System Development for Placement Modules (Pre-Start of Placement) Final Report

Author: Samuel Meehan
Supervisor: Martin Caminada
Moderator: Jing Wu

School of Computer Science and Informatics, Cardiff University

Date of Completion: 15/05/2020

Abstract

Managing student placements presents an increasingly complex challenge to staff at Cardiff University. While commercially available tools exist, those found so far by staff are generally expensive and do not fully meet the requirements of the university. A request has therefore been made to develop a bespoke, web-based tool to assist students and staff involved in this process.

This project resulted in the development of a website which supports the completion of common tasks both kinds of users face during the placement process prior to a student's departure for their placement, including document and event management. This is intended to be used alongside the results of a sister project to provide comprehensive support throughout the placement process.

As of the submission of this report, almost all critical and medium priority requirements of the website have been met and the website is in a functional state. However, additional work is required to implement some lower priority features and better integrate it with the sister project.

Acknowledgements

I would like to thank my supervisor, Martin Caminada, and for his support throughout the project.

I would also like to thank Catherine Teehan for acting as a regular source of contact within the Cardiff University placement staff, and Mathew Trimby for his assistance in helping our two projects to better complement one another.

Table of Contents

1.0 Introduction

For many years students studying Computer Science have been able to supplement their degree programs with an additional 12-month industrial placement at an IT company of their choice. This placement module, while excellent for the future of those who choose to take it, places an increasingly large burden on the placement team staff responsible for running it as more and more students choose to take it each year.

This project aims to deliver a web-based solution to support the management of the university's industrial placement schemes. This particular project's scope is restricted to supporting users performing tasks that are expected to occur before a student begins their placement at a company. Combined with an additional sister project focusing on the period after that point, the two solutions aim to significantly improve the experience of students on placement as well as the placement team staff responsible for organising and managing them.

To achieve this, it was decided to develop a website that would provide the following functionality to users:

- Allow the user to upload key information and documents to a cloud storage system, and allow authorised users to retrieve these documents.
- Allow staff to easily set up and advertise any events that may take place over the pre-placement period.
- Allow students to register their interest in attending these events.
- Provide helpful analytical tools for staff to assess interest

During development, an Agile methodology was utilised to better meet the needs of the users. While the approach taken mostly resembles Rapid Application Development (RAD), elements of DSDM such as using MoSCoW prioritisation are also employed.

The solution implemented is intended to support staff involved in organising student placements, and additionally the students themselves. Staff will primarily make use of the system to track student progress towards starting their placements and manage the various employment events that take place each year. Students meanwhile will primarily use the site to upload their completed and supporting documents as well as to sign up to any events that strike their interest. In its current state the solution developed meets most of the requirements set for it, though there are some exceptions that could not be met before the current deadline.

2.0 Background

2.1 Context

The current process used to manage student placements is needlessly complex and has significant room for improvement. All required information about student placements (i.e. company/due dates/supervisor details) is currently recorded on spreadsheets, which along with other relevant documents are uploaded using Learning Central. Hard copies of the documents are also kept.

Placement Document Listing

Document Reference	Type	Number	Iterations	Name	Description	Last updated	Users
CSP1	Form	1	1	Form 1 – Consent to Share CV	Form giving permission for the placement team and consultants to shar a student's CV with potential employers	15/03/2019	Students and Placement team
CSP2	Form	2	1	Form 2 – Confirmation of Placement	For providing details of placement	15/03/2019	Students and Placement team
CSP3	Form	3	1	Form 3 - H&S Questionnaire	Form detailing H&S	20/07/2019	Company, Students and Placement team
CSP4	Form	4	1	Form 4 – Placement Agreement	Placement Agreement between Student, Company and COMSC Placement Team	20/07/2019	Company, Students and Placement team
CSP5	Form	5	1	Form 5 Induction Checklist	Induction checklist to ensure students have a full and complete induction into the company	20/07/2019	Students and Placement team
CSP6	Form	1	1	Overseas Status Letter	Letter to be sent to employers to confirm the conditions of an overseas student tier 4 visa	20/07/2019	Students and Placement team
CSP7	Form	2	1	Placement Authorisation Letter	Letter to be sent to students once forms 2-4 have been completed and the student has successfully progressed to the placement year	20/07/2019	Company, Students and Placement team
CSP8	Form	6	4	Employer Evaluation Form	Evaluation form to be completed by work-based supervisor each quarter	20/07/2019	Students and Placement team
CSP9	Form	7	2	SFIA Mapping Document	Evaluation form to be completed by student and employer mid-way and at the end of placement. This will identify the skills and level at which the student is working	20/07/2019	Company, Students and Placement team
CSP10	Doc	1	2	Placement Handbook	Handbook detailing rules, regulations, processes and procedures for both UG and PGT placement students.s	20/07/2019	Company, Students and Placement team
CSP11	Doc	2	2	Expectations for Placement			
CSP12	Form	8	1	Confidentiality release form			
CSP13							
CSP14							
CSP15							

Highlighted rows are documents handled prior to placement start.

Figure 1: List of documents required over the course of a placement

According to a placement team representative, without a dedicated tool keeping track of all these documents requires a considerable investment of time. Furthermore, as they tend to be kept separately important information can occasionally be lost outright. An alternative system is therefore required to keep up with the increasing demand as more and more students opt for a Placement.

RE: Existing Process For Project

CT **Catherine Teehan <TeehanC@cardiff.ac.uk>**
21/04/2020 19:00

To: meehansam98@gmail.com

Hi Sam,

Currently we use Learning Central to upload documents and use spreadsheets keep track of what student is placed where, supervisor details, start dates, visit dates, report due dates and document tracking. We also keep hard copies of all the documents as well. At the moment everything is separate meaning somethings get missed and it's time consuming and messy to keep track of everything.

Catherine

Figure 2: Screenshot of an email from a placement team representative on the current process

2.2 Existing Solutions

There are several pre-existing Place Management solutions available commercially, however staff have so far been unable to find one that sufficiently meets their needs at an acceptable cost. Some general research into these was done during the planning phase of the project to get a better understanding of current industry approaches to solving the problem, which is summarised below.

Name: InPlace
Summary: InPlace is advertised as a generic placement management system, boasting responsive design, "power communication capabilities" and integration with existing student/financial systems. Includes a document repository for storing placement agreement files. It appears to provide tools for better interacting with companies (i.e. online advertisement of students), however does not provide much if any support for event management. As a generic system, it can handle multiple different kinds of placement, such as internships, graduate placements and general volunteering. Also supports multiple methods of placement allocation, i.e. self-selection/rule matching.
Impact on Project Website: -Website UI should be responsively designed. -Website could ideally integrate with existing university systems i.e. UI, user accounts, etc. -Generic approach unnecessary for the project, though potentially useful if system expanded beyond School of Computer Science or if significant change to placement process occurs later? -Strong interaction with companies could be beneficial, but almost certainly too great a task to complete within the scope of this project.
URL: https://www.inplacesoftware.com/

Name: Sonia
Summary: An IMS Certified solution that aims to provide a comprehensive and flexible approach to placement management. Comes with an importer tool to upload existing data from Excel/LTI compatible systems as well as cloud support for Microsoft Azure and AWS. Students can access information and submit sheets via a mobile app as well as though the website.
Impact on Project Website: -While out of scope, developing a mobile app to work alongside website may help enhance the student experience. -Website could provide a modifiable dashboard for reporting purposes. -Importing new/existing data in bulk from an integrated tool would significantly reduce administrative workload spike, as students are likely to be added to the program in batches.
URL:

https://www.qsrinternational.com/sonia-student-management-software/home/

Besides complete Placement Management solutions, many companies also offer cloud document storage and event management services, such as Microsoft SharePoint, Amazon Web Services' S3 Bucket for the former or Eventbrite and Cvent for the latter. The underlying issue with these options is they do not provide a single unified solution to the issues facing expected users. A bespoke solution combining elements from these would better suit the project's requirements.

That said, as these existing tools can already be used to accomplish individual tasks, it would be efficient to try and incorporate them directly into the proposed solution where possible. Particularly as some, i.e. Eventbrite, provide well-documented API's to assist with this.

2.3 Planning

As stated, an Agile approach was taken to planning and developing the project, combing elements of RAD and DSDM (What is DSDM and the 8 principles. 2018, Singh 2019). This was done so that prototyping could begin quickly and hopefully deliver a viable product on time, which was of some concern given the scale of the system and risk of feature creep. It also meant the project could benefit from frequent assessments by the end-users. To that end from very early on a weekly meeting was organised with the developer of the sister project, the project supervisor, and a representative from the Placement Team. These meetings were used to deliver progress reports and demonstrations as well as to provide an opportunity for the Placement Team to provide feedback and suggestions.

3.0 Specification/Design

3.1 Requirements

The following section details the website's requirements. The requirements can generally be considered an extension of the Project Aims found in the Initial Plan created at the start of the project.

Requirement	**Must Have**
Students will be able to upload their CV, required placement documents and additional supporting documents to the system	
Acceptance Criteria	

• Students can successfully store (in PDF format):
  ○ Their latest CV
  ○ Confirmation of Placement
  ○ Health & Safety Questionnaire
  ○ Placement Agreement
  ○ Additional miscellaneous documents
• The system will prevent unauthorised users (i.e. other students) from accessing said documents.	

Requirement	**Must Have**
Members of the placement team will be able to upload required placement documents for a specific student.	
Acceptance Criteria	

• Members of the placement team can successfully store the following documents (in PDF format) for a specified student:
  ○ Confirmation of Placement
  ○ Health & Safety Questionnaire
  ○ Placement Agreement
  ○ Overseas Status Letter
  ○ Health & Safety Questionnaire
  ○ Placement Agreement
  ○ Placement Authorisation Form
  ○ Additional supporting documents
• The system will prevent unauthorised users (i.e. other students) from accessing said documents.	

Requirement	**Must Have**
Users will be able to download and export documents.	
Acceptance Criteria	
• Students can successfully download documents belonging to them. • Staff can download documents belonging to any specific student. • Documents can be downloaded in PDF and CSV form.	

Requirement	**Must Have**
Student profile information will be modifiable.	
Acceptance Criteria	

• Students can edit non-administrative information including their password, email address and anticipated company. • Staff will be able to list additional notes about a student and assign them to a supervisor.

Requirement	**Must Have**
Students will be able to allow or prevent their CV from being shared with 3rd parties outside the university	
Acceptance Criteria	
• The system will present students with an option to make their CV sharable. • Staff will be able to identify whether the student has selected to make their CV shareable.	

Requirement	**Should Have**
Placement Staff will be able to set up, edit and delete upcoming events using the system	
Acceptance Criteria	
• Placement Staff can generate valid events from the user interface that students can view and sign up to. • Existing events can be edited with new names/descriptions/capacity/times. • Events can be selected and deleted by staff.	

Requirement	**Should Have**
Students should be able to view and register their interest in upcoming events	
Acceptance Criteria	
• The student should able to see a list of upcoming events created by placement staff, including necessary details such as description/time. • Students should not be able to “double book” events when registering. • The event list should indicate to students which events they have signed up to. • Students should be given reminders about events they have signed up to.	

Requirement	**Should Have**
Students should be able to view and register their interest in upcoming events	
Acceptance Criteria	
• The student should able to see a list of upcoming events created by placement staff, including necessary details such as description/time. • Students should not be able to “double book” events when registering. • The event list should indicate to students which events they have signed up to. • Students should be given reminders about events they have signed up to.	

Requirement	**Could Have**
Staff could be able to keep a record of which students actually attend events they have registered interest in.	
Acceptance Criteria	
• Placement staff should be able to see which events a student has registered interest in.	

• After the listed events start time, they should be able to enter whether an interested student actually attended the event or not (assuming yes by default).

Requirement	**Could Have**
The system could provide automated alerts via multiple mediums for important events.	
Acceptance Criteria	
• The system can send alerts to students via email/text for upcoming deadlines/events.	

