Ulster University Attendance Management System

COM547 Final Report

Student: Neil Browning

Student Number: B00652112

Course: Computing Systems PT

Date: 2017/2018

Supervisor: Professor Colin Turner

Neil Browning

B00652112

# Abstract

The goal of this project was to create a tool for managing student attendance and module feedback for students and staff of Ulster University: the uploading of such data; viewing the information in meaningful formats; and reporting back to users via email. The aim was to replace an existing manual-effort approach in terms of student attendance and provide new functionality for students to provide feedback at any time for their enrolled modules.

In the existing system at Ulster University, student attendance would be collected by lecturers per session that they hold with students and entered into a system that outputs a spreadsheet – a flat file with limited capability for searching, analysing or reporting on the data. This manual process of managing attendance data was limiting, and a new system for processing audience data was proposed which would start with student attendance, with the ability to expand it for other opportunities as well – another facet of audience data that this project addressed was feedback from students. The feedback cycle of the existing system was too long, and a faster turnaround would allow for both staff and students to benefit.

The project followed an agile software development lifecycle, with development and testing of the system being performed between periodic demos to the project supervisor and key stakeholder as the system grew.

The final system met all of the requirements expected of it and received good feedback for its functionality and design. As an extensive proof of concept for an additional university component, it demonstrated the potential usefulness of such functionality that the university could add to its arsenal to better help management of attendance and student feedback.

# Acknowledgements

To start, I’d like to thank Professor Colin Turner for his time and supervision during the year. His insight and familiarity with the technologies involved, plus his willingness to invest time and thought into the project helped tremendously and his feedback was invaluable.

Thanks to Dr Kenneth Adamson for his time at the beginning of final year, lending some time every week for any queries that we had, organising the supervisor assignments, and supplying general advice for final year as well.

Thanks to Dr Don McFall, the course director for the Computing Systems course. His organising of the course over the last 4-5 years allowed for minimal obstacles in the way of our academic career.

Thanks to all the lecturers and study advisors that have given their time and effort to teaching and supporting myself and my peers as we made our way through the course.

Thanks to Ulster University for supplying a trouble-free experience during my time enrolled in the Computing Systems course, made possible only by the efforts of the University staff.

Thanks to my family, friends, and peers for their support and understanding over a busy and lengthy time spent doing university work over the past few years. Special thanks to Rebecca Sands, Sam Henry, and Samuel Moore from the course for their cooperation inside and outside of university.

It has been a pleasure and was only allowed by the efforts of everybody involved to be as supporting and selfless as they were.

Contents

[Abstract i](#_Toc511573134)

[Acknowledgements ii](#_Toc511573135)

[Listing of Figures vi](#_Toc511573136)

[Listing of Tables viii](#_Toc511573137)

[Listing of Code Listings ix](#_Toc511573138)

[1 – Introduction 1](#_Toc511573139)

[1.1 Background 1](#_Toc511573140)

[1.2 Project Aims 1](#_Toc511573141)

[1.3 Project Objectives 1](#_Toc511573142)

[1.4 Outline of Dissertation Structure 2](#_Toc511573143)

[2 – Literature Review 4](#_Toc511573144)

[2.1 Existing Solutions 4](#_Toc511573145)

[2.1.1 Current System 4](#_Toc511573146)

[2.1.2 Desired Solution 4](#_Toc511573147)

[2.1.3 Potential Existing Solutions 6](#_Toc511573148)

[2.2 Weighing up the options 9](#_Toc511573149)

[3 – Technical Background 10](#_Toc511573150)

[3.1 Choosing a Software Development Life-Cycle 10](#_Toc511573151)

[3.1.1 Considerations 10](#_Toc511573152)

[3.1.2 Software Development Life-Cycles 10](#_Toc511573153)

[3.1.3 Choosing a life-cycle model 12](#_Toc511573154)

[3.2 Choosing the tools 12](#_Toc511573155)

[3.2.1 Identifying the needs 12](#_Toc511573156)

[3.2.1 Choosing a web framework 12](#_Toc511573157)

[3.2.2 Choosing a database backend 14](#_Toc511573158)

[3.2.3 Development Environments and Other Tools 15](#_Toc511573159)

[4 – Project Planning 17](#_Toc511573160)

[4.1 Resource Consideration 17](#_Toc511573161)

[4.2 Risk Assessment 17](#_Toc511573162)

[4.3 Data Management 20](#_Toc511573163)

[4.4 Knowledge and Skills Required 20](#_Toc511573164)

[4.5 Involving Stakeholders 21](#_Toc511573165)

[5 – System Requirements 22](#_Toc511573166)

[5.1 Requirements Gathering 22](#_Toc511573167)

[5.2 Requirements 23](#_Toc511573168)

[5.2.1 Functional Requirements 23](#_Toc511573169)

[5.2.2 Non-Functional Requirements 24](#_Toc511573170)

[6 – Design 25](#_Toc511573171)

[6.1 UX Design 25](#_Toc511573172)

[6.1.1 Design Principles 25](#_Toc511573173)

[6.1.2 UI Designs 25](#_Toc511573174)

[6.1.3 UI Implementations 31](#_Toc511573175)

[6.2 Data Modelling 41](#_Toc511573176)

[6.2.1 Database Design 41](#_Toc511573177)

[6.2.2 Data Models 42](#_Toc511573178)

[7 – Implementation 44](#_Toc511573179)

[7.1 Architecture Overview 44](#_Toc511573180)

[7.1.1 Architecture 44](#_Toc511573181)

[7.1.2 Additional libraries used 45](#_Toc511573182)

[7.2 Code Overview 46](#_Toc511573183)

[7.3 Obstacles Encountered 55](#_Toc511573184)

[8 – Testing & Evaluation 60](#_Toc511573185)

[8.1 Testing 60](#_Toc511573186)

[8.1.1 Unit Testing 60](#_Toc511573187)

[8.1.2 System Testing 61](#_Toc511573188)

[8.1.3 Manual Testing 62](#_Toc511573189)

[8.1.4 Compatibility Testing 62](#_Toc511573190)

[8.2 Evaluation 64](#_Toc511573191)

[8.2.1 Evaluation of technologies used 64](#_Toc511573192)

[8.2.2 Evaluation of system 65](#_Toc511573193)

[9 – Results 67](#_Toc511573194)

[9.1 Fulfilment of project objectives 67](#_Toc511573195)

[9.1.1 Fulfilment of functional requirements 67](#_Toc511573196)

[9.1.2 Fulfilment of non-functional requirements 68](#_Toc511573197)

[9.1.3 Additional work completed 69](#_Toc511573198)

[10 – Conclusion 70](#_Toc511573199)

[10.1 Reflecting on the project 70](#_Toc511573200)

[10.1.1 Project scope 70](#_Toc511573201)

[10.1.2 Products produced 70](#_Toc511573202)

[10.1.3 Managing the process 71](#_Toc511573203)

[10.2 Suggestions for future improvements 75](#_Toc511573204)

[References 77](#_Toc511573205)

[Appendices 81](#_Toc511573206)

[Appendix 1 – Initial Gantt Chart 81](#_Toc511573207)

[Appendix 2 – Unit tests listing 84](#_Toc511573208)

# Listing of Figures

[Figure 2.1 – GradeXpert 11 trial version (2018) 7](#_Toc511572920)

[Figure 2.2 – TeacherKit Support Pages (2018) 8](#_Toc511572921)

[Figure 4.1 – Probability-Impact matrix before controls 18](#_Toc511572923)

[Figure 3.2 – Probability-Impact matrix after controls 19](#_Toc511572924)

[Figure 6.1 - Index page mock-up 26](#_Toc511572927)

[Figure 6.2 – Module page attendance tab mock-up 27](#_Toc511572928)

[Figure 6.3 – Module page feedback tab mock-up 27](#_Toc511572929)

[Figure 6.4 – Course page mock-up 28](#_Toc511572930)

[Figure 6.5 – Staff page mock-up 28](#_Toc511572931)

[Figure 6.6 – Student page mock-up 29](#_Toc511572932)

[Figure 6.7 – Lecture page mock-up 30](#_Toc511572933)

[Figure 6.8 – Settings page mock-up 30](#_Toc511572934)

[Figure 6.9 – Index screen 31](file:///C:\Users\Neil\Documents\Uni\CW\Final%20Report\B00652112_Report.docx#_Toc511572935)

[Figure 6.10 – Index screen entity containers 32](file:///C:\Users\Neil\Documents\Uni\CW\Final%20Report\B00652112_Report.docx#_Toc511572936)

[Figure 6.12 – Index screen attendance upload results 33](file:///C:\Users\Neil\Documents\Uni\CW\Final%20Report\B00652112_Report.docx#_Toc511572937)

[Figure 6.11 – Index screen attendance upload form 33](file:///C:\Users\Neil\Documents\Uni\CW\Final%20Report\B00652112_Report.docx#_Toc511572938)

[Figure 6.13 – Single module attendance view (staff user) 34](file:///C:\Users\Neil\Documents\Uni\CW\Final%20Report\B00652112_Report.docx#_Toc511572939)

[Figure 6.14 – Single module attendance charts view 34](file:///C:\Users\Neil\Documents\Uni\CW\Final%20Report\B00652112_Report.docx#_Toc511572940)

[Figure 6.15 – Single module attendance view (student user) 35](file:///C:\Users\Neil\Documents\Uni\CW\Final%20Report\B00652112_Report.docx#_Toc511572941)

[Figure 6.16 – Module feedback tab (staff user) 35](#_Toc511572942)

[Figure 6.17 – Module feedback submission 36](#_Toc511572943)

[Figure 6.18 – Module feedback tab (student user) 36](#_Toc511572944)

[Figure 6.19 – Single course view 37](#_Toc511572945)

[Figure 6.20 – Single lecturer view (table) 37](#_Toc511572946)

[Figure 6.21 – Single lecturer view (charts) 37](#_Toc511572947)

[Figure 6.22 – Single student view 38](#_Toc511572948)

[Figure 6.23 – Single lecture view 38](#_Toc511572949)

[Figure 6.24 – Settings view 39](#_Toc511572950)

[Figure 6.25 – Module / Course settings view 39](#_Toc511572951)

[Figure 6.27 – Module attendance mobile view 40](file:///C:\Users\Neil\Documents\Uni\CW\Final%20Report\B00652112_Report.docx#_Toc511572952)

[Figure 6.26 – Index mobile view 40](file:///C:\Users\Neil\Documents\Uni\CW\Final%20Report\B00652112_Report.docx#_Toc511572953)

[Figure 6.29 – Settings mobile view 40](file:///C:\Users\Neil\Documents\Uni\CW\Final%20Report\B00652112_Report.docx#_Toc511572954)

[Figure 6.28 – Module feedback mobile view 40](file:///C:\Users\Neil\Documents\Uni\CW\Final%20Report\B00652112_Report.docx#_Toc511572955)

[Figure 6.30 – Database tables design 41](#_Toc511572956)

[Figure 6.31 – Final database tables generated by Django 43](#_Toc511572957)

[Figure 7.1 – Architecture overview 44](#_Toc511572958)

[Figure 7.2 – University attendance data example 48](#_Toc511572962)

[Figure 7.3 – Email attendances command ‘help’ output 52](#_Toc511572968)

[Figure 8.1 – Travis CI automated unit testing 61](#_Toc511572980)

[Figure 8.2 – Web browser market share statistics (via https://netmarketshare.com/browser-market-share.aspx) 62](#_Toc511572981)

[Figure 8.3 – Cross-browser compatibility screens (from left to right, top to bottom: Chrome, Firefox, Edge, IE11, mobile device (simulated by Chrome dev tools) 63](#_Toc511572982)

[Figure 10.1 – Github repository build status 71](#_Toc511572985)

[Figure 10.2 – Github repository README documentation 72](#_Toc511572986)

[Figure 10.3 – Github repository issues 73](#_Toc511572987)

[Figure 10.4 – Github repository closed issue 73](#_Toc511572988)

[Figure 10.5 – Github repository pull requests 74](#_Toc511572989)

[Figure 10.6 – Github repository merged pull request 74](#_Toc511572990)

# Listing of Tables

[Table 4.1 – Risk Assessment 17](#_Toc511572994)

[Table 5.1 – Functional Requirements 23](#_Toc511572997)

[Table 5.2 – Non-functional Requirements 24](#_Toc511572998)

[Table 9.1 – Functional requirements evaluation 67](#_Toc511573055)

[Table 9.2 – Non-functional requirements evaluation 68](#_Toc511573056)

# Listing of Code Listings

[Code Listing 7.1 – File upload code fragment 46](#_Toc511573102)

[Code Listing 7.2 – File upload file type saving code fragment 47](#_Toc511573103)

[Code Listing 7.3 – Duplicate data detection code fragment 48](#_Toc511573104)

[Code Listing 7.4 – Attendance session row parser code fragment 49](#_Toc511573106)

[Code Listing 7.5 – Attendance data row parser code fragment 49](#_Toc511573107)

[Code Listing 7.6 – Pie chart building code fragment 50](#_Toc511573108)

[Code Listing 7.7 – Line chart building code fragment 50](#_Toc511573109)

[Code Listing 7.8 – Email command arguments code fragment 51](#_Toc511573110)

[Code Listing 7.9 – Email command arguments usage code fragment 52](#_Toc511573112)

[Code Listing 7.10 – Email command send email code fragment 53](#_Toc511573113)

[Code Listing 7.11 – Email attendance report template 53](#_Toc511573114)

[Code Listing 7.12 – Module feedback form code fragment 54](#_Toc511573115)

[Code Listing 7.13 – Module feedback save code fragment 54](#_Toc511573116)

[Code Listing 7.14 – Encrypted user model code fragment 55](#_Toc511573117)

[Code Listing 7.15 – Password reset form override code fragment 56](#_Toc511573118)

[Code Listing 7.16 – Password reset view form override code fragment 56](#_Toc511573119)

[Code Listing 7.17 – Multiple file upload placeholder HTML fragment 57](#_Toc511573120)

[Code Listing 7.18 – Add file upload row code fragment 58](#_Toc511573121)

[Code Listing 7.19 – Submit attendance data code fragment 59](#_Toc511573122)

# 1 – Introduction

## Background

Ulster University is one of the largest universities in Northern Ireland, with campuses in Belfast, Jordanstown, Coleraine and Magee.

