2020/
2021

Computer Science GCE Unit 5 Project Report

DOFE SCHEME PLANNING APPLICATION

LUCA HUELLE

*This document does **not** contain instructions on how to run the application. Please refer instead to the* `README.md` *file located within the same directory as this file.*

TABLE OF CONTENTS

DISCUSSION

For my project, I plan to work on a solution to meet the current organisational needs of the teachers coordinating the DofE award and associated expeditions within my school. The award has been running in the school for many years but with larger groups of students, it can become difficult to keep track of individuals' progress.

Arranging the scheme consists of coordinating which students are on which trips and seeing the section progress of lots of students (~80/academic year) across multiple year groups, each within their separate award schemes (Bronze, Silver, Gold) and each encompassing four sections (volunteering, physical, skill and expedition). Due to the final award being externally moderated, the data organisation must be streamlined and transparent and data easily accessible when required.

For the students themselves, they also have a complicated task managing what sections of the award they still have left to complete and what evidence/material they still need to submit in order to achieve their award.

Broad aims

CURRENT SYSTEM

Currently, the teachers use a combination of a paper based (manually filed) and an online standardised system (eDofE) – this can lead to filing errors, with documents being misplaced, and makes it hard to quickly view overall student progress. Getting new data/information (i.e., for new award levels, expeditions etc.) from students can also be time-intensive and this is often recorded via multiple multi-page paper forms which are easy to lose track of; not to mention that these can be very difficult to categorise and sort efficiently.

Students also need to remember long lists of tasks (including things like what equipment they need to pack, what their expedition dates are, etc.) and to actually fill out and return the forms handed to them. Currently, the system facilitates some of these requirements in terms of allowing students to provide details of the activities they are undertaking but others, like expedition planning in particular, are left for the student to organise by themselves with only some verbal guidance from staff.

Feedback considered

I conducted a brief presentation of an outline of my proposed user requirements (listed in *Involvement of stakeholders*), and my above perspective of the current system, to the current DofE staff team at school (via Skype). This involved discussing the different user groups in turn and examining an overview of how the current process operated. Later on, during the *Investigation*, I returned to these points in more detail on my own.

Their verbal feedback informed the aims below that my new system will try and address. They were also very helpful in pointing out some of its potential limitations, listed below; in particular, those related to the privacy aspects of the software and ensuring appropriate levels of access to restrict access to potentially sensitive student data.

BROAD AIMS OF NEW SYSTEM

My aim is to tie these two systems closer together so that, while keeping their roles distinctly separate (so that there is no duplication of functionality and data), they become more integrated with each other, making data overview easier for the staff involved. Providing students and staff with a way to quickly see the outstanding tasks required in order to complete the award scheme is my main goal.

Overall, I aim to create an application, with a Tcl/Tk Graphical User Interface, GUI (to enable intuitive use), that:

- Provides an account system with a separate interface for students compared to teachers, along with restricted levels of access to stored data
- Allows staff to manage and plan expeditions easily and more effectively than using assorted paper records
- Enables students to manage their progress through the award effectively and submit information to their teachers as required
- Allows staff to view the progress of all or specific students overall as well as within different aspects of the award
- Enables coordinated expedition planning between teachers (sharing trip information) and students (submitting/'ticking off' the listed requirements while viewing the provided information)

POTENTIAL LIMITATIONS OF NEW SYSTEM

- Due to my planned usage of tkinter to create the GUI, the interface will be quite basic and will focus on functionality over aesthetics. Due to the single-threaded nature of Tcl/Tk implementation, some longer processing operations may also cause the GUI to freeze and become unresponsive while Python continues to work in the background.
- The system will not authenticate the credentials of moderators (as this would require additional linking to the external DofE database) so it will be up to the staff to ensure that this feature is not abused for malicious purposes to access potentially sensitive and personal data stored.
- Local data storage will not be designed to be secure due to time constraints on this project. The software will *not* have its own encrypted file format so anyone could theoretically access the raw data within the database as well as any login details stored. This is an issue in terms of GDPR and other privacy laws (especially concerning student data) so access to the software's source and database directories must be controlled separately.
- The data for the system will be stored locally within the database, so in order for a cross-machine/network-wide (i.e., multi-user) setup to function correctly, it would need to be stored/hosted remotely (on the school server for instance).
    - The above could also introduce new file handling scenarios (such as multiple people trying to alter the same record within the database at the same time) which are outside the scope of this project. The software will not support these instances and will be limited to single machine interaction – imagining that all users access the software from a single shared machine.
- The system would not be able to act as a full substitute for paper documents as physical copies of certain forms, such as those detailing parental consent, are needed for school records. Things like signatures to signify consent will also not be able to be replaced by this new system for similar reasons.

INVESTIGATION

Methods of investigation

Due to the COVID restrictions, I was limited in the variety of methods of investigation I could do into the existing system. For example, direct observation of use of the current system was not possible. Added to this was the fact that the DofE scheme was running in a much more limited capacity this year with no out-of-school expeditions planned.

Nevertheless, I conducted all aspects of my investigation with the following question framework:

- How does the current system, as a combination of eDofE and paper records, operate?
- What data on students is needed (inputs), how is it collected and which parts are stored?
- What features/processing capabilities do different users need and what data needs to be output/displayed?
- How can the information obtained be displayed in an easily over-viewable and accessible way? Additionally, which pieces of data are the most important to see quickly?
- What are the main issues with the current system?

These informed the *Questionnaire*'s structure, which parts of the *Analysis of existing* student interface I investigated more closely, priorities during *Document collection* and the areas of focus for my *desk-based research*.

QUESTIONNAIRE

Adapting to the COVID restrictions, I decided to create a Google Form that I could email to the two main DofE management staff for them to complete digitally. This would allow me to collect the most important information on the current system with relative ease – having to act as a substitute for full detailed interviews. However, I would need to ensure that the questions were designed in such a way as to ensure that I collected sufficient and relevant data to build the new system.

The final questionnaire (*Document 1 Questionnaire* shown in *Appendix A – Documents*) hence asks several questions as to the sort of data that is needed on each student and the process of organising expeditions. I also tried to get an idea for what the issues with the current system were and how it could be improved so as to ensure the best user experience for the staff who would use the new system.

Analysis of responses

Regrettably, only one member of the key staff had the time to get the questionnaire back to me (*Document 2 Questionnaire response*) but the basic data provided still proved useful in investigating the existing system. It just meant that I would place greater weight on the other sections of my investigation in terms of informing decisions throughout my project.

The response highlighted how important the current eDofE system was but also raised the fact that the software was not very intuitive and that it could be difficult to find information – clearly these are both issues I need to address. Interestingly, the main stated reason for the continued use of paper records was for "ease of use of parents". This encouraged me to consider parents as a prospective end user of the system. However, in the following question, parents were not stated as a potential user, suggesting to me that a more useful potential feature would be to allow paper records to be printed off from the application to

then be handed to parents – a kind of 'student report' – rather than adding parents as a fully-fledged user to the system.

Another insightful question was identifying the three most important pieces of student data. (I had created this question in order for me to suitably design the student overview section of the application.) These turned out to be the current award level, the progress per section and the student's session attendance.

The questionnaire also revealed that, currently, the staff are also able to edit student data, send messages to students and export records.

DOCUMENT COLLECTION

Thanks to my past involvement with the scheme, I was able to obtain blank copies of the two main forms that I had had to fill in while signing up for the scheme*: Document 3 Enrolment Form* and *Document 4 Parental Consent Form*. These step through the basic data that was needed from each student and helped inform the details of *Data flow* below. They also helpfully categorise it into information concerning: each student; each separate expedition; and the parental consent required.

Analysis of documents

Participant enrolment form – Document 3

This form splits the data required from the student into several key sections, making it easier for staff to gather information from just a glance and for the student to see what the different sections of data they fill in will actually be used for. These sections are:

- DofE Centre information – *centre ID and year group*
- Details on the DofE level – *level itself and whether the student is a new applicant*
- Personal details – *name, gender, date of birth, language and enrolment date*
- Contact details – *phone, email, address, emergency contact*

The latter part of the form is then simply a declaration of consent. This analysis suggested to me that I should organise the interface of my student forms in a similar fashion and perhaps structure the tables within my database likewise. Additionally, I should make sure to include a prompt for the user to confirm that they are happy for their data to be stored within the system.

Parental consent form – Document 4

The first part of the form provides information to the student and parent about the details of the proposed activity – completed by DofE staff beforehand – including the destination, activities and the date/time. The form is then split into separate headers for student information, emergency contact details, medical and any additional information. The risk assessment that follows has also been completed by the teacher in advance and exists as reference for the parent and student. Finally, the form closes with passages on data protection and conduct during the trip in addition to the actual declaration of consent.

ANALYSIS OF EXISTING STUDENT INTERFACE

Again, due to my previous participation in the award scheme, I was easily able to investigate the tools and user interface design of the student-side of the system.

Home Page – Screenshot 1

The home page shows a number of key features including my progress through each of the sections of the award (volunteering, physical, skill and expedition) as a simple graphic. These dials only have three different positions – 'not started', 'in progress' and 'completed'. On the right sidebar, there is a small tab to access messages send to be by staff while on

the left there is an overview of my personal information (which I have the option to edit), as well as menus to access and complete the various stages of my award.

Award details – Screenshot 2

This page allows the user to input details concerning each stage of their award. Shown in the screenshot is a *completed* skills section. The key data fields [data types given in square brackets] are: the timescale [integer] (this is from a choice of 3 or 6 months with a compulsory distribution changing based on award level – see *Document 5 Timescales for each level*); the start date [date]; the type of activity [string]; detail of activity (inc. location) [string]; goals [string]; assessor's name and position [string]; assessor's email [string-format] and their phone number [string-format]. The student then has the option to save this information as a draft or submit it to staff for review and approval. Once they have completed the activity, the student is able to upload evidence (e.g., photos and assessor's reports) from the same page. Once this is done, they can submit the section for final moderation. On the same page, the student can also view any comments that staff have left.

Section Progress – Screenshot 3

This page provides a very useful overview for students of their current progress through each of the four sections of a particular award (with options on the sidebar to view historic progress through previous levels). It provides a graphic of completion status once again alongside other important overview information, mainly concerning dates of completion and details about the type and arrangements of the activity the student has chosen to carry out.

RESEARCH INTO EXISTING SOLUTIONS TO SIMILAR PROBLEMS

Since the system presents an example of a relatively standard data storage and planning tool, desk-based research into existing planning and student moderation applications will be quite useful. It will allow me to see what is common to these systems and the most important functions that are often added to work with available data. Alongside the observations I made in this section, I also considered how I could implement what I had learnt into my final design.

Planning applications

I decided to look at *Google Calendar* as an example of a general-purpose planning application in order to gain an understanding for what information the user entered as well as how this was then displayed. Even though my calendar interface will likely not be as detail-rich, I will be able to use the knowledge gained from this research to appropriately design and layout my interface for students and staff when it comes to organising lessons and other key dates.

The main page of the calendar (*Screenshot 4*) displays **what events** are scheduled and for what **times**. Different event types are **colour coded** – I could adopt a similar styling to differentiate between group sessions, final expeditions and kit checks etc. Times are clearly displayed under each date with a small **superscript for the day of the week** – again, I could include a function that adds the weekday onto any dates displayed to the user.

When planning an event (*Screenshot 5*), the user can enter a **title** as well as **start/end dates** and **times**. The page also offers an option to add specific **'guests'** to an event, set **timed reminders** and include a **description**. I might consider replicating a 'guest' feature in the application by allowing staff to add specific students to meetings (from a set list restricted based on the award level). This would allow me to create a personalised 'events' list for students. Linked to this could also be a reminders/alerts system for students based on these same events.

Academic marking applications

Our school currently uses *Google Classroom* to coordinate assigning work to students and measuring performance, etc. The software has many features which are not applicable to this application but it does have a 'Grades' tab (*Screenshot 6*) which displays a summary of student performance for teachers to easily overview. It allocates a row to each student (which can be sorted by name) and then uses columns for each task the student has undertaken and the grade they received. Every assignment has a **date superscript**, a **class average** and the **status** of each student (which ranges from 'Turned in' to 'Missing'). This table/spreadsheet-like layout is a common standard across these sorts of applications so I will likely use the same style for my staff overview page where each student's progress in the award can be referenced.