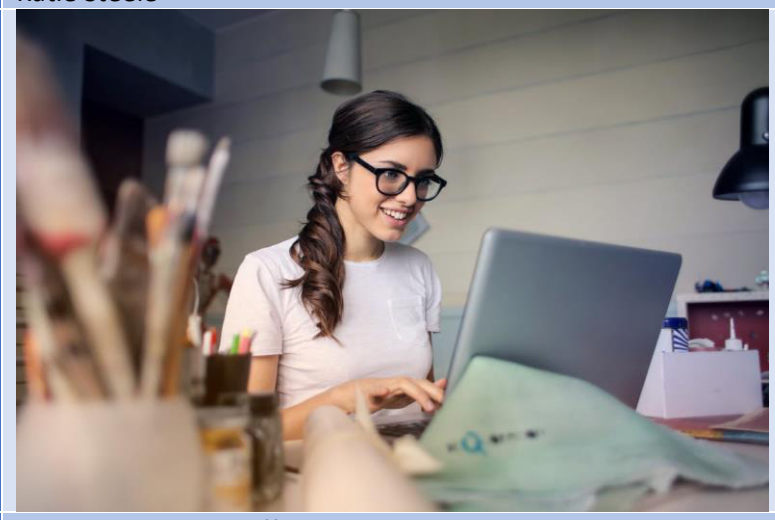
In addition to these requirements, several features were requested by the Placement Team representative during the weekly demonstration meetings throughout the project such as integration with the existing university account systems to allow Single Sign On. Where possible these were implemented, but this was not always the case.

3.2 Personas

To assist in improving the usability of a website it is generally recommended to create several "personas" to represent key user groups for future reference (see Usability.gov). In accordance with this advice, the following personas were created for this project.

Name:	William Blake
Photo:	
Role:	Student
Description:	William has just started his second year of study and is planning to take an industrial placement next year. He currently does not have a clear idea what company he wishes to join but has a reasonable understanding about which subjects he enjoys and which he is good at. He has also begun writing up a CV and is having it reviewed by a member of staff for improvements. He has good technical knowledge and access to the internet both on and off-campus. He sometimes changes his contact details.
Goals and Tasks:	-Be able to view easily upcoming events and judge which he is interested in attending. -Be able to update profile information i.e. email address.

	-Be able to submit documents requested by the university. -Be able to submit additional supporting documents like a CV for review from other users.

Name:	Katie Steele
Photo:	
Role:	Placement Team Staff Member
Description:	Katie is a member of staff at the university who is responsible for organising various employment events throughout the year. Events, though optional, are often a considerable investment to run so she is interested in maximising attendance and understanding what worked well or not. She is also part of a team who keep track of upcoming placements and ensuring all students have submitted the relevant documentation and have been assigned a supervisor. She regularly has to communicate with students who have failed to secure a placement or are taking their time to submit key documents required for the placement to go ahead. She primarily works from her office at the university. She has a decent level of technical knowledge and frequently uses tools such as Excel and Microsoft Outlook to manage a tight schedule.
Goals and Tasks:	-Set up employment events more efficiently. -Advertise these events to Students to generate interest and improve attendance. -Analyse event statistics (i.e. attendance/number of students given offers) to determine how best to run them in future. -Identify students who are missing key documentation or who have not been assigned a supervisor. -Assign supervisors to students, ensuring no supervisor is responsible for more than a maximum number of students. -Alert students in danger of failing to meet key deadlines. -Minimise time spent looking up information and documents.

Name:	Emma Green

Photo:	
Role:	University Professor and Placement Supervisor
Description:	Emma primarily works at the university as a lecturer/researcher but is also assigned several students each year to act as a supervisor for. She mainly only deals with her assigned students after they have begun their placements but may need to gather information or get in touch beforehand.
Goals and Tasks:	-Easily find and download information/documentation regarding the students she has been assigned.

These personas are intended to represent the three primary types of user that the solution is expected to be used by. Specifically, that is the students, their supervisors and the staff managing the placements. The student persona is based on the experience of actual students who have already been through the placement process (specifically myself and several other students in my academic year), while the remaining two are based on meetings with the project supervisor (who has experience as a placement) and a representative from the university staff in charge of managing placements.

3.3 Use Cases

The following Use Cases were constructed from the requirements and additional requests from the client later in development. They show the basic flow of the primary tasks that users are expected to complete using the tool.

Use Case ID: 1	**User logs into website**
User	Any
Pre-Requisites	Currently on login page
Basic Flow	User enters valid username and password pair into text boxes. User is redirected to landing page and access level identified
Alternate Flows	User enters an invalid username and/or password and login is rejected. An error message explaining this is displayed.

Use Case ID: 2	**User requests password reset**
User	Any
Pre-Requisites	Currently on login page

Basic Flow	User selects “Forgot Password” button and is redirected to “Forgot Password” page. User inputs valid username. An email is sent to the user’s address, explaining the request and providing a link to the user to follow. User clicks on the email link, taking them to Password Change page. User inputs new password (double entry for confirmation) and submits. User is redirected to login page and can log in with the new password.
Alternate Flows	The user may stop progressing through the flow at any point before submitting the new password. User enters an invalid username on the Forgot Password page. An error message stating this is displayed. The two entries of the new password in the Password Change screen do not match. An error message stating this is displayed.

Use Case ID: 3	**Student submits documents**
User	Student
Pre-Requisites	User must be logged in to an account with Student level access. User on Document/Profile Page.
Basic Flow	User clicks Choose File button in a section they want to upload a document for (sections indicated by labels). They navigate to the file they want to upload using the file browser and select it. They repeat the above steps for the remaining documents they wish to upload, using the relevant sections File Upload buttons. They select Upload Files Button. Page refreshes.
Alternate Flows	Multiple files may be selected for uploading to Supporting Documents section. If no files are submitted, error message displayed. If any of the uploaded documents are invalid (too large/non-PDF format) file upload fails and an error message is displayed.

Use Case ID: 4	**Student updates profile information**
User	Student
Pre-Requisites	User must be logged in to an account with Student level access. User on Document/Profile Page.
Basic Flow	User enters email and company name to relevant fields, replacing current values. User selects Update Profile button.

	Page refreshes. The default values of the fields reflect the new values.
Alternate Flows	

Use Case ID: 5	**Student changes permission for CV sharing**
User	Student
Pre-Requisites	User must be logged in to an account with Student level access. User on Document/Profile Page.
Basic Flow	User clicks on CV sharing checkbox
Alternate Flows	

Use Case ID: 6	**Placement Team Member uploads student documents**
User	Placement Team Member
Pre-Requisites	User must be logged in to an account with Placement Team access. User on Student Overview Page
Basic Flow	User selects a student from the drop down list, causing student profile info and document table to appear. User clicks Update Profile button and is taken to Document/Profile page for that student User clicks Choose File button in a section they want to upload a document for (sections indicated by labels). They navigate to the file they want to upload using the file browser and select it. They repeat the above steps for the remaining documents they wish to upload, using the relevant sections File Upload buttons. They select Upload Files Button. Page refreshes.
Alternate Flows	Multiple files may be selected for uploading to Supporting Documents section. If no files are submitted, error message displayed. If any of the uploaded documents are invalid (too large/non-PDF format) file upload fails and an error message is displayed.

Use Case ID: 7	**Staff updates student profile information**
User	Placement Team Member
Pre-Requisites	User must be logged in to an account with Placement Team access. User on Student Overview Page
Basic Flow	User selects a student from the drop-down list, causing student profile info and document table to appear.

	User clicks Update Profile button and is taken to Document/Profile page for that student User enters values (i.e. Supervisor ID) to relevant fields, replacing current values. User selects Update Profile button. Page refreshes. The default values of the fields reflect the new values.
Alternate Flows	Staff go back without clicking Update Profile button. No change to database.

Use Case ID: 8	**Student downloads an existing document**
User	Student
Pre-Requisites	User must be logged in to an account with Student level access. User on landing page
Basic Flow	Student finds desired document on table and clicks download link. The PDF is downloaded and can be opened from the file browser or download bar.
Alternate Flows	

Use Case ID: 9	**Non-Student downloads an existing document**
User	Placement Team Member/University Staff
Pre-Requisites	User must be logged in to an account Staff/Placement Team Access User on Student Overview page
Basic Flow	User selects a student from the drop-down list, causing student profile info and document table to appear. User selects desired document from the table. The PDF is downloaded and can be opened from the file browser or download bar.
Alternate Flows	User may select another student after the first selection. The info/table will be changed to reflect the currently selected student.

Use Case ID: 10	**Creating a new event**
User	Placement Team Member
Pre-Requisites	User must be logged into an account with Placement Team access. User on Event Management Page
Basic Flow	User clicks Create Event button and is taken to Create Event page. User enters name, description, event capacity, start/end times and clicks button to submit.

	User is redirected to Event Management page. New event should be visible on event list.
Alternate Flows	User may return from Create Event page without submitting. User may enter invalid event details, causing an error message to display on submission and no event to be created.

Use Case ID: 11	**Editing an existing event**
User	Placement Team Member
Pre-Requisites	User must be logged in to an account with Placement Team access. User on Event Management Page
Basic Flow	User clicks Update Event button and is taken to Event Update screen. User makes desired changes to event details in relevant fields and submits changes by clicking button. User redirected to Event Management page. Event list should reflect the changes made.
Alternate Flows	User may return from Create Event page without submitting. User may enter invalid event details, causing an error message to display on submission and no event to be created.

Use Case ID: 12	**Deleting events**
User	Placement Team Member
Pre-Requisites	User must be logged in to an account with Placement Team access. User on Event Management Page
Basic Flow	User selects checkbox for events they want to delete. User clicks Delete Event button. Event list refreshes missing the deleted events.
Alternate Flows	User attempts to delete no events. An error message tells them to select an event to delete. User attempts to delete an event that has already had tickets bought for it. Due to Eventbrite constraints this is not allowed, so error message is displayed informing the user.

Use Case ID: 13	**Registering interest in an event**
User	Student
Pre-Requisites	User is logged into a student account. An outstanding event is available for the student to sign up to.
Basic Flow	User navigates to event list. User selects an event to sign up to.

	User selects an event ticket. User confirms ticket selection. Page refreshes, the selected event is indicated to have been registered to, and an email is sent to the user confirming their registration.
Alternate Flows	During ticket selection and confirmation the user may back out, returning to event list with no update taking place. After confirming, the user reselects the event and unregisters

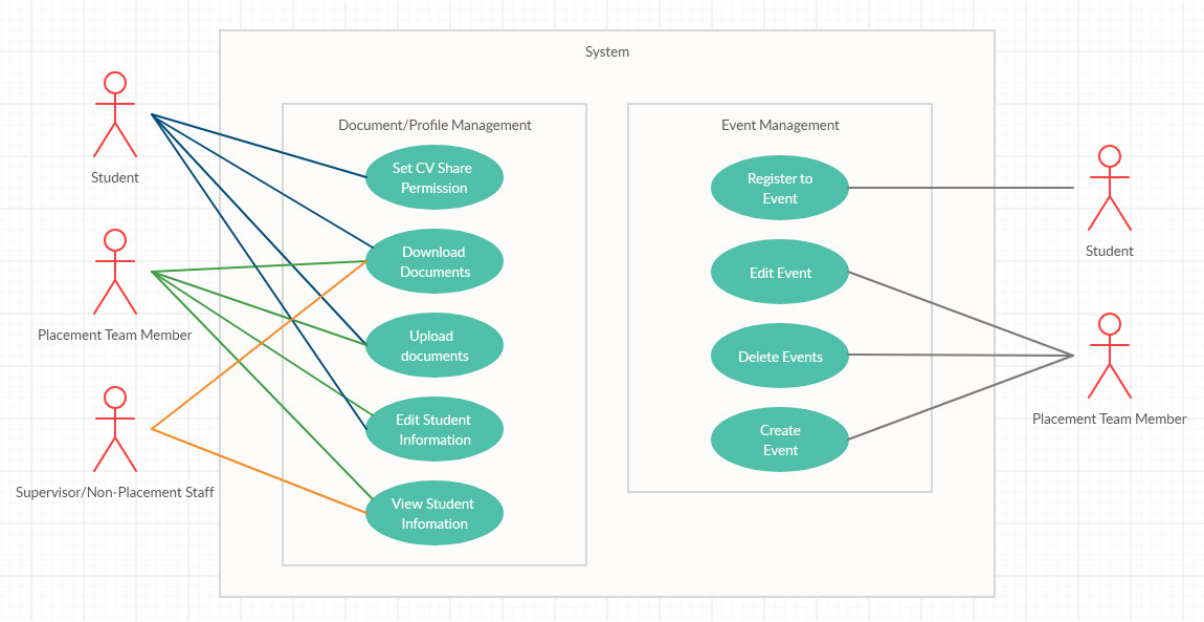


Figure 3: Diagram showing functional use cases each type of user is expected to follow

3.4 User Interface Design

During discussions with the Placement Team representative and sister project it was decided that the user interface of both projects should attempt to emulate that found on other university websites, and particularly those found on the university intranet, in order to fit in with these existing webpages and present a familiar environment to students and staff. This would also save time during the integration of the two projects when that eventually took place.