In the existing system at Ulster University, student attendance would be collected by lecturers per session that they hold with students and entered into a system that outputs a spreadsheet – a flat file with limited capability for searching, analysing or reporting on the data.

This manual process of managing attendance data was limiting, and the idea for this project was put forward by Dr Alan Brown, associate head of the school of engineering at the Jordanstown campus of Ulster University, who played a role in the existing system and proposed that a technical solution would allow for better use of audience data. This system could be extended to cover other aspects of audience data that the university deals with as well, to provide benefits in multiple different areas.

## Project Aims

The main aim for this project was to create a technical solution to manage student attendance and module feedback data to provide a more modern and effective way to view, process, and report on this data to benefit both students and staff members of the university.

## Project Objectives

To be able to start working toward the aim of the project, the following activities must first be completed:

1. Review the current process and ask questions of the relevant parties to understand the state of the current system and what improvements are desired to get to the proposed system.
2. Investigate the potential choices for tooling, both in programming languages and the development environments to support them and decide on the tools most suited for the project.
3. Gather high-level requirements to give a general outline of the proposed system.
4. Investigate and decide on the most appropriate software development life-cycle to use for the project.

To achieve the project aim, the following activities must then be completed:

1. Work with stakeholders to refine system requirements.
2. Design how data will be represented and create database to hold the data in this format.
3. Design how the website will look and create components of the website according to this design, making sure to adhere to best practices including consistency and user-friendliness.
4. Build the web application to fulfil the system requirements while adhering to the website designs.
5. Perform testing on each feature as they’re developed to keep the system’s correctness and integrity intact as development progresses.
6. Perform final system testing of the application and fix any issues found.

## Outline of Dissertation Structure

This report will be structured as follows:

Chapter 2 will review the existing literature as it relates to this project to give context to the project and to review existing solutions that could potentially meet our needs.

Chapter 3 will focus on the choices made to drive the project from a development standpoint including the development life-cycle and the tooling to be used to develop the project.

Chapter 4 will cover any relevant aspects as they relate to planning the project, including risk assessment and the skills required of the developer.

Chapter 5 will cover the systems requirements and the methods used for requirements gathering.

Chapter 6 will present the various system designs, including UX design and the approach to modelling the data.

Chapter 7 will focus on the implementation of the system, covering the actual implementation that realises the system requirements. This chapter will include a high-level overview of the system architecture, code fragments and any significant challenges that were encountered (with respective solutions).

Chapter 8 will cover the testing and evaluation carried out to help validate the project and verify its correctness.

Chapter 9 will reflect on the project’s results and evaluate the fulfilment of the initial project objectives.

Chapter 10 will conclude the report, including a critical evaluation of the processes and management involved in the project, and suggestions for future improvements.

# 2 – Literature Review

## 2.1 Existing Solutions

Before jumping into any work, it was necessary to look at the current system to understand its shortcomings and research any possible solutions that already exist out there that may meet our needs. Even if nothing suited our needs there may be facets of similar solutions that could inspire some features in a system that we would create, so this exercise can always be valuable.

### 2.1.1 Current System

In the current system, lecturers collect attendance after each “session”, be it a lecture or tutorial, and feed this attendance data into a 3rd-party system which outputs a report in an Excel file format. The result is a flat file with limited capacity for searching and linking the data, especially with multiple such files across different sessions. Another area in which the university could improve their data handling was found to be in module feedback. The only formal approach to student feedback in the existing system comes at the end of modules, where a module survey can be carried out by students who had undertaken that module. However, feedback at this stage in the academic year is already too late, as the students have already finished their work on the module – any improvements to the content or methods of teaching because of this feedback only benefits the next students who undertake that module. In other words, the feedback cycle is too long; a faster turnaround is required for students to see more benefit in their studies.

### 2.1.2 Desired Solution

To address these shortcomings, an ideal solution would try to address these 2 fronts:

1. The attendance front could be improved (without outright replacing the current system) by allowing the upload, storage, and processing of these existing files. Such a system would allow retroactive loading and processing of this existing data in addition to any future attendance data and would provide more a more user-friendly way to interact with and analyse the information, with the benefit of built-in reporting of such information as well.
2. The feedback front would be addressed by allowing students to give feedback on modules in a similar fashion to the current system, but instead allowing them to directly tie feedback to the module at any time rather than at a pre-determined point in time when it’s too late for those students to see any improvement.

These components of the university ecosystem are important, both from a student’s and a staff member’s point of view. For attendance, the ability for the system to automatically report on the data via email to relevant staff members can be important as it relates to particular students. Lecturers can manage many students at one time which can make it difficult for them to individually manage students. With the stresses of university – financing, workload, and balancing their academic work with a job to name a few – having a lack of a watchful eye can compound the problem. Depression and mental health issues have become increasingly recognised as legitimate issues in recent years, and university students dealing with these stress factors as (mostly) young adults makes their emotional well-being an important thing to keep an eye on. A survey of British students held by YouGov in 2016 found that 27% of responders ‘report having a mental health problem of one type or another’, with depression and anxiety the most common at 77% and 74% respectively (Aronin & Smith, 2016). If a student begins to lose motivation in their studies, becomes unwell, faces external issues with home life etc., it is likely that their studies and attendance drop in priority. Automatic reporting on a sudden or prolonged absence for individuals can highlight that a student may be struggling and allows lecturers or studies advisors to proactively reach out to help. The developer was also interested in reducing the feedback loop time, as it had been witnessed first-hand through their time in university that collecting formal module feedback only after a module has already been completed provides no benefit to the students who dealt with a poor approach to teaching, and less accountability for lecturers who are expected to meet a university-level standard for teaching.

Such a solution stands to benefit both lecturers and students: lecturers will be able to manage their attendance feedback more easily and gain some feedback on how any students that they manage are doing; and students can potentially receive help without needing to actively reach out, can take more ownership in the quality of their teaching, and provide more granular feedback as the academic year progresses.

### 2.1.3 Potential Existing Solutions

There are already a few solutions in this domain already, which we’ll go into detail on to see how they fit to our needs. As the main objective of this project is to improve the attendance monitoring process, we’ll look at solutions as they relate to this facet.

StREAM by SolutionPath (<https://www.solutionpath.co.uk/>)

StREAM is a learning analytics solution for university institutions, designed to ‘support decision making, personalise engagement and improve student attainment’ (SolutionPath, 2017). StREAM appears to be quite sophisticated and provide a lot of useful functionality, claiming to help predict which students are at risk of early withdrawal from studies, provide alerts and to aid with attendance monitoring – some of the obstacles that our desired solution aims to address.

Some of the downsides of StREAM as it relates to our needs:

* SolutionPath only have bases in mainland UK, which could mean that system support could be made slightly more difficult.
* The website gives no indication of fees, but the depth of the product’s features suggest that it might be rather costly.
* There doesn’t appear to be any trial version or examples available without booking a demo for the product – unable to quickly see how this solution might fit our needs.

GradeXpert (<http://www.gradexpert.com.au/>)

GradeXpert is a system for schools with provides attendance functionality, and includes data management, analytics and reporting (GradeXpert, 2017). Although targeted to schools, its claims on flexibility and customisation plus our limited requirements as they relate to attendance mean that this is potentially another viable solution for our problem. This product appeared to be a lot simpler than the first and this may even align with our requirements more closely as our proposed solution doesn’t necessarily need to perform very complex operations. It also provided a free trial so that the product could be demoed quickly with just a download.

Some of the downsides of GradeXpert as it relates to our needs:

* Based in Australia, this again could cause some difficulty if ever needing support for the product.
* The trial version didn’t inspire much confidence. Figure 2.1 shows a dialog that prevented any actions while trying to use the trial, with no follow-up emails to provide any such registration codes after already signing up for the trial version with an email address.

A screenshot of a cell phone

Description generated with very high confidence

Figure 2.1 – GradeXpert 11 trial version (2018)

TeacherKit (<http://teacherkit.net/>)

TeacherKit boasts versatility and being intuitive to the point that ‘there’s no need for specialist skills or training’ (TeacherKit, 2017). At a price point of only $40 annually and statistics such as 1M+ downloads and support for multiple languages, this appears like another potential candidate for our needs as the product has functionality for logging student attendance.

Some of the downsides of TeacherKit as it relates to our needs:

* No further clarification on the price point, but assuming that this low price is per user means it could quickly become very expensive with hundreds of students using the system.
* Not much information about the product on the website, and lack of a trial version.
* Their website doesn’t inspire much confidence – their support page results in a 404 (see Figure 2.2) which isn’t exactly what we’re looking for if we were to require help with the product.

A screenshot of a cell phone

Description generated with very high confidence

Figure 2.2 – TeacherKit Support Pages (2018)

## Weighing up the options

After researching some potential solutions that are already out there, it became clearer that a tailor-made solution for our situation would make the most sense. In addition to the shortcomings previously listed for each product, there are other considerations that make them less feasible for our needs:

* Our requirement to upload data in a certain file format and structure means we need a very specific solution that, if even possible with an existing solution, will require a certain set-up to achieve.
* None of the solutions explored appear to provide any feedback functionality, which was the other half of our desired functionality.

Based on this, it was decided to develop a system that we could tailor to our specific needs, down to the exact formats that Ulster University operates with. Building a custom solution gives us benefits in cost, flexibility, support, and being able to create proof of concepts that can be checked and modified to further refine for our needs.

# 3 – Technical Background

## 3.1 Choosing a Software Development Life-Cycle

### 3.1.1 Considerations

The following information was gathered prior to the project and may play a factor into deciding which life-cycle is most appropriate for this project:

1. The developer has regularly scheduled meetings and direct communication with the project stakeholders
2. The project is open to constant expansion for different types of audience data
3. There is minimal non-coursework related project documentation to be completed
4. The key stakeholder is very invested in the solution and is open to frequent cooperation regarding the project as it progresses

With this information, the project’s needs can be evaluated against some well-known software development lifecycle models and a choice can be made for whichever fits the project best.

### 3.1.2 Software Development Life-Cycles

Waterfall

The waterfall model is a distinct set of stages from analysis through requirements gathering to the actual development, testing and ending in documentation and maintenance. Waterfall is straightforward and easy to follow but quite restrictive – there’s little room for overlap in each phase and the developer would only consider it for the project if there was no capacity for constant contact with the end user(s), which isn’t the case.

Spiral

The spiral model differs from waterfall in that it is an iterative model, which is a step in the right direction for the project. The spiral model iterates over 4 steps, the time spent on each growing as the project progresses and grows in complexity. This model places a large amount of focus on prototyping and risk analysis, and while the prototyping aligns with the project goals, the risk analysis is not as important. The spiral model is more appropriate for large projects where mistakes can be costly, and as such it isn’t the right model for this project.

Incremental

A purely incremental approach to software development focuses on building features to full completion, feature by feature. Each solution goes from zero to fully fleshed out, and so an incremental approach is an iterative approach. This type of approach is a viable candidate for the project, but as the development team consists of only a sole developer, this approach could be difficult as features will take longer to fully develop for demoing to stakeholders.

Iterative

A purely iterative approach builds the whole of the project, but to a lower level of capability than the purely incremental approach. This method allows for a barebones-level project to be built, and then refined and extended until each feature is fully completed. This approach is another viable candidate for the project, but just as in the purely incremental approach, with such a small development team capacity it can be difficult to build the whole application to even a basic level as it will take some time, and the stakeholders’ involvement can’t be used as advantageously.

Agile

An agile approach to software development is both incremental and iterative, allowing for the product to be built piece by piece, with each feature gradually improved upon over time (Scotland, 2009). Agile approaches focus on communication with the customer / end user, and continuous delivery of working software. The agile methodology also focuses on flexibility and tailoring each approach to the individual projects that they’re applied to. One of the more popular agile frameworks is Scrum, where demonstrations to the end user or a product owner are frequent (every 2-4 weeks at the end of a ‘sprint’). This aligns nicely with the stakeholders’ availability, and they can also act to keep the project requirements prioritised for the system if the schedule began to slip, which is always a real possibility with software projects. There’s also a minimum need for documentation as the actual software is continually reviewed by the product owner.

