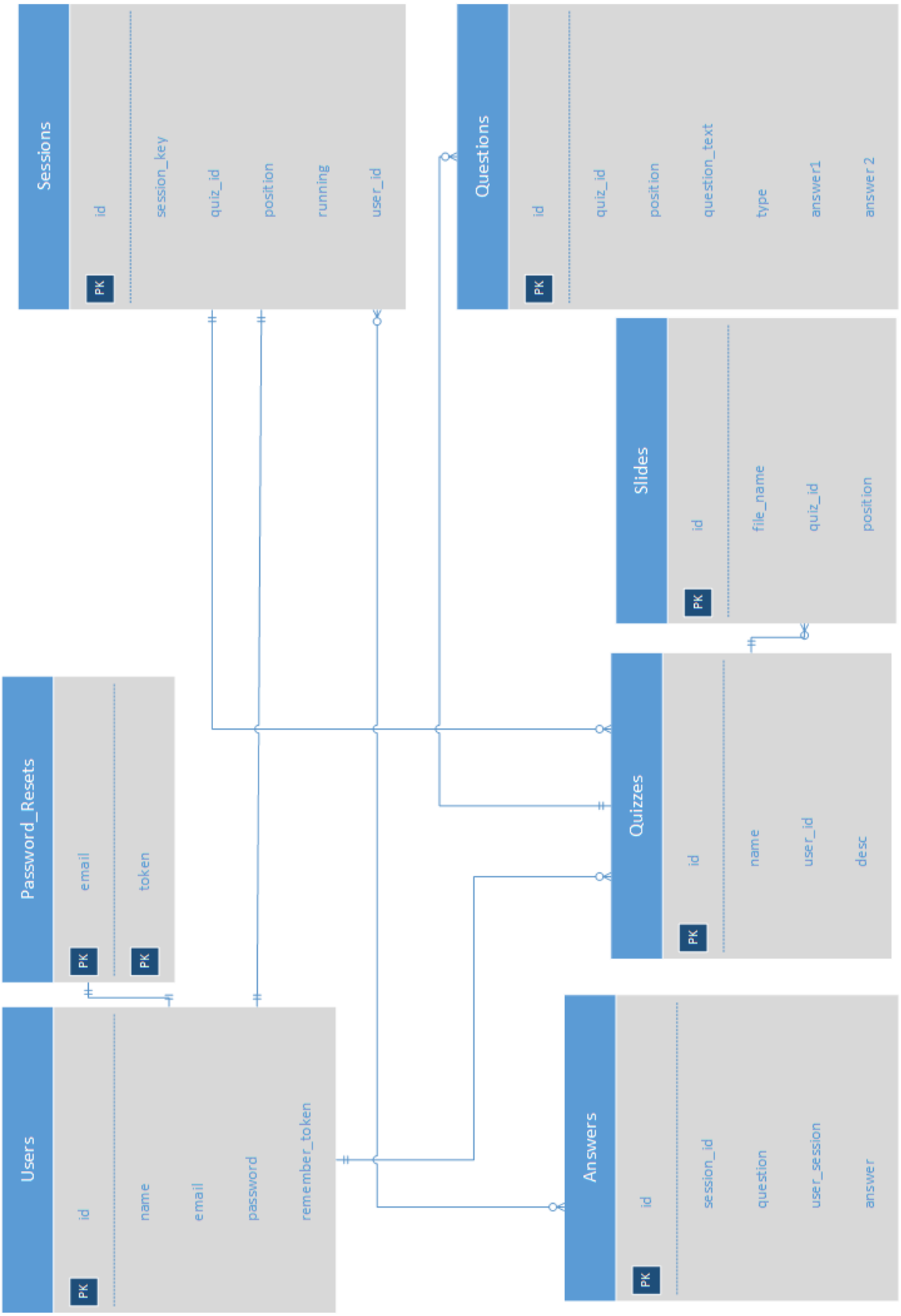


Figure 3.4: Entity relationship digram for final database

There are seven tables within the application. Two of these were generated by running a Laravel command to make the authentication part of the site, the users and password_resets tables. These provide basic user login. the other five, answers, questions, quizzes, sessions and slides were made manually.

The quizzes table contains information about the quizzes themselves, and are associated with a user. The questions and slides tables both store information about their respective parts of a quiz, with each row within these tables associated with a quiz. Questions store information about the actual question including all the answers. Not present in the design for the answers table above are the other eight fields for answers up to answer10. The slides table stores the file name of a slide image that has been converted from a pdf slide. Both of these tables store the positions of their items within a quiz.

The sessions table stores the information about the runnable session, each user has one associated session row. Within this row, the session_key is stored, which is the key that students would use to connect to a session. Additionally it stores information about the session when it is running, specifying if it is running, and what position it is at. This position references the positions specified in the questions and slides tables, though there is no actual foreign key relationship between them.

The final table, answers, stores all the responses from users to questions. It stores which question, the answer given and the user that submitted the answer. The user_session is a cookie value rather than a user from the database.