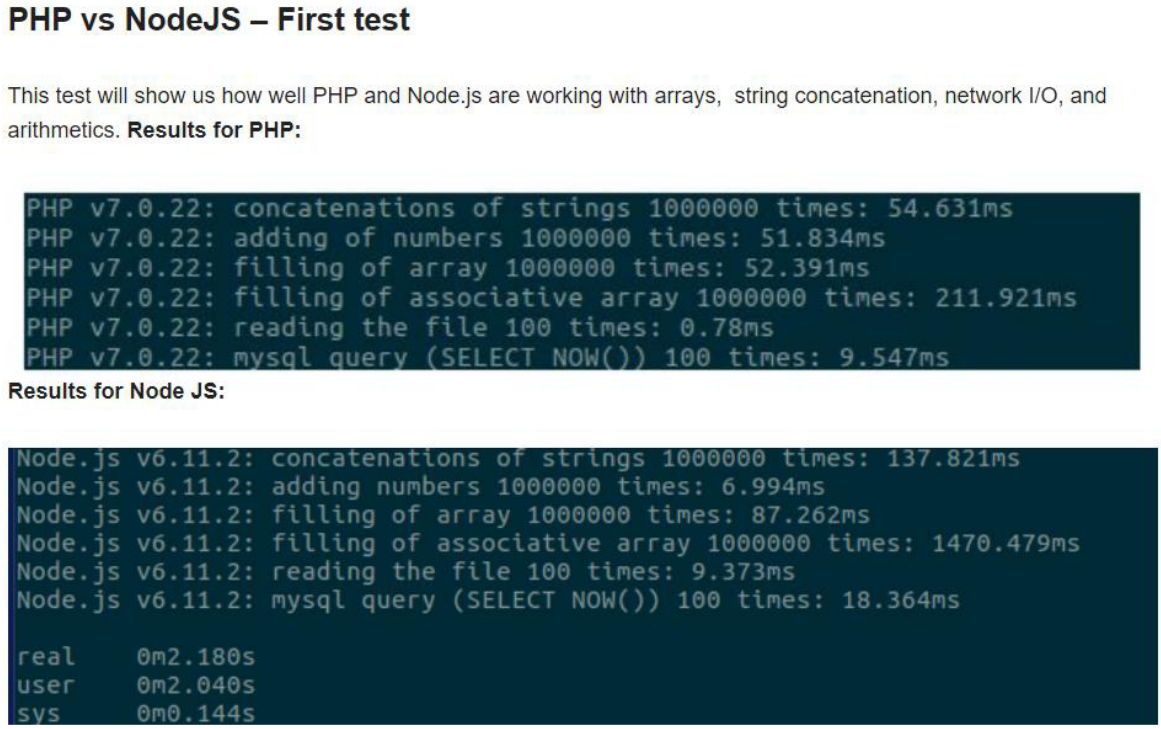
The UI design also attempts to follow Jakob Nielson's 10 Usability Heuristics (Nielson 1994) and aims to be responsive, though it was designed primarily for desktop rather than mobile like some of the existing solutions.

3.5 Tools

The following section provides a general overview of the tools and technologies used to develop the website, including a summary and justification for their use over alternatives where applicable.

PHP

Most of the website's functionality was coded using PHP, as it has several advantages over alternatives such as Node.js. PHP has better performance for tasks the website is expected to frequently carry out such as reading files and querying MySQL.



PHP vs NodeJS – First test

This test will show us how well PHP and Node.js are working with arrays, string concatenation, network I/O, and arithmetics. **Results for PHP:**

```
PHP v7.0.22: concatenations of strings 1000000 times: 54.631ms
PHP v7.0.22: adding of numbers 1000000 times: 51.834ms
PHP v7.0.22: filling of array 1000000 times: 52.391ms
PHP v7.0.22: filling of associative array 1000000 times: 211.921ms
PHP v7.0.22: reading the file 100 times: 0.78ms
PHP v7.0.22: mysql query (SELECT NOW()) 100 times: 9.547ms
```

Results for Node JS:

```
Node.js v6.11.2: concatenations of strings 1000000 times: 137.821ms
Node.js v6.11.2: adding numbers 1000000 times: 6.994ms
Node.js v6.11.2: filling of array 1000000 times: 87.262ms
Node.js v6.11.2: filling of associative array 1000000 times: 1470.479ms
Node.js v6.11.2: reading the file 100 times: 9.373ms
Node.js v6.11.2: mysql query (SELECT NOW()) 100 times: 18.364ms

real    0m2.180s
user    0m2.040s
sys     0m0.144s
```

Figure 4: Screenshot of PHP vs NodeJS test comparison, taken from ThinkMobiles

PHP is used on more than 80% of web servers (ThinkMobiles in 2020) and is extremely flexible. Thanks to this it is also well supported by a large community, meaning there is an abundance of online support for accomplishing most common tasks. Furthermore, many companies (notably Amazon and Eventbrite as discussed below) have created SDKs, APIs and frameworks to help developers easily integrate PHP applications with their products. Lastly, as I have previous experience coding with PHP I was confident it would be able to meet the requirements of the website. It therefore did not seem worthwhile to learn an alternative language from scratch, especially as during the Risk Analysis running out of time to implement desired features was considered to be highly likely to occur.

Selenium Webdriver

It was initially planned that automated tests would be developed alongside the code using Selenium Webdriver. This would have been a significant help during the testing process but was unfortunately not implemented due to time constraints. More details can be found in the Testing section of this report.

Bootstrap

In addition to regular HTML and JavaScript the Bootstrap 4 framework was also used to construct the website's front-end. Thanks to BootstrapCDN, Bootstrap can be integrated into a webpage with a single line of code, making it extremely easy to use. In addition, there is plenty of online support

for it due to its popularity and the documentation and tutorials provided by the Bootstrap Team and third parties such as W3Schools.

Using Bootstrap provides several advantages. Firstly, thanks to its built-in classes and styles it considerably reduces the amount of effort required to create a good-looking webpage. Its Grid System for instance makes it extremely simple to layout the page's structure using rows and containers. Furthermore, Bootstrap is designed from a mobile-first perspective and as such provides helpful tools for designing a responsive webpage. Unfortunately, as this was not considered a key requirement for the project and as I have less experience with responsive design I did not make much use of the latter features, but if mobile support is desired in a future version using Bootstrap should make the update considerably easier.

jQuery

jQuery was also used to extend the core JavaScript used for the website front-end. This was, in part, due to it being a requirement of Bootstrap, but it also provides many useful functions that the website benefits from. Most notably this includes the ability to use AJAX to call server-side code (i.e. PHP) without refreshing the webpage. This is extremely helpful for some of the website's functions that require some input from the user (i.e. selecting Student profiles).

Like the other tools mentioned, jQuery is extremely popular and well supported. While I have little experience using AJAX thanks to the wealth of documentation around it reduced the overall time spent learning considerably.

Amazon Simple Storage Solution (S3) & SDK

Many of the critical requirements of the project revolve around the key documents students and the placement team are required to submit prior to the start of placements. While it would be possible to use a local file system, due to the risk of limited space on the cloud instance it was decided that the best way to store and retrieve these documents was using a cloud storage solution. S3 was eventually settled on as it is highly flexible, scalable (Rouse 2018) and there is a considerable amount of support provided for it due to its popularity. While not free, it remains extremely affordable for the level of performance and space expected to be required (costing only about 8 pence per month during development and testing).

Detail	
Amazon Simple Storage Service	$0.10
Charges	$0.08
VAT **	$0.02

Figure 5: Screenshot of invoice for S3 Bucket use in April 2020

While initially my own AWS account was used, prior to deployment the website will need to be configured to use an account managed by the university.

The structure of the S3 bucket is relatively straightforward, with documents separated first by user and then by type. All directories except for SupportDocs are set to automatically replace existing documents with the most recently submitted, reducing the amount of space required per student (albeit at the cost of storing multiple versions of documents, though for most documents this is unnecessary).

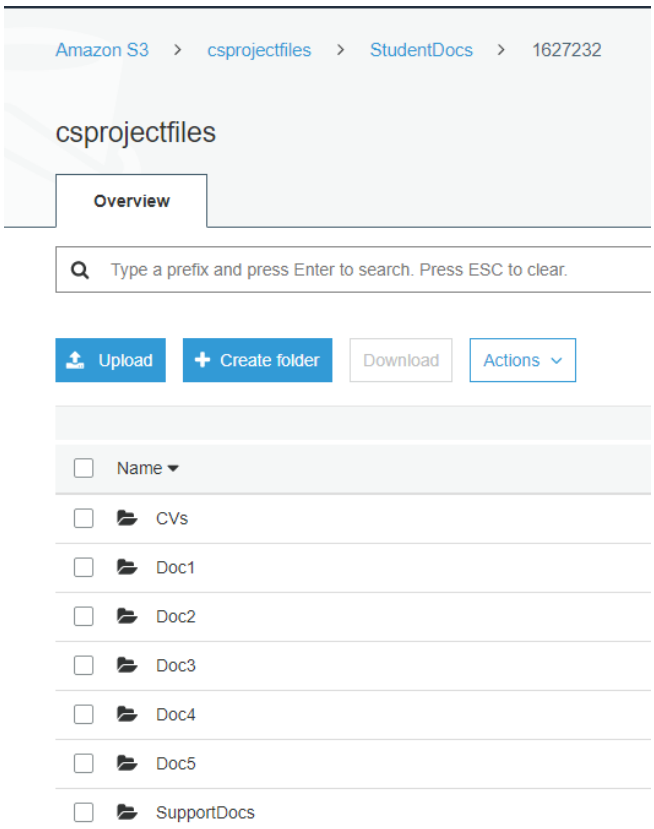


Figure 6: Screenshot of S3 Bucket structure as seen on the AWS interface.

Amazon also provides an SDK so that developers can easily connect with their AWS services via PHP code. Like S3 itself this SDK came with plenty of documentation and support, making it relatively simple to start interacting with the S3 Bucket from the website, though there was still quite a learning curve to scale. For instance, in an S3 Bucket, directories are also considered to be objects like files, which caused quite a few confusing results during development before this was fully understood.

Eventbrite API & Checkout

As the placement team already makes use of Eventbrite to some extent when organising events, it made sense to continue using their services for the website. Fortunately, Eventbrite has released an API to enable third-party developers to create and manage events, as well as an embedded checkout service that can be easily integrated into other websites. An Eventbrite account for the placement team was required to use these tools, however students do not need to have an account created in order to sign up to any events.