### 3.1.3 Choosing a life-cycle model

From the above evaluation, it seems that an agile approach is the best fit for the project. While an agile approach makes more sense for a team of developers and testers rather than a single developer / tester, the methodology is what suits the project’s situation and can be adapted to be more suitable e.g. not placing as much importance on retrospective sessions. The opportunity for the developer to have frequent face to face meetings with the key stakeholder, who will act as the product owner, allows for the ability to constantly demo new work / evaluate and re-prioritise tasks based on progress. The product owner also has ideas for future expansion of the proposed project, so even if the progress made on the project was better than expected the project can still be built upon with (currently) unplanned features.

## 3.2 Choosing the tools

### 3.2.1 Identifying the needs

The proposed system would consist of a web application with a database backend to store the associated data with attendance and feedback required of the application. When choosing the tools for development, these needs should be considered to choose an appropriate tech stack to effectively do the job.

### 3.2.1 Choosing a web framework

When deciding on a web framework, the options were already limited to frameworks that the developer was at least aware of, if not familiar with the language. With so many options out there, it would be infeasible to consider them all, and considering only the most popular frameworks guarantees a large network of users that can be queried when issues crop up during development. As the developer is mostly familiar with Java, we’ll begin with looking at Java frameworks:

Play Framework – Java / Scala (<https://playframework.com/>)

Play touts the ability to create highly-scalable applications due to its ‘predictable and minimal resource consumption’, among other traits such as developer-friendliness and being built specifically for modern web and mobile applications (PlayFramework, 2017).

Benefits:

* IDE support for IntelliJ IDEA, a powerful IDE that the developer is familiar with.
* Built in test helpers for controllers and views, plus strong support for testing with Java’s JUnit, Selenium, and Mockito libraries.

Downsides:

* Released in 2012, Play 2 (an overhaul of Play 1) is quite immature with a smaller community than other large Java frameworks which have stood the test of time (Brikman, 2013).

Spring Framework – Java (<https://spring.io/>)

Spring is a very popular Java framework with a large community that promotes a quick start-up with Spring Boot (Pivotal Software, 2017).

Benefits:

* Massive support available from other users on StackOverflow and lots of guides on their website means getting up to speed and resolving issues is made easier.
* Promotes developer productivity tools such as live reload.
* IDE support for IntelliJ IDEA.
* Boasts the ability to get up and running very quickly with minimal configuration of Spring with Spring Boot

Downsides:

* Large – the framework has lots of files and for a relatively small application or proof-of-concept, it might be overkill.

Django – Python (<https://www.djangoproject.com/>)

Django is a web framework for Python that ‘encourages rapid development and clean, pragmatic design’ (Django Software Foundation, 2017).

Benefits:

* Is designed for a fast start-up from concept to a real project.
* Strong support for security, aiding developers to avoid common security mistakes.
* Takes care of common web development tasks out of the box e.g. user authentication, forms
* Extensive built-in unit testing support

Downsides:

* Its templating language goes to lengths to prevent logic in templates. While adhering to good practices, these kinds of restrictions can sometimes be more trouble than they’re worth (Gujar, 2016).

While the developer is very familiar with Java, the development language and framework of choice chosen for this project is Python with Django. Django was chosen for its ability to get started very quickly by bootstrapping common tasks such as user authentication. In addition to this, the project supervisor was familiar with this framework and the developer wanted to use this project as an opportunity to learn a new language and framework.

### 3.2.2 Choosing a database backend

Following on from above, one of the benefits that Django provides is built in ORM (object-relational mapping) via Django models. These models typically map to single tables in the database, and when the data structures are represented in the code via these models, SQL “migration” scripts can be automatically generated for newly created tables and modifications to existing tables. Because of this, the database layer is cleanly separated from the application code and as a result, the database backend doesn’t matter as much as the code isn’t tied down to a specific implementation – the backend can be changed with minimal effort and the migration scripts and code should continue to work seamlessly.

The default backend for Django applications is SQLite. While SQLite doesn’t stand up to database engines like MySQL or PostgreSQL, it works fine as a backend for development purposes and is useful for building proof-of-concepts. With this in mind, SQLite will be used to store data during the development of the project, and if the project is eventually deployed the backend can be changed to be more production-ready as the code and SQL scripts created by Django will be engine-independent and should work for any backend that is decided on at that stage. SQLite would be less useful for a deployed production-ready application as its built into the application. To separate the data from the application in this case would work but would be less performant, where a client/server database engine would be the more appropriate option. SQLite would also be less appropriate in the case of concurrent writes as the files only allow one writer at a time. While this likely wouldn’t be an issue with this application (database writes would be queued for the small amount of time they’re in use, and the number of potential users in the system wouldn’t be large enough to make this a problem in most cases), it’s still wise to consider the traditional client/server approach as it’s more appropriate for the use (SQLite, 2017).

### 3.2.3 Development Environments and Other Tools

For the main IDE, the choice for development with Python and Django is PyCharm. PyCharm is an IDE by JetBrains, the creators of IntelliJ IDEA. The developer of the project is familiar with IntelliJ as an IDE for Java and PyCharm operates almost identically. PyCharm offers intelligent code completion tailored to Python and checks for and allows automatic formatting to adhere to PEP-8, a styling convention for Python which means that the code remains consistent both within the project and with other Python projects.

For source control, Git and Github were chosen to keep track of changes in files, allow for reverting changes effectively, and to keep a back-up version hosted on Github both for peace of mind and for development on multiple different machines. These were also chosen based on familiarity and are also some of the most popular tools in this domain.

For issue tracking, a Gantt chart (Appendix 1) was initially created to estimate and group tasks to understand how much could be feasibly done in the allotted time for this project. After this, Github’s issues were created from most of the tasks on the Gantt chart and used to track issues during the actual development so that the features could be tied to actual code and closed off after completion, and so that the developer and project supervisor could interact with these together.

# 4 – Project Planning

## 4.1 Resource Consideration

As this project only plans to deliver a web application, the resources required are minimal:

* Development machine – desktop computer or laptop
* Software to facilitate writing / running / testing code – all free and available on the Internet

## 4.2 Risk Assessment

As the project isn’t safety critical and doesn’t deal with very sensitive information, the risk associated is small. Nevertheless, a plan of action should be developed for any potential risks that may occur, including risks that are common to any software project.

Table 4.1 – Risk Assessment

|  |  |  |  |
| --- | --- | --- | --- |
|  | Risk | Impact | Controls |
| a | Data loss | Time spent on re-doing work which could be spent on new work – falling behind schedule | Use of version control which also acts as a backup, hosted on a popular site that is unlikely to fail (Github) |
| b | Schedule slippage | Failure to meet critical deadlines and overall project deadline | Early use of the Gantt chart and constant referral to the Github issues to manage the project’s scope and time constraints |
| c | Feature creep | Project schedule lengthened (can’t afford this for this project) or quality of work decreases to perform more in less time. Could result in increased costs depending on the new requirements | Overall system requirements are defined and locked down with stakeholders at the start of the project, as well as frequent communication with stakeholders |
| d | Unforeseen circumstances | Unforeseen circumstances, both technical and personal, could lengthen project schedule or decrease quality of work. This project only has a single developer with no replacements, so this could lead to serious setbacks | Accounted for to a certain degree with a substantial buffer at the end of the Gantt chart before the project’s deadline. Beyond a certain point there’s nothing that can be done to fix this, but it should mitigate the risks to a reasonable degree |
| e | Unavailable resources | Project schedule could be lengthened or could even be made impossible depending on the resources required. Workarounds could decrease quality of project | Resources required are identified at the start of the project, and no acceptance of any requirements past this point that change these expected resources |

Figure 4.1 and 4.2 show probability-impact matrices for these risks before and after the proposed controls used to mitigate / prevent their potential impact.

**Before Controls**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **Probability** | | | | |
|  |  | 1 | 2 | 3 | 4 | 5 |
| **Impact** | 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  | **c, e** |  |  |
| 4 |  |  | **b** |  |  |
| 5 |  | **a** |  | **d** |  |

Figure 4.1 – Probability-Impact matrix before controls

**After Controls**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **Probability** | | | | |
|  |  | 1 | 2 | 3 | 4 | 5 |
| **Impact** | 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 | **c, e** |  |  |  |  |
| 4 | **b** |  |  |  |  |
| 5 | **a, d** |  |  |  |  |

Figure 3.2 – Probability-Impact matrix after controls

In Figure 4.1 and 4.2 above, the combination of probability and impact of each risk is shown, each falling into a coloured zone. Green represents safety – either the probability or the impact is low enough that the project can be accepted with confidence. Yellow represents a warning – if the risk cannot be reasonably mitigated any further, then it’s still okay to accept the project but it must be kept in mind and monitored. Red represents a problematic risk – the combination of the risk’s probability to occur and the severity of the impact of its occurrence are too problematic to safely accept the project.

As shown, after the controls are applied all the project risks can be considered safe. Risk ‘d’ – unforeseen circumstances – was initially considered to be the largest risk, but with over a month’s buffer added before the end of the project deadline, this should be reasonably mitigated.

## 4.3 Data Management

While the project doesn’t deal with very sensitive data, it does handle some personally-identifying information regarding its users. The Data Protection Act 1998, amongst other things, specified that data must be kept safely and securely (GOV.UK, 2017). With that in mind, any personally-identifiable information will have to be stored securely. An appropriately secure encryption method will be used to store user information and the same for hashing user passwords. As the code is in a public Github repository, any sensitive code-related information e.g. keys will be stored as environment variables on any machine that needs to run the project, as it’s important for such keys to remain secret in the context of software applications to enable things such as encryption. In addition to this, no database data will be stored in the repository, only the project code used to process the data, and for the majority of the project’s lifecycle all development will be done on the developer’s local system with obfuscated de-personalised data.

## 4.4 Knowledge and Skills Required

The system to be developed doesn’t involve the implementation of any hardware components to use, so the only new skills required will be related to actual programming. The project developer has 4 years of experience in the software industry and is familiar with programming and programming concepts, however the proposed project is to be built with Python and Django, which the developer has had no exposure to. There will be some upskilling with this language and framework to be considered as part of the project’s development. The developer is also familiar with databases and there shouldn’t be a need for advanced database functions, so the type of RDBMS used shouldn’t matter much aside from syntactical differences. The developer’s career doesn’t typically involve web-app development so there may be some additional knowledge required in this area, but they’ve had some exposure to this type of development through their university course and personal projects. Any gaps in any of these areas should be sufficiently filled through practice as the project progresses, so no specific planning is needed for upskilling.

## 4.5 Involving Stakeholders

The key stakeholder, Professor Colin Turner, will also act as the product owner in an Agile setting over the course of the project. Meetings have been scheduled on a weekly / fortnightly basis via Skype or in face-to-face meetings. This will help to ensure that the project is kept on track and will allow for feature demos and to resolve any queries regarding the system’s requirements or design. Professor Turner also, as the head of the engineering school at the university, will maintain a network of other lecturers who will be likely to use the final system, so he can collect any pertinent additional information from them and pass it directly to the project developer. Dr Kenneth Adamson plays a smaller role in the development of the project but will be available on a weekly basis for face-to-face meetings regarding any overall project queries and deadlines. Dr Alan Brown will not be directly involved in the project but can communicate with Professor Turner to collate feedback through one end user. Having the stakeholders available so frequently and for face-to-face meetings will allow any hiccups to be resolved quickly and effectively, which will reduce the risk of the project failing to meet its deadlines. As a university student, the project’s developer will also have access to a network of student peers who can be used to gather requirements and feedback from regarding the system.

# 5 – System Requirements

## 5.1 Requirements Gathering

The Software Engineering Body of Knowledge (SWEBOK) v3 defines a software requirement as ‘a property that must be exhibited by something in order to solve some problem in the real world’ (IEEE Computer Society, 2014). Such requirements can be categorised either as functional or non-functional requirements. Functional requirements describe the functions that the software realises, also known commonly as features, while non-functional requirements are constraints on the solution. Examples of non-functional requirements are those related to the performance, security, maintainability etc. of the system.

To gather requirements for this system, the bulk of the information was gathered via correspondence with Professor Turner. Professor Turner was able to provide information about the needs as they relate to the attendance upload and email reporting functionality. Professor Turner was a very appropriate stakeholder as it relates to this information, having a wealth of domain knowledge about the university’s processes as the head of the school of engineering and having technical experience himself, even having prior experience with developing similar email reporting functionality for other systems. The relevant information was gathered during informal meetings with Professor Turner to understand the problem domain and refined into requirements as Professor Turner and the system developer came to an understanding about what was technically viable and made sense in the context of the domain. Information about the feedback feature was mainly gathered from other students of the university, which provided valuable information for requirements such as the desire for anonymity when supplying feedback.

When documenting the requirements, they were initially scheduled as tasks to be performed in a Gantt chart at the beginning of the project. These were later migrated to the Github repository for the project as Github issues to allow them to be worked against as features.