I could also mimic the way work is assigned to students in the application with teachers currently being given the option to assign students class material, homework or a quiz – i.e., I could present teachers with various options as to the degree of urgency of the dates/tasks they give to students.

Involvement of stakeholders and their requirements

Informed by my initial discussions, as well as my desk-based research, I am now able to produce the following list of stakeholders in the current system and their requirements of the new system.

STUDENTS

DofE students need a system that can:

- Have separate accounts for different students to allow them to manage their own data relating to the award
- Displays their current progress within the award, along with relevant details such as dates of expeditions
- Prompt them to complete the necessary forms and provide the required data (including uploading assessor's reports and evidence for their various sections) in order to go on expeditions and achieve their award
- Allow them to prepare for an expedition by uploading their route map, referencing an equipment list, etc. and enable them to work through these systematically

TEACHERS/STAFF

The staff DofE team need a system that:

- Has restricted levels of access in order to limit access of sensitive student data to only certain individuals
    - Provides a basic interface to students, which is appropriate in functionality (i.e., limited in the extent of data access compared to staff)
- Can 'push' the details of an expedition (including equipment lists, a route map, dates etc.) to students' accounts so that they can view details on their next login
- Allows them to view the details (level, age, etc.) and progress (in the activities within each section) of students currently undertaking the award – ideally in an easily overview-able form – e.g., in a simple table
- Can produce hard copies of specific records/group info for future reference

MODERATORS

The DofE moderating team need a system that can:

- Output the data required in an easily sharable and overview-able format
- Filter/search all stored data so that only specific requested data is visible

- Save the data requested to a file and load previously saved/backed up data from an existing file

Data flow

INPUT DATA

The following list of data points is a culmination of this investigation process (addressing my *second initial prompt* at the beginning of this process) and will serve as a highly useful reference point during the *Design**Error! Reference source not found.*** section in terms of the data structures I create.

Per Student

- User ID
- DofE Centre ID
- Year Group
- DofE Level – *Bronze/Silver/Gold*
- Full Name
- Gender
- Date of Birth
- Address
- Phone and email contact
- Emergency contact details
- Primary language
- Enrolment date
- Fee/payment status

Per award section (per student) (volunteering, physical, skill)

- Start date
- Length (selection limited by level)
- Current progress (variable)
- Type of activity
- Activity details
- Goals
- Assessor's full name
- Assessor's phone and email contact
- Activity evidence/assessor's report

Per expedition/event

- Start date
- Event length
- Students involved
- Location/destination
- Equipment list
- Route map
- Main activities
- Risk assessment/Hazards

PROCESSING/OUTPUT DATA

Based on the information I gathered through my questionnaire, analysis of the student interface and by investigating similar systems above, I have also been able to compile this outline of the processes carried out by the existing system together with the outputs these produce:

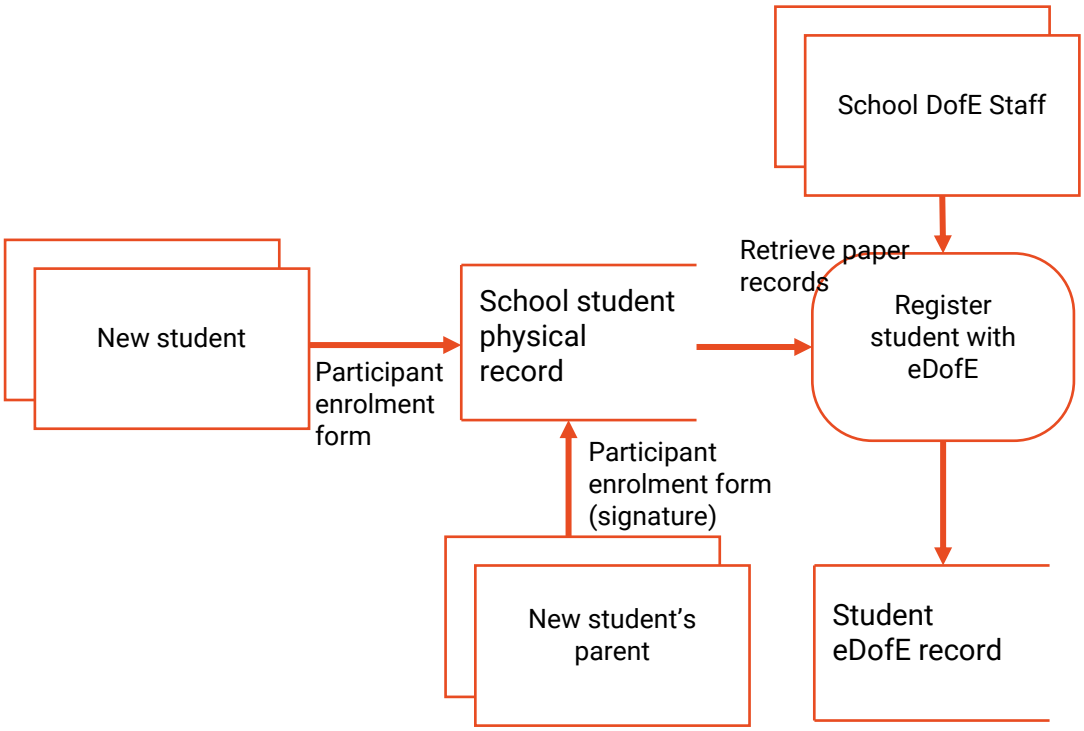
- Allow staff to edit student submitted data
- Allow staff to send messages to students
- Export PDFs/physical copies of student data for permanent physical records
- Export digital copies of student tables for later retrieval/backup
- Allow students to upload evidence for each section
- Visualise a student's progress through an award section
- Produce an overview of all participants' progress

DATA FLOW DIAGRAMS FOR CURRENT SYSTEM

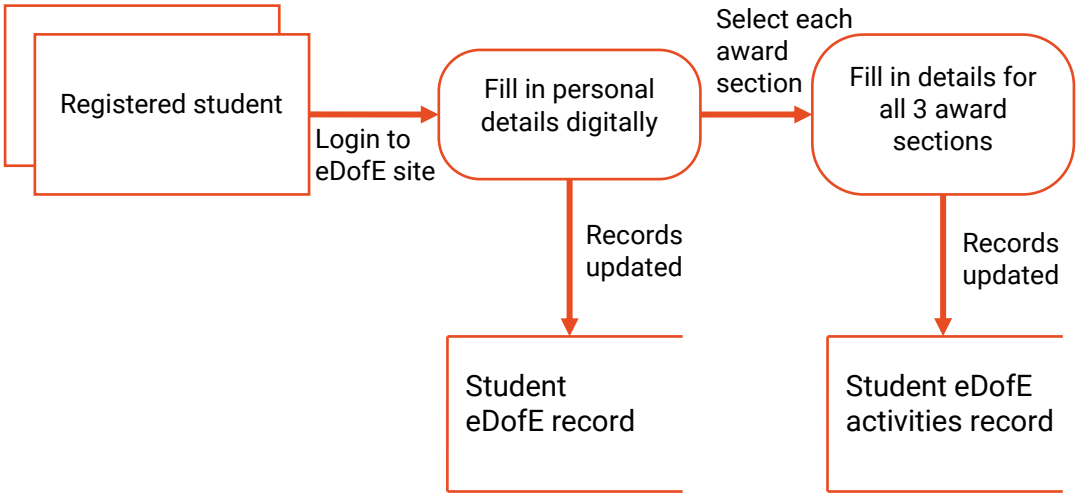
Data flow diagrams can be used to show the main processes within the current system and how data moves into, out of and between different areas/aspects of the system.

A key of symbols used is shown in Appendix C – Diagram and notation keys.