Chapter 4

Testing

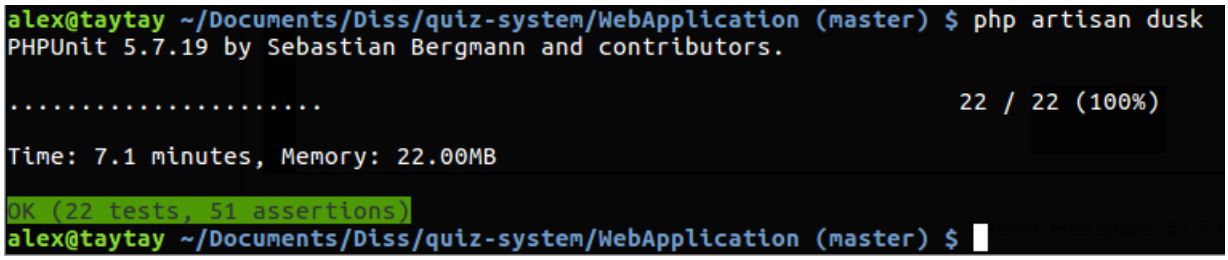
4.1 Overview

At the start of the project, testing was to use two practices from Agile. These were Test Driven Development (TDD) and Continuous Integration (CI). 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 testing. Overall there were 22 tests that tested the overall functionality of the system, see figure 4.1. Unit tests were dropped for a number of reasons in favour of application tests.

Unit testing suits a situation where more of the base code is written by the developer, in this case the majority of the code is built on a pretested framework. Testing the final implementation would fit better than trying to test if a view is rendered by a controller correctly. Application tests can also be used to test the stories, meaning that they can give a much clearer indication of whether a story is complete and whether the end user will be happy.

The time for extensive testing was also limited, meaning some tests had to be cut. The unit tests were the easiest to cut and seeing as the application tests tested the overall functionality of the system the lack of unit tests is not a terrible missing feature.

Figure 4.1: Output of the tests



4.2 Application testing

To perform application tests, a testing framework called Laravel Dusk was used, which is the default testing framework provided with Laravel 5.4 [35] [36]. 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 on the 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, 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 database 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 database could be migrated before any tests were run to allow the use of transactions, this would mean that any future changes to the migration files would mean that the developer would have to remember to re-migrate 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.

4.2.1 Limitations of dusk

There were various problems encountered with Dusk throughout the project. These are likely due to the age of Dusk having only been released in January 2017. A major difficulty was that Dusk could not be run on Travis, the CI tool chosen for this project, which meant that CI had to be abandoned. The reasons for this are not understood, but it was better to run the tests manually than to spend a large amount of time trying to fix it, if at all possible.

As for the actual tests, there were a number of problems with the asserts and navigating pages that slowed how quickly tests could be written. The selectors in particular did not seem to all have the same functionality; some of them can take a HTML selector, similar to JQuery, where an HTML object could be selected by id or class. However, some could only be found by the name attribute and others by the content of the HTML tag and nothing else. The problems with lack of consistency was that a lot of assumptions were made about the way selectors worked, and the errors produced tended to simply say that the item being looked for was not found. This lead to a lot of frustration and confusion.

Other problems with selectors included that they could not see inside iframes or canvas tags, a

major part of showing results of quizzes. And if a selector did find a collection, it returned the first item on the page, and not a collection of items that can be iterated over. This problem in particular meant more work adding various unique ids or classes to items that were repeated on the page so that they could be selected without any issues.

There was additionally the issue that due to the amount of initial database seeding used within each test, the actual tests could get quite long in terms of line numbers. There was a lot of code repetition and this is something that could be improved upon in future by refactoring out some of the functionality into initialiser 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 Qwizdom. 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 (XSS) and Cross-Site Request Forgery (CSRF) attacks. SQL injection attacks involves attackers trying to execute their own SQL code on the server, thereby giving them access to the system or to try and damage the system by say dropping some tables. XSS attacks are when attackers attempt to inject their own JavaScript into the system that may be run on another users page. Typically this would include injecting a script tag with malicious code into the database that would then be executed when written to the page by another user viewing a page. These tests are located in SecurityTest.php. There are five tests that test both SQL Injection and XSS attacks.

For the SQL injection tests, a string that tries to insert a new database entry is used:

```
"INSERT INTO `quizzes`
(`name`, `desc`, `user_id`, `created_at`, `updated_at`)
VALUES ('SQL Injection', 'this is an attack', '1', now(), now());"
```

And for the XSS attacks it tries to add a script tag with an alert:

```
<script>alert('hey xss');</script>
```

There are two main places that these attacks could be carried out. The first is on the front page, where the session search field could be used to insert malicious data. Though nothing is saved from this input, it does query the database, which means it could be subject to an SQL injection attack. If the search returned the original search query, then it could be used to embed some JavaScript, and if the search was then linkable (i.e. in the URL parameters), it could be sent on to other users. The other place that these two attacks could be carried out are on the admin backend, in the quiz and question creation pages where SQL or XSS code could be added to the body of the various quiz and question fields.

For SQL injection attacks, the easiest way to check that this data has not been added is to look for the record in the quizzes table. With the XSS attack, the page source can be searched for the

script tag. Laravel copes with these attacks by default by escaping the inputs [37]. With SQL, the entire string is quoted and not seen as an actual SQL command. With the XSS the HTML tags are escaped in a similar fashion, the <and >symbols are turned into < and > which are not rendered as HTML when written to the page [37].

The tests prove that the system works as intended and these two attack types are not possible. Cross site request forgery attacks are another form of attack but harder to test as they rely on the browsers weaknesses and come from another website. Dusk cannot simulate this sort of environment. However, for forms to be submitted within the system a CSRF token has to be submitted with the form. This token is compared to the user session to ensure they are the ones submitting this data, not a third party site. Laravel forces forms to contain this token, ensuring this attack cannot take place.