## 5.2 Requirements

### 5.2.1 Functional Requirements

Table 5.1 – Functional Requirements

|  |  |  |
| --- | --- | --- |
| ID | Requirement | Reason |
| F01 | Staff members shall be able to upload attendance data for a specific module in CSV/Excel format | To store student attendance data for analysis and reporting |
| F02 | Attendance data that is re-uploaded shall overwrite existing data rather than append duplicate data | To allow bulk uploads that may include files that were already uploaded; so that staff members don’t need to keep track of which attendance data was already uploaded |
| F03 | Attendance data shall be able to be uploaded in the university’s format | To allow direct translation from the existing files to the new system; to reduce manual work done by staff members to prepare the data for upload |
| F04 | Attendance data shall be able to be uploaded for multiple files at a time | To reduce tedium and time spent uploading and submitting batches of files in sequence |
| F05 | Staff members shall be able to choose the modules and courses that they want to view on their dashboard and to receive email reporting on | To allow staff members to narrow their focus to only the modules and courses that they’re involved or interested in receiving updates on |
| F06 | Students shall be able to view only the modules and lectures that they have attendance data for on their dashboard and in email reporting | To allow students to only focus on their own attendance data; to hide unnecessary information about other students |
| F07 | All users shall be able to change their password and request a password reset | To give flexibility to the user regarding their security and allow account recovery |
| F08 | Staff members shall be able to add modules, courses, and other student / staff users | To allow staff members to maintain the system by adding other entities on which the system depends |
| F09 | All users shall be able to view appropriate visualisations of the attendance data | To allow a concise and user-friendly representation of the data that quickly conveys necessary information |
| F10 | All users shall be able to receive email reporting to their registered email address to inform on their attendance over a weekly or monthly period | To give an overview of attendance data passively without having to actively use the application; to give warnings for low attendance |
| F11 | Students shall be able to give feedback on modules that they have attendance data for, and view only their feedback given for that module | To give students the ability to let their voice be heard more throughout their course and give them more potential to influence their education; to hide potentially sensitive feedback information from other students |
| F12 | Students giving feedback shall be able to choose whether their feedback is anonymous or not | To give students the confidence to give honest feedback regarding their module |
| F13 | Staff members shall be able to only view feedback for a given module | To give staff insight on how a module is being received and make any adjustments as required |

### 5.2.2 Non-Functional Requirements

Table 5.2 – Non-functional Requirements

|  |  |  |
| --- | --- | --- |
| ID | Requirement | Reason |
| N01 | Newly-created user accounts shall be initialised with a random password to force a password reset request for the registered email address | Security: allowing the creator to set the initial password or generating the same password for all new accounts presents security vulnerabilities |
| N02 | Passwords shall be forced to meet certain restrictions, including length checks and not being too common | Security: having such restrictions means that users are encouraged to set more complex passwords that are harder to guess and exploit |
| N03 | Passwords shall be hashed when stored so that they cannot be read or translated to plaintext | Security: plaintext passwords present a critical security vulnerability |
| N04 | The system shall be accessible by users who are colour-blind: any information that is presented with the use of colour, e.g. graphs, shall be equally accessible by such users | Accessibility: ‘As many as 8 percent of men and 0.5 percent of women … have the common form of red-green color blindness’ (National Institutes of Health, 2015). With the plan to display key information using charts, this is an important measure needed to ensure that users aren’t left out |
| N05 | The styling of the application shall remain consistent throughout | Usability: a consistent interface allows users to understand the system wherever they travel in it |
| N06 | The application shall be displayed on all major web browsers and devices as equally as possible, such that all information is conveyed in an appropriate way | Usability: users should be able to access the application and use it equally regardless of their preference of the client browser or device |
| N07 | The system should hide information to unauthenticated users, and hide irrelevant information based on user | Security: unauthenticated users may be malicious actors who plan to use the information for nefarious purposes |

# 6 – Design

## UX Design

### 6.1.1 Design Principles

When considering the design of the website, it was important to adhere to common design patterns that users are familiar with. Employing such techniques means that the users of the system could more quickly understand it by using their knowledge of existing sites. A 2015 study by Orbit Media of the 50 top marketing websites found that 88% of these sites used horizontal navigation across the top of the site (Crestodina, 2015), which was considered for this system for logout / settings navigation. Another area in which the system strives for consistency is in its use of visual metaphors. In the use of icons, this consistency is seen with the cog used to represent settings, as well as a tick and cross used to represent the attendance value for students. These icons were supplied by Font Awesome, which is used by over 15 million websites including Deadspin, Merriam-Webster, and Gizmodo (BuiltWith, 2017). With usage in popular websites, even the icons themselves will be more recognisable for users. As well as being consistent with other websites, the system should also strive to be internally consistent. Being internally consistent with its components, colour scheme and layout means that the user has a smooth experience when navigating through and using the site, avoiding potentially jarring inconsistencies.

### 6.1.2 UI Designs

Some high-level UI designs for the system are shown below. For most components of the system, the views slightly differ between the staff and student user types, mostly related to the privileges that staff users are afforded but also for some functions that only students have access to e.g. giving feedback.

Index Page

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.1 - Index page mock-up

On the index page for the system, an overview of each entity visible to the user is displayed. For a student this would be limited to their own modules and lectures with attendances, and for a staff member this would show everything as they don’t need any information hidden from them. Staff members will also have access to the ability to upload attendances for modules from this area. The index page sets the tone for the rest of the system, so the other views should appear consistent with the design choices present here. On the top right of the page, a user indicator will drop down to reveal navigation to settings and an option to log out, and ample space is present in the margins of the page and between components so as to not over-crowd the page. These will be seen through all pages of the site, with other pages having some navigation back to this page or a different related page such as the settings page.

From this page, each of the listed entities can be navigated to for further information, e.g. for information on particular modules or lectures.

Module Page – Attendance

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.2 – Module page attendance tab mock-up

On the single module page, 2 tabs will be present to separate the attendance and feedback information. For staff users, each student’s attendance for that module will be shown while a student will only be able to view their own attendance. A pie chart showing the overall module attendance will be shown below, and a line chart showing how the attendance fluctuates per lecture.

Module Page – Feedback

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.3 – Module page feedback tab mock-up

For the feedback tab of the single module page, staff will see the feedback given by students for that module, sorted by the newest first. Students will only see the feedback that they’ve given and will have the option to give feedback as well. Giving feedback will consist of a separate page with a simple form to submit feedback, so it won’t be included in these designs.

Course Page

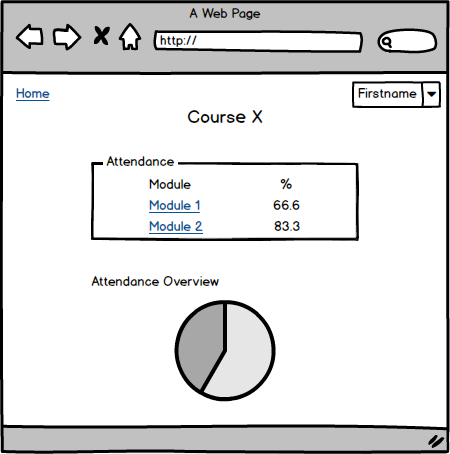


Figure 6.4 – Course page mock-up

For a single course, the attendance for each module linked with that course will be shown, and another pie chart showing the overall attendance below.

Staff Page

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.5 – Staff page mock-up

For a single staff member, the modules linked with that user will be shown, along with a pie chart showing the attendance overview and a bar chart comparing the attendance between modules.

Student Page

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.6 – Student page mock-up

For a single student view, the page is slightly different from other single entity views. As multiple modules are shown that the user has attendance for, the information was decided to be separated in an accordion per module. A single table couldn’t be used to display this information as before as modules may have different numbers of lectures. As previously used, a pie chart will also be present to give the overall attendance percentages.

Lecture Page

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.7 – Lecture page mock-up

The view for a single lecture will display the students and whether they attended this lecture. A pie chart will show the overall percentage value for all students.

Settings Page

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.8 – Settings page mock-up

The settings page offers some user-specific settings and admin controls available to staff members. This page links to some options such as changing the user password and creating new entities from the admin panel. As with the other pages, this page remains consistent in style.

Other pages will also be in the system but will either play the function of form submission (feedback, entity creation) or are common to all websites (login, change and reset password). These pages don’t need to have specific designs but will still be consistent with the rest of the system in look and function. In addition, charts used to display information will use a red/green colour scheme as these commons are commonly associated with positive/negative or pass/fail, so these are appropriate for displaying attended vs. not attended. As the system also aims to aid colour-blindness, these colours will change to blue/orange when this setting is enabled for users as this is a common colour-blind-friendly palette (Shaffer, 2016).

### 6.1.3 UI Implementations

Some of the screens of the implemented system are shown below to show and discuss the similarities and differences from the designs. Some mobile-sized screens will be shown as well to show how the website scales to smaller devices.

A screenshot of a cell phone

Description generated with very high confidenceIndex Page

Figure 6.9 – Index screen

The initial view of the index page will hide the entities so that it doesn’t display an overwhelming amount of information. This shows the staff view of the page, where the student would only see the modules and lectures.

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.10 – Index screen entity containers

Figure 6.10 shows the same page as in Figure 6.9, except with the ‘Modules’ and ‘Students’ containers expanded. The first thing to note is settings cog shown in the top right of the modules container. Also shown in the courses container, this is only shown for staff members, and leads to a module/course view configuration page for that user.

Another design choice shown is the alternating table row colours, to help distinguish between different items – this persists in other tables in the application as well.

A final thing to notice is the pagination present, where the items are limited to display only 5 at a time. This is important for when a lot of entities are present e.g. see the 44 pages of students. This helps reduce the clutter and makes the views consistent regardless of the number of items to view.

A screenshot of a cell phone

Description generated with very high confidenceA screenshot of a computer

Description generated with very high confidenceAt the bottom of the index page is the attendance upload functionality for staff members. Uploads can be added, with a file upload and module choice for multiple uploads at once, and Figure 6.12 shows the results after upload.

Figure 6.12 – Index screen attendance upload results

Figure 6.11 – Index screen attendance upload form

Module Page – Attendance

A screenshot of a cell phone

Description generated with very high confidenceThe single module view defaults to the attendance tab, and the attendance table shows the attendance per lecture for each student, with individual summaries in the last cell. The attendance charts are also shown below.

Figure 6.13 – Single module attendance view (staff user)

A close up of a map

Description generated with high confidence

Figure 6.14 – Single module attendance charts view

A screenshot of a cell phone

Description generated with very high confidenceThe view as a student shows only the current student’s attendance. Another difference to note is the colour usage. This screen shows what is displayed for a user with default settings, while the previous screens showed what a user with colour-blind options enabled sees.

Figure 6.15 – Single module attendance view (student user)

Module Page – Feedback

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.16 – Module feedback tab (staff user)

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.17 – Module feedback submission

A screenshot of a social media post

Description generated with very high confidence

Figure 6.18 – Module feedback tab (student user)

Figure 6.18 shows the feedback submitted in Figure 6.17, and the ‘Give Feedback’ button which is only present for student users. One thing to notice here is that the line breaks are preserved from submission.

Other single entity pages

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.19 – Single course view

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.20 – Single lecturer view (table)

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.21 – Single lecturer view (charts)

A screenshot of a social media post

Description generated with very high confidence

Figure 6.22 – Single student view

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.23 – Single lecture view

The other single entity views follow in the same fashion (shown in Figures 6.19 to 6.23), mostly adhering to what was laid out in the designs.

Settings

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.24 – Settings view

The settings page is similar to the design, except for the addition of an ‘attendance’ section which allows for attendance ranges to be defined which set colour coding on attendance averages to highlight different ranges of attendance to quickly gather information at a glance.

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.25 – Module / Course settings view

The module/course view settings, available to staff users, allows staff members to define which modules and courses they see and receive reporting on.

Mobile-device views

A screenshot of a cell phone

Description generated with very high confidenceA screenshot of a cell phone

Description generated with very high confidence

Figure 6.27 – Module attendance mobile view

Figure 6.26 – Index mobile view

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.29 – Settings mobile view

A screenshot of a cell phone

Description generated with very high confidence

Figure 6.28 – Module feedback mobile view

Figures 6.26 – 6.29 show some of the views as displayed in mobile-device dimensions, as provided by Google Chrome’s developer tools. The site remains responsive by maintaining proportions and displaying content appropriately. Tables that are too wide, e.g. module attendance with a dynamic number of lectures, become horizontally scrollable, and scrolling is minimised where possible to keep the content within the view at all times. The charts are one area where the responsiveness is limited due to the nature of Google Charts, which requires some custom Javascript to achieve. This could be considered for future improvements, but in the current system this only affects the charts, and the chart can still be viewed by scrolling across. The rest of the content on these pages remains responsive and is unaffected by the charts.

## Data Modelling

### Database Design

Figure 6.30 below shows the tables that the models in the final system would map to.

A screenshot of a video game

Description generated with high confidence

Figure 6.30 – Database tables design

Students and staff objects have a 1-1 relationship with a user, with any additional fields for each entity fitting neatly into their respective tables. Each user also maps 1-1 with a settings object so that each user can have custom settings tailored to suit their individual needs. As staff members can choose the modules and courses they see, there is a many-to-many relationship with both of these tables, which were to be expressed via associative tables ‘staff\_courses’ and ‘staff\_modules’. A similar relationship is shown between courses and modules, and also between students and modules. Lectures are each linked to one module, and student attendances each have one student and one lecture, plus the value of whether the student actually attended that lecture. Finally, each instance of module feedback is linked with one student and one module each.