Unfortunately, Eventbrite does come with some restrictions for managing events. Most notably, it is extremely difficult to delete, unpublish or fully modify events once a ticket has been sold (even if the ticket is free). No way around this restriction was found before the end of the project, which unfortunately means these restrictions also apply in the final implementation.

Additionally, by design (Antwonne [no date]) the embedded checkout only function on webpages served over HTTPS. Attempting to use it over HTTP will cause it to fall back on the classic widget, which while functional does require leaving the project website and prevents the student account and the event being registered from being associated on the database, which does cause several

non-trivial bugs. As it is generally good practise to use HTTPS anyway for security reasons (Jackson 2016), so this should not be a serious concern.

OpenStack

During development it was possible to host the website on my university project site. Unfortunately, as this was not generally accessible without my login details and additionally is expected to be deleted a short period after the project deadline. A simple solution would be to find a cloud storage solution such as Amazon S3 to host it, however as the website relies on server-side PHP code these would not be sufficient. S3 and systems like it are only able to host static websites. The best, albeit most complicated, option remaining was to host the finished website on a full cloud computing stack.

Eventually, it was decided to have a project created by the university on OpenStack. There are several reasons for this. Having a project created on OpenStack by the university also comes with additional support from the IT Service desk and supporting documentation created by the university, which was particularly helpful due to a lack of experience setting up instances. Alternatives such as Amazon EC2 also come with plenty of support, however it was believed having experienced staff on-site would more beneficial. Also, unlike alternatives such as EC2 OpenStack is free for students to use.

Unfortunately using OpenStack does come with the significant drawback that the site can only be accessed from machines on the university network or using its VPN. It was concluded that as the project was focusing on pre-placement work this would be acceptable to start with, as students and staff are expected to be on-site. If this is found to be insufficient later on (or causes difficulties when integrating with the sister project's implementation) migrating to another service may prove necessary.

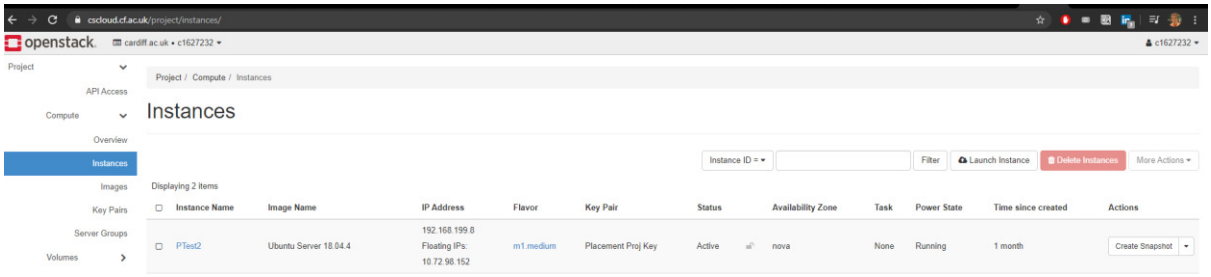


Figure 7: Screenshot of instance hosting the website as displayed on OpenStack interface.

An Ubuntu Server (18.04.4) was chosen for the instance that eventually hosted the site. As of the time of writing it is assigned a medium flavour, meaning it has access to 2 virtual CPUs, 2GB of RAM and 10GB of disk space. As documents are stored off-site on the S3 bucket this should be more than enough space for the foreseeable future, as the MySQL database is the only aspect of the site that will increase in scale as the number of users goes up. If necessary, a request can also be put in to grant additional storage space in the future.

MySQL

It was initially hoped that the website would be able to extend existing student and staff profiles kept by the university for things such as the intranet and/or Office 365, as this would reduce data

redundancy, provide a consistent experience to users logging in and perhaps most importantly reduce the workload required from staff to add students to the system.

Mid-way through the project attempting to use an Azure database alongside Microsoft Identity Platform Authentication Libraries (MSAL) was considered to achieve this, however due to the time requirements required to find, understand and implement this it seemed impossible to deliver within the timeframe of the project. Particularly as while there are compatible client libraries for working with PHP, none are directly supported by Microsoft (Microsoft identity platform authentication libraries, 2019) and it was difficult to follow what documentation there was. As such, the simpler but less desirable option of a MySQL database hosted alongside the website itself was eventually used.

List of Students on a Programme

Input:

Course Co	U%
Programm	2

Results:

Student C	Surname	Forename	Forename	Course	Course Tit	Acad Year	Block	Occl	Moac	Fee Status	Reg Statu	Univ Emai	Home Em	Tutor(Hon	Tutor(Assoc)
1234567	Meign	Paul		UFBSCMSI	BSc Comp	2020				E					
1638432	Smith	Bill		UFBSCMSI	BSc Comp	2020									
1845999	Terry	James	T	UFBSCMSI	BSc Comp	2020				H					
1632822	Johnson	Jill		UFBSCMSI	BSc Comp	2020									

Figure 8: Partially completed example of the CSV file currently used for keeping track of student information.

It should be noted that the Post-Placement project instead opted to use MongoDB so there may be some additional work required for further integration of the two projects. Also, as only a single table is currently used it could be argued that a relational database such as MySQL is unnecessary as the data does not follow a relational model. That said, MySQL is still adequate for the tasks expected of the website, and its simplicity ensures that a functional version could be set up quickly. Attempting to use MongoDB or other alternatives may have been too time-consuming in the short term, even if it provides benefits during integration later.

XAMPP/LAMPP

While as previously mentioned it was possible to develop and host the website on the university project site, eventually a cloud-based web server would be required. The LAMP stack was an obvious choice to use considering MySQL and PHP were both already being used before for prototyping before a webserver was selected. Newer versions of XAMPP have swapped MySQL for MariaDB, however due to the similar nature of the two

A major advantage of using this stack is its simplicity. Apache is a relatively simple webserver to learn to use, and the stack comes with PHP and MySQL/MariaDB pre-installed (depending on version), alongside PhpMyAdmin which made managing the database considerably easier during testing (and will likely be helpful to admin users in the future). XAMPP can easily be set up on a local machine and on Windows comes with an easy to use dashboard, which was helpful earlier on in development before the OpenStack instance was set up.

4.0 Implementation

4.1 Account Types

In the final implementation all user profiles are stored as records on the Placement Table on the database. For most of the development phase this was my personal database on School of Computer Science's MySQL server, however as stated this was later migrated to the webserver hosting the site itself.

Besides useful real0world information such as the users name/email/etc. each user is assigned an integer AdminLevel in the range of 0-2, with 0 being assigned to Students, 2 to members of the placement team (and any other profiles who require additional privileges) and 1 for remaining members of staff and other users. This value is retrieved during the login procedure and stored as a Session variable which most pages will check for when accessed.

Certain pages intended for use by specific types of users will redirect the user to the login page if they do not have the appropriate AdminLevel. Others, such as the landing page, navigation bar and student document uploader, are altered depending on this value to display different elements to the user (i.e. the Navbar will only display certain links to appropriate users).

```
<ul class= "navbar-nav ml-auto">
    <li class = "nav-item" id="home"><a class="nav-link" href = "index.php">Home</a>
    <?php
        //Admin Nav
        if($_SESSION['accessType']==2){
            echo "<li class = 'nav-item' id= 'navEvents'><a class='nav-link' href = 'viewEvents.php'>Event Management</a>";
            echo "<li class = 'nav-item' id= 'navDocManager'><a class='nav-link' href = 'awsDocManager.php'>Document Management</a>";
            echo "<li class = 'nav-item' id= 'navListStudents'><a class='nav-link' href = 'listStudents.php'>Update Student List</a>";
        //Student Nav
        }elseif($_SESSION['accessType']==0){
            echo "<li class = 'nav-item' id= 'studentEvents'><a class='nav-link' href = 'studentEvents.php'>Events</a>";
            echo "<li class = 'nav-item' id= 'uploadDocs'><a class='nav-link' href = 'studentDocUpload.php'>Upload Documents</a>";
        }
    ?>
    <li class = "nav-item" id= "changePass"><a class="nav-link" href = "changePassword.php">Change Password</a>
    <li class = "nav-item" id= "logout"><a class="nav-link" href = "logout.php">Log Out</a>
```

Figure 9: Source code for Navbar

4.2 Accessing the MySQL Database

When user data is required the database is queried using PHP's built-in MySqli Extension (an improved version of the deprecated MySQL extension). Using the credentials found in *dbConnect.php,* a connection is created with the MySQL database. Following the inclusion of this code, MySQL statements can be constructed and executed as required.

```
<?php
$sqlServerName = "localhost";
$sqlUsername = "root";
$sqlPassword = "tempPass1";
$sqlDatabase = "Webdata";
$conn = new mysqli($sqlServerName , $sqlUsername ,$sqlPassword ,$sqlDatabase );

if($conn->connect_error){
        die("Connection Failed: ".$conn->connect_error);
}
?>
```

Figure 10: Source code for dbConnect.php

If the user can provide some input to alter the query (i.e. when updating profile information), prepared statements are constructed to prevent SQL Injection attacks.

```
function updateDB($selectedID){
    include("dbConnect.php");
    $email = $_POST['email'];
    $company = $_POST['company'];
    $supervisor = $_POST['supervisor'];
    //Handle notes
    $notes= "<pre>".$_POST['notes']."</pre>";
    //Prepared statement to prevent sql injection
    $preparedSql = $conn->prepare("UPDATE Placement SET Email = ?, Company = ?, SupervisorID = ?, Notes = ? WHERE userID = ?");
    $preparedSql->bind_param("sssss", $email, $company, $supervisor, $notes, $selectedID);
    $preparedSql->execute();

    mysqli_close($conn);
}
```

Figure 11: Source code for updateDB() function

4.3 General User Tasks

Logging In

To log into the website, the user must submit their ID and password. The system hashes the entered password using PHP's *password_hash()* function. It then retrieves the current password hash and admin level of the user stored on the database using a prepared statement. The two hashes are compared using the *password_verify() function*. If successful, the 'userID' and 'Admin Level' Session variables are initialised with the ID and the admin level respectively, and the user is redirected to the home page.

```
$result=$preparedSql->get_result()->fetch_assoc();
$preparedSql->close();
if(result!="") {
    if(password_verify($password, $result['PasswordHash'])==true){
        //get Admin Level
        $_SESSION["accessType"] = $result['AdminLevel'];
        $_SESSION["userID"] = $username;
        mysqli_close($conn);
        header("Location: index.php");
    }
    else{
        mysqli_close($conn);
        echo "Incorrect Username or Password";
    }
}
```

Figure 12: Part of the login function handling SQL execution and Session variable assignment

Forgotten Password

If a user forgets their password when logging in, they can reset it by following the Forgot Password link on the login page. After submitting their user id, the system will retrieve their email address from the database and update the forgotPassword attribute with a randomly generated 50-character token. An email will then be sent to the address requesting they follow a link to follow, with the token as a URL parameter using PHP's *mail()* function.

```php
if(isset($_POST['passwordRemind'])){
    $preparedSql =  $conn->prepare("SELECT Email, Forename FROM Placement WHERE userID = ?");
    $preparedSql->bind_param("s",$username);
    $preparedSql->execute();
    $result=$preparedSql->get_result()->fetch_assoc();
    $preparedSql->close();

    if($result == true){
        $token = bin2hex(random_bytes(50));
        $email = $result['Email'];
        $forename = $result['Forename'];

        $preparedSql = $conn->prepare("UPDATE Placement SET resetToken ='".$token."' WHERE userID=?");
        $preparedSql->bind_param("s", $username);
        $preparedSql->execute();
        $preparedSql->close();
        $message = "Hello ". $forename . "\r\n A password reset request has been submitted for your account.
        \r\n Please follow this link to reset your password: <a href=\"https://10.72.98.152/resetPassword.php?token=".$token."\">Reset Password</a>";
        $message = wordwrap($message, 75, "\n");
        $headers = 'MIME-Version: 1.0' . "\r\n";
        $headers .= 'Content-type: text/html; charset=iso-8859-1' . "\r\n";
        $headers .= 'From: PlacementPortalAuto@cardiff.ac.uk' . "\r\n";

        mail($email,"Pre-Placement Portal Forgotten Password", $message, $headers);
    }
    else{
        echo "No user with that ID found.";
    }
}
```

Figure 13: Part of code responsible for requesting a new password after forgetting. Checks that user ID exists then sends an email with the link to the reset password page

On the resetPassword page, the user will be prompted to enter and confirm their new password. On submission, the two entries are compared to ensure they match. If so, the password is hashed. Finally, the database is queried to update the passwordHash in the record containing the token with the new value, and the token is deleted from the record. If successful, the user is redirected to the login page, else an error message explaining what went wrong is displayed.