4.4 User testing

At the end of development, the system was tested in a first year workshop, where they were asked to give feedback about the module. Feedback asked about their experience answering questions, viewing the graphs on the lecturers screen and the general user interface. The quiz was made by the lecturer beforehand rather than the developer, allowing some feedback from both a lecturer and the students. Feedback from students was split between those using mobile devices and desktops, with a larger number using desktops and two responses for mobile. The feedback from desktops was very positive, with all the responses rating four or five out of five regarding their experience. Although, there were comments that indicated a problem was lack of feedback when an answer was submitted, which would be an easy task for any further work.

As for mobile, the lack of responses make it difficult to tell what users thought. However the two responses received were split on their experiences. The first response indicated a somewhat positive experience, with a final rating of three out of five. The other response was much more negative, with a final rating of one. In the feedback about any bugs encountered it was stated that the quiz boxes were out of position and the wrong sizes. They also stated that they were using the default Samsung browser, which is not a browser that was tested on beforehand. Without knowing what the positive response was using, it is hard to tell if this is restricted to the Samsung browser. Mobile testing did use Chrome for Android which may be why this was missed as it appears like they behave differently.

Both mobile users also stated that the application did not update within a reasonable time, however this was not observed during development. This could be due to several reasons including bad Wifi signal in the lecture room or the devices themselves.

The lecturer for the session also gave some feedback based on their experience of using both the quiz creator and running the quiz in the workshop. The results indicated a generally positive experience, with all but one question receiving four or five out of ten. Just one question received a three out of five, which was how easy it was to run the quiz from the backend. There were also suggested additions and extensions. They stated their satisfaction by saying ”Overall, very nice system that I think I could use.”

For a more detailed breakdown of results, see appendix D.

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 give an idea of the stress the system could take. It was able to support about twenty students connecting and answering questions without any issues. While the system is supposed to be up to 300, this is the best indicator at this point, and there were no problems.

4.6 Automated testing

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.

Chapter 5

Evaluation

5.1 Story/ requirements comparison

For the final list of stories see appendix 5.2. When looking at the final application and the final stories, they can all be argued as being completed with the exception of stories 7, 9 and 12.

Concerning story 7, whilst technically the application only supports up to a hundred users simultaneously, this is not throttled by the application. But instead is due to the WebSocket provider Pusher, which for free use is limited to one hundred users. If the system were to go live, then this would be replaced by using an in house Redis server, or by paying Pusher for more slots. These would be able to meet this upper capacity and therefore meet the story. For story 9, a percentage of answers was decided to be too hard to do without relying too much on the WebSocket server being used, which as mentioned above could be changed. Therefore a simpler solution was added, just giving the number of answers submitted. This number would give a rough estimate of the percentage, as if the lecturer knew the class size, they could compare it to the number answered. It was important to not rely on Pusher and instead offer an alternative solution that did not quite meet the story.

As for the final story that was not completed, 12, the downloading of stories was done in the last iteration. It was decided that the reuploading of these to view on the application would have taken too long given how much other work was yet to be completed. Additionally, due to the download being a csv rather than the other potential format (XML), if the lecturer wished to view the results in a more graphical view, their preferred spreadsheet can easily be used to create graphs of the data. Whilst this is more work for the lecturer, it seemed like a minor issue and one worth causing if it meant more time for writing the report.

There were significant differences between the initial list of stories, 5.1, and the final list. A couple of new ones were added, such as number 14 and 15 on the final list. However, all the stories for part two were changed. The reasons for these changes have already been explained but it is obviously different from the initial set of stories. However, the redesign and production of new stories fills the same functionality as originally specified and therefore it can be argued that the original specifications are still met.

5.2 Methodology

XP's main strength is its adaptability and the first thing to do was adapt its practices to fit the needs of the project. This was very beneficial as some practices were not relevant and would have not worked or even hindered the development. As development progressed, some of the practices had to be further adapted, such as dropping test driven development, allowing even more flexibility to the way I worked.

I particularly liked working to stories and in iterations. The iterations allowed small bits of working functionality to be written on a weekly basis. They were particularly useful in that they enforced small pieces of design, implementation and testing each week, meaning there was a good mix of things to do. But most importantly, the iterations helped adapt to change. The best example is the part two redesign, which was originally planned to include a lot of work, two or three iterations worth, but instead came down to less than a full iteration of work. The iteration also included the creation of a review document at the end. This was used to evaluate the work done and the methodology itself, such as deciding that TDD should be dropped.

There were some problems though, the design and testing sections sometimes suffered within the iterations. This was because I was more interested in getting the implementation done. Its not to say that design and testing didn't happen when they were supposed to, its just that occasionally it had to be done in the following iteration.

Another major problem is that due to a lack of overall design at the start, as encouraged by the use of XP, there was some redesigning and reimplementation happening throughout due to some bad or short sighted design choices. A good example would be when the position was added to the questions, which whilst quite an important part of the implementation, was not in the original designs at all. Storing this position in the question and session tables and using those both to find the right question could have instead used a separate table to store the position and a reference to the relevant question. This would have put less work on the database and required less coding. Another example of where upfront design might have worked is that the QuizController is quite large and some of its functionality could be split into sub controllers or split better among existing ones.

It is problems like these that could have been avoided if the system had been designed from the ground up. An upfront design might also fix what sometimes feels like lots of small sections of functionality tied together into something that feels more cohesive and robust. On the flip side though, it might have caused there to be too much work, for example without the iterations, part two might have been designed in a way similar to the original proposals which appeared to be a lot of work.