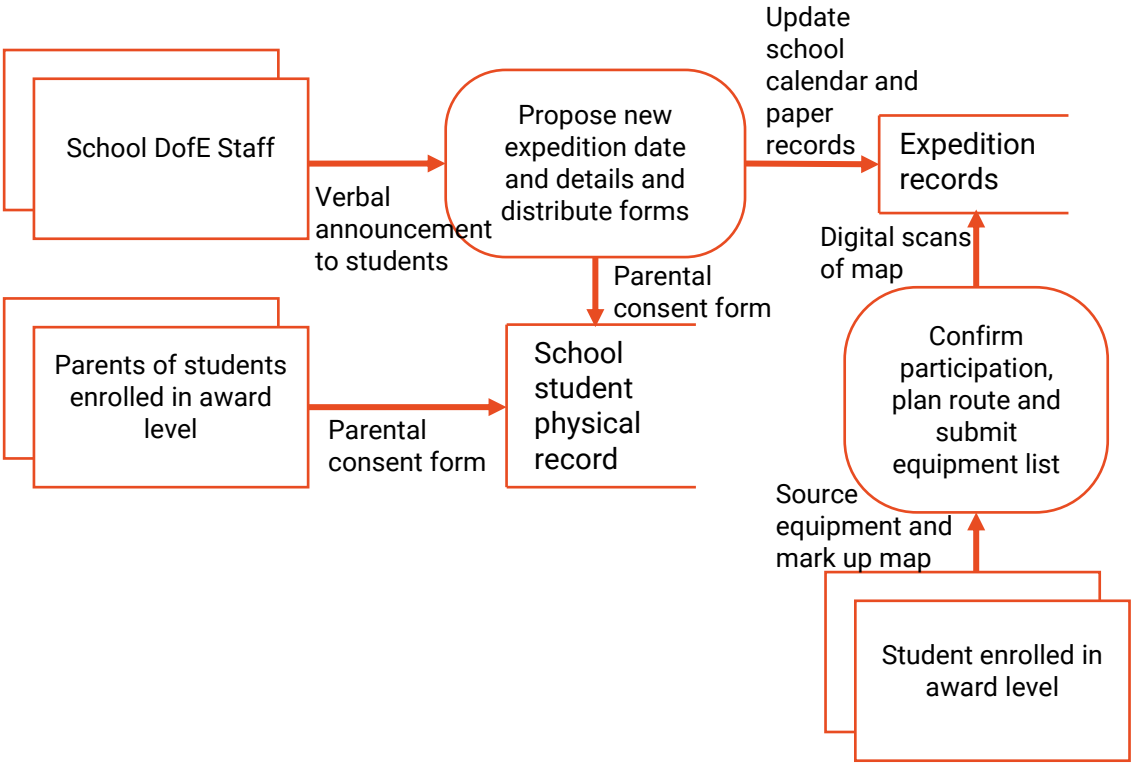
Registering a new student



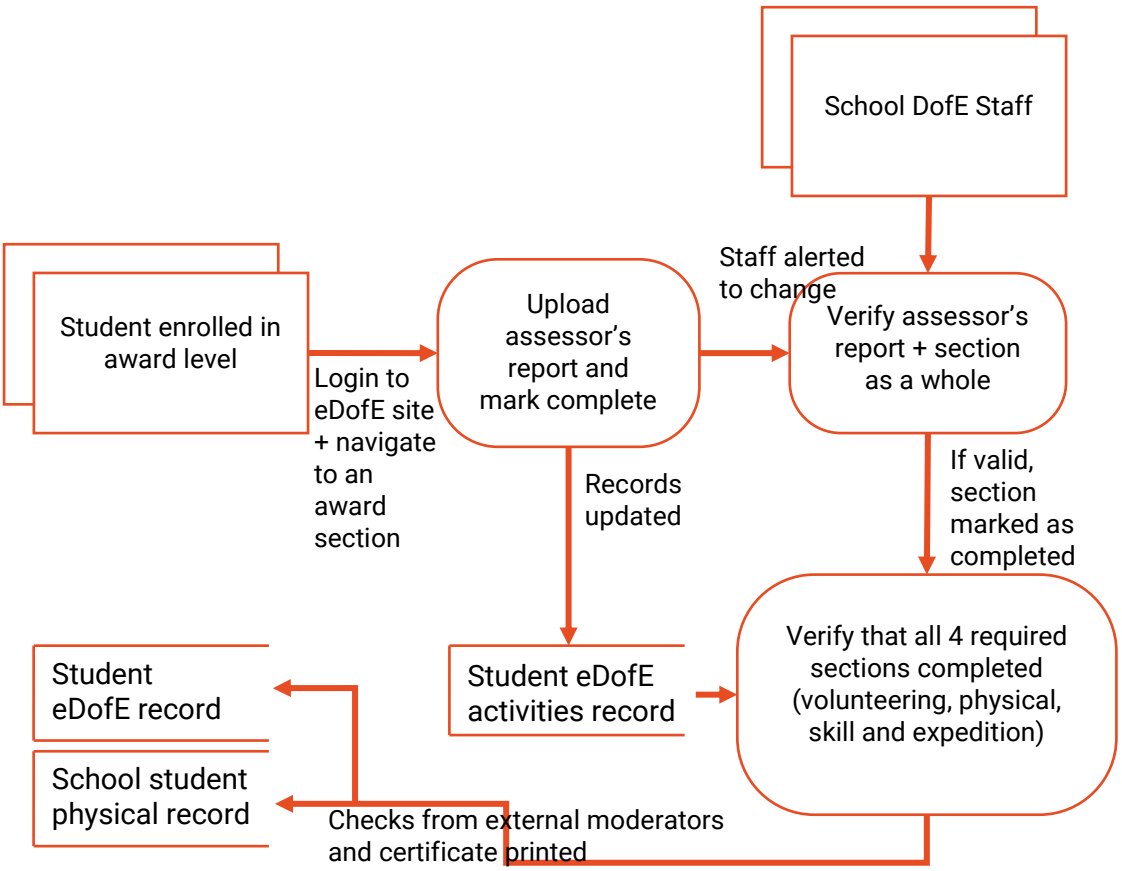
Setting up a student 'profile'/detail



Preparing for an expedition (on an award level basis)



Completing a level of the award



Limitations of current system

I have identified several weaknesses in the current system which therefore become key aims for me to address during the development of the new system.

Please note that because of the inability to observe the use of the current system and the fact that the scheme is running in limited capacity this year, some of the issues discussed here are potential/hypothetical issues rather than concretely observed ones.

- The online system (eDofE) can be difficult to use and unintuitive at times leading to frustration and increasing the time it takes for minor tasks. This is due to a variety of factors including interface layout with many submenus and the way data is grouped, which can result in information being difficult to find (raised in *Analysis of responses*).
- The current system is spread across both digital and paper records meaning organising what needs to be collected from students can be confusing. Additionally, it can make it hard for the students themselves to know where they need to be submitting information. Sometimes the school requires a physical consent form, other times they need the information entered into eDofE.
    - The paper forms themselves are an issue too as they will often end up being submitted digitally anyway, making the paper copy redundant. Time is wasted re-entering data (in the case of students) or manually copying up data into the digital system (for staff registering new students).
    - Data duplication is also an issue with several paper forms requesting the same data (e.g., contact details). This could potentially lead to clashes within the main digital database due to any conflicting erroneous information submitted (e.g., students accidentally completing two copies of the same form differently).
- Other additional information, such as equipment lists and expedition routes are spread across various different places from the official DofE website to a folder in the school shared drive. Staff must therefore direct students to different places each time for various task, adding to the confusion.

Specification

SUMMARY OF PURPOSE OF PROJECT

I aim to create an intuitive and reliable computerised system for managing the DofE scheme at Rougemont, reducing dependence on paper aspects of the current system. The system will utilise various levels of access (with a login system) and store data on students' respective progress through various stages of the award in a relational database. Staff will be provided an overview of stored student data as well as the ability to search through this data and inspect it more closely.

METHODS

Due to my familiarity with the language and thanks to the abundance of additional useful libraries it provides (even with the default installation), I will be using Python 3 to program my system. This will allow me to easily modularise different parts of my project into different files, reducing the line count of individual files and significantly improving my potential overview of the program as a whole. The libraries that I am likely to be using include tkinter (Python's Tcl/Tk GUI implementation), datetime (for handling dates and related time information) and ReportLab (for producing PDF printouts of program data).

OBJECTIVES

I use the SMART framework (Specific, Measurable, Achievable, Relevant and Time-bound) for my criteria below in order to ensure that the objectives I create are suitable for the task and provide a clear goal for my project.

The required/measurable performance of the system (in terms of speed, usability, etc.) is highlighted in **bold**.

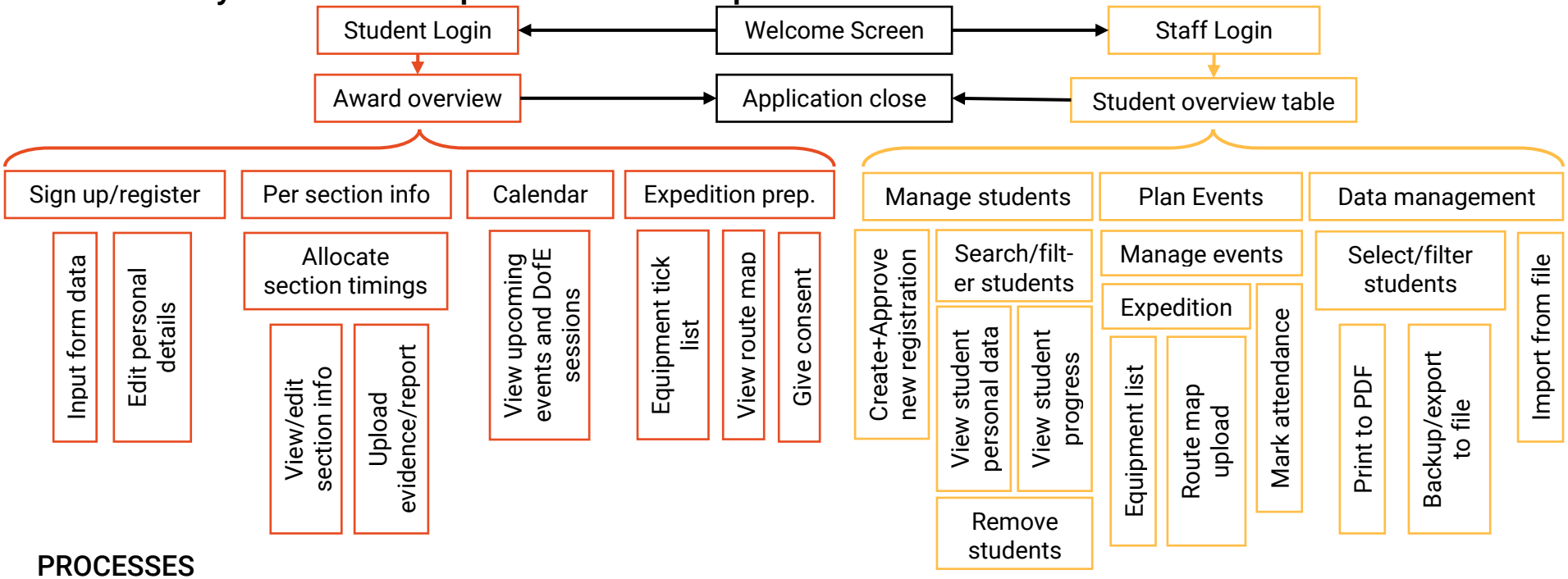
Success criteria

By the end of my project, I should have produced an application for the staff running the DofE scheme that:

- Has an intuitive interface with a clear layout – a new user (student or staff) should be able to find the information they need with **no guidance** other than from within the application itself
- Allows students to enrol in the DofE scheme
- Allows the same student to go on to further levels of the scheme after completing one
- Allows students (and authorised staff) to edit data after initial submission
- Enables students to plan the required sections of their award
- Allows students to submit an assessor's report once their section is complete and for staff to then moderate this submission
- **Only** shows a student's **own data** to them, not any others students' – to be accomplished via a login system and tiers of authorisation
- Allows staff to track **all** students' progress through their respective levels of the award and see what is in need of moderation
- Offers a facility to search and sort student records by name, progress, etc. (by at least **5 different variables**) – a search should not take longer than **3 seconds**
- Provides a facility to export PDFs of records for printing, moderation, physical backup, etc.
- Allows staff to plan expeditions to be planned in advance
- Enables information to be shared easily with students (defined as **less than 3 actions**)
- Provides the option to quickly (defined as less than **5 minutes**) make equipment lists and upload route information to the program
- Shows students relevant dates (defined as any events within the **next 7 days**) and information, including route maps and equipment lists, for any upcoming events
- Ensures all data is saved when the user closes the program and allows this data to be loaded when the program is started again (both in under **3 seconds**)
- Saves data in a format that can be easily exported for moderation and re-imported into a program on a different computer system

DESIGN

Overview of system – decomposition into sub-problems



PROCESSES

1. **Welcome Screen/Login**: Users choose login type then use unique username and password combination
2. **Award Overview**: Students see a home-screen with an award summary of outstanding tasks/documents and key dates
3. **Sign up/register**: Students register for scheme and input details
4. **Per section info**: Students view info for each section and add appropriate evidence after filling in all required details
5. **Calendar**: Students see upcoming dates/events (up to a week in advance)
6. **Expedition preparation**: After details are 'pushed' by staff, students prepare for an expedition
7. **Student overview table**: Staff are provided with a comprehensive overview of students' progress through the award
8. **Manage students**: Staff can create new logins for students and then approve the info they submit.
   The existing student database can be searched and viewed.
9. **Plan Events**: Staff fill in details about upcoming events/expedition for it to then be 'pushed' to students/groups of students (by level)
10. **Data management**: Stored student data can be exported as a file, for the program to load back into memory at a later date, or PDF
11. **Application close/exit**: The program's entire database/memory is saved to a file at this time to be restored on next start-up

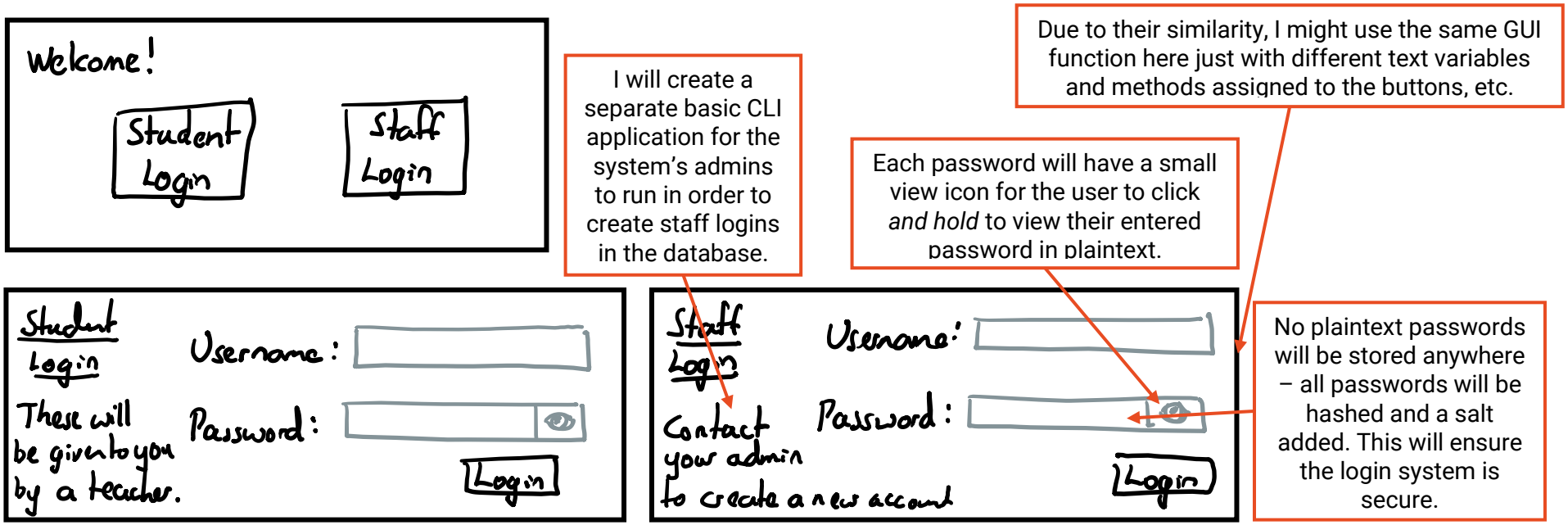
Breaking down the problem into subproblems above using my *Objectives*, has provided me with a much better overview of the project as a whole and will allow me to modularise its different components more easily. I will be able to focus on each named process (and its sub-processes) in turn and then combine them to create the final solution. This will also have the benefit of making bugs easier to locate and allow for more code reuse.