Changing Known Password

Changing a known password after logging on is much simpler than the previous method. Rather than sending a token to the users email address, the form simply requires the user to input their current password as well as the new one. When submitted, the hash of the current password is compared with the hash stored on the database using PHP's *password_verify()* method. If the verification is successful and both entries of the new password match, the current hash in the database is updated with the hash of the new password.

Adding New Records to Database

Besides administrators adding students to the database directly using PhpMyAdmin, all Placement Team Members can add new records by submitting a CSV file on the listStudents page.

List of Students on a Programme																
Input:																
Course Co	U%															
Programm	2															
Results:																
Student C	Surname	Forename	Forename	Course	Course Tit	Acad Year	Block	Occl	Moac	Fee Status	Reg Statu	Univ Emai	Home Em	Tutor(Hon	Tutor(Assoc)	

Figure 14: Example CSV file for uploading as seen on Excel. Files must be in this format with student records below the schema.

The system first checks the file format before attempting to insert each row of student information as a new record in the database. For each new record, a 10-character alpha-numeric string is randomly generated to act as a temporary password. The hash of this password is stored on the record with the other data.

After each record is inserted, an email is automatically sent to the address using PHP's *mail*() function. This email informs the student that their account has been created and includes their temporary password.

If any student record is missing an ID or email address on the CSV that record will not be added to the database and an error message will be displayed stating that this occurred.

4.4 Document Management

Staff Experience

Getting General Summary

When non-student accounts open the awsDocManager the *getSummary()* PHP function is automatically called as the page is loaded. This method is used to identify which students have not submitted necessary documentation and populate the summary div element based on this information.

After initialising counts for the number of missing documents, instantiating the S3 Client object, and creating a connection to the database and SQL query is executed to retrieve the id and names of all student accounts on the system. The function iterates through the list of students, searching for any missing documents.

This is achieved by repeatedly calling the S3 Clients *listObjectsV2()* method using a for loop, providing the bucket ID as well as by using a prefix generated using the Student ID and current iteration of the loop to check each document type. This works because when a document is uploaded for/by a student the directory in the bucket is set up in a particular way, with documents being separated by type. The method returns the child contents of the prefix as an array.

If the array returned is empty, then no document of the current type being checked has been found and the function increments that document types missing counter. The student's name is also appended to the relevant array storing a list of all students missing the document.

```
for($i=1;$i<=5;$i++){       //Search through each of the 5 required document types
    $searchDoc = $s3Client->listObjectsV2([
        'Bucket'=>"csprojectfiles",
        'Prefix'=>"StudentDocs/".$targetStudent."/Doc".$i."/"
    ]);

    if(empty($searchDoc['Contents'])){      //Check if the current doc type is found. If
        switch($i){
            case 1;
                $confirmationMissing++;
                $confirmationArray[]=$targetStudentName;
                break;
            case 2;
                $hsMissing++;
                $hsArray[]=$targetStudentName;
                break;
            case 3;
                $agreementMissing++;
                $agreementArray[]=$targetStudentName;
                break;
            case 4;
                $overseasMissing++;
                $overseasArray[]=$targetStudentName;
                break;
            case 5;
                $authorisationMissing++;
                $authorisationArray[]=$targetStudentName;
                break;
        }
    }
}
```

Figure 15: Code snippet from getSummary() function

Once the directories of each student have been searched, the counts are used to display the total number of students missing each document type on the summary element. Beneath each total the names of the students are also displayed so they can be easily found using the profile selector.

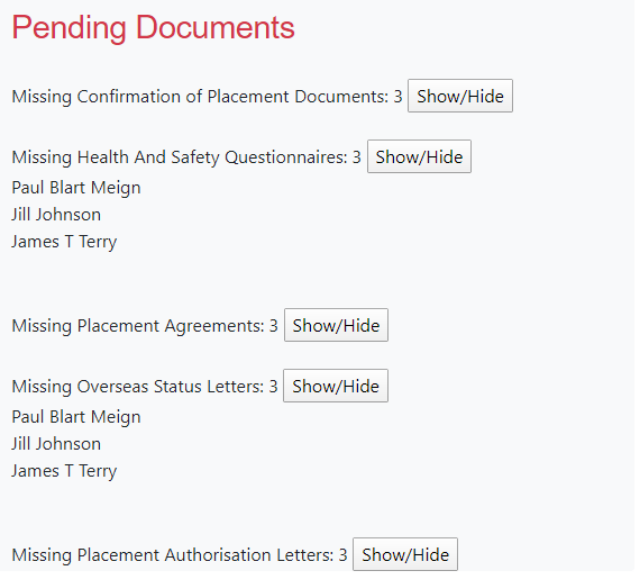


Figure 16: UI for Pending Document Summary.

Clicking the Show/Hide button next to each count triggers the showHide() JS function, which simply toggles the display mode of the HTML elements containing the name lists. This helps reduce the amount of clutter and information on the page by default without sacrificing the user's ability to identify who is missing key document types.

Displaying Student Information

On page load for the awsDocManager the *populateProfileSelect()* PHP function is automatically called. Like the *getSummary()* function this opens a connection to the database and executes a SQL query requesting the ID and name values of each student account stored. It then iterates through these to generate the options for the ProfileSelector drop-down list, with each option being assigned the students ID as its value.

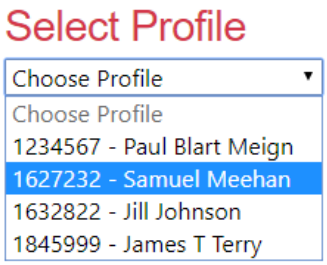


Figure 17: UI for profile selector. Options are based on current student records in the database.

When the user selects an option from the list the *getProfileInfo()* JS function is called. This function posts the selected students ID to getProfile.php. This code assigns the selected students ID to a Session variable and constructs an HTML response for the Ajax request.

```
function getProfileInfo(){
    var selectElement = document.getElementById("profileSelector");
    var selectedID = selectElement.options[selectElement.selectedIndex].value;
    $.ajax({
        url: 'getProfile.php',
        type: 'post',
        data: { "callFunc1": "1", "selectedID":selectedID},
        success: function(response){
            $("#output").html(response);
            }
    });
}
```

Figure 18: Source code for getProfileInfo() function.

To do this another SQL query is executed, this time retrieving the selected students profile information for display. Additionally, an S3Client object is instantiated and the objects *getPaginator()* and *listObjectsV2()* methods are used to retrieve download links for all the students files.

```
if(sizeof($totalfiles['Contents'])>0){  //If any files are found
    echo ("<table border=1 cellpadding=5 cellspacing=0 class=whitelinks>\n");
    echo("<tr><th>Document</th><th>Download</th></tr>\n");

    $results = $s3Client->getPaginator('ListObjects', [
        'Bucket' => "csprojectfiles",
        'Prefix' => "StudentDocs/".$selectedID
    ]);

    foreach ($results as $result) {             //For each file found
        foreach ($result['Contents'] as $object) {
            $objectExploded = explode("/",$object['Key']);
            $docType = $objectExploded[2];
            $docName = end($objectExploded);
            if($docName != ""){
                echo("<tr><td>$docType</td><td><a href=https://csprojectfiles.s3.eu-west-2.am
                echo("</tr>\n");
            }
        }
    }
    echo "</table><br>";
}
else{
    echo "No Files Found";
}
```

Figure 19: Code snippet for searching for existing documents to display

When successful, the HTML response to the Ajax request is set as the inner HTML of the output element via JS.

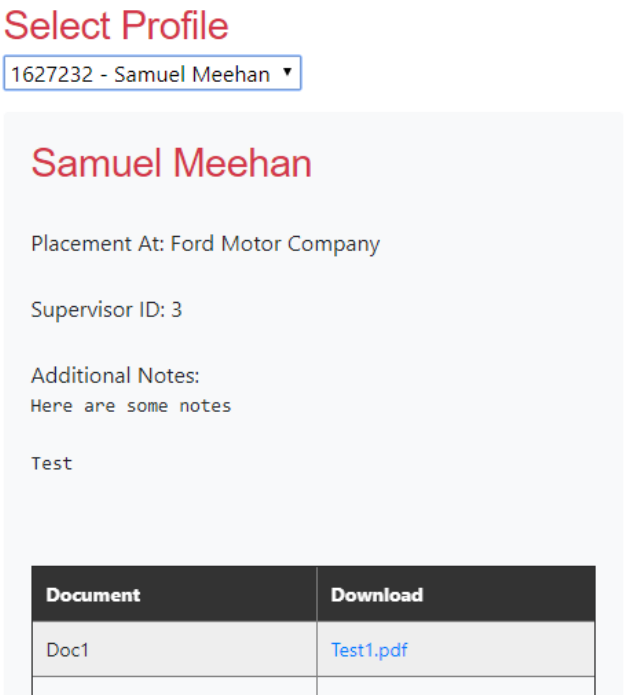


Figure 20: UI of profile selector after a student has been selected.

Uploading Student Documents

When the user has selected a student and clicks the Update Profile button for a student they are taken to the studentDocUpload page. This page serves all users, however depending on the admin level of the user (found by checking the session variable in PHP), some HTML elements may or may not be displayed.

```
<div class = bg-light rounded style="<?php if($_SESSION['accessType']!=0){echo 'display:none;';}?>
```

This prevents staff from uploading a student's CV or changing the permission state for sharing said CVs in the database. Besides that, they are able to upload all documents, though in most cases the student is expected to upload some document types.

For each document type the *getFiles()* PHP function is called to check if a file has already been uploaded, with the document type's location in the S3 bucket being passed as an argument. The function instantiates an S3Client object and retrieves any files at the location using the *listObjectsV2()* method. The function then outputs an HTML response stating if any files were found or not. Initially the names of the files were also output, however this was removed as it seemed redundant, and if many files were already stored the output could potentially take up too much space.

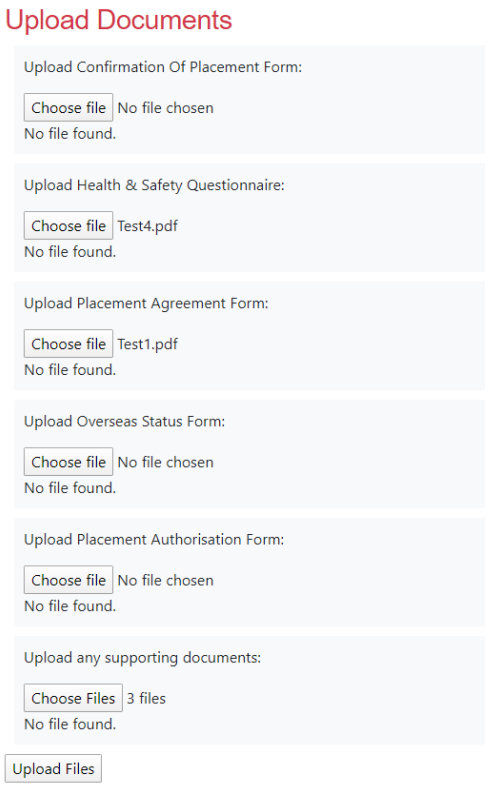


Figure 21: File Uploader UI after selecting several files.