Whilst not really a part of the methodology, the time spent refactoring and documenting the code really helped. As mentioned above, some stories built on top of old ones, and some of the implementations changed a lot. Without the time spent documenting and refactoring the functions the code base would have been impossible to navigate and modify. Just writing the PHPDoc for function calls helped tremendously when doing any future work.

5.3 Tools and technologies

A number of tools and technologies were used during development and overall I am happy with the majority. Using PHP and Laravel seemed to fit the project very well as there were no features that were impossible to do in the language or framework. Not all parts of the framework were great however; the testing framework, Dusk, was particularly troublesome. This was mainly due to Dusk being new, and the problems slowed progress. There was no suitable alternative for application tests however, so Dusk was probably still the best choice. And over time, the framework will hopefully iron out its problems in time for any future work.

The JavaScript side of things could probably have been handled better because I never got all the JS into a single minified file as you are supposed to with Laravel. This meant that it was a little disorganised and sometimes hard to navigate and change things. It probably would have been better to spend some more time figuring out how to get it all into JS files rather than as script tags, however I deemed it not a high priority and to better focus on getting the system working.

On reflection I would not change the use of WebSockets. They were easy to set up, did exactly what was needed without the need for any unorthodox behaviour, such as forcing a page refresh, and are relatively future proof due to being a new standard. Most of the tools I used I had no problem with and would also use without hesitation a second time. Git was very important and whilst I did not need to use it for much version control, it gave me the option should it ever arise. My diary and Trello however were very important in day to day development. They were used to keep track of all progress in the project and also to try and plan some of the upcoming work. Without them, the project would have been much tougher and I am happy with the way I used them.

Other tools I am happy with include the Chrome developer tools which I used for debugging a lot and my editor, Vim. A program that could potentially have replaced both of these is an editor called PhpStorm [38]. As an editor it filled Vims role but it also provides a number of other features like a debugger. Debuggers for PHP are quite unusual as the code is executed by the server and not locally meaning debugging is usually manual with a lot of ”die()” commands being used to figure out what is wrong. This is clearly not the best as a good debugger can speed up finding problems very easily. PhpStorm also comes with a live preview feature, this can be debugged and edited in much the same way the Chrome dev tools are used. I did not however use it because I thought that overall the lack of a debugger and other features did not slow me down enough to warrant learning a new piece of software.

5.4 User testing evaluation

As discussed in the Testing chapter, the results were overall positive, with the majority of users and the lecturer having a good experience with the system. There were a number of issues and possible improvements raised, though none of them need rectifying immediately. I am a little disappointed in not having much feedback on the mobile side of things. Hopefully the one user with an issue was an outlier and from my experience of the system on phones when developing it, it was working. Before the system were to go live, more testing should be conducted with mobile devices.

5.5 Things to do differently and future work

There are a number of design choices that could have been done differently in retrospect. I think the main one is a different way to track the positions of slides and questions. It was designed initially without much thought as to how it was going to work further down the line, leading to a conflicting design. As mentioned before, using a reference table for keeping track of positions may have been a good choice for this.

The other main thing that could have changed was the use of testing. When TDD failed, a possible replacement that I did not look into at the time was Behaviour Driven Development [39]. BDD is generally used to explain the behaviour of the application in regards to the story, and then used to evaluate the outcome based on the expected outcome. Whilst BDD and TDD could have been used together, BDD can itself be used alone. It also would have suited the development process followed in this project, as the stories were worked on in a roughly one by one basis and the Dusk tests are a form of application tests. The way that BDD would evaluate the story would involve application tests. It would have fit in quite nicely but it never got suggested and might have taken some times to set up.

The last major change would be to the way a lot of the models are used, as almost all model functionality is static. The main reason for this is that the majority of functions in the models update the database, retrieve small bits of information from the tables or only needed to be called once in the process so there was no point in instantiating so many objects. There is object orientation in the controllers however there could have been a lot more within the models. A lot of the position changing functionality could work very well if done as an object parameter and then saved at the end. Currently the position variable is retrieved from the database and passed around a lot rather than used as an object parameter. Another reason a lot of the later potential non static functions were building on the already written static functions so it was easier just to add more static functions than refactor the whole system. From my perspective I would have preferred more object orientated programming in the application however it was far easier to just keep writing the static functions.

As for future work there are a number of potential tasks. The most pressing would undoubtedly be moving the WebSocket host from Pusher to Redis or something similar. This would be an important first task as it raises the limit on number of users without costing anything.

A highly requested feature in the user testing was some form of feedback when an answer was submitted, this should be quite an easy addition to the system and should be high priority if the system were to be used in the future.

An extension that may be quite important for the system would be to integrate the university login system, which I believe uses Lightweight Directory Access Protocol, LDAP. This feature was suggested in the lecturers feedback from user testing. This addition would mean lecturers would have accounts linked to their university accounts meaning easier login and authentication. In general the user authorisation could be improved substantially. All lecturers can see all the other lecturers quizzes and questions, and do anything to them. Limiting their actions was not a part of the original set of stories and therefore the authentication and authorisation is very basic. More authorisation should realistically be added to the system.

There are also a number of smaller things that could be added with little development time, such as adding more question types or adding some keyboard commands for the quiz controls allowing the lecturer to press enter for the next slide.

5.6 Summary

Overall, I am happy with the project, I think there are a lot of different ways it could have been implemented which might have been better now that successfully completed the project. There are also a number of potential improvements and extensions that could be developed but the final application meets the majority of its requirements and the user feedback was good.