Input and Output

GENERAL USER INTERFACE

In implementing all the following interface designs, I will be using tkinter's ttk theme library to automatically style windows according to the OS.

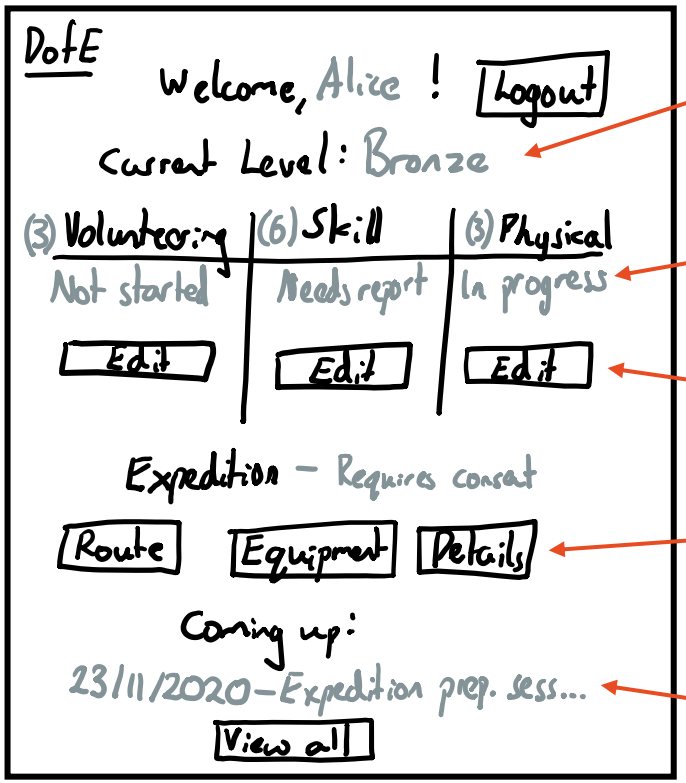
Welcome and Login



STUDENT INTERFACE

Award Overview

Linked from '*Login*' button in **student login** *Welcome and Login* window



This will only appear as a particular level if the student has filled in and completed the relevant sign up/enrolment details. Otherwise, there will be a 'Complete Enrolment button to click.

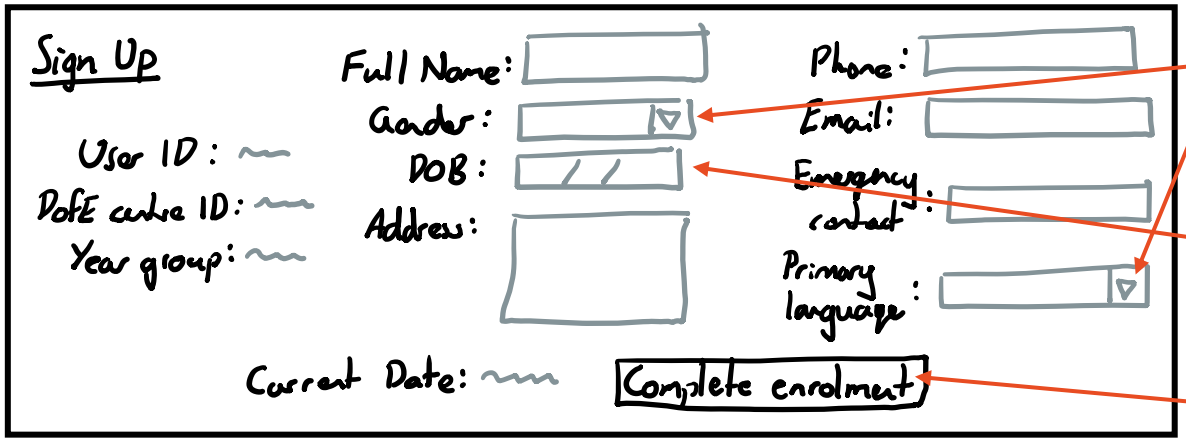
Sections can be "Not Started", "In Progress", "Needs Report" or "Completed"

Should the student not have completed the required details above, or has already started, these buttons will be greyed out.

These buttons relating to expedition preparations are only available if a teacher has 'pushed' an expedition to the student.

Students (who will have fewer events than staff in general) are shown their next 3 key dates in a listed view. They can choose to view more detail by clicking the button.

Sign up/Register
Linked from '*Complete Registration*' button in *Award Overview* window

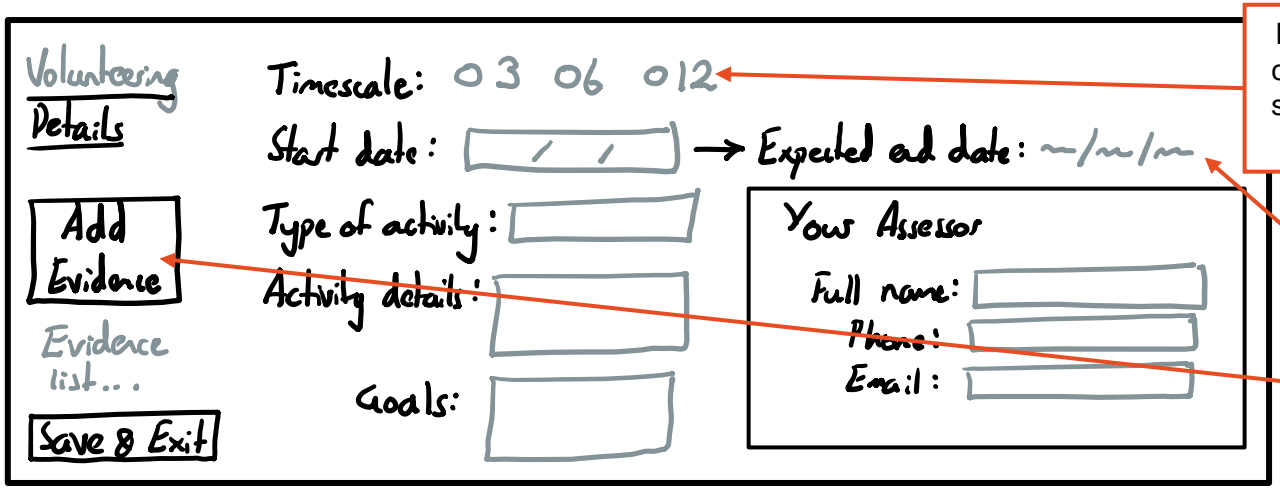


These dropdowns exist for fields which have a limited set of options. e.g., Gender: Male/Female/Other/Prefer not to say

This date entry field will be validated to ensure that an appropriate (exists and in the past) date is entered.

Submits completed form to teacher (after validation and verification) for final approval before the student can begin using the system fully.

Per Section Info
Linked from each section's '*Edit*' button in *Award Overview* window

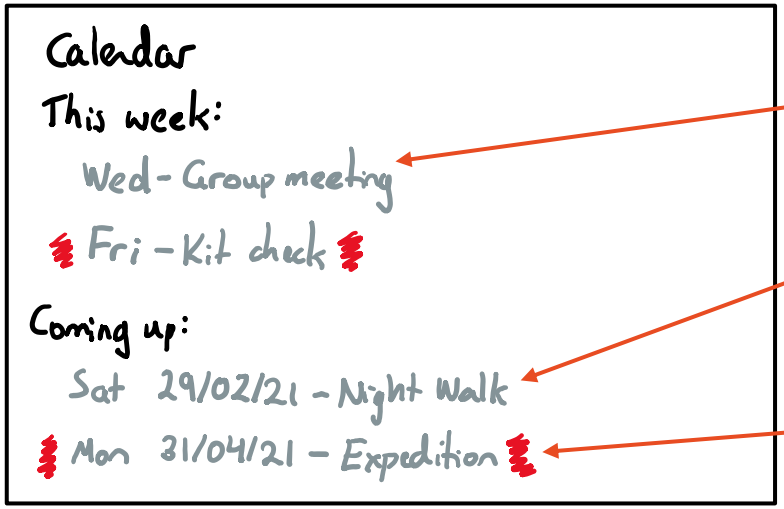


Depending on the level and the timescales chosen for the other sections of the award, some options here will be greyed out – see *Document 5 Timescales for each level*

Calculated from the start date (check if in future) & timescale selected above.

This button will open a file section dialogue box allowing the user to select files to then be saved in the application directory (an 'upload' essentially).

Calendar
Linked from '*View All*' button in *Award Overview* window

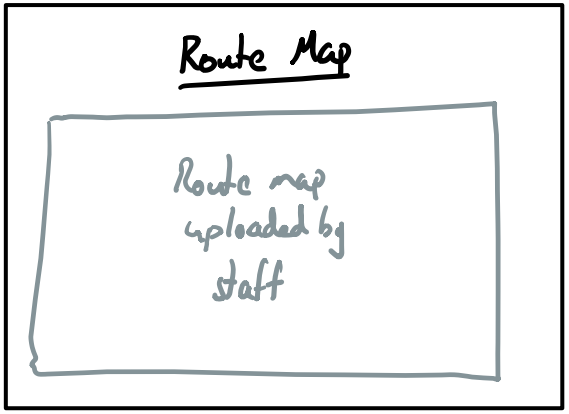


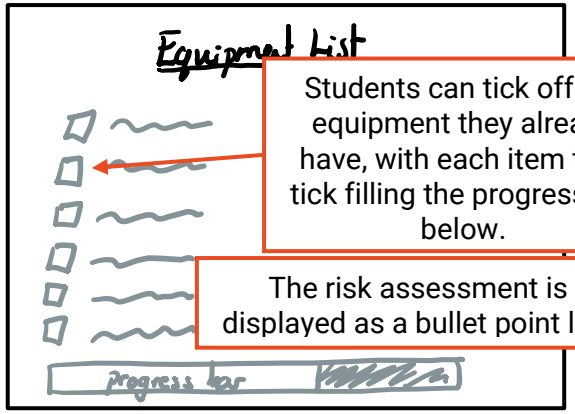
On hover of an event, a tooltip will show with 'Click to view more details'. Clicking will then open a separate message box with further information about the event that was provided by staff.

Students shouldn't have as many events as staff so a simple list view will suffice. If there are no events in the next week, the interface will simply say 'No events'. Each event consists of a weekday (generated from the date), a full date and the event's name.

Different categories of events will be displayed in different colours – key events will thus be highlighted to students.

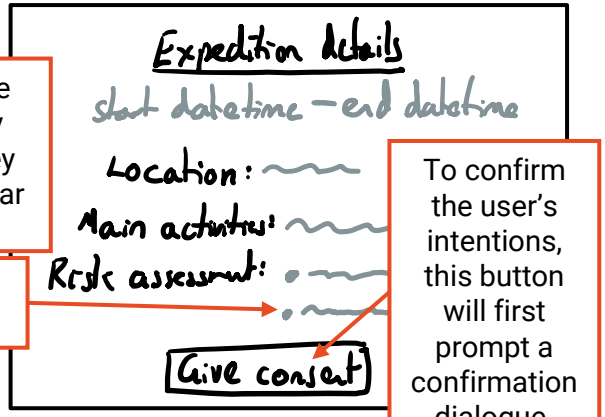
Expedition prep
Linked from '*Route*', '*Equipment*' and '*Details*' buttons in *Award Overview* window respectively





Students can tick off the equipment they already have, with each item they tick filling the progress bar below.