The user can select which files they want to upload for each document type using the file upload buttons. With the exception of supporting documents, they are only able to select a single file to upload at once, as there should not be any need for files of the same type. However multiple generic supporting documents can be uploaded simultaneously.

Upon clicking the Upload Documents button at the bottom of the form the *cvShare()* and *docUpload()* functions are called. *cvShare()* simply checks if the checkbox used to indicate whether CVs are permitted to be shared has been changed, and is therefore irrelevant to staff. *docUpload()* checks that each of the files uploaded meets the size and format requirements (currently files must be under 10MB and in PDF format). If these are not met for any file, an error message is displayed explaining the violation and the upload is stopped.

If all files meet the requirements, the function iterates through each file and tries to move them from their temporary location to the correct location within the S3 Bucket using the S3Client objects *putObject()* method. The code also checks if any files already existed in the S3 Bucket at the targeted directory and deletes them, with the exception of supporting documents. Therefore, only the most recently uploaded version of each file is stored. Finally, a message is displayed stating whether that the upload was successful, or in the event of an error with the S3 bucket what went wrong.

Updating Student Account Information

The student profile update form is accessed from the same page as the document upload form. Similarly to that form, the Admin Level session variable is used to set which elements of the form are displayed. The *getProfileInfo()* function is also called to query the database for the students

information, and prepopulate the form with the results.

```php
function getProfileInfo($selectedID){
    include("dbConnect.php");
    $sqlQuery="SELECT Email, SupervisorID, Company, Notes FROM Placement WHERE userID =".$selectedID;
    $results = mysqli_query($conn, $sqlQuery)->fetch_assoc();

    $email = $results['Email'];
    $supervisor = $results['SupervisorID'];
    $company = $results['Company'];
    $notes = strip_tags($results['Notes']);
    mysqli_close($conn);

    return array($email, $supervisor, $company, $notes);
}
?>
```

Figure 22: Source code for retrieving student profile details from database

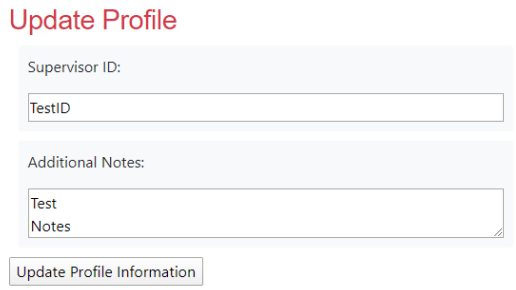


Figure 23: UI for profile update form as seen by a member of the placement team.

When the Update Profile button is clicked the *updatedDB()* PHP function is called. This function connects to the database using *dbConnect.php* then executes an SQL query to update the selected student's data with the data posted from the form. For the most part this is done simply uploads the posted data as is, however due to using a textarea input the contents of "Additional Notes" is wrapped in <pre> HTML tags to preserve their formatting when retrieved later. Also, as the user is able to input their own data a prepared statement is used in place of mysqli.

```php
function updateDB($selectedID){
    include("dbConnect.php");
    $email = $_POST['email'];
    $company = $_POST['company'];
    $supervisor = $_POST['supervisor'];
    //Handle notes
    $notes= "<pre>".$_POST['notes']."</pre>";
    //Prepared statement to prevent sql injection
    $preparedSql = $conn->prepare("UPDATE Placement SET Email = ?, Company = ?, SupervisorID = ?, Notes = ? WHERE userID = ?");
    $preparedSql->bind_param("sssss", $email, $company, $supervisor, $notes, $selectedID);
    $preparedSql->execute();

    mysqli_close($conn);
}
```

Figure 24: Source code for updating database with newly input profile data.

Student Experience

For the most part students upload documents and update their profiles in the same way as staff. However unlike staff who can select specific students, students are only able to view and modify their own accounts and documents. They therefore do not have access to the awsDocManager page and will in fact be redirected if they attempt to access it. When accessing the studentDocUpload page rather than a "selectedID" session variable their own "userId" Session variable set at login is used.

Uploading Documents

Uploading documents works more or less the same for students as staff. The only difference is that where staff are able to upload all document types, students are unable to upload Overseas Status letters or Placement Authorisation Letters, as these elements will not be displayed on the form if the "accessType" Session variable shows the user is a student.

Figure 25: UI for uploading documents as seen by students.

Updating Account Information

Likewise, updating account profile information works similarly for students as it does for staff. The only difference is that when the form is loaded student accounts will be shown fields for their email and company, and not be able to update their supervisor or additional notes fields.

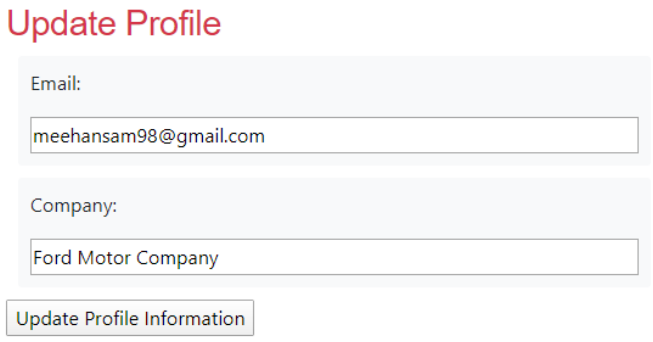


Figure 26: UI for updating profile information as seen by students.

Setting CV Sharing Permission

Students can set whether they allow their CV to be shared using the checkbox on the document upload form. When the studentDocUpload page is loaded the checkbox's state is set using the

getAllowStatus() PHP function. This function simply queries the database for the value of the shareCV field, checking the box if it is set to 1.

As previously stated, the when the form is submitted the *cvShare()* function is called prior to the documents being uploaded. This function sends a SQL query updating the shareCV value for the student with a Boolean value depending on the current state of the checkbox on submission.

Viewing Profile Information/Downloading Documents

As students do not have access to the awsDocManager page their profile information and uploaded documents are instead on the website's landing page. There is otherwise very little change from how The table is constructed using the *outputData* PHP function identical to the one constructed when a student is selected on the awsDocManager, albeit using the users "userID" session variable in place of an ID they select themselves.

Amazon S3 Client

In order to communicate with the Amazon S3 bucket used for document storage the website makes use of the AWS SDK for PHP provided by Amazon. Any function that interacts with the S3 bucket is required to instantiate an S3 client object. For the sake of maintainability, the code for doing this is written separately in *s3Client.php* and requested when needed by said functions. Therefore, if the credentials for the S3 bucket ever change (i.e. if it needs to be moved to another AWS account), future developers can easily do so at a single point.

```php
<?php
    require 'Aws/aws-autoloader.php';

    use Aws\S3\S3Client;
    use Aws\S3\Exception\S3Exception;

    include("s3Client.php");
```

```php
<?php
    $s3Client = new  Aws\S3\S3Client([
        'version'     => 'latest',
        'region'      => 'eu-west-2',
        'credentials' => [
            'key'    => 'AKIA4OL25UH63BOA5ZPP',
            'secret' => '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',
        ],
    ]);
?>
```

Figure 27: Code snippet of initialisation of S3Client object

4.5 Event Management

This section will provide a more in-depth explanation of how the implementation meets the requirements for managing the various events hosted throughout the placement process. As mentioned earlier, the website makes extensive use of Eventbrite for these tasks, connecting to an

existing Eventbrite account via their API. It should be noted that in order to do this an OAuth token set up for the account (and sometimes the organisation ID value) must be passed with the API calls. As the account being used is not expected to change once set up, these credentials are currently hard-coded.

Staff Experience

Creating Events

To create events staff must navigate to the createEvent page. To successfully submit the form, the user must provide the event with a name as well as a valid capacity and start/end times. Failing to provide valid required data will result in the event not being created, and an error message will be displayed to the user explaining what needs to be added or changed. The user may also provide a description of the event, though this is optional.

Upon filling out and submitting the form the *createEvent()* PHP function is called with the token and organisation ID. This function creates an array with the submitted data and then converts this array into an HTTP query using the in-built *http_build_query()* function. Using curl this is then posted to the API, which will attempt to create the event as requested on the Eventbrite account. The response from the server is then checked to see if the event was successfully created.

```
//Create array of API attribute values
$data = array(
        'event.name.html'=>$eventName,
        'event.start.timezone'=>"Europe/London",
        'event.start.utc'=>$startTime.":00Z",
        'event.end.timezone'=>"Europe/London",
        'event.end.utc'=>$endTime.":00Z",
        'event.currency'=>"GBP",
        'event.capacity'=>$capacity,
        'event.invite_only'=>true

);
//url-ify the data for the POST
$data = http_build_query($data);

//Post data to API
$url = "https://www.eventbriteapi.com/v3/organizations/".$orgID."/events/?token=".$token;
$curl = curl_init($url);
curl_setopt($curl, CURLOPT_URL, $url);
curl_setopt($curl, CURLOPT_POST, 1);
curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
$serverOutput = json_decode(curl_exec($curl),true);
curl_close($curl);
```

Figure 28: Code Snippet of createEvent() function

If so, the *createTicket()* function is then called. Tickets are necessary for users to register to events. Eventbrite can support the sale of multiple different kinds of tickets for an event, however according to the Placement Team representative in meetings this is not necessary for the events being hosted. The function for creating the ticket works similarly to the previous one. As only a single free generic ticket is required, the user is not required to enter any details except the quantity of tickets available (aka Event Capacity), with the remaining information for the API call being hardcoded.

```php
function createTicket($eventID,$token, $ticketQuantity){        //Currently only allowing free tickets
    $url = "https://www.eventbriteapi.com/v3/events/".$eventID."/ticket_classes/?token=".$token;
    $ticket = array(
        'ticket_class.name'=> "Standard",
        'ticket_class.capacity'=> $ticketQuantity,
        'ticket_class.quantity_total'=> $ticketQuantity,
        'ticket_class.free'=>true,
        'ticket_class.cost'=>"GBP,0"
    );
    //url-ify the data for the POST
    $ticket = http_build_query($ticket);

    //API Call
    $curl = curl_init($url);
    curl_setopt($curl, CURLOPT_URL, $url);
    curl_setopt($curl, CURLOPT_POST, 1);
    curl_setopt($curl, CURLOPT_POSTFIELDS, $ticket);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    $serverOutput = json_decode(curl_exec($curl));
    curl_close($curl);
```

Figure 29: Code snippet of createTicket() function

Finally, while the event and tickets have been created it has not yet been published. This prevents anyone but the owner of the account from seeing and registering to the event. This is done using the *publishEvent()* function, which sends a last curl request to the API requesting the event is published. Finally, the user is redirected to the viewEvents page, which should now display the newly created event in the table.

Editing Events

To edit an event, upon navigating to the updateEvents page, select an event from the drop-down list and submit the updated details. On page load the *getEvents()* PHP function is called to retrieve an array of all currently live events on the account by sending the API a request using curl and decoding the JSON response. This is then used to populate the forms drop-down list, allowing the user to select an event to update.