Appendices

Appendix A

Third-Party Code and Libraries

1.0.1 Laravel

The PHP framework used as the basis for the application [2]. It is an MVC framework and comes with a number of tools such as Artisan [8] that are used to build parts of the application quickly and efficiently.

1.0.2 Laravel Dusk

The testing framework used for application testing the project [35]. It allows application tests, simulating what a user would do through the use of a Chrome browser.

1.0.3 Bootstrap

Bootstrap is popular HTML, CSS and JavaScript framework for building responsive, mobile-first websites and applications [13]. It was used for the majority of styling in the application.

1.0.4 PusherJS

Pusher is a third-party service that provides an easy way to implement WebSockets [21]. They provide a JavaScript library for connecting to channels and listening for incoming data which is a major component of the application.

1.0.5 JQuery

JQuery is a JS library that that simplifies much of the JS functionality, such as how to select parts of the DOM and interact with them [40]. It was used throughout the project as it was much quicker than writing the equivalent JS.

1.0.6 ChartJS

ChartJS is a library used to generate the graphs of data, it was used for rendering the results graphs shown by the lecturer [27].

Appendix B

Ethics Submission

AU Status
Undergraduate or PG Taught

Your aber.ac.uk email address
amt22@aber.ac.uk

Full Name
Alexander Michael Levi Taylor

Please enter the name of the person responsible for reviewing your assessment.
Reyer Zwiggelaar

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment
rrz@aber.ac.uk

Supervisor or Institute Director of Research Department
cs

Module code (Only enter if you have been asked to do so)
CS39440

Proposed Study Title
MMP - Classroom Quiz System - Providing a similar system to the Qwizdom system already used by lecturers, that allows lecturers to run interactive quizzes during lectures to enhance student participation and learning.

Proposed Start Date
30/01/2017

Proposed Completion Date
08/05/2017

Are you conducting a quantitative or qualitative research project?
Mixed Methods

Does your research require external ethical approval under the Health Research Authority?
No

Does your research involve animals?
No

Are you completing this form for your own research?
Yes

Does your research involve human participants?
Yes

Institute
IMPACS

Appendix C

Code Examples

3.1 Combining slides

Code for combing slides and questions into one array:

```
for ($i=1; $i<=$total; $i++) {
    foreach($questions as $question) {
        //Simple check to speed things up
        //No need to check it again
        if ($question->position < $i){
            continue;
        }
        if ($question->position == $i) {
            $combined[] = $question;
            continue;
        }
    }

    foreach($slides as $slide) {
        //Simple check to speed things up
        //No need to check it again
        if ($slide->position < $i){
            continue;
        }
        if ($slide->position == $i) {
            $combined[] = $slide;
            continue;
        }
    }
}
```

3.2 Multi user test

Test that showcases Dusks ability to use multiple browsers that are needed to test the WebSockets.

```
/**
 * Test two users joining different sessions
 */
public function testTwoUsersJoinDifferentSessions()
{
    $user1 = factory(User::class)->create();
    $user2 = factory(User::class)->create();
    $quiz1 = factory(Quiz::class)->create(['user_id' => $user1->id]);
    $quiz2 = factory(Quiz::class)->create(['user_id' => $user2->id]);
    factory(Session::class)->create([
        'user_id' => $user1->id,
        'quiz_id' => $quiz1->id,
        'running' => true,
    ]);
    factory(Session::class)->create([
        'user_id' => $user2->id,
        'quiz_id' => $quiz2->id,
        'running' => true,
    ]);

    $this->browse(function ($first, $second) use ($user1, $user2,
     $quiz1, $quiz2)
    {
        $first->visit('/quiz/' . $user1->session->session_key)
            ->assertSee($quiz1->name)
            ->assertDontSee($quiz2->name);

        $second->visit('/quiz/' . $user2->session->session_key)
            ->assertSee($quiz2->name)
            ->assertDontSee($quiz1->name);
    });
}
```

3.3 Multi selection saving

This is part of the function that saves answers to the database. It highlights the extra logic needed to deal with multiple selection answers:

```
//If its an array, its a multi select question
if (is_array($answer)) {
    foreach($answer as $a) {
        if (strlen($answerToUse) == 0) {
            //If its the first item, we dont want a comma
            $answerToUse = $a;
        } else {
            $answerToUse = $answerToUse . ', ' . $a;
        }
    }
} else {
    $answerToUse = $answer;
}
```

3.4 CSV downloader

A rather large function that creates the CSV of results:

```
        //loop over each question and add the relevant data to the csv
        foreach ($questions as $question) {
            //Add the title of each question
            $answersArray[] = [$question->question_text];

            //Get all the answers from the session
            $answers = Answer::where('question', $question->position)
                ->where('session_id', $session->id)->get();

            //Get the answers and how many times they were answered
            //Loop over all of them and add them to an array, if
            //the item already exists, increment
            $answerValues = [];
            foreach($answers as $answer) {
                if(array_key_exists($answer->answer, $answerValues)) {
                    $answerValues[$answer->answer]++;
                } else {
                    $answerValues[$answer->answer] = 1;
                }
            }

            //Now to add two arrays to the csv
            //The first is the keys, the answers themselves that
            //people click
            //The second is the number of times each was answered
            //This is so each array goes on a line in the csv
            $keys = [];
            $values = [];
            foreach($answerValues as $key=>$value) {
                //So if its a multi selection question,
                //need to change answer1, answer2
                //to the various answers, split the string
                //and loop over replacing with
                //the actual names
                if (strpos($key, ',')) {
                    $multiSelectKeys = explode(', ', $key);
                    $answerNames = '';
                    foreach($multiSelectKeys as $multiKey) {
                        $answerNames .= $question[$multiKey] . ', ';
                    }
                    $answerNames = rtrim($answerNames, ', ');
                    $keys[] = $answerNames;
                    $values[] = $value;
                } else {
```

```
                $keys[] = $question[$key];
                $values[] = $value;
            }
        }
        $answersArray[] = $keys;
        $answersArray[] = $values;
    }

    Excel::create($quizName . "-results", function($excel)
     use ($answersArray, $quizName) {
        // Set the spreadsheet title, creator, and description
        $excel->setTitle($quizName);

        // Build the spreadsheet, passing in the payments array
        $excel->sheet('sheet1', function($sheet) use ($answersArray) {
            $sheet->fromArray($answersArray, null, 'A1', false, false);
        });
    })->download('csv');
```