The risk assessment is displayed as a bullet point list.

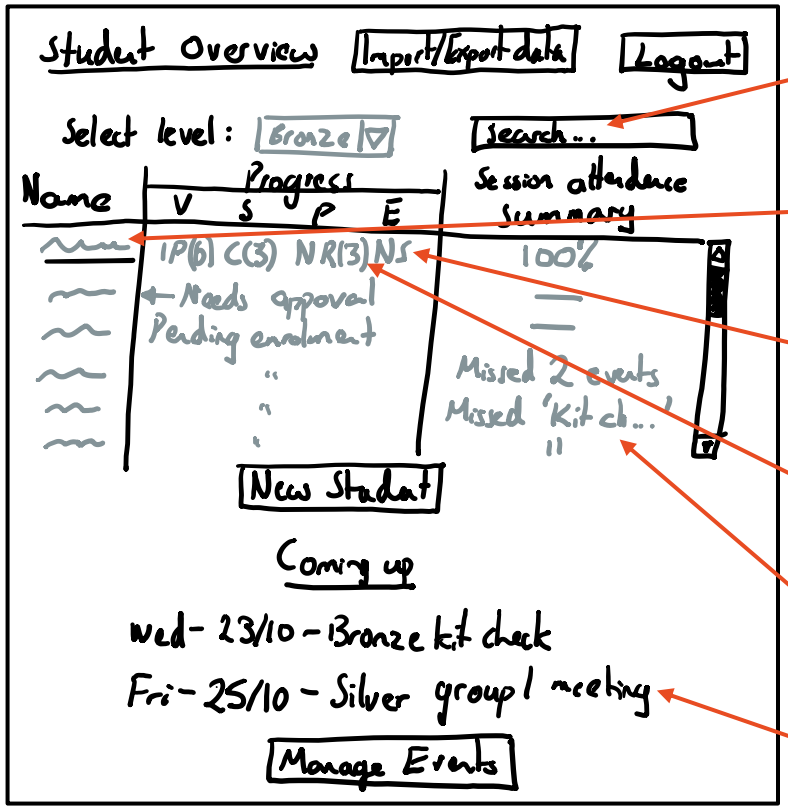


To confirm the user's intentions, this button will first prompt a confirmation dialogue.

STAFF INTERFACE

Student Overview Table

Linked from '*Login*' button in **staff login** *Welcome and Login* window



The search bar at the top of the table can be used to the filter the list of students displayed by name. Student's names will be checked for the substring entered. Staff always view students filtered by the level selected in the dropdown.

The user can double-click a student's name to approve their enrolment, see more/edit details or remove them from the database entirely.

This table uses these acronyms (for 'In Progress', 'Completed', 'Needs Report' and 'Not Started') in order to make it more compact – these could also be colour coded. The user can view the key for these acronyms at any time by hovering over them.

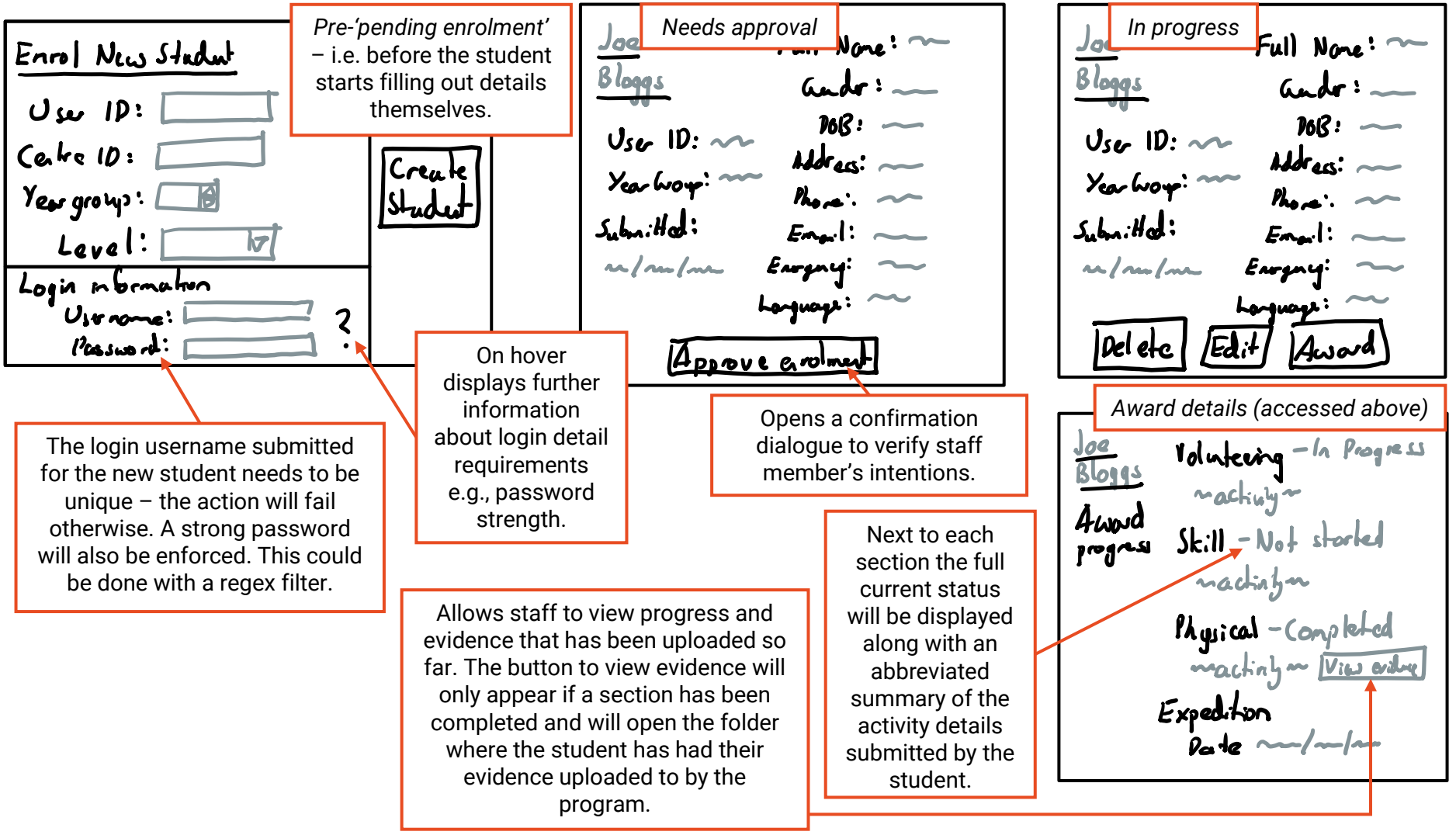
The table also displays the timescales the student has selected (where available) in brackets.

The attendance column will contain a number of 4 different types of messages: "100%", "N/A" (–), "Missed # events" (up to 3) and "Missed 'name of ev...' " (if only one missed in last 3).

Similar to student's calendars with the list view of events within the next week or so. More details are then available to view via the 'Manage Events' button.

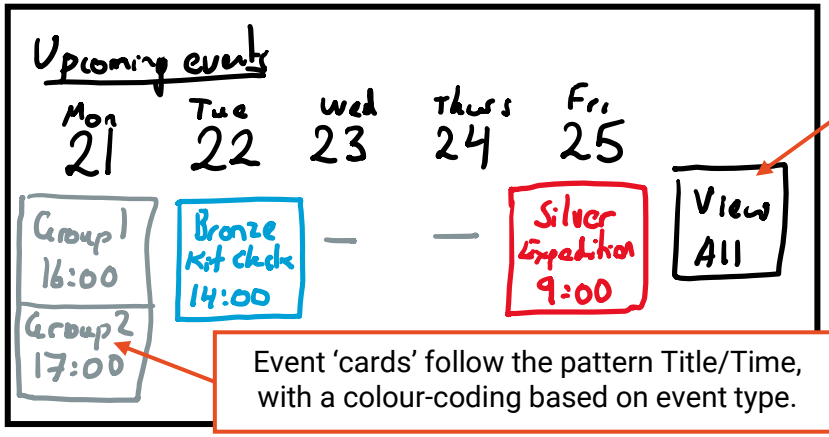
Manage Students

Accessed for each individual student by clicking their name in the *Student Overview Table* window (the window opened is dependent on status of student) or by clicking '*New Student*'



Manage Events and Expeditions

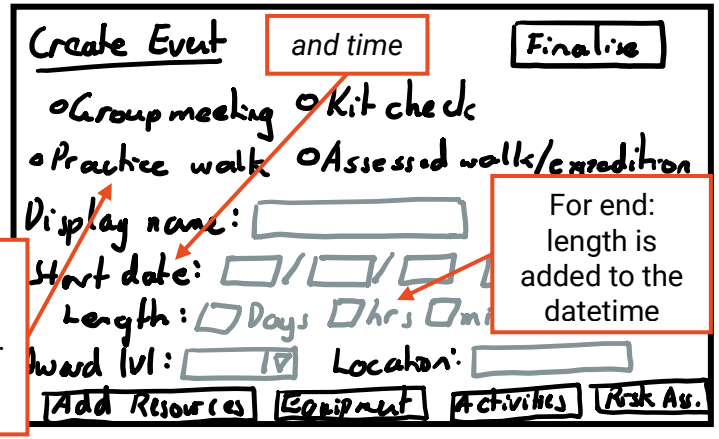
Linked from '*Manage events*' button in the *Student Overview Table* window



'View All' present a scrollable list view of all events – this is meant less as an overview and more for reference.

Event 'cards' follow the pattern Title/Time, with a colour-coding based on event type.

The lower fields are inactive until a category is selected. Determines colour, listing info, etc.



For end: length is added to the datetime

Data Management

Linked from '*Import/Export data*' button in the *Student Overview Table* window

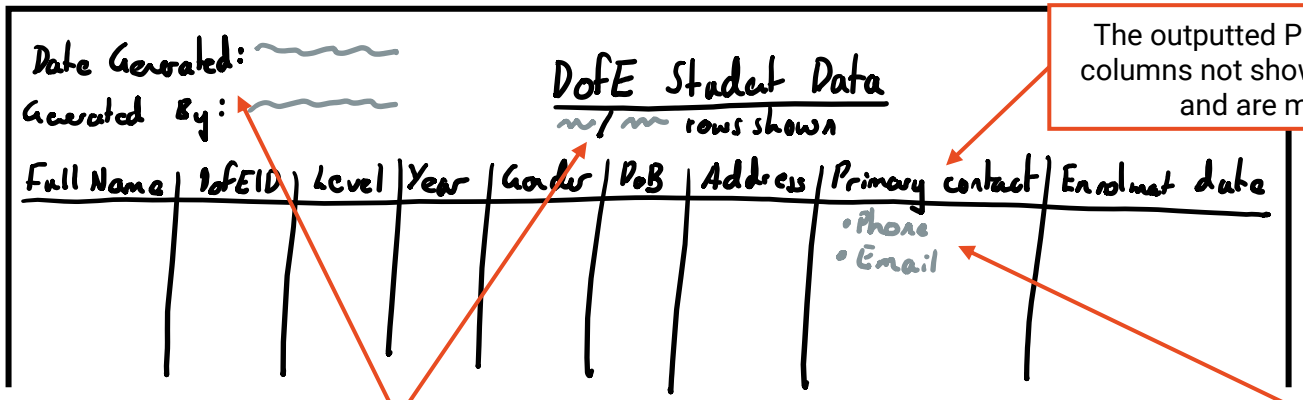
Button displays a warning before import file selection that this will overwrite all data stored – it will **not** merge data, just replace the current tables with the imported ones.

The export button creates a JSON file with **all** data from **all** tables currently stored within the application. These import and export buttons will **not** take into consideration selected checkboxes (due to the complex linking of tables) – a tooltip will inform the user of this.

The staff can select only certain students for PDF export by ticking each individual checkbox next to the students' names or select all students by using the checkbox at the head of the table.