Upon selecting an event, a JS function is called. This function sends a GET request to the API using xhttp to find the specific events data, and then updates the form fields with the data that is returned.

```javascript
<script>
//Following script sends a get request to API for selected event details, and sets values of form fields to current value.
   eventSelector.onchange = function(){
       var selectedEvent = document.getElementById("eventSelector").value;
       var xhttp = new XMLHttpRequest();
       var token = "<?php echo $token; ?>";
       xhttp.open("GET", "https://www.eventbriteapi.com/v3/events/"+selectedEvent+"/?token="+token);

       xhttp.onload = function() {

       var currentInfo = JSON.parse(xhttp.response);
       var currentName = currentInfo.name.text;
       var currentStart = currentInfo.start.local;
       var currentEnd = currentInfo.end.local;
       var currentDescription = currentInfo.description.text;
       var currentCapacity = currentInfo.capacity;
       var currentlyListed = currentInfo.listed;
       var currentlyInvite= currentInfo.invite_only;

       document.getElementById("name").value = currentName;
       document.getElementById("startTime").value = currentStart
       document.getElementById("endTime").value = currentEnd;
       document.getElementById("description").value = currentDescription;
       document.getElementById("capacity").value = currentCapacity;
       document.getElementById("listed").checked = currentlyListed;
       document.getElementById("invite").checked = currentlyInvite;

       };
       xhttp.send();

       xhttp.onerror = function() { // only triggers if the request couldn't be made at all
         alert(`Network Error`);
       };
   }
```

Figure 30: JS Code Snippet that handles loading of form data to reflect selected event

The user is then free to modify these fields. Upon submitting the form, the *updateEvent()* PHP function is called. Like when creating an event from scratch, this function assigns the submitted values to an array, uses this to create an http query and finally posts this to the API using curl.

Deleting Events

Staff can select the events they wish to delete on the view table using the checkboxes on the side of each row. On clicking the "Delete Events" button, the *deleteEvents()* PHP function will be called. This function will iterate through each selected event using a foreach loop. For each event, the *getOrders()* function is called. This function simply sends a GET request to the API to see if any tickets have already been "sold". If not, a DELETE request is sent to the API using curl, else an error message is displayed stating the user cannot delete events with existing orders placed.

The reason for this check is because it is currently impossible to delete events if any tickets have been "sold". This is due to a restriction set by Eventbrite to prevent events from being cancelled by the organiser without attendees being handled/refunded. Unfortunately, this also applies to free events. So far, no workaround has been found.

Student Experience

Viewing Events

When the studentEvent page is opened, the server calls the *getOwnedEventAPIDetails()* function with the accounts OAuth token. This function calls the Eventbrite API using curl to retrieve the details of all live events on the tokens account. These details are extracted using json_decode() into an associative array.

Following this, an SQL query is made requesting the contents of the Events field for the current user. This returns a comma-separated list of all the events the user has registered to.

Event	Start Time	End Time	Description	Tickets Remaining	
Test	2020-05-26 - 15:00	2020-05-26 - 17:00		39	Already Registered

Figure 31: View Event table UI as viewed by a student, after registering to the event

A table to display the upcoming events is then constructed. After constructing the header row, a for loop is used to iterate through the array generated earlier to extract the necessary event details. Most of the required information can be taken directly, or in the case of Start/End dates and times by taking a substring of the relevant array values. However in order to get the remaining ticket number for each event another curl request to the Eventbrite API must be made by calling the *getEventCapacityDetails() function,* returning this data as another associative array. The data includes the events total capacity and number of tickets already reserved, which are used to calculate the remaining number of tickets for each event.

Finally, the *registeredTo()* function is called where strpos() is used to find if the event ID currently being worked with exists somewhere in the comma-separated list. If found, the table informs the user that they have already registered to it and cannot register again. If not, a button for registering to the event is displayed on the row.

Registering Interest in Events

To register interest in an event the user clicks on the Register button for the desired event. The system uses an embedded checkout widget that was provided by Eventbrite. The code was modified using PHP so that the widget for each row of the table is given the correct EventID during the table's construction, which ensures the user is signed up to the correct event.

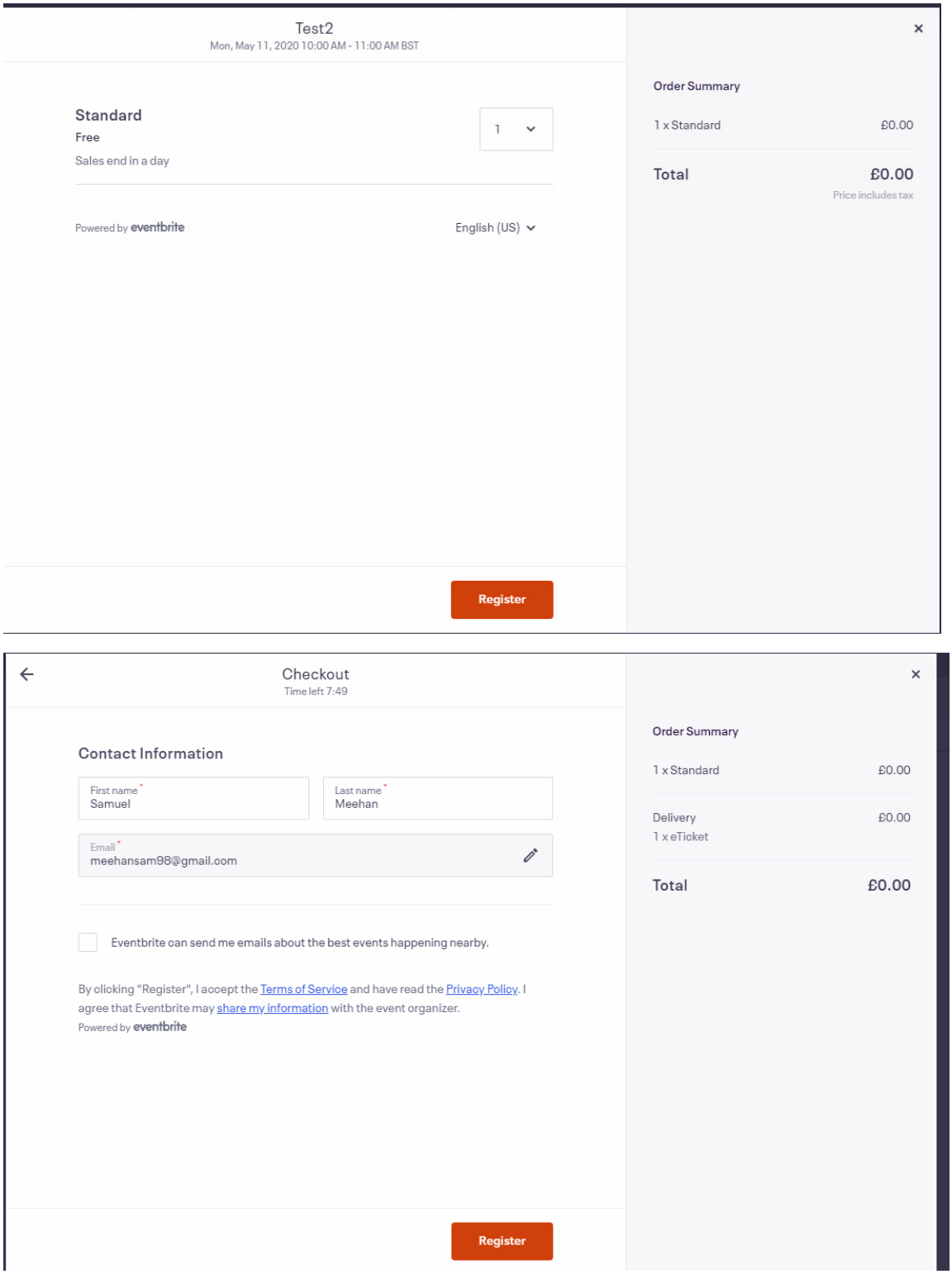


Figure 32: UI of embedded checkout provided by Eventbrite

Once the user clicks a Register button the embedded checkout for that event is opened. The User can proceed to select a ticket and checkout or otherwise cancel the order. Once the checkout is complete, an Ajax call clears the webpage body (in order to prevent quick users from trying to register again), sets the $_POST['eventRegistered'] value to the ID of the event the user has registered to, and refreshes the page. The widget also automatically sends two emails to the event owner and ticket buyer, confirming the order.

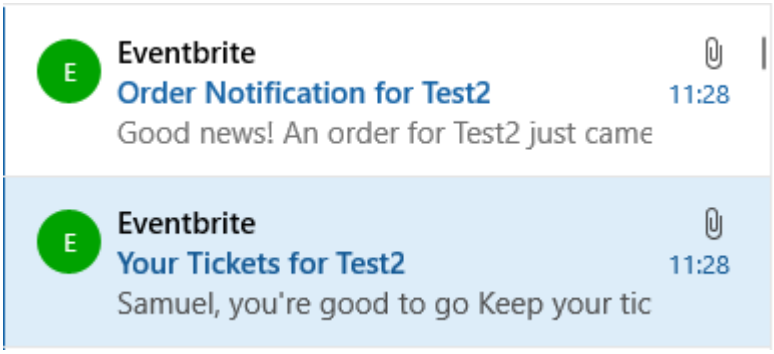


Figure 33: Automated emails sent by Eventbrite on student registration to event. Note, in normal circumstances the student only receives the bottom email.

When the page is refreshed the change to the $_POST value is detected using the isset() method. This causes the *updatedRegisteredEventsDB* function to call, which connects the system to the database and runs an UPDATE query to append the posted value onto the users Event comma-separated list. This ensures that when the event table is constructed afterwards, the user cannot sign up to the event again.

4.6 Maintainability & Supporting Documentation

Maintainability

As this website is intended to be used in a live environment and will require additional development after the end of the project, ensuring the code base is maintainable and easy to understand is an important consideration.

The implementation therefore attempts to follow common standards and principles to improve readability and make it easier to maintain (Coding Standards The Way to Maintainable Code. 2018).

- Basic steps like ensuring code is correctly indented providing comments and consistent, useful variable names.
- Code cohesion and coupling were also considered, though in the latter case with mixed results (i.e. the decision to significantly modify some pages/functions based on user type rather than keeping them separate).
- Deep nesting was generally avoided. Select statements were used as appropriate.
- Frequently used code (i.e. connecting to the database/S3 Client) and most static credentials were written once and from then on simply referred to, in order to prevent code duplication and reducing the work required if that code needs to be changed.

However, there are several ways this aspect of the website could have been handled better. Maintainability was often improved after refactoring the code rather than during its initial development. For example, towards the end of development there was a push to remove duplicated

code by reusing existing functions. As a result, code quality is not consistent across the entire site and may ultimately have cost more time than it saved.

Documentation

Supporting documentation for the website can be found in the Appendix: Website Documentation.

User Guide

The website's user interface has been designed to be intuitive for users, and in particular those familiar with the university intranet, so that users should be able to use it without additional training or support. However as is good practise a User Guide has also been created to help direct visitors through most common functional tasks.

Technical Documentation

As additional work is planned to finalise the website and better integrate it with the post-placement project's implementation later on, additional supporting documentation has also been created to assist future developers and administrators. As opposed to the user guide, this documentation is intended to assist the completion of non-functional tasks such as:

- Accessing the instance and webserver through common tools like WinSCP/Putty.
- Accessing the database via PhpMyAdmin.
- Working with the various tools used by the website i.e. the S3 Bucket/Eventbrite account.

Hosting the Website

One of the most challenging tasks during the implementation of this project was getting the website hosted and accessible from the web. Firstly, to use OpenStack a request must be made to the university's IT Service desk, who would set up a project that could be used. This took considerably longer than anticipated, though fortunately the request was made early enough that it did not significantly impact the project schedule. Similar delays were caused by ongoing maintenance to OpenStack which resulted in it periodically being taken offline, though again this was primarily during the early part of the project and so did not significantly delay development.

More importantly I had no prior experience with the task and found the existing documentation for OpenStack and its alternatives difficult to understand. The time spent learning how to set up the instance and webserver to run the site was an unanticipated delay which took up valuable time. Additionally, while for the most part the migration of code from the project site went smoothly the move did break the automated email functionality of the website. While the actual fix was relatively simple, it took a very long time to work out what the cause was and several attempts to fix it may have affected the overall quality of the implementation i.e. additional libraries downloaded to the server.