Appendix D

User survey results

Questionnaires were sent out to the students and lecturer via Google Forms, with the consent form as the description of the questionnaire.

For all the raw results or an example questionnaire, please contact the author.

4.1 Desktop

There were eleven questionnaires completed for desktop users. Overall the users responded positively. All users stated that questions and non question slides were easy to read, and only one stated that the answers were not easy to click on and submit. They all agreed that the pages updated within reasonable time when the lecturer changed question, which was less than five seconds. When asked to rate the UI of their view and the results view on the lecturers screen, all gave either four or five out of five. They also all rated the overall experience fours and fives.

4.2 Mobile

There were unfortunately only two respondents meaning there is no clear consensus. The two responses had very differing views, with one agreeing that the UI was easy to read and use whilst the other disagreed. They both agreed that the pages did not update within a reasonable time however, suggesting a potential issue with the WebSockets. This has not been observed in development and may potentially be because of the Wifi signal or the mobile devices themselves.

They gave a two and three out of five for the rating of the UI, with the person who had no issues with UI in previous question giving it a middling score of three. This would suggest some issues with the UI on mobile although with such a small sample this is hard to determine. Overall their final scores were three and one out of five.

The user who had a bad experience gave information that they were using a Samsung phone with the default browser, which had not been tested against during development. Testing against all phone variations during development would be extremely hard, and for this more user testing would be recommended.

4.3 Comments

A number of extra comments were left by the students, of which all the desktop responses stated that some form of feedback once they hit submit would have been nice.

The mobile users also left comments stating the same, but also that the size of buttons could be scaled down.

4.4 Lecturer results

The lecturer questionnaire covered more functionality as it included both the running of tests in the lecture and the creation of quizzes. The responses about the backend concerning quiz creation, adding slides and the UI were all positive, with either four or five out of five.

The responses concerning the running of quizzes were similarly positive with mostly four and five ratings for the UI, quiz controls, update times, and results. There was a rating of three given for the ease of running a quiz from the backend. Overall, they gave a four out of five for their experience.

There were a number of comments submitted in addition to the ratings, primarily about potential future work. This included adding authentication via university credentials, some feedback for when an answer is submitted as students requested and quiz cloning functionality.

They also left a very positive comment which backs up their final rating of four out of five: ”Overall, very nice system that I think I could use. It’s good that PDF slides can be interleaved with the quiz questions.”

Appendix E

User stories

A list of stories was produced that act as the functional requirements of the system. The stories have a difficulty ranking associated with each item in relation to the other stories. 1 being the easiest and 10 the hardest.

5.1 Initial user stories

Stories for the first part (only quizzes):

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

These stories fit the second part (streaming slides), though there are some stories from the first part that are applicable within this part:

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

5.2 Final user stories

Due to the redesign of part two, all the stories fit into the same part, and do not need to be split up. This final list was refined over the development period:

1. (6) ”Admins can create quizzes via the website”. This is then broken into these sub stories:
   (a) (3) Admins can log into a backend
   (b) (2) They are presented a list of their quizzes
   (c) (5) They can create a new quiz in the backend
   (d) (3) They can edit an existing quiz they own
2. (4) Quizzes contain a variety of questions
3. (7) Admins can login to the website and run a session with a quiz
4. (8) Multiple different sessions can be run simultaneously
5. (3) The Admin specifies which question is being run by clicking next/prev question etc
6. (5) Sessions can be joined by users via the website
7. (3) Up to 300 users should be able to join and answer questions
8. (1) Users answer the question being displayed by the quiz
9. (2) The Admin can see what percentage of users connected to the session have answered
10. (4) The Admin can show the results of the question in a sensible format e.g. graph
11. (2) Admin can then save results as CSV or XML
12. (2) The Admin can load from saved file to display again
13. (4) Users should not be able to submit their own answers by altering the HTML, as they did with Qwizom
14. (5) The site should be mobile responsive
15. (2) Admins should be able to change their session key
16. (1) The admin creates slides in their preferred editor and exports them as a PDF
17. (4) The admin can upload these slides to a quiz they have created in the past
18. (3) The admin can reorder the questions within the quiz to move them around the slides
19. (2) When this quiz is run, it should render the slides as well as questions in the order specified

Appendix F

Maintenance guide

6.1 Installing and running

See the readme.md file in the accompanying submission folder.