PDF Export

An example of the report generated by using the '*Export as PDF*' functionality in the *Data Management* window (A4 landscape)



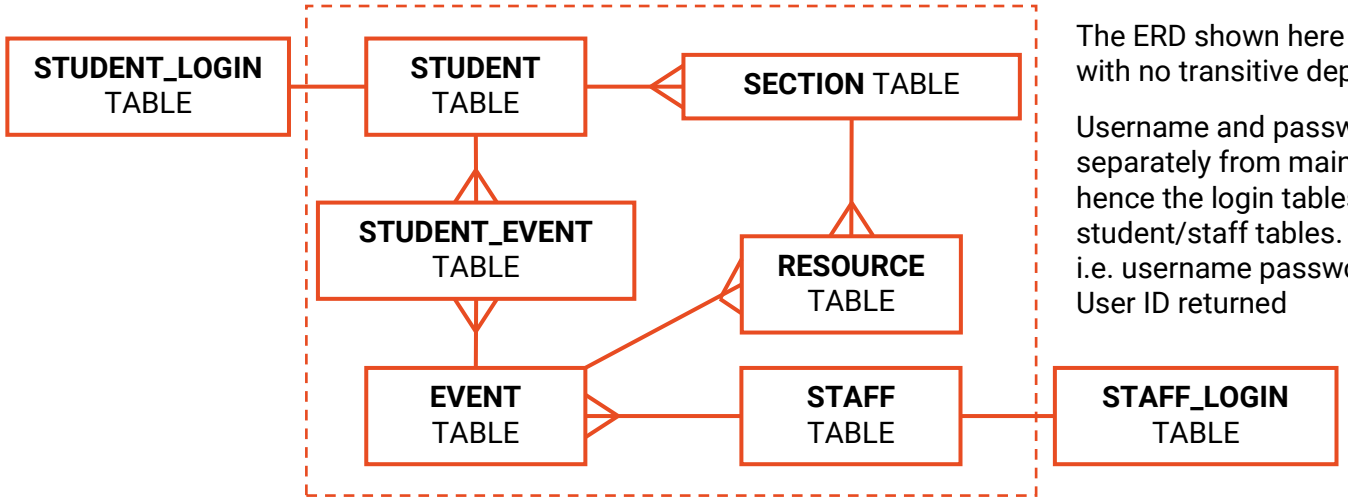
The outputted PDF export includes a few additional columns not shown in the Data Management window and are merely included for reference.

The report includes in its header a few useful bits of information to aid record keeping including: the date and time that the report was generated; the username of the staff member logged in when the data was exported; and the number of rows/students selected for export out of the total number available.

Each individual row/student has two 'sub-rows' in this 'Primary contact' column with each bullet pointed: the primary phone number; the primary email address.

Data structures

ENTITY RELATIONSHIP DIAGRAM (ERD)



The ERD shown here is already in third normal form with no transitive dependencies between any fields.

Username and password validation is carried out separately from main table access (dashed box) hence the login tables are separated from the main student/staff tables.
i.e. username password checked → corresponding User ID returned

DATA STURCTURE TABLES (INC. VALIDATION)

Any primary key fields are assumed to be unique (since they are almost always handled and generated by the program and not the user).
All fields include presence checks in their validation unless stated otherwise.

Student login table

Field Name	Key type	Description	Data type	Length	Example	Validation	Validation details
username	Primary Key	A user's username – unique to each student	string	2-30	lucahuelle20	length	username needs to be at least 2 characters long but no longer than 30
password_hash	-	A salt + hash of the user's password	string	192 (salt 64)	3953…e60e (shortened)	-	Generated within program
user_id	Foreign Key	The student_id of the username /password combination	integer	-	5	-	Generated within program linking to student table immediately

Student table

Field Name	Key type	Description	Data type	Length	Example	Validation	Validation details
student_id	Primary Key	Uniquely identifies a student	integer	-	5	-	Generated within program
fullname	-	Student's full name	string	2-30	Luca Huelle	length	A student's name should be a reasonable length (between 2 and 30 characters, including spaces)
centre_id	-	DofE centre ID of student	integer	-	68362	-	
year_group	-	Student's year group	integer	-	12	range	Must be between 7-13 (inclusive)
award_level	-	The level of award the student is taking part in	string	4-6	bronze	lookup	Must be one of 'bronze', 'silver' or 'gold'
gender	-	Gender that the student identifies as	string	4-6	other	lookup	Must be one of 'male', 'female', 'other', 'pnts' (prefer not to say)
date_of_birth	-	Student's date of birth	date	-	23/10/2002	range	Must be at least 10 years in the past but no more than 25 (age limit for DofE)
address	-	Student's home address	string	5-100	26 Dorallt Close, Henllys, NP45 3HW	-	
phone_primary	-	Student's main contact phone number	string (due to leading 0)	9-11	07428 124796	length	A valid personal UK phone number (including leading 0) is always between 9 and 11 digits.
email_primary	-	Student's main email	string	5-40	example@ domain.com	format	Validating an email rigorously is extremely complex. This program will

							perform only a basic check for the format ' ?? @ ?? . ?? '
phone_emergency	-	The student's preferred emergency phone contact	string (due to leading 0)	9-11	07431 184796	length	A valid personal UK phone number (including leading 0) is always between 9 and 11 digits
primary_lang	-	The student's primary language (for reference and to determine language of any correspondence)	string	5-7	english	lookup	Must be either 'english' or 'welsh'
enrolment_date	-	The date that the student was successfully enrolled in the DofE scheme	date	-	30/11/2020	-	Generated within program when record is initially created
skill_info_id	Foreign Key	student_section_id for the student's skill section	integer	-	5	-	Generated within program. Can be None if section not yet started
phys_info_id	Foreign Key	student_section_id for the student's physical section	integer	-	6	-	Generated within program. Can be None if section not yet started
vol_info_id	Foreign Key	student_section_id for the student's volunteering section	integer	-	7	-	Generated within program. Can be None if section not yet started

Section table

Field Name	Key type	Description	Data type	Length	Example	Validation	Validation details
student_section_id	Primary Key	Uniquely identifies each entry of section information	integer	-	5	-	Generated within program

section_type	-	Determines the section type	string	3-5	phys	lookup	Must be one of 'vol', 'phys' or 'skill'
activity_start_date	-	Specifies the date that the activity is planned to start	date	-	10/02/2021	range	Must be in the future
activity_length	-	Dictates how long the student plans to do the activity for	time	-	180 days	lookup	Must be one of 90, 180 or 360 days (3/6/12 months)
activity_type	-	A short name for activity	string	3-20	animal shelter	length	This should just be a short name to identify the activity by
activity_details	-	Extended text providing description of activity	string	10-200	I will volunteer at my local animal shelter and look...	length	The description should be between 10 and 200 characters in length
activity_goals	-	A brief text detailing the student's goals for the section	string	10-100	I will gain skills in working with small animals.	length	The goal paragraph should be between 10 and 100 characters in length
assessor_fullname	-	The student's selected assessor for this section's full name	string	2-30	John Smitherton	length	An assessor's name should be a reasonable length (between 2 and 30 characters, including spaces)
assessor_phone	-	The assessor's preferred phone contact	string (due to leading 0)	9-11	08431 184396	length	A valid personal UK phone number (including leading 0) is always between 9 and 11 digits
assessor_email	-	Assessor's main email contact	string	5-40	example. email@ domain.com	format	Validating an email rigorously is extremely complex. This program will perform only a basic check for the format ' ?? @ ?? . ?? '

Student event table

Field Name	Key type	Description	Data type	Length	Example	Validation	Validation details
student_event_id	Primary Key	Uniquely identifies each user-event pairing. Used for no other purpose	integer	-	4	-	Generated within program
student_id	Foreign Key	Identifies a student in the student table	integer	-	5	-	Generated within program
event_id	Foreign Key	Identifies an event in the event table	integer	-	12	-	Generated within program

Event table

Field Name	Key type	Description	Data type	Length	Example	Validation	Validation details
event_id	Primary Key	Uniquely identifies each event	integer	-	17	-	Generated within program
event_type	-	Refers to the type of event selected by the teacher	string	9-24	Group meeting	lookup	Must be one of 'Group meeting', 'Kit check', 'Practice walk' or 'Assessed walk-expedition'
event_name	-	The name of the event displayed to students and staff	string	3-50	Bronze practice kit check	length	This title should not be too long as it should summarise the event
event_start_date	-	The date *and time* of the start of an event	datetime	-	16:30 03/02/2021	range	Date must be in the future and date and time must both exist
event_length	-	Dictates how long the event is planned for and thus the end datetime	time	-	1 hr 30 min	-	Generated within program from controlled user input
event_award_level	-	Specifies which award level the event is for and, by	string	4-6	silver	lookup	Must be one of 'bronze', 'silver' or 'gold'

		extension, which students will take part					
event_location	-	The address/name of the location where the event will take place	string	3-50	Sport Classroom 1	-	
event_equipment	-	A *semi-colon separated* string of equipment students should bring	list/array (string)	-	backpack; hiking shoes; waterproof coat	-	The program will split the string by ';' to create an internal list for displaying as a tick-list, etc.
event_summary	-	A short description of the event and its main activities to summarise its purpose to students	string	10-200	We will go through the route for the expedition.	length	The summary should be relatively short since it will be displayed briefly alongside other information
event_risks	-	A *semi-colon separated* string of the potential risks	list/array (string)	-	trip hazard; slip hazard; hay fever	-	The program will split the string by ';' to create an internal list for displaying as a tick-list, etc.
staff_admin	Foreign Key	The staff_id of the staff member in charge of the event	integer	-	5	-	Generated by program based on the staff member logged in when event was created

Resource table

Field Name	Key type	Description	Data type	Length	Example	Validation	Validation details
resource_id	Primary Key	Uniquely identifies each resource in the table	integer	-	102	-	Generated within program
file_path	-	When the student/staff selects a file to	string	-	\uploads \students	-	Generated within program after the file has been 'uploaded'/copied to the

		'upload' the *relative* path to that file is saved here			\userid \ass_report.txt		application directory automatically (so that it can't be deleted elsewhere and lost)
date_uploaded	-	Details the datetime when the file was selected for 'upload' into the system	datetime	-	14:58 30/12/2020	-	Generated within program. The datetime the file is selected *not* the file's own created datetime
section_report	-	Allows the student to mark whether the file uploaded is an assessor's report for a section	boolean	-	True	-	Automatically marked as False for any event-related uploads. Student given the option for section evidence uploads
resource_type	Foreign Key	Dictates whether the resource is for an event or used as evidence for an award section	string	5-16	event	lookup	Automatically set within program depending on what is being created. Must be either event or section_evidence
parent_id		Either an event_id or student_section_id depending on resource_type field	integer	-	17	-	Automatically linked to correct table row within program

Staff login table

Field Name	Key type	Description	Data type	Length	Example	Validation	Validation details
username	Primary Key	A user's username – unique to each staff member	string	2-30	mcmahonp18	length	username needs to be at least 2 characters long but no longer than 30
password_hash	-	A salt + hash of the user's password	string	192 (salt 64)	4063…f60e (shortened)	-	Generated within program
staff_id	Foreign Key	The id of staff member of the username	integer	-	5	-	Generated within program linking to staff table immediately

		/password combination					

Staff table

Field Name	Key type	Description	Data type	Length	Example	Validation	Validation details
staff_id	Primary Key	Uniquely identifies a member of staff	integer	-	5	-	Generated within program
fullname	-	Staff member's full name	string	2-30	Joe Jefferson	length	A name should be a reasonable length (between 2 and 30 characters, including spaces)
centre_id	-	DofE centre ID of staff member	integer	-	68362	-	
gender	-	Gender that the student identifies as	string	4-6	male	lookup	Must be one of 'male', 'female', 'other', 'pnts' (prefer not to say)
date_of_birth	-	Staff member's date of birth	date	-	16/11/1978	-	Must be a valid date (i.e., exists in the past)
address	-	Staff member's home address	string	5-100	12 Dorallt Way, Henllys, NP42 3HW	-	
phone_contact	-	Staff member's main contact phone number	string (due to leading 0)	9-11	07428 124796	length	A valid personal UK phone number (including leading 0) is always between 9 and 11 digits.
email_contact	-	Staff member's main email	string	5-40	example email3@ domain.com	format	Validating an email rigorously is extremely complex. This program will perform only a basic check for the format ' ?? @ ?? . ?? '

METHODS OF DATA ACCESS

Object-oriented data storage during execution

While the program is running, an object-oriented approach will be used in order to store data. For example, new students will be instantiated from a Student class within a StudentTable object.
Each class will have a 'tabulated output' method that returns a string representation of all data contained within an object, padded and with a separator string used. This can then be used to 'save' an object's current state and its data to a text file – one row per instance/student/etc. For example:

```
>>> student_A = Student(fullname='Joe Bloggs', ...)
>>> student_A.tabulate()
'1        \&sJoe Bloggs      ...'
```

\&s used as a value separator here

student_table.txt *after program is closed*

```
1           \&sJoe Bloggs...
...         ...         ...
```

Text-file data storage and user uploads

Data will be written to text files (from program memory) automatically when the program is closed and will be automatically read back into memory on start-up (i.e., raw data from text files will be converted back into individual objects within the program). Handling the data in this way, with objects while the program is running, avoids editing the database live (meaning any program errors/crashes will not be saved to/corrupt the main text file database). It will also allow for greater flexibility in Python by taking advantage of direct access (hash mappings/dictionaries) to access data immediately by a key (hashable) value, instead of having to read in file data each time a query is made and using slower search procedures.