Lockdown

Another significant disruption to the project was the global outbreak of Covid-19 and subsequent lock-down. While this did not prevent work on the project from continuing, it did significantly hinder it in several ways.

Minor delays included additional tasks like setting up a VPN to continue accessing the website and files which the university network would otherwise block access to. Additionally, while the IT Service

Desk was not closed it was more difficult to quickly get support after losing physical access to the staff, along with mixed results when submitting support tickets by email. Finally, and perhaps most significantly, the lockdown had a substantial negative impact on my overall approach to the project. Working from the university had provided a sense of separation and structure that I struggled to remain focused without. This exasperated the existing time management issues considerably.

Automated Testing

During the initial design phase of the project it was planned that unit tests would be created to test individual pieces of the website's functionality as they were developed. It was hoped that Selenium Webdriver would be used to construct automated tests for this purpose. This would allow the code to be frequently tested throughout development, hopefully resulting in better quality code.

However due to the time it would take to refamiliarize myself with Selenium and set them up, development of these automated tests was ultimately abandoned.

5.0 Testing and Evaluation

Results

To ensure the website meets the project requirements a number of test cases were created based on the Use Cases constructed during planning. These cases are designed to test how the system responds when the users carry out tasks, using both valid and invalid data where applicable.

The following table shows a summary of the Test Case results. To view the Test Cases in full, see Appendix: Test Cases.

Test Case ID	Test Case	Pass/Fail
TC1	Check user can log in with valid student credentials	Pass
TC2	Check user can log in with valid staff credentials	Pass
TC3	Check user login with invalid credentials	Pass
TC4	Check Forgot Password function works	Pass
TC5	Check student can download their uploaded files.	Pass
TC6	Check students can upload files	Pass
TC7	Check staff can select, view information on and download files belonging to a specific student	Pass
TC8	Check staff can upload files for a selected student	Pass
TC9	Check that invalid files are handled correctly during file upload	Pass
TC10	Check that staff can create new event listings with valid details	Pass
TC11	Check that event validation works correctly during creation	Pass*
TC12	Check that staff can edit existing event listings	Pass*
TC13	Check that staff can delete existing event listings	Pass
TC14	Check that students can register interest in event listings and cannot register multiple times to same event.	Pass
TC15	Check that staff can upload CSV list of students to database	Pass

In the current implementation all test cases successfully pass, though in a few instances i.e. TC12 the pass rate is not 100%. These tests occasionally fail due to the UI not correctly reflecting the new state of the system after a refresh. This is believed to be due to a small delay on Eventbrite's end. This does not affect the back-end functionality of the website and can be easily resolved by simply waiting a short time before refreshing the page again. Even so, given some more development time this behaviour would hopefully be resolved.

User Testing

Initially it was planned that user testing would occur towards the end of the project, with several staff and students currently involved in the placement process taking part and providing feedback on the new system. Unfortunately, due to time constraints and the impact of the coronavirus lockdown this did not take place. However, both the project supervisor and a member of staff responsible for managing placements have been attending weekly meetings from very early on in the project, where they have had the opportunity to provide feedback on the requirements, influence the design and have also been shown live demos of the software as it has been developed.

While not an ideal substitute, this should hopefully ensure that the final result remains fit for purpose and adequately meets the user's needs.

6.0 Conclusion

The website currently meets the vast majority of “Must” and “Should Have” requirements and has stood up well to testing. The main aim of this project was to provide support for the pre-placement process, primarily through managing documents and upcoming events. The website developed is a bespoke solution that, unlike existing solutions, meets this aim at almost no cost to the university.

It cannot be called a total success. Several requirements either did not completely meet their acceptance criteria or were simply not implemented (i.e. exporting files as CSVs). Issues with time management resulted in the testing process being noticeably stunted compared to what was initially planned, resulting in some uncertainty as to whether the current version is suitable for live deployment. While the testing that was done did not detect any significant issues the lack of user testing and unit tests could reasonably be considered cause for concern.

Even so, the end result of the project is a website that if released tomorrow should not only be functional but significantly improve the experience of both students and staff, providing the vast majority of the features and functionality expected of it when it was first proposed and a strong foundation to build on with future development.

7.0 Future Work

7.1 Missing Features

As mentioned, several lower-priority functions were not during development due to time constraints and will require further work to implement. These include:

- Data analysis of topics such as actual student event attendance.
- Detecting and flagging students who meet certain criteria (i.e. those who do not regularly attend events)
- Additional alerting methods besides email (i.e. texting)
- Exporting documents as CSVs

In addition to these many other potential features could be added that did not fit within the initial scope of the project (see Existing Solutions for examples).

7.2 Improvements and General Changes

In addition to these missing features, several aspects of the implementation that were successfully implemented could potentially be revised or improved.

The current UI attempts to emulate the existing intranet pages and succeeds to some extent. However, it remains very barebones and could be significantly improved. This is particularly obvious with aspects like the error/confirmation messages, which could be significantly improved (i.e. moving to modals/JS alerts/etc). Additionally, since responsive design was never only a minor consideration during development this could be significantly improved across the website.

As the website is currently hosted on a university OpenStack project, it requires visitors to either be using one of the university machines or otherwise logged into the Computer Science and Informatics VPN. While this is good for security, it is arguably too much of a hindrance for staff and students, who may wish to access the site from an off-site location. It may therefore be necessary to migrate the website to an alternative hosting service, such as an AWS EC2 instance.

Another issue that could be investigated is website performance, which suffers in some places. This seems particularly apparent when viewing many events. It may be worth investigating whether the source code or hosting solution can be improved to speed up loading times.

7.3 Integration with Post-Placement Project

As was mentioned in the introductory section, another project was developed alongside this one to handle tasks occurring *after* students begin their placements. When combined these two projects are intended to provide a comprehensive tool to support students and staff throughout the entire placement process.

While both project's solutions work independently, and effort was taken to keep one-another informed of development choices, there is currently little if any integration between the pre and post-placement projects as was originally intended. To remedy this, funding has already been secured for additional development to combine the two into a single cohesive solution. This could include such improvements as:

- Integrating the MSAL login used in the post-placement project, so that users do not need to separately log into both applications.
- Combining style/pages/functionality from both projects to provide a single cohesive user experience.
- Eliminating duplicate data by combining both projects databases where appropriate (i.e. account information).

8.0 Reflection

8.1 Difficulties

Effectively managing time was a regular issue throughout the project. While a well-defined work plan had been written up as part of the initial plan, the project fell behind schedule as early as the initial design phase. This was due in part to frequent deviations from the plan (i.e. realising that it would be easier in practise to handle UI work later on in development), and a lack of commitment to following the structure of the agile methodology in favour of a completely ad hoc approach.

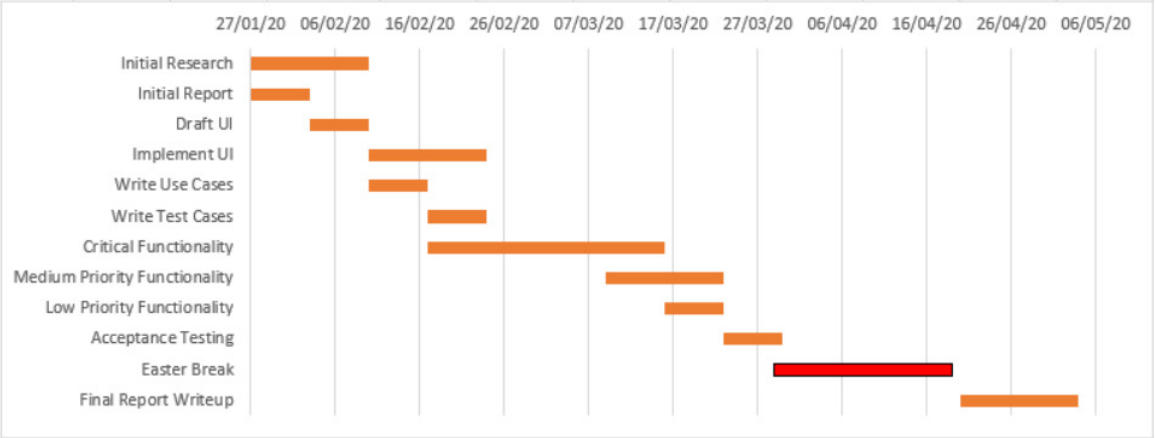


Figure 34: Work schedule taken from Initial Plan

Maintaining discipline when following Agile practises should be a priority in future projects. However, while this and experience will help to incrementally improve planning a more immediate step I could take is to use planning tools such as *Rally* or *Trello.* Tools like this would allow me to easily track progress and break down objectives in a much more structured way than my current ad-hoc approach. In addition, it may be helpful to create more documentation at this stage in future projects i.e. UML diagrams, as this would made recollecting information much easier.

Linked to the above, the failure to ensure code quality was kept high throughout development eventually proved quite costly. While some of this has been fixed through refactoring, even in the latest version of the code there are several ways maintainability could be improved if I had the time. Going forward this should be treated as a higher priority, as it has been made clear that any time saved cutting corners in the initial development will be spent on refactoring later.

8.2 Successes

The decision to frequently demonstrate what had been achieved over the course of the project inadvertently reduced the impact not having time for user tests had on the project. This has not only reinforced my perception that planning should take a greater priority going forward but has also changed my position on some Agile procedures (i.e. sprints), which I previously considered to be unnecessarily frequent.

Developing this website required me to learn how to use tools and software I had previously never touched, so I am quite happy that I was able to successfully use them. Learning to use AWS' cloud storage and Eventbrite's API was a particularly enjoyable challenge and has given me a much better understanding of these kinds of systems and how to use them. Working with the instance and web-server was much more frustrating and has made it clear that I need to do more research on those topics in future, but even so I'm proud of what I've achieved despite the time restrictions.

Bibliography

Antwonne, D. [no date]. How to sell Eventbrite tickets on your website. Available at: https://www.Eventbrite.com/support/articles/en_US/Multi_Group_How_To/how-to-sell-Eventbrite-tickets-on-your-website-through-an-embedded-checkout [Accessed: 20 February 2020].

Coding Standards The Way to Maintainable Code. 2018. Available at: https://www.maansoftwares.com/coding-standards-the-way-to-maintainable-code [Accessed: 22 February 2020].

Fowler, M. 2014. UnitTest. Available at: https://martinfowler.com/bliki/UnitTest.html [Accessed: 28 April 2020].

Jackson, B. 2016. What is the Difference Between HTTP and HTTPS? - KeyCDN. Available at: https://www.keycdn.com/blog/difference-between-http-and-https [Accessed: 9 May 2020].

Microsoft identity platform authentication libraries. 2019. Available at: https://docs.microsoft.com/en-us/azure/active-directory/develop/reference-v2-libraries [Accessed: 4 May 2020].

Nielson, J. 1994. 10 Heuristics for User Interface Design: Article by Jakob Nielsen. Available at: https://www.nngroup.com/articles/ten-usability-heuristics/ [Accessed: 13 February 2020].

Personas | Usability.gov. 2013. Available at: https://www.usability.gov/how-to-and-tools/methods/personas.html [Accessed: 4 May 2020].

PHP vs NodeJS comparison and benchmarks | Thinkmobiles. 2020. Available at: https://thinkmobiles.com/blog/php-vs-nodejs/ [Accessed: 28 April 2020].

Rouse, M. 2018. What is Amazon S3? - Definition from WhatIs.com. Available at: https://searchaws.techtarget.com/definition/Amazon-Simple-Storage-Service-Amazon-S3 [Accessed: 18 February 2020].

Singh, A. 2019. What Is Rapid Application Development (RAD)?. Available at: https://blog.capterra.com/what-is-rapid-application-development/ [Accessed: 7 May 2020].

What is DSDM and the 8 principles. 2018. Available at: https://agilekrc.com/resource/168/what-dsdm-and-8-principles [Accessed: 7 May 2020].