### 6.2.2 Data Models

The database tables shown in figure 6.30 show the design of how the data would be modelled normally in a database, but the data representation in the final system was actually managed entirely through the code. This was possible because of Django’s models system, which abstracts away the need to directly interface with and manage the database tables or build any SQL queries. For example, the staff-modules relationship that was previously discussed was achieved by simply linking the staff and module Django models together with a ManyToManyField relationship. This allows us to directly grab the models and courses for a staff user in the code without having to make queries to look up the modules and courses that map to that staff member. This ORM behaviour supplied by Django made it easier to make changes to data structures during development and its auto-generated migration scripts allow for changing the type of database without needing to worry about the minutiae.

The real database tables generated by Django for the final system are shown in Figure 6.31. Additional helpful tables are generated and maintained automatically by Django – one of its many useful features. Tables such as django\_session, used to hold details on user sessions, help provide additional functionality like session management right out of the gate.

A black and silver text on a screen

Description generated with high confidence

Figure 6.31 – Final database tables generated by Django

# 7 – Implementation

## 7.1 Architecture Overview

### 7.1.1 Architecture

A high-level overview of the architecture consists of the system code and some interaction with other entities or external services i.e. the database and external Javascript.

The code written with the Django framework can be looked at as resembling an MVC pattern (model-view-controller). Django’s models abstract all direct interaction with the database into objects in the code, known as models, and these play the role of the model in the pattern as well, handling the representations of data and any functions that relate to the data. The views in Django handle the business logic in the application and provide a way to act on the data from the model and pass the necessary processed information to the view (in MVC pattern). This makes the Django views, somewhat confusingly, play the role of the controller in MVC. The Django URL resolving, defined by urls.py, also plays a part in the ‘controller’ role, defining which Django view is resolved by the visited URL. Finally, the Django templates play the role of the MVC views. These templates receive any data that’s needed to present to the user via the controller and ideally only deal with logic that’s related to how the information should be presented. Figure 7.1 shows a high-level overview of the architecture of the system.

A screenshot of a cell phone

Description generated with very high confidence

Figure 7.1 – Architecture overview

For the Google charts Javascript usage, it was required that this remained external as Google’s terms of service do not allow this code to be downloaded for local use (Google, 2017). There is some consideration required as to the security implications of using 3rd party services in a live application, removing the ability for the application to be fully autonomous. This introduces the possibility that the code found at this URL (<https://www.google.com/jsapi>), assumed to be the Javascript code for the application’s charts functionality, instead points to something malicious at some point in the future. The risk of this is mitigated by 2 concepts. Firstly, the script is accessed via HTTPS, meaning that the contents are encrypted in transit and are protected from tampering. Secondly, as Google-provided code, it’s unlikely that the script will be changed to something malicious any time soon. As a bonus, this code will be kept up-to-date with any bug fixes or improvements made to the libraries.

### 7.1.2 Additional libraries used

For the front-end implementation, Bootstrap (<https://getbootstrap.com/>) was used for additional CSS and JS functionality. jQuery (<https://jquery.com/>), Chosen.js (<https://harvesthq.github.io/chosen/>), and Pagination.js (<http://pagination.js.org/>) were used for additional JS functionality as well. Font Awesome (<https://fontawesome.com/>) was used for certain icons. For Python/Django, Django Graphos (<https://github.com/agiliq/django-graphos>) was used for chart functionality to pass a chart to the HTML to be interpreted by Google Charts (<https://developers.google.com/chart/> and <https://www.google.com/jsapi>). Xlrd (<https://github.com/python-excel/xlrd>) was used to parse excel files on attendance uploads and the Django Encrypted Model Fields library (<https://github.com/lanshark/django-encrypted-model-fields>) was used to provide encryption for user data. Finally, Selenium WebDriver (<https://github.com/SeleniumHQ/Selenium>) was used for automated testing purposes for the system.

## 7.2 Code Overview

Some code snippets will follow which show how some requirements were addressed or other interesting snippets.

Requirement F01

Requirement F01 addresses the need to upload data in CSV/Excel format. Firstly, Code Listing 7.1 shows the uploaded file from a form being validated on file type and passed to the appropriate method for parsing the uploaded data.

A screenshot of a social media post

Description generated with very high confidence

Code Listing 7.1 – File upload code fragment

Code Listing 7.2 shows the ‘DataSaver’ class where this information is passed to, which performs any further data re-structuring before passing it on to a common function that saves the uploaded data, independent of where the data originally came from.

A screenshot of a social media post

Description generated with very high confidence

Code Listing 7.2 – File upload file type saving code fragment

Requirement F02

Requirement F02 relates to the desire to overwrite existing attendance data where appropriate rather than appending duplicate data when duplicates may be uploaded. Code Listing 7.3 shows a code fragment related to this requirement, which checks for each piece of data involved whether an existing object already exists. In the case that the uploaded data already maps to data stored in the database, this instance will be identified, and the stored data will be overwritten such that it will remain the same if there are no changes but allows for data overwrite if required.

A screenshot of a social media post

Description generated with very high confidence

Code Listing 7.3 – Duplicate data detection code fragment

Requirement F03

Requirement F03 addresses the need for attendance data to be uploaded in the format output by the university’s systems. Figure 7.2 shows an example of this data, in an excel spreadsheet, for reference.

A screenshot of a video game

Description generated with high confidence

Figure 7.2 – University attendance data example

To handle this, 2 classes were used to parse the data. One was used for the attendance column headers, which gives information about the lectures for this attendance data. The other was used for the attendance data itself, including the student that the attendance values were for. Code Listings 7.4 and 7.5 show the code for each of these data rows respectively.

A screenshot of a social media post

Description generated with very high confidence

Code Listing 7.4 – Attendance session row parser code fragment

A screenshot of a social media post

Description generated with very high confidence

Code Listing 7.5 – Attendance data row parser code fragment

Requirement F09

Requirement F09 relates to visual representations of the attendance data. Code Listing 7.6 shows the code used to build a pie chart from the Django Graphos library, which is later passed to the template and displayed using the Google Charts Javascript libraries. Also shown in this code is some user-dependent configuration, relating to the user’s colour-blind settings.

A screenshot of a cell phone

Description generated with very high confidence

Code Listing 7.6 – Pie chart building code fragment

Code Listing 7.7 shows the code used to create a line chart for the single module view, with the same interactions with external libraries later down the line.

A screenshot of a social media post

Description generated with very high confidence

Code Listing 7.7 – Line chart building code fragment

Requirement F10

Requirement F10 relates to the email reporting that users can receive regarding attendance information on their modules. This was achieved with the use of a Django command, which allows specific code to be invoked from the command line and provides facilities for supplying arguments to configure how the command is interpreted. With this, the email reporting implementation was able to be isolated from the rest of the system, provided easier access for unit testing coverage, and allows for a cron job or something similar to schedule the reporting instead of requiring a complicated setup to deploy code etc. Code Listing 7.8 shows some of the metadata and configuration options relating to the command, and Figure 7.3 shows the corresponding output of the ‘--help’ argument supplied for this command.

A screenshot of a social media post

Description generated with very high confidence

Code Listing 7.8 – Email command arguments code fragment

A screenshot of a cell phone

Description generated with very high confidence

Figure 7.3 – Email attendances command ‘help’ output

Code Listing 7.9 shows how some of these arguments are used to pick up options such as the type of user to send to or the time range, and then opening a connection to send off the emails.

A screenshot of a social media post

Description generated with very high confidence

Code Listing 7.9 – Email command arguments usage code fragment

Code Listing 7.10 shows the actual sending of the email further down the chain, where a text template is used and markers relating to the user, attendance values etc. are inserted to prepare the email for sending to a specific user.

A screenshot of a social media post

Description generated with very high confidence

Code Listing 7.10 – Email command send email code fragment

Code Listing 7.11 shows the email template in which values are inserted so that the emails are custom for each user’s attendance information.

A screenshot of a social media post

Description generated with very high confidence

Code Listing 7.11 – Email attendance report template

Module Feedback

The module feedback functionality allows students to give feedback for modules they undertake, and for staff to view that feedback. To submit that feedback, the students need a form, and this is one example of where Django’s built-in form functionality helped remove some of the mundane form building while again providing strong support for unit testing. Django’s forms allow for mapping a model to a form while providing some flexibility on the fields shown, HTML classes to add to fields etc. This can be shown in Code Listing 7.12, where the module feedback data model is mapped to a form which can then be passed to the Django template and automatically translated to HTML.

A screenshot of a cell phone

Description generated with high confidence

Code Listing 7.12 – Module feedback form code fragment

Usage of this form can be seen in Code Listing 7.13, where the data is passed to the form and saved via the form object.

A screenshot of a cell phone

Description generated with very high confidence

Code Listing 7.13 – Module feedback save code fragment

## 7.3 Obstacles Encountered

During the implementation of the system, some parts of development work presented more difficulty than others. Shown below is a few examples of these challenges and the code that was used to solve them.

User Encryption

While Django has built-in support for hashing user passwords, there is a lack of support for encryption/decryption for regular fields. To achieve the goal of encrypting user’s names and email addresses, the ‘Django Encrypted Model Fields’ library was used (<https://github.com/lanshark/django-encrypted-model-fields>).

Code Listing 7.14 shows the new EncryptedUser model that was created. Previously, the project was referencing Django’s built in User model, and these references were all updated to use EncryptedUser instead. This model inherited from Django’s AbstractUser base class and overrode the name and email fields with encrypted equivalents. However, this created a problem where users couldn’t be directly looked up via their email (as the data was now encrypted), which was needed for Django’s built-in password reset functionality that the system was using. The get\_by\_email() function was added to this model class to achieve this required functionality, which simply iterated through all users in the system and comparing the emails in its decrypted state.

A screenshot of a social media post

Description generated with very high confidence

Code Listing 7.14 – Encrypted user model code fragment

Now there was an alternative for fetching users by email address, but this needed to be plugged in to our password reset functionality which was previously fully supplied by the Django libraries and out of sight. To solve this issue, we would again override Django’s built-in library code, but this time target the password reset form. Code Listing 7.15 shows this new form, which overrides the get\_users() method from Django’s PasswordResetForm class, allowing the password reset functionality to send an email to the correct user upon a password reset request.

A screenshot of a cell phone

Description generated with very high confidence

Code Listing 7.15 – Password reset form override code fragment

Finally, Code Listing 7.16 shows our new form being used for our password reset view instead of the default, giving us full functionality with encrypted user data.

A screenshot of a cell phone

Description generated with very high confidence

Code Listing 7.16 – Password reset view form override code fragment

Multiple attendance uploads

Requirement F04 relates to the desire to be able to upload multiple attendance data files at once. While implementing this functionality, the biggest obstacle was representing this on the UI, as each upload required some manual input from the user to indicate to the system which exact module that the attendance data was for. This meant that the form used to upload attendance data would be more complex than the typical form, with a need for repeatable sections that would each allow for attendance data file upload and module selection.

To tackle this problem, it was decided to create a placeholder upload row in the HTML which could then be acted upon from the Javascript to allow for new rows to be added based on this placeholder template. Code Listing 7.17 shows the HTML for the upload form, with the placeholder upload shown which contains a file upload input and a module select.

A screenshot of a social media post

Description generated with very high confidence

Code Listing 7.17 – Multiple file upload placeholder HTML fragment

The next step was to handle new upload row creation based on this placeholder in the Javascript. This code can be seen in Code Listing 7.18 where the placeholder is cloned and its attributes overwritten with indexed ids or names to sequence and uniquely identify the rows. On deletion of any row, the remaining rows would be re-indexed to maintain an unbroken sequence to make processing easier for the backend. Code Listing 7.19 then shows how the attendance data is handled on submission, which manually adds the file uploads to the JS form data and makes an AJAX request to upload the attendance data while a progress spinner showed that the files were being processed. This allowed for a more user-friendly experience where the user could receive upload validation messages without a page refresh. Finally, the file submissions could just be iterated over and processed at the backend to handle multiple uploads.

A screenshot of a cell phone

Description generated with very high confidence

Code Listing 7.18 – Add file upload row code fragment

A screenshot of a cell phone

Description generated with very high confidence

Code Listing 7.19 – Submit attendance data code fragment

# 8 – Testing & Evaluation

## 8.1 Testing

### 8.1.1 Unit Testing

SWEBOK v3 defines unit testing as ‘[verifying] the functioning of software modules in isolation from other software elements that are separately testable’ (IEEE Computer Society, 2014). During the development of the system, unit tests were added alongside each piece of new functionality to maintain the correctness of the system and confidence that regression bugs would be kept to a minimum.

One of the benefits provided by using Django as a web framework was its extensive support for unit testing. Django’s testing framework allowed for testing views by mocking HTTP requests and returning responses to validate, which meant that tests could be created for views once, ran often, and give a look at the health of the system very quickly. These tests can cover anything from the expected content returned to the type of response e.g. success, redirect etc. Each of the views in the system has tests to cover the expected content per user type, how the system reacts when an unauthorised user attempts to visit the page, how any form submissions are handled from the backend with unexpected data, and any specific functions that are expected of the view. For the final system, there were 129 such tests in total, running in sequence for ~55-60 seconds each time. Appendix 2 gives a breakdown of these tests and what functionality each test covers.