The data-containing text files will be stored within the application directory. For files/'resources' that are 'uploaded' to the program as evidence, etc. (see *Resource table* above), these will be copied to the application directory under '/uploads/staff/user_id' (or '…student/user_id' for students), where user_id is replaced by the id of the student/staff member. This will help keep the directory organised as new files are added since all files will be kept associated with their original uploader and their unique user id.

Algorithms/Processes

Some of these algorithms utilise Python's standard libraries. Where this is the case, an `import` *statement is used at the beginning of the pseudocode.*

Some algorithms also access the tables/data currently in memory. They are referred to as 'databases' within variable declarations and accessed like dictionaries using key values/field names. Each object's data is then accessed with dot notation. Pseudocode example:

```
student_table = {1: student_1, 2: student_2, ...}
# where 1, 2, ... are the primary keys of student table
# where student_1, student_2, ... are instances of the Student class
# i.e. Student Objects

# to access information:
student_1_name = student_table[1].fullname
student_2_year = student_table[2].year_group
```

HASH USER PASSWORD

Given an entered password, the algorithm will hash this, while also adding a 'salt' of random data, for it to be stored in the database. Thereby, no plaintext passwords are stored.
This code is adapted from *https://www.vitoshacademy.com/hashing-passwords-in-Python/*

```
import hashlib, binascii, os

password_str is string
salt is string
pwdhash is string

def hash_password(password_str):
    salt = get_random_bytes_from_os(n=60)
    hash salt
    convert salt to hexadecimal str
    encode salt as ascii

    encode password_str as utf-8
    pwdhash = hash password_str with salt using sha512 and 100000 iterations

    convert pwdhash to hexadecimal str

    return (salt + pwdhash) decoded as ascii
enddef
```

VALIDATE/AUTHENTICATE LOGIN

Given a username, verifies the password provided against the database or, if that username doesn't exist, raises an error. If password successfully verified, returns the appropriate student/staff ID.
This code is adapted from *https://www.vitoshacademy.com/hashing-passwords-in-Python/*

```
import hashlib, binascii, os

username is string
password_str is string
user_table is database
salt is string
pwdhash is string

def verify_password(username, password_str):
    if username in user_table:
          # gets first 64 characters of stored hash
```

```
            salt = user_table[username].password_hash[:64]

            # gets all remaining character of stored hash
            stored_password = user_table[username].password_hash[64:]

            encode password_str as utf-8
            pwd_hash = hash password_str with salt
                        using sha512 and 100000 iterations

            convert pwdhash to hexadecimal str
            decode pwdhash as ascii

            if pwdhash == stored_password:
                  return match
                  # function execution halts here – output statement not reached
            endif
      endif
      output 'Username/Password combination not found or incorrect'
      return False
  enddef
```

GENERATE SECTION STATUS

Based on the section fields from the table, this algorithm returns a categorisation label for that section. Sections can be 'Not Started', 'In Progress', 'Needs Report' or 'Completed'.

```
  import datetime

  student_id is integer
  student_table is database
  section_name is string
  section_progress_table is database
  resource_table is database
  table_key is integer
  section_obj is object
  resources is list
  finish_date is datetime
  report_found is boolean

  def get_status(student_id, section_name):
      table_key = None
      # If the field is blank, these queries return None
      if section_name == 'skill':
            table_key = student_table[student_id].skill_info_id
      endif
      elif section_name == 'phys':
            table_key = student_table[student_id].phys_info_id
      endelif
      elif section_name == 'vol':
            table_key = student_table[student_id].vol_info_id
      endelif

      if table_key != None:
            section_obj = section_progress_table[table_key]
            finish_date = get_end_date(
                        section_obj.activity_start_date, t.activity_length)

            if finish_date > datetime.datetime.now():
                  resources = resource_table WHERE
                    resource_type == section_evidence AND parent_id == table_key
```

```
                report_found = False
                for r in resources:
                        if r.section_report == True:
                                report_found = True
                        endif
                endfor

                if report_found == False:
                        return 'Needs Report'
                endif
                else:
                        return 'Completed'
                endelse
        endif
        else:
                return 'In Progress'
        endelse
    endif
    else:
            return 'Not Started'
    endelse
enddef
```

SHORTEN TEXT

If text if longer than a specified amount, this function truncates the text and appends an ellipsis (…) – this is used across the GUI where there wouldn't be enough space to display the full text.

```
main_text is string
length is integer
truncated_str is string

def shorten_text(main_text, length):
    if len(main_text) > length:
            truncated_str = main_text[:length-3]
            truncated_str = truncated_str + '...'
            return truncated_str
    endif
    else:
            return main_text
    endelse
enddef
```

CALCULATE END DATE

Given a start date and a length of time in days, hours and minutes, this algorithm calculates the corresponding end date.

```
import datetime

start_date is datetime
time_length is datetime.timedelta  # time lengths will be stored as timedeltas

def get_end_date(start_date, time_length):
    return start_date + time_length
enddef
```

GET WEEKDAY ABBREVIATION AND DATE PARTS

From a datetime object, returns the first three letters of the weekday as well as the date and month as a dictionary. This data is used in various simplified listings throughout the GUI.

```
import datetime, calendar

date is datetime
weekday is string

def get_date_info(date):
    weekday = calendar.day_name[date.weekday()]
    return {'abr': weekday, 'day': date.day, 'mon': date.month)
enddef
```

GET UPCOMING EVENTS WITHIN TIMEFRAME

From a list event objects, returns those within a given timeframe from the start date.

```
import datetime

date is datetime
timestep is datetime.timedelta
end_date is datetime
event_table is database
events_in_range is list

def get_upcoming_events(date, timestep, events):
    end_date = get_end_date(date, timestep)
    for e in event_table:
          if date < e.event_start_date < end_date:
                events_in_range.append(e)
          endif
    endfor
    return events_in_range
enddef
```

ENFORCING PASSWORD STRENGTH

This algorithm will return True only if the given password meets all the requirements. Otherwise, False is returned.

```
import string  # includes several helpful variables/constants

password_str is string
lower_list is list
upper_list is list
int_list is list
char_count is integer

def check_password(password_str):
    if len(password_str) < 6:
          return False  # password must be at least 6 characters long
    endif

    lower_list = list(string.ascii_lowercase)  # list of lowercase letters
    upper_list = list(string.ascii_uppercase)  # list of uppercase letters
    int_list = list(string.digits)  # list of digits

    # takes each requirement list in turn. i.e. ???=lowercase, uppercase etc.
    for char_list in (lower_list, upper_list, int_list):
          char_count = 0
```

```
            for char in char_list:
                  if char in password_str:
                        char_count += 1
                  endif
            endfor
            if char_count == 0:
                  return False  # password must contain at least 1 ??? character
            endif
      endfor
      return True  # if the program has reached this far, the password is valid
  enddef
```

Altrernativerly, the following regex pattern could be used (with Python's `re` library) to achieve the same effect:

```
(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,}
# at least one digit, one lowercase letter,
# one uppercase letter and at least 6 characters long
```

WRITE ACTIVITY SUMMARY

This algorithm creates a short summary text to be used in the award details dialogue (see *Manage Students*) given a section object (with activity detail fields) and student information.

```
student_id is integer
section_obj is object
summary is string

def activity_summary(student_id, section_obj):
    summary = ''
    if get_status(student_id, section_obj.section_type) != 'Not Started':
          # .upper() makes string upercase to make title stands out
          summary += shorten_text(section_obj.activity_type, 10).upper()
          summary += ':\n'  # \n is a newline character
          summary += shorten_text(section_obj.activity_details, 20)
          summary += '\nAssessed by: '
          summary += section_obj.assessor_fullname
          return summary
    endif
    else:
          return 'No summary available yet.'
    endelse
enddef
```

TABULATE OUTPUT

This **method** will be possessed by each object and will return a standardised input with each attribute value padded appropriately ready to be inserted into a table/text file. The example below uses the student login table but the same process will be applied to every class/table.

```
student_login_table is database
self is a student_login object contained within student_login_table
return_string is string

def tabulate(self):
    return_string = ''

    # 30 is length limit of username field
    return_string += self.username.ljust(30)  # left padded to 30 characters
    # separator inserted between entries
    return_string += r'\%s'  # a raw string to avoid having to escape slash
    return_string += self.password_hash.ljust(192)
```

```
        return_string += r'\%s'
        return_string += self.user_id.ljust(4)
        return return_string
    enddef
```

PARSE TABULATED INPUT

This algorithm takes a line produced from the method above (likely read from a text file) and extracts the actual elements of data to be parsed as an object. The example here is generic (hence '???') and will need to be replicated for each object in the program.

```
???_table is database
ObjectAdd is the initialisation method of the ???_table class
input_string is string  # read from a table txt file – one row of table
padded_input is list
raw_input is list

def add_new_table_row(input_string, ???_table, ObjectAdd):
    padded_input = input_string.split(r'\%s')  # splits into individual fields
    for f in padded_input:
        raw_input.append(f.strip())  # removes whitespace from each field
    endfor

    # Class initialisation method – ObjectAdd(...)
    # passed list of paramaters – raw_input
    # then object is added to table/database in memory – ???_table.add(...)
    ???_table.add(ObjectAdd(*raw_input))
enddef
```

WRITE DATA TO FILE

This algorithm takes all the tables and objects currently in memory and writes them each into tables (organised by class) in their own file for permanent storage when the application is closed.

```
from pathlib import Path

student_login_table is database
student_table is database
section_progress_table is database
resource_table is database
student_event_table is database
event_table is database
staff_table is database
staff_login_table is database  # i.e. all tables in memory
table_list is list
save_file_path is string

def save_db_state():
    table_list = [student_login_table, student_table, section_progress_table,
                  resource_table, student_event_table, event_table,
                  staff_table, staff_login_table]
    for table in table_list:
        save_file = Path('data_tables/')
        save_file /= str(table) + '.txt'
        with save_file.open(mode='w+') as fobj:  # creates the file
            for row in table:
                fobj.writeline(row.tabulate())
            endfor
```

```
                fobj.close()
            endwith
        endfor
    enddef
```

READ DATA FROM FILE

This algorithm goes through all the database text files and then inputs that data into the object structure used during the actual running of the program.

```
from pathlib import Path

student_login_table is database
student_table is database
section_progress_table is database
resource_table is database
student_event_table is database
event_table is database
staff_table is database
staff_login_table is database
table_list is list
save_file_path is string

def load_db_state():
    table_list = [student_login_table, student_table, section_progress_table,
                  resource_table, student_event_table, event_table,
                  staff_table, staff_login_table]
    for table in table_list:
         save_file = Path('data_tables/')
         save_file /= str(table) + '.txt'

         # if a file is missing then none loaded since tables are linked
         if not save_file.exists():
             raise FileNotFoundError
         endif
    endfor

    for table in table_list:
         save_file = Path('data_tables/')
         save_file /= str(table) + '.txt'

         with save_file.open(mode='r') as fobj:
             for line in fobj:
                 # .strip() removes trailing newline character
                 add_new_table_row(line.strip(), table, table.ObjectAdd)
             endfor
             fobj.close()
         endwith
    endfor
enddef
```

EVALUATION

Evaluating the effectiveness of my chosen programming language

Having chosen Python 3 due to my familiarity with the language, it ended up being a very suitable choice for the project. Python's clear implementation of classes and tight integration within my IDE (PyCharm) meant that I was able to write clear and well-structured code. I managed to accomplish this thanks to my decision to modularise related functions (using Python's `__init__.py` code functionality) and by collecting all my data structures together as subclasses of Rows/Tables. Typing function arguments and attributes also helped enormously during development, as my IDE was immediately able to flag any errors I had made with data types in my code.