6.2 Useful commands

There are a number of artisan commands that are used to build the system and debug:

- php artisan migrate - migrates the database, use migrate:refresh to rollback database and migrate. Add –seed option at the end to seed using the seeders.
- php artisan make - used to create most components of the system, such as models and controllers.
- php artisan dusk - runs the dusk tests. Can also supply a file to run a subset of the tests
- php artisan route:list - lists all the routes for the application, a good debugging tool
- php artisan tinker - enters a custom console in which Eloquent commands can be executed to interact with the database. Might be easier to use this if you prefer an ORM to however you interact with the database. Example command: App\Session::all();

It is recommended that you visit the official doc pages for more information [12] or the Laracasts guides [11].

For building CSS or JavaScript:

- npm run dev - builds the JS and CSS in development mode, unminified.
- npm run production - builds the JS and CSS in production mode, minified into build files.
- npm run watch - continously builds the JS and CSS in development mode, meaning whenever a file is changed and saved, it is rebuilt saving having to run dev repeatedly.

6.3 Layout

The model files are located in \app. The controllers in \app\Http\Controllers. The views under \resources\views.

CSS and JS is in the \resouces\assets folder.

The views are built in Blade, a templating language that combines PHP and HTML. The views use a number of base templates locates under \layouts which are then extended by other view files. For example, the quiz.index page extends the layouts template, and adds the content with the blade @content tag.

The database files are located under \database. The migrations folder within this holds all the migrations that are used to create the database.

The \routes folder contains the web.php file which is the primary routing file used to determine all the routing in the application.

The \tests folder contains all the tests. Dusk tests are under Browser and Unit tests would be under Feature.

The \storage folder is used to store all the files that the system uses, namely the slides images and pdfs.

6.4 Major components

6.4.1 Broadcasting and running quizzes

Most of the functionality for running a quiz is located within the QuizController. It controls the running of quizzes and also their creation. The other controllers concern themselves with their namesakes on the whole with few exceptions.

The Event used to broadcast to the WebSockets is located under \app\Events. This Event is called whenever a quiz is started or next or prev is pressed. It broadcasts the relevant quiz information for the position in the quiz that it is moving to. This information is then received on the client end within the PusherJS code within the \rousources\views\quizzes\run\quiz.blade.php file.

Within this file, the JavaScript is used to update the content of the page by making an ajax call to the needed content and then replacing the pages content with this ajaxed content. Similarly the admin panel uses an ajax request to get the render the results and then renders them using a third party library.

6.4.1.1 Change WebSocket provider

A free alternative to Pusher is using Redis and Socket.IO servers. A guide for them can be found here [41].

6.4.2 Creating quizzes

The quiz creation is mostly handled within the Quiz and Question controllers which display the relevant views and call the creation and update functions in the respective models. Changing question position is within the Question model.

6.4.3 Slide uploading

Adding slides is handled inside the SlideController, which uploads them to the storage folder and then uses a third party library to convert the uploaded PDF to images before adding them to the slides database table.

Annotated Bibliography

[1] Qwizdom, Inc., "Experts in Audience Response Systems & Training — Qwizdom UK," http://qwizdom.com/uk/, 2017.

Qwizdom is the software that this project is primarily based off.

[2] T. Otwell, "Laravel - the php framework for web artisans," https://laravel.com/, 2017.

Main site for the Laravel framework, used as the primary framework in this project.

[3] B. Skvorc, "The best php framework for 2015: Sitepoint survey results," https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/, March 2015.

A comparison of the most popular PHP frameworks

[4] Microsoft, "Get an office add-in for powerpoint," https://support.office.com/en-gb/article/Get-an-Office-Add-in-for-PowerPoint-41e3e84e-17e7-4fa7-807a-d027046230bd?ui=en-US&rs=en-GB&ad=GB, 2017.

Microsoft guide on how to get and use add-ins for PowerPoint.

[5] B. Kent, *Extreme programming explained : embrace change*, 2nd ed. Boston, MA : Addison-Wesley, 2005.

Book on extreme programming, by one of the ideas originators, Kent Beck.

[6] "Trello," https://trello.com/home, 2017.

Homepage for Trello, the project management and feature tracking site used during development.

[7] "Github," https://github.com/, 2017.

Github, one of the leading Git repository hosting services available, and used in this project.

[8] "Artisan - laravel," https://laravel.com/docs/5.4/artisan, 2017.

Documentation describing Artisan, the custom CLI provided with Laravel. Its used to run just about everything Laravel related, from database migrations to making new controllers etc.

[9] "Flowchart maker and diagramming software - micrisoft visio," https://products.office.com/en-gb/visio/flowchart-software?tab=tabs-1, 2017.

Description of Visio, a tool for building diagrams that was used extensively in the design stages of the project.

[10] "Laravel recipes," http://laravel-recipes.com/, 2017.

Laravel Recipes is a good source for many Laravel functions.

[11] J. Way, "Laravel 5 from scratch," https://laracasts.com/series/laravel-from-scratch-2017, February 2017.

Laracasts

[12] "Laravel - the php framework for web artisans," https://laravel.com/docs/5.4/, 2017.

The documentation pages for Laravel

[13] "Bootstrap," https://getbootstrap.com/, 2017.

Bootstap is a CSS, HTML and JAvaScript framework that makes styling and building mobile responsive sites very easy. The basic Bootstrap CSS was used extensively throughout this project.

[14] "Validation - laravel," https://laravel.com/docs/5.4/validation, 2017.

Documentation for how server side validation works within Laravel

[15] "Csrf protection - laravel," https://laravel.com/docs/5.4/csrf/, 2017.