To ensure that these tests were ran when needed, Travis CI was integrated with the project. Travis is a CI (continuous integration) build tool that helps with automated testing and deploys (Travis CI, 2018). For each commit and pull request made to the Github repository for this project, Travis would pick up these changes and run tests based off a travis.yml file at the root of the project which tells Travis what language the project is, how to install dependencies and how to run the tests. The result of this was automatic unit test builds started on each change to the repository, ensuring visibility when some functionality would cause regression bugs. Figure 8.1 shows how this appeared on Github, displaying a pass/fail indicator alongside commits for a pull request when adding changes to the codebase.

A screenshot of a social media post

Description generated with very high confidence

Figure 8.1 – Travis CI automated unit testing

### 8.1.2 System Testing

SWEBOK v3 defines system testing as ‘[being] concerned with testing the behavior of an entire system’ (IEEE Computer Society, 2014). When the system was in early development stages, it was sufficient to test this manually as there were few components and the complexity of the system hadn’t been fully realised yet. As the system came closer to completion, a small number of automated system tests were added to remove human error and make validation for the whole system a lot faster. These tests were implemented using Selenium WebDriver’s python library, which makes it easy to spin up an instance of a browser of choice and interact with it, e.g. going to a URL and handling common fields such as buttons and text boxes. To build on this, Django offers a test case class for test classes to inherit from, ‘StaticLiveServerTestCase’, which starts the application on an unassigned port with a fresh database for automated system testing. The power of Selenium coupled with Django’s extensive testing support allows for a brand-new instance of the application to be created and interacted with in a live browser, making test assertions along the way.

The automated system tests were chosen to run against Firefox, as Chrome was the main browser used during development and debugging. By choosing Firefox, it meant that a different browser was accounted for as it relates to cross-browser compatibility. 2 such automated tests were created: one for testing the whole system functionality as a staff user, and the other for testing the system as a student. These tests were able to test the full path of the system for the 2 different user types in just over a minute, ran sequentially.

### 8.1.3 Manual Testing

After the implementation of the automated system tests, the bulk of manual testing came from debugging any new code in the browser before the automated tests would be updated to reflect the changes if necessary. By the end of development, manual testing was minimal as any substantial changes could be checked by the selenium tests.

### 8.1.4 Compatibility Testing

When testing on different browsers, the browser market share was considered to ensure that the most widely-used browsers were supported by the system. Figure 8.2 shows the statistics for the most-used browsers recently, and these browsers were targeted for compatibility testing (Chrome, Firefox, Edge, & IE) (NetApplications.com, 2018).

A close up of a map

Description generated with very high confidence

Figure 8.2 – Web browser market share statistics (via <https://netmarketshare.com/browser-market-share.aspx>)

Chrome is the most-used browser by a substantial margin, and this is where most of the testing was done during development giving it the most confidence that the system will behave as expected here. As previously mentioned, Firefox was also used to run the automated system tests, giving it good test coverage as well. Edge and Internet Explorer 11 were tested on after completion to give the same results as Chrome and Firefox, and comparison screens can be seen in Figure 8.3. Also shown in Figure 8.3 is the same comparison done for a mobile-sized screen, shown through the Google Chrome developer tools.

A screenshot of a computer

Description generated with very high confidence

Figure 8.3 – Cross-browser compatibility screens (from left to right, top to bottom: Chrome, Firefox, Edge, IE11, mobile device (simulated by Chrome dev tools)

## 8.2 Evaluation

### 8.2.1 Evaluation of technologies used

What follows is an evaluation of the technologies used during the development of the system, and their appropriateness for the task required. Overall, the developer was very pleased with the technologies and tools used to achieve the project’s goal. The chosen technology stack helped to bootstrap many common mundane development tasks and allowed for the time spent developing and testing the system to be spent effectively.

IDE – PyCharm

PyCharm’s intelligent code completion, Python formatting help and other checks like adherence to PEP-8 made development a lot easier to see what was going on and to avoid and solve hard-to-find errors. An example of where PyCharm helped with this is with the use of the word ‘module’, a key word in Python/Django. As this system dealt with modules in the academic sense, naturally a lot of variables were named after this and in some cases if a parameter was forgotten, PyCharm would highlight that the keyword was being used rather than a regular variable. Auto-import of required libraries also helped to find the required functionality of external libraries as well.

Development language – Python

This was the developer’s first real experience with using Python, and it gave minimal trouble in getting up to speed with it due to its simplicity. A lax syntax meant that early development time wasn’t wasted on having to deal with the burden of where braces and parentheses should go, and pip (a package management system for Python) was simple to use as well. Pip came in especially handy for 2 situations. The first was the choice to change development machine a couple of months into development, and this allowed for the required dependencies to be installed on the new machine quickly and hassle-free. The second was for the automated build tests, where each build required Travis to set up the project on what was essentially a fresh machine. This allowed the dependencies to be set up for these tests painlessly.

Web framework – Django

Django provided a web development framework for Python, and it was definitely the correct tool for the job. Django’s models removed the pain of interacting with the database directly, and its in-built handling for common web application requirements such as user authentication and login / password reset forms allowed the common tasks to be handled quickly and correctly by Django so that the system-specific requirements could tackled and implemented.

Database - SQLite

For the development of the system, SQLite was a great choice for local development and testing. As a lightweight database with no additional programs required, it meant that no configuration was required when changing development machine or when running automated builds with Travis. An SQLite database backend was the perfect choice for development purposes.

CI tool – Travis CI

Travis was a great way to run automatic unit tests for this system. Free for open source projects, Travis only required the repository to be linked with an account on the Travis website and a short script placed in the root of the project to tell Travis how to install the required dependencies and run the needed tests. At the time of writing, there were exactly 100 builds ran in total over the course of the project, and in that time the build script only changed once from the initial test script due to some project changes (project build history can be seen here: <https://travis-ci.org/nbrowning1/FYP/builds>). Travis required no maintenance unless the project changed significantly enough to require the build setup to change, while continuing to work and spot any issues with the project code as it changed over time.

### 8.2.2 Evaluation of system

The bulk of the system appraisal came from the project’s supervisor, Professor Colin Turner. As a member of staff at the university and the head of the school of engineering, Professor Turner initially helped to craft the requirements of the system and was the most appropriate person to evaluate whether the system did its job correctly.

As the project went on, multiple meetings were held with Professor Turner to evaluate the progress of the system’s development and to demo new functionality: both how it looked in the UI and the actual code itself. One of the ideas that came out of an early meeting like this was the ability for users to set the attendance ranges for colour-coding attendance value percentages e.g. setting the range for what was considered ‘very low’ so that it could be highlighted as such in the UI. Professor Turner’s input was also valuable for some design choices. Initial designs had the sign out link in the top left of the index page, which is in the same position as the ‘Home’ button on other pages. It was pointed out that this could cause accidental logouts, and this design was changed to avoid this issue, with the ability to sign out from any page in the application. Views for looking at a student’s information were also slightly modified to allow multiple modules in the accordion to be opened at once, so that the modules could be compared. Insight like this from a staff member was necessary for the system to be validated, especially as most of the functionality for the application was specific to staff users.

The system was positively received by the project supervisor and other students as peers of the developer, from informal sessions during which the system was actually used and explored. The few potential issues raised by Professor Turner were addressed and the system was validated by these test users. In addition to this, some extra future improvements were suggested that would require larger effort such as mass uploading of new users, but this will be covered in more detail in Chapter 9.

# 9 – Results

## 9.1 Fulfilment of project objectives

For each functional and non-functional requirement, a status will be assigned to it to determine whether the objective was fulfilled, partially fulfilled or unfulfilled. As the requirements lay out what was expected of the initial system, this is a way to objectively evaluate to what extent the project can be considered completed, and its goals achieved.

### 9.1.1 Fulfilment of functional requirements

Table 9.1 – Functional requirements evaluation

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Description | Status | Comments |
| F01 | Staff members shall be able to upload attendance data for a specific module in CSV/Excel format | Fulfilled | Discussed in more detail in Section 7.2. Unit test coverage in Appendix 2 – tests 108-110 |
| F02 | Attendance data that is re-uploaded shall overwrite existing data rather than append duplicate data | Fulfilled | Discussed in more detail in Section 7.2. Unit test coverage in Appendix 2 – test 111 |
| F03 | Attendance data shall be able to be uploaded in the university’s format | Fulfilled | Discussed in more detail in Section 7.2. Unit test coverage in Appendix 2 – tests 20, 24, 108-110 |
| F04 | Attendance data shall be able to be uploaded for multiple files at a time | Fulfilled | Discussed in more detail in Section 7.3. Unit test coverage in Appendix 2 – test 117 |
| F05 | Staff members shall be able to choose the modules and courses that they want to view on their dashboard and to receive email reporting on | Fulfilled | Unit test coverage in Appendix 2 – test 76 and other related tests |
| F06 | Students shall be able to view only the modules and lectures that they have attendance data for on their dashboard and in email reporting | Fulfilled | Unit test coverage in Appendix 2 – tests 36, 89, 105, 128 |
| F07 | All users shall be able to change their password and request a password reset | Fulfilled | Bulk of support supplied by Django. Also, unit test coverage in Appendix 2 – test 18 |
| F08 | Staff members shall be able to add modules, courses, and other student / staff users | Fulfilled | Unit test coverage in Appendix 2 – tests 2-14 |
| F09 | All users shall be able to view appropriate visualisations of the attendance data | Fulfilled | The complex SVGs generated by the Google Charts library meant this was manually tested for the system. Should already be thoroughly tested by proxy as a Google library. Discussed in more detail in Section 7.2. |
| F10 | All users shall be able to receive email reporting to their registered email address to inform on their attendance over a weekly or monthly period | Fulfilled | Discussed in more detail in Section 7.2. Unit test coverage in Appendix 2 – tests 34-37 |
| F11 | Students shall be able to give feedback on modules that they have attendance data for, and view only their feedback given for that module | Fulfilled | Discussed in more detail in Section 7.2. Unit test coverage in Appendix 2 – tests 42-46 |
| F12 | Students giving feedback shall be able to choose whether their feedback is anonymous or not | Fulfilled | Unit test coverage in Appendix 2 – test 44 |
| F13 | Staff members shall be able to only view feedback for a given module | Fulfilled | Unit test coverage in Appendix 2 – tests 41, 88 |

### 9.1.2 Fulfilment of non-functional requirements

Table 9.2 – Non-functional requirements evaluation

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Description | Status | Comments |
| N01 | Newly-created user accounts shall be initialised with a random password to force a password reset request for the registered email address | Fulfilled | Unit test coverage in Appendix 2 – tests 9, 12 |
| N02 | Passwords shall be forced to meet certain restrictions, including length checks and not being too common | Fulfilled | Uses Django’s built-in password management, which include length check (8 minimum), common passwords check among other things (Django Software Foundation, 2017) |
| N03 | Passwords shall be hashed when stored so that they cannot be read or translated to plaintext | Fulfilled | Uses Django’s built in password management, which uses PBKDF2 with SHA256 (Django Software Foundation, 2018) |
| N04 | The system shall be accessible by users who are colour-blind: any information that is presented with the use of colour, e.g. graphs, shall be equally accessible by such users | Fulfilled | Some of this shown in screens in Section 6.1.3. Unit test coverage in Appendix 2 – tests 82-84 |
| N05 | The styling of the application shall remain consistent throughout | Fulfilled | Shown in screens in Section 6.1.3 |
| N06 | The application shall be displayed on all major web browsers and devices as equally as possible, such that all information is conveyed in an appropriate way | Fulfilled | Discussed in more detail in Section 8.1.4 |
| N07 | The system should hide information to unauthenticated users, and hide irrelevant information based on user | Fulfilled | Unit test coverage in Appendix 2 – tests 1-2, 33, 39, 41-43, 73-74, 80-81, 88-90, 92-94, 96-98, 100-102, 104-106, 119-120, 123, 125-128 |

### 9.1.3 Additional work completed

During the development of the system, the developer realised that even when the attendances for data was tabulated, e.g. to display each student’s attendance in a module, the overall attendance percentages for the students in that module had nothing to distinguish or rank them. These attendance percentage cells were modified to be colour-coded based on the value, with 0-25% being red, 25-50% being orange, 50-75% light green, and 75%+ a darker green. This made it easier to parse at a glance as to how each student was doing. To build on this, the project supervisor suggested the ability to adjust these range values as a setting e.g. so that <50% values could be displayed as the lowest range etc. This extra work was completed in full, with unit test coverage in Appendix 2 – tests 85-87.

# 10 – Conclusion

## 10.1 Reflecting on the project

### 10.1.1 Project scope

The developer believes the project’s scope was appropriate, as development of the system was completed on time and all the initial objectives were met. The Gantt chart used in Appendix 1 was a great way to schedule the initial tasks, even if it wasn’t used after the beginning of the project. Making the initial estimates and having to schedule them in the Gantt chart revealed whether everything was doable in the time frame allotted and gave an initial structure to the work to carry out to complete the project. Early plans included the development of a dedicated mobile application to link with this system as well, and the time estimates/scheduling showed that this would have been too difficult to achieve in the time provided.

One area in which there could’ve been some room for improvement was in the amount of capacity estimated by the developer to work on the project. With work and other commitments, the planned capacity of 2-3 man-days a week was infeasible, and the project buffers helped to mitigate any issues arising from this.

A suggestion for future students undertaking a similar project is to take a realistic look at how much time they may have to spend on the project, and to leave a healthy buffer in the schedule as well. Circumstances can change drastically over several months, and everybody needs time off now and again.

### 10.1.2 Products produced

The developer is proud of the system developed for this project, picking up a language and web framework that they had no prior experience in and creating a fleshed-out application from the client through to the back-end, with carefully designed data modelling, email reporting for custom reporting per user, command line operations to manage aspects of the system, and concise, well-documented code which was covered by extensive unit tests to ensure a robust product. As shown in the results, each requirement expected of the system was met rigorously, using all appropriate technologies to provide an excellent user experience from the design of the UI supported by client-side scripting through to the back-end, and each verified with thorough unit testing to culminate in a complete system that is strong the whole way through. In addition, some extra work was completed for the project in the attendance range highlighting functionality. The system gained praise in the meetings used to evaluate it in functionality and design. As a fleshed-out proof of concept, this system gave good insight into what’s achievable and its usefulness in this area.

A suggestion for future students undertaking a similar project is to use it as an opportunity to learn new technologies. In the world of software engineering a wider breadth of knowledge is a massive boon, and it may reveal useful technologies to help in their future endeavours.

### 10.1.3 Managing the process

The developer believes that the process was managed very well. While the project met all of its goals, it was also a well-managed software project. Project documentation, an extensive suite of unit tests, use of version control from the beginning, and continuously running these tests on any changes to the repository made a software project that would be fit for other developers to jump in on and collaborate on. Figure 10.1 shows the build link in the repository, with a link to the Travis build status image meaning that the state of the system can always be quickly assessed by seeing how the most recent build was built. Figure 10.2 shows the project’s README file in the repository that was used to start up on new development machines quickly, and which could be used by other developers to start the system as well.

A close up of a mans face

Description generated with high confidence

Figure 10.1 – Github repository build status

A screenshot of a cell phone

Description generated with very high confidence

Figure 10.2 – Github repository README documentation

The use of Github issues to manage the development tasks gave an insight into where the project was at and where it needed to be at any stage and allowed for the code to be tracked as the codebase grew as each issue was linked with the code that fulfilled the requirements. Figure 10.3 shows some of the Github issues used to track feature development, and Figure 10.4 shows one of these issues in more detail including information about its implementation after development was completed. Figures 10.5 and 10.6 show the same for pull requests which helped group all the code for a particular issue in one place and into one merge commit which could be reverted later if necessary.

A screenshot of a cell phone

Description generated with very high confidence

Figure 10.3 – Github repository issues

A screenshot of a cell phone

Description generated with very high confidence

Figure 10.4 – Github repository closed issue

A screenshot of a cell phone

Description generated with very high confidence

Figure 10.5 – Github repository pull requests

A screenshot of a cell phone

Description generated with very high confidence

Figure 10.6 – Github repository merged pull request

A command that was used to set up test data also meant that fundamental changes could be made at any point during development and a hefty amount of test data could be instantly loaded into a fresh database to simulate the real system.

Meetings scheduled with the project supervisor on a regular basis allowed the supervisor to stay up to date with all developments that had been made in the project and provide feedback, which is where an Agile process thrives. Small modifications could be made here and there as needed and some new functionality was added during the project as discussed in the previous chapter.

A suggestion for future students undertaking a similar project would be to prioritise unit testing and try to automate as much manual work as possible. Automated test builds were especially useful in this project and enforced the integrity of the system without relying on the developer to kick off tests manually.

## 10.2 Suggestions for future improvements

As with any project, there are always improvements that can be made: new functionality, increased effectiveness or efficiency, integration with other systems etc.

Focusing on the most important functionality first, the developer would re-work the uploading to automatically detect the module from the attendance data rather than have it selected via a dropdown. This would make the attendance data uploads smoother by only requiring file uploads. Another improvement in this area could see over-written attendance data give this information back to the user to ensure that this was intentional, as currently the system will overwrite data where appropriate but gives no feedback to the user regarding these changes.

To make the application more data-focused, the addition of the ability to sort tables could be a useful future improvement. Being able to sort by e.g. attendance of students in a module could help in finding the kind of data that a staff member might be looking for. Search functionality in multiple areas of the system could also benefit users to narrow down on particular criteria.

As far as system maintenance is concerned, new user additions could allow bulk changes instead of the current one-at-a-time approach. This could be implemented similarly to the attendance data uploads, with the information coming via way of a file with student codes and names etc.

Integrations with other systems would benefit this system. As a system designed for a university, there are applications already commonly in use by students and staff in the university which the system could be integrated into e.g. Blackboard. Integration with university systems to automatically create users for existing users in the system would also provide convenience so that the system wouldn’t need to be manually maintained in terms of its users.

# References

Aronin, S. & Smith, M., 2016. *One in four students suffer from mental health problems.* [Online]   
Available at: https://yougov.co.uk/news/2016/08/09/quarter-britains-students-are-afflicted-mental-hea/  
[Accessed 24 October 2017].

Brikman, Y., 2013. *What are the pros and cons of Play Framework 2, for a Java developer?.* [Online]   
Available at: https://www.ybrikman.com/writing/2013/03/31/what-are-the-pros-and-cons-of-play-framework-2-for-a-java-developer/  
[Accessed 30 September 2017].

BuiltWith, no publication date given. *Web Technology Usage Trends.* [Online]   
Available at: https://trends.builtwith.com/  
[Accessed 3 October 2017].

Crestodina, A., 2015. *Web Design Standards: 10 Best Practices on the Top 50 Websites.* [Online]   
Available at: https://www.orbitmedia.com/blog/web-design-standards/  
[Accessed 3 October 2017].

Django Software Foundation, no publication date given. *Databases | Django documentation.* [Online]   
Available at: https://docs.djangoproject.com/en/2.0/ref/databases/  
[Accessed 16 November 2017].

Django Software Foundation, no publication date given. *Password management in Django.* [Online]   
Available at: https://docs.djangoproject.com/en/2.0/topics/auth/passwords/  
[Accessed 27 March 2018].

Django Software Foundation, no publication date given. *The Web framework for perfectionists with deadlines.* [Online]   
Available at: https://www.djangoproject.com/  
[Accessed 30 September 2017].

Google, 2017. *Frequently Asked Questions | Charts | Google Developers.* [Online]   
Available at: https://developers.google.com/chart/interactive/faq  
[Accessed 6 January 2018].

GOV.UK, no publication date given. *Data protection.* [Online]   
Available at: https://www.gov.uk/data-protection  
[Accessed 24 October 2017].

GradeXpert, no publication date given. *GradeXpert.* [Online]   
Available at: http://www.gradexpert.com.au/  
[Accessed 1 November 2017].

Gujar, P., 2016. *What are the advantages and disadvantages of using Django?.* [Online]   
Available at: https://www.quora.com/What-are-the-advantages-and-disadvantages-of-using-Django  
[Accessed 30 September 2017].

IEEE Computer Society, 2014. *SWEBOK V3.* [Online]   
Available at: https://www.computer.org/web/swebok/v3  
[Accessed 5 January 2018].

National Institutes of Health, 2015. *Facts About Color Blindness.* [Online]   
Available at: https://nei.nih.gov/health/color\_blindness/facts\_about  
[Accessed 1 October 2017].

NetApplications.com, no publication date given. *Browser market share.* [Online]   
Available at: https://netmarketshare.com/browser-market-share.aspx  
[Accessed 23 March 2018].

Pivotal Software, no publication date given. *Spring: the source for modern java.* [Online]   
Available at: https://spring.io/  
[Accessed 30 September 2017].

PlayFramework, no publication date given. *The High Velocity Web Framework For Java and Scala.* [Online]   
Available at: https://playframework.com/  
[Accessed 30 September 2017].

Scotland, K., 2009. *Agile | Fidelity – The Lost Dimension of the Iron Triangle.* [Online]   
Available at: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/  
[Accessed 22 October 2017].

Shaffer, J., 2016. *5 tips on designing colorblind-friendly visualizations.* [Online]   
Available at: https://www.tableau.com/about/blog/2016/4/examining-data-viz-rules-dont-use-red-green-together-53463  
[Accessed 5 October 2017].

SolutionPath, no publication date given. *Improve academic attainment, retention and support student well-being with StREAM.* [Online]   
Available at: https://www.solutionpath.co.uk/  
[Accessed 1 November 2017].

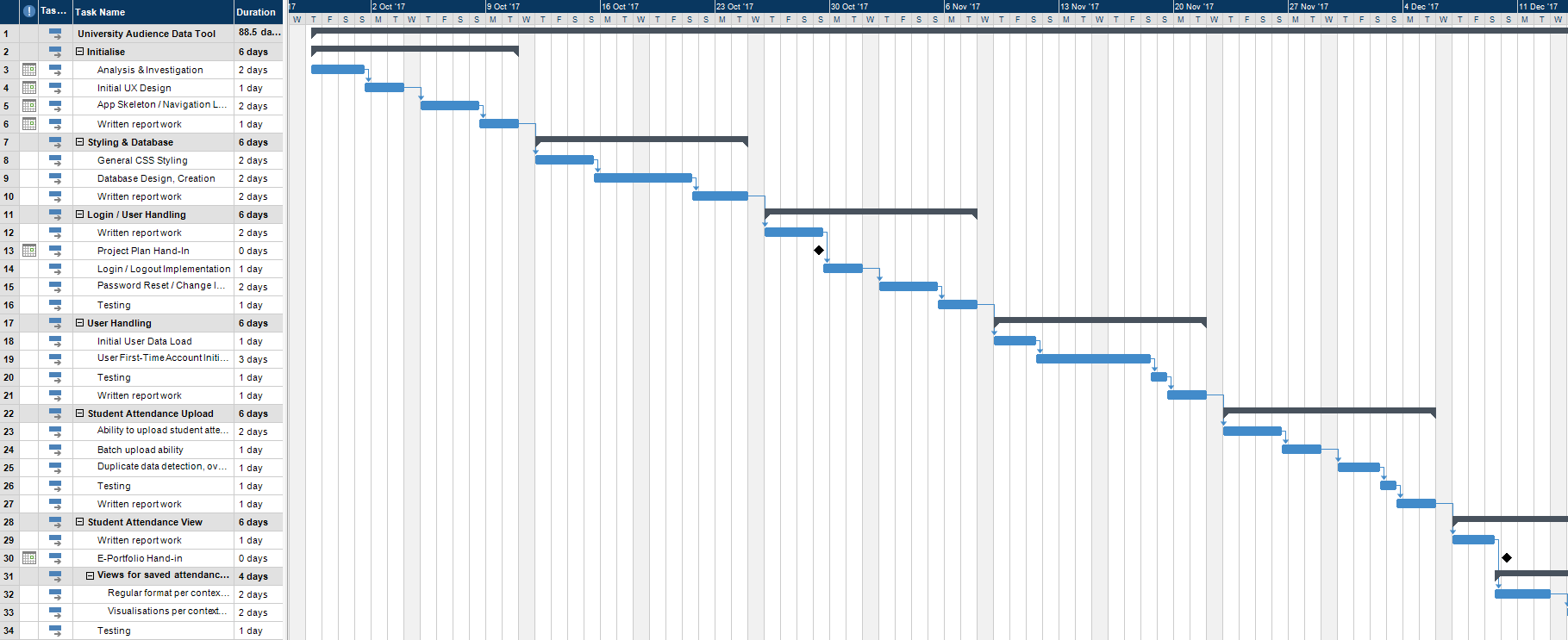
SQLite, no publication date given. *Appropriate Uses For SQLite.* [Online]   
Available at: https://www.sqlite.org/whentouse.html  
[Accessed 17 November 2017].

TeacherKit, no publication date given. *Over 1 Million Teachers are using TeacherKit.* [Online]   
Available at: http://teacherkit.net/  
[Accessed 1 November 2017].

Travis CI, no publication date given. *Travis CI - Test and Deploy Your Code with Confidence.* [Online]   
Available at: https://travis-ci.org/  
[Accessed 20 March 2018].