Documentation for how Laravel handles protection from cross site request forgery attacks

[16] gaddygab and georaldc, "[question] why dusk page test uses databasemigrations instead of databasetransactions? issue #110 laravel/dusk," https://github.com/laravel/dusk/issues/110, February 2017.

Describes why migrations have to be used rather than transactions, posted as an issue to the test frameworks Github page

[17] "Controllers - laravel," https://laravel.com/docs/5.4/controllers#resource-controllers, 2017.

A list of the routes automatically bound when a controller is specified as a resource controller.

[18] "Websockets," https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API, 2017.

Description of WebSockets, a technology for pushing information to browsers in real time.

[19] "Action cable overview - ruby on rails guides," http://edgeguides.rubyonrails.org/action_cable_overview.html, 2017.

Describes how RoR uses WebSockets. Good for a comparison of other languages and another framework.

[20] "Broadcasting - laravel," https://laravel.com/docs/5.4/broadcasting, 2017.

Documentation for broadcasting within Laravel, which is what it calls WebSockets

[21] "What is pusher? building real-time laravel apps with pusher," https://pusher-community.github.io/real-time-laravel/introduction/what-is-pusher.html, 2017.

Guide on what pusher is and how to set it up with Laravel, though this guide was not used for that purpose, merely to gain an understanding of how Pusher worked.

[22] "Writing realtime apps with laravel 5 and pusher - pusher blog," https://blog.pusher.com/writing-realtime-apps-with-laravel-5-and-pusher/, June 2016.

Guide that was used to get the basics of Pusher and how to do use WebSockets within Laravel

[23] "Introducing laravel echo: an in-depth walk through," https://mattstauffer.co/blog/introducing-laravel-echo, June 2016.

Guide that was used to get the basics of Laravel Echo and how to do use WebSockets within Laravel

[24] phuclh, "Cannot find module cross-env," https://github.com/JeffreyWay/laravel-mix/issues/478, February 2017.

Github issue about Laravel Mix screipts not working, including a fix

[25] Way, Jeffrey and calebeoliveria, "Best practices for custom helpers on laravel 5," https://laracasts.com/discuss/channels/general-discussion/best-practices-for-custom-helpers-on-laravel-5, May 2016.

Discussion of how global helper functions should be registered and used and used within Laravel

[26] "Middleware - laravel," https://laravel.com/docs/5.4/middleware, 2017.

Description of Middlware in Laravel, a way of filtering incoming HTTP requests to a controller. An example would be like checking if a user is logged in.

[27] "Chartjs — open source html5 charts for your website," http://www.chartjs.org/, 2017.

Library used for rendering the results as a bar chart.

[28] "Media queries - css . bootstrap," https://getbootstrap.com/css/#grid-media-queries, 2017.

Bootstraps recommended media query sizes for phones, tablets and desktops.

[29] "What's new in laravel 5.3: Super simple file uploading," https://laracasts.com/series/whats-new-in-laravel-5-3/episodes/12, August 2016.

Video tutorial on how to upload and save files in Laravel.

[30] Spatie, "spatie pdf-to-image: Convert a pdf to an image," https://github.com/spatie/pdf-to-image, 2017.

A package for converting pdfs to images, used in the slide conversion part of the system.

[31] Maatwebsite Team, "Facades - laravel," http://www.maatwebsite.nl/laravel-excel/docs, 2017.

Library used for the creation of CSV files that the lecturers can download.

[32] "Laravel excel," https://laravel.com/docs/5.4/facades, 2017.

Documentation on facades in Laravel.

[33] E. Freeman, E. Robson, K. Sierra, and B. Bates, *Head First design patterns*. Sebastopol, CA : O'Reilly, 2004, p. 51.

Description of the Model-View-Controller design pattern in more depth

[34] "Architecture of laravel applications - laravel book," http://laravelbook.com/laravel-architecture/, 2017.

Describes the architecture of the Laravel framework in more detail

[35] "Browser tests (laravel dusk) - laravel," https://laravel.com/docs/5.4/dusk, 2017.

Laravel Dusk documentation page

[36] M. Stauffer, "Introducing laravel dusk," https://mattstauffer.co/blog/introducing-laravel-dusk-new-in-laravel-5-4, February 2017.

A comprehensive introduction to Laravel Dusk which gives a better introduction than the documentation

[37] W. Gilmore, "How laravel 5 prevents sql injection, cross-site request forgery, and cross-site scripting," http://www.easylaravelbook.com/blog/2015/07/22/how-laravel-5-prevents-sql-injection-cross-site-request-forgery-and-cross-site-scripting/, July 2015.

Description of how Laravel copes with various common attacks used on the web, focussing on SQL injection, XSS and CSRF.

[38] "Phpstorm ide," https://www.jetbrains.com/phpstorm/, 2017.

PHPStorm is a dedicated PHP editor with a lot of built in functionality to help development, including a debugger and native support for major frameworks like Laravel.

[39] Agile Alliance, "Behavior driven development (bdd)," https://www.agilealliance.org/glossary/bdd/, 2017.

Behaviour Driven Development information from the Agile Alliance. Desribes what it is and how it is beneficial.

[40] "Jquery - write less, do more," https://jquery.com/, 2017.

JQuery is a JavaScript library used throughout this project to speed up development.

[41] J. Way, "Real-time laravel with socket.io," https://laracasts.com/series/real-time-laravel-with-socket-io, 2017.

Guide for setting up WebSockets with a Redis server and Socket.IO, a free alternative to Psuher as it self hosted.