# Appendices

## Appendix 1 – Initial Gantt Chart



A picture containing indoor

Description generated with very high confidence

A screenshot of a computer

Description generated with high confidence

## Appendix 2 – Unit tests listing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Filename | Class name | Test name | Description |
| 1 | Tests\_admin\_views.py | GeneralTests | test\_unauthenticated | Tests unauthenticated user shouldn’t see all admin create views |
| 2 |  |  | test\_permissions | Tests that students shouldn’t see admin create views, while staff members should |
| 3 |  | CreateModuleTests | test\_create\_module | Tests creation of a valid module is successful |
| 4 |  |  | test\_invalid\_module | Tests creation of a module with invalid code fails |
| 5 |  |  | test\_module\_already\_exists | Tests creation of a module that already exists fails |
| 6 |  | CreateCourseTests | test\_create\_course | Tests creation of a valid course is successful |
| 7 |  |  | test\_invalid\_course | Tests creation of a course with invalid code fails |
| 8 |  |  | test\_course\_already\_exists | Tests creation of a course that already exists fails |
| 9 |  | CreateStudentTests | test\_create\_student | Tests creation of a valid student user is successful |
| 10 |  |  | test\_invalid\_student | Tests creation of a student user with invalid values fails |
| 11 |  |  | test\_student\_already\_exists | Tests creation of a student user that already exists fails |
| 12 |  | CreateStaffTests | test\_create\_staff | Tests creation of a valid staff user is successful |
| 13 |  |  | test\_invalid\_staff | Tests creation of a staff user with invalid values fails |
| 14 |  |  | test\_staff\_already\_exists | Tests creation of a staff user that already exists fails |
| 15 | Tests\_auth.py | AuthTests | test\_login\_valid\_credentials | Tests login with valid credentials for existing user is successful |
| 16 |  |  | test\_login\_invalid\_credentials | Tests login with invalid credentials for existing user is unsuccessful |
| 17 |  |  | test\_login\_nonexistent\_user | Tests login with for non-existent user is unsuccessful |
| 18 |  |  | test\_password\_reset\_recognised\_email | Tests password reset request with existing email results in email being sent |
| 19 |  |  | test\_password\_reset\_unrecognised\_email | Tests password reset request with unrecognised email sends no email |
| 20 | Tests\_data\_rows.py | AttendanceSessionRowTests | test\_valid\_row | Tests valid session row data is parsed correctly |
| 21 |  |  | test\_row\_invalid\_session\_format | Tests expected error message for session row data with invalid format |
| 22 |  |  | test\_row\_invalid\_date\_format | Tests expected error messages for invalid dates |
| 23 |  |  | test\_row\_empty | Tests expected error messages for various empty sections of session row data |
| 24 |  | AttendanceRowTests | test\_valid\_row | Tests valid attendance data row is parsed correctly |
| 25 |  |  | test\_student\_by\_username\_and\_device\_id | Tests attendance data row parsing can find student by student and code device ID |
| 26 |  |  | test\_invalid\_student\_message | Tests expected error message for invalid student |
| 27 |  |  | test\_invalid\_multiple | Tests expected error message for multiple invalid fields in a data row |
| 28 |  |  | test\_it\_trims\_spaces | Tests that spaces are trimmed from data fields when data row is parsed |
| 29 |  |  | test\_incorrect\_column\_count | Tests expected error messages for unexpected number of data columns compared to number of lectures |
| 30 | Tests\_download.py | DownloadTests | test\_download | Tests download of valid file is successful for staff member |
| 31 |  |  | test\_download\_nonexistent\_file | Tests download of non-existent file is unsuccessful for staff member |
| 32 |  |  | test\_download\_usertype\_permissions | Tests download is denied for student and allowed for staff member |
| 33 |  |  | test\_unauthenticated\_download | Tests download for unauthenticated student redirects to login |
| 34 | Tests\_email\_attendances\_command.py | EmailAttendanceReportTest | test\_staff\_monthly\_report | Tests email reporting to staff with month range |
| 35 |  |  | test\_staff\_weekly\_report | Tests email reporting to staff with week range |
| 36 |  |  | test\_student\_report | Tests email reporting to students |
| 37 |  |  | test\_all\_report | Tests email reporting to all users |
| 38 |  |  | test\_test\_only | Tests email reporting in ‘test’ mode doesn’t actually send emails |
| 39 | tests\_feedback\_views.py | ViewsTests | test\_unauthenticated | Tests redirect to login for unauthenticated user for this view |
| 40 |  |  | test\_nonexistent\_module | Tests 404 for URL for non-existent module |
| 41 |  |  | test\_staff | Tests staff user shouldn’t be able to visit page to give feedback |
| 42 |  |  | test\_student | Tests student user should be able to visit page to give feedback |
| 43 |  |  | test\_student\_unlinked\_module | Tests student user shouldn’t be able to visit page to give feedback for module they’re not linked with |
| 44 |  |  | test\_valid\_data | Tests submission of valid feedback, with and without anonymous option |
| 45 |  |  | test\_empty\_data | Tests validation for feedback submission with empty data |
| 46 |  |  | test\_too\_long\_data | Tests validation for feedback submission with long data |
| 47 | Tests\_forms.py | ModuleFormTests | test\_empty\_form | Tests validation for empty create module form |
| 48 |  |  | test\_valid\_form | Tests valid data for create module form |
| 49 |  |  | test\_invalid\_code | Tests invalid module code in create module form |
| 50 |  |  | test\_length\_check | Tests invalid lengths for data in create module form |
| 51 |  |  | test\_existing\_module | Tests validation for existing module in create module form |
| 52 |  | CourseFormTests | test\_empty\_form | Tests validation for empty create course form |
| 53 |  |  | test\_valid\_form | Tests valid data for create course form |
| 54 |  |  | test\_length\_check | Tests invalid lengths for data in create course form |
| 55 |  |  | test\_existing\_course | Tests validation for existing course in create course form |
| 56 |  | UserFormTests | test\_empty\_form | Tests validation for empty user form |
| 57 |  |  | test\_valid\_student\_form | Tests valid data for student user form |
| 58 |  |  | test\_invalid\_student\_form | Tests invalid data for student user form |
| 59 |  |  | test\_valid\_staff\_form | Tests valid data for staff user form |
| 60 |  |  | test\_invalid\_staff\_form | Tests invalid data for staff user form |
| 61 |  |  | test\_existing\_user | Tests validation for user form for existing user by username and email |
| 62 |  | StudentFormTests | test\_empty\_form | Tests validation for empty student form |
| 63 |  |  | test\_valid\_form | Tests valid data for student form |
| 64 |  |  | test\_invalid\_form | Tests invalid data for student form |
| 65 |  |  | test\_existing\_student | Tests validation for student form for existing student by device ID |
| 66 |  | ModuleFeedbackFormTests | test\_empty\_form | Tests validation for empty module feedback form |
| 67 |  |  | test\_valid\_form | Tests valid data for feedback form |
| 68 |  |  | test\_length\_check | Tests validation for feedback form with invalid lengths |
| 69 | Tests\_init\_test\_data\_command.py | InitTestDataTest | test\_minimal\_data\_loaded | Tests data load for an initial system with minimal data i.e. 1 staff user |
| 70 |  |  | test\_all\_data\_loaded | Tests data load for system with lots of data for development purposes |
| 71 |  |  | test\_single\_datatype\_reload | Tests data load for single data type e.g. students |
| 72 |  |  | test\_no\_args\_provided | Tests data load command when no required argument supplied |
| 73 | Tests\_module\_course\_view\_settings.py | ModuleCourseViewSettingsTests | test\_user\_type\_access | Tests that only staff members have access to this view |
| 74 |  |  | test\_unauthenticated\_redirects | Tests that unauthenticated user redirects to login |
| 75 |  |  | test\_can\_see\_modules\_courses | Tests that staff can see all modules and courses on page |
| 76 |  |  | test\_set\_modules | Tests that modules and courses can be set for staff |
| 77 |  |  | test\_set\_modules\_invalid\_data | Tests that modules and courses aren’t set when invalid data supplied |
| 78 | Tests\_parse\_excel.py | ExcelParserTests | test\_parse\_xls | Doesn’t test any components but is good tool to test excel parsing for xls files |
| 79 |  |  | test\_parse\_xlsx | Doesn’t test any components but is good tool to test excel parsing for xlsx files |
| 80 | Tests\_settings.py | GeneralSettingsTests | test\_unauthenticated | Tests unauthenticated user is redirected to login when trying to access settings |
| 81 |  |  | test\_user\_types | Tests settings that student user sees vs. what staff user sees |
| 82 |  | ColourBlindSettingsTests | test\_colourblind\_options\_set\_value | Tests setting colour-blind options enabled and disabled |
| 83 |  |  | test\_pass\_fail\_colours\_returned | Tests pass/fail colours when colour-blind settings enabled vs. disabled |
| 84 |  |  | test\_colourblind\_usages | Tests colours shown on all single entity views with colour-blind settings enabled vs. disabled |
| 85 |  | AttendanceRangeSettingsTests | test\_attendance\_range\_invalid\_values | Tests validation for multiple invalid data types for attendance range settings |
| 86 |  |  | test\_attendance\_range\_valid\_values | Tests attendance range settings for valid values |
| 87 |  |  | test\_attendance\_range\_usages | Tests attendance range settings on single entity views before and after changes |
| 88 | Tests\_single\_views.py | SingleModuleViewTests | test\_single\_module\_view\_staff | Tests single module visibilities for staff |
| 89 |  |  | test\_single\_module\_view\_student | Tests single module visibilities for student |
| 90 |  |  | test\_module\_unauthenticated | Tests unauthenticated user is redirected to login for module view |
| 91 |  |  | test\_nonexistent\_module | Tests 404 for non-existent module URL |
| 92 |  | SingleCourseViewTests | test\_single\_course\_view\_staff | Tests single course visibilities for staff |
| 93 |  |  | test\_single\_course\_view\_student | Tests students can’t see single courses |
| 94 |  |  | test\_course\_unauthenticated | Tests unauthenticated user is redirected to login for course view |
| 95 |  |  | test\_nonexistent\_course | Tests 404 for non-existent course URL |
| 96 |  | SingleLecturerViewTests | test\_single\_lecturer\_view\_staff | Tests single lecturer view visibilities for staff |
| 97 |  |  | test\_single\_lecturer\_view\_student | Tests students can’t see single lecturer views |
| 98 |  |  | test\_lecturer\_unauthenticated | Tests unauthenticated user is redirected to login for lecturer view |
| 99 |  |  | test\_nonexistent\_lecturer | Tests 404 for non-existent lecturer URL |
| 100 |  | SingleStudentViewTests | test\_single\_student\_view\_staff | Tests single student view visibilities for staff |
| 101 |  |  | test\_single\_student\_view\_student | Tests single student view visibilities for students |
| 102 |  |  | test\_student\_unauthenticated | Tests unauthenticated user is redirected to login for student view |
| 103 |  |  | test\_nonexistent\_student | Tests 404 for non-existent student URL |
| 104 |  | SingleLectureViewTests | test\_single\_lecture\_view\_staff | Tests single lecture visibilities for staff |
| 105 |  |  | test\_single\_lecture\_view\_student | Tests single lecture visibilities for students |
| 106 |  |  | test\_lecture\_unauthenticated | Tests unauthenticated user is redirected to login for lecture view |
| 107 |  |  | test\_nonexistent\_lecture | Tests 404 for non-existent lecture URL |
| 108 | Tests\_upload.py | UploadTests | test\_upload\_valid\_data\_csv | Tests valid data is uploaded for CSV files |
| 109 |  |  | test\_upload\_valid\_data\_xls | Tests valid data is uploaded for xls Excel files |
| 110 |  |  | test\_upload\_valid\_data\_xlsx | Tests valid data is uploaded for xlsx Excel files |
| 111 |  |  | test\_sequential\_upload\_replaces | Tests data upload for slightly different data overwrites existing data |
| 112 |  |  | test\_upload\_unrecognised\_module | Validates error message for attendance data upload with unrecognised module |
| 113 |  |  | test\_upload\_unrecognised\_student | Validates error message for attendance data upload with unrecognised students |
| 114 |  |  | test\_upload\_invalid\_attendance\_data | Validates error message for attendance data upload with unrecognised attendance values |
| 115 |  |  | test\_upload\_incorrect\_file\_extension | Validates error message for attendance file upload with incorrect file extension |
| 116 |  |  | test\_upload\_no\_file | Validates error message for attendance upload with no file |
| 117 |  |  | test\_upload\_multiple\_files | Tests multiple file uploads for single attendance data upload |
| 118 |  |  | test\_upload\_multiple\_files\_invalid | Tests that data is still uploaded for valid upload when invalid file upload is included in multiple file upload for attendance data |
| 119 |  |  | test\_upload\_usertype\_permissions | Tests that staff are the only users who can upload attendance data |
| 120 |  |  | test\_unauthenticated\_upload | Tests unauthenticated user is redirected to login on upload attempt |
| 121 | Tests\_views.py | ViewsTests | test\_login\_view | Tests login view |
| 122 |  |  | test\_login\_redirects\_if\_authenticated | Tests redirect to index page if login URL is hit as authenticated user |
| 123 |  |  | test\_unauthenticated\_index\_view | Tests index view redirects to login page for unauthenticated user |
| 124 |  |  | test\_authenticated\_index\_view | Tests index view for authenticated user |
| 125 |  |  | test\_no\_content\_staff | Tests expected messages for index page for staff with no content to look at |
| 126 |  |  | test\_no\_content\_student | Tests expected messages for index page for student with no content to look at |
| 127 |  |  | test\_content\_staff | Tests content on index page for staff user |
| 128 |  |  | test\_content\_student | Tests content on index page for student user |
| 129 |  |  | test\_invalid\_user | Tests unrecognised user type (i.e. not staff/student) is logged out and redirected to login |