While the tkinter module was annoyingly verbose at times, it was very powerful and allowed me to create a complex GUI with relative ease by including useful widgets like notebooks. There was also a wealth of documentation and support available online (including TkDocs and StackOverflow) which helped me immensely. In particular when I was structuring my code (e.g., using layered tkinter frames to create 'page' objects) and when I was considering implementations (e.g., restricting Entry fields to integers/digits – DigitEntry).

Python's default argparse module for creating command line interface also proved to be very helpful as it removed the need to create another fully-fledged interface for system administrators, reducing the scale of the project.

TOOLS AND TECHNIQUES USED

As mentioned above, I made heavy use of both Python's OOP capabilities and the default libraries in order to reduce my workload. In line with maintaining good development practices, I also used Python's unittest framework and followed the PEP8 style guides as much as possible. This helped make my code more robust and well-formatted. I made frequent use of my IDE's refactoring and debugging facilities (including breakpoints and variable watches) to ease my workflow when working with so many lines of code across many files. Use of the logging module also helped since I was able to easily track the stages that my program was going through (loading tables, GUI creation, etc.). My IDE also helped me when creating a virtual environment to isolate the project on my computer and prevent it interfering with other Python installations.

I used Git Version Control Software (VCS) to easily keep track of changes throughout my entire codebase, allowing me to easily rollback any edits that were causing issues and enabling me to work on a separate branch to 'master' to help categorise changes further in addition to commits. I used this in combination with my IDE's TODO functionality to help me keep track of which parts of the code still needed to be worked on/completed.

All these features helped significantly in organising the project as a single developer while also providing strong supporting evidence of my process in tackling the software development task in addition to this report. That is to say, the commit history and merged pull requests of the repository for instance can be used as a 'timeline' of key parts of the project.

PERSONAL STRENGTHS AND WEAKNESSES

I think I have written code that is very well documented and structured allowing for easy inspection and maintenance by a competent third party. I used self-documenting identifiers, included function doc-strings and typed the majority of my functions/methods to help with debugging. The README.md file provides additional detail on the use of the program

alongside inline comments in key sections of code (which I myself used during development to understand the structure of code that I had perhaps written some time ago). I think I also managed to write highly Pythonic code that utilises programmatic idioms and takes advantage of the Python language (e.g., list comprehensions). Not only does this make my code more readable but it also reduces unnecessary clutter and simplifies the code base.

Perhaps one of my greatest weakness during the project however, was spending too much time trying to perfect minute aspects such as the position of buttons within the GUI. This meant that I was not able to implement all the functionality I had originally planned including major features such as the calendar and expedition planning, due to time constraints. While still very feature-dense, the project is technically therefore still in an unfinished state.

Another potential weakness during the project was my use of tkinter as my GUI framework. While feature-rich and relatively straightforward, it perhaps wasn't the best choice for the sort of multi-page application I wanted to design. The frame-based approach I ended up using can become unresponsive on some machines as all frames are present at any one time, even if not visible. The dynamic updating of labels and fields when some pages are first navigated to also means that the window's size can change unexpectedly with widgets shrinking/expanding to fit their new contents. This means that the GUI, while functional, can be frustrating and confusing at times to actually use. However, I still believe this was better than spawning additional windows for each separate operation as this can quickly clutter a user's desktop. This way at least, the program's actions are confined to one window which is easily managed.

Identification of success criteria met

As discussed above, several of the original *objectives and success criteria* remain unmet since the program isn't fully complete. However, the following objectives **were** met:

The final program…

- has an intuitive and straightforward GUI (for students and staff)
- allows students to enrol in the DofE scheme with systems to ensure valid and appropriate data entry (approved by staff)
- enables students to plan different sections of the award
- provides students with the facility to upload evidence to completed sections including an assessor's report
- uses a login system to restrict students' view of other users' potentially sensitive data and restricts student-wide views to staff users
- provides an interface for staff to track the progress through the award of all enrolled students at all award levels
- offers functionality to search students by (user)name
- saves and loads data from permanent storage files in order for data to persist between runnings of the program
- saves this data in text files that can be easily shared between systems and imported into the program on another system

COMPARISON TO EXISTING SYSTEM

There are several aspects by which to compare my solution and the existing system which will serve as a basis for further evaluation. Regrettably, since I was unable to obtain any screenshots of the teacher/staff eDofE interface, I will not be able to compare these elements of my solution. In a future project, this is definitely an area where I would seek to collect more evidence.

Initially, it is possible to compare the dashboards presented to students – see *Screenshot 1 eDofE – student home screen* and *Screenshot 9 My solution – student dashboard*. Comparing

the information presented, there are several similarities – most notably including displaying the current status of each section. My solution presents several additional pieces of information including clearly identifying current level and the selected timescale of each section. Meanwhile, the eDofE system instead includes a messages menu, news about the award and options to edit your own information along with additional menu options in the sidebar. The first of these was outside the scope of my system while the second was not really possible since I would not have access to the same news feed. However, in a future system I could very much have implemented a process to allow for students to edit their own details. Theoretically, I am only missing the UI for this operation – the OOP design of my solution already allows attributes to be changed easily. Additionally, the eDofE system is aesthetically superior as it displays lots of its information with images and diagrams, making it more visually appealing than my purely functional solution.

The other major interface element to compare is that of section information – see *Screenshot 2 eDofE – section overview*, *Screenshot 3 eDofE – completed section information* and *Screenshot 10 My solution – section details*. The eDofE system splits this across multiple pages with a student having to navigate to the section overview page first from the dashboard and then into each separate section. In my system, section information can be directly navigated to from the dashboard, streamlining the experience. However, the existing system does provide greater flexibility in terms of being able to edit the activity information – although, as above, this again would be relatively easy to implement. It also enables staff to comment directly on a section – related to the message system referred to above. One thing that I believe my solution accomplishes better though is the organisation of information. In my UI, the assessor's information is clearly grouped instead of simply being appended to the end of the information listed. Critical information (e.g., the explanation as to why a section cannot yet be submitted) is also only shown to the user when needed (via a message box or tooltip) and therefore does not clutter the UI like in the eDofE system.

Finally, while it is not possible to compare the sign-up processes exactly since I was unable to collect screenshots of eDofE sign-up and the associated staff interface, I can examine my own process of student creation and login– see *Screenshot 13 My solution – student creation*, *Screenshot 7 My solution – student login* and *Screenshot 8 My solution – student enrolment*. Staff must first create a student with basic information (e.g., centre number, year group) and login details. The student themselves then has to log in using these provided details before filling out their own personal details (e.g., full name, date of birth, etc.). Before the student can start the award, a staff member must then approve these details. Once this is complete, the student is free to start all sections of the award via their UI. While this system of sign-up works and allows staff to approve student details, there is no facility to reset passwords or for staff to vet section information (as there are in eDofE). The second of these could be added relatively easily by building on the existing approval process for initial student details. A password recovery/reset facility would also definitely be added in a future design. It could be done through an additional command line argument, therefore only being available to system admins.

However, at this point I should also consider the security of the application. In my case, security relies on the operational files of my program being 'locked-down' by the operating system and device administrator; i.e., commands (such as `Python main.py -s`) can only be executed in this directory, and database files can only be edited by those authorised to do so. There is **no** additional security in my application preventing students from using the command line tools so this must be considered by staff instead. eDofE meanwhile is web-based meaning no data is stored locally. Without a staff login, there no physical way a student can even view sensitive data since the student portal is entirely separate.

Potential future changes in approach and improvements to avoid future problems

There are two main issues with my application that I believe could be addressed by a change in approach during any future development.

The first of these is of course the incomplete nature of the project with many features still left unimplemented. One potential way to address this could be a more agile approach to development by prototyping all parts of the application first and then carrying out timed 'sprints' to develop certain features in more finalised detail. This lack of concrete/timed targets during development was, along with the pandemic, a major factor in the project's incomplete final state: I simply spent too much time trying to needlessly perfect minutia instead of adding the required functionality. I should also have prioritised the *key aims from my initial discussion* in order to meet as many stakeholder requirements as possible. However, this too would have been difficult due to the large number of *interlinking* systems I would have needed to create.

The visually unattractive and at times laggy interface is the second major issue with my current solution. If I were to develop the application again, I might have used PyQt instead of tkinter. This would have enabled me to use *Qt Designer* to speed up the GUI creation process while also allowing me to split up the UI and functionality elements of code into separate files, providing greater flexibility and more readable code. Currently, my functional and GUI code is combined together, making it harder to initially understand which code is responsible for what.

ADDITIONAL FUTURE FUNCTIONALITY AND FEATURES

I have already discussed several potential improvements in the sections above. Those ideas and some additional ones are presented below:

- Messaging/comment functionality to allow staff to directly feed back to students on submitted information.
- Ability to allow students to edit personal and selected section information after submission
- Enable staff to vet/approve submitted section information before the student starts working on it.
- A password reset/recovery facility along with an option to set a security question
- Allow student accounts to increase in level in order for one account to cover multiple stages of the award – i.e., ensure only one login and student object is needed to progress all the way from Bronze to Gold.
- Greater emphasis on visual elements to help display information (such a section progress).
- A new UI for the assessors (nominated by students) who can log in to upload section assessor reports.
- A new UI for DofE moderators in order to approve the final submitted information from each student and view the progress across a whole centre's candidates
- A system for handling students that join part way through the overall scheme – e.g., starting directly from Silver (without doing Bronze), a student must complete an additional six months in any section.
- Greater investigation of compatibility across operating systems – particularly GUI elements and file handling/copying. Currently, the program has only been tested on Windows 10.

Features that were in the original design of the system but left unimplemented in the current version of the program would form a priority in any future development. These include:

- Functionality for running/planning expeditions. This includes features to share equipment lists as tick lists for students to complete and sharing route map images to allow students to plan the expedition.
- Calendar functionality to keep track of events for students and pupils. Calendars would be added to by staff and viewable by students in order to plan different sections of the award.
    - Staff would also be able to keep a register of pupil's attendance and mark whether a pupil attended a required session or not. The student overview table would then display an attendance column summarising students' attendance record over the past few events.
- Ability to view all student data in overview or to export this data to a PDF form for review/permanent storage.
    - The ability to sort students in the student overview table by their name/section status simply by clicking on the section headings.
    - Functionality to search by different attributes of students (not just their user/name) and perhaps refine searches by category (e.g., only search the students who have fully completed their award).
    - Buttons to import data from text files while the program is actually running. Such a data import function could be done on either a merge or a replace basis – i.e., combine new data with that already in memory or use only the new data.
    - Buttons to actually delete student accounts (and all associated files as well as all evidence, sections and login objects related to the student).
- In terms of the source-code itself, unit-testing scripts to test all elements of the program would be included in a more fully developed project. These would test database methods and potentially simulate the user interacting with the GUI.

APPENDIX A – DOCUMENTS

Document 1 Questionnaire

DofE Scheme Management - Current system

Forming the basis for the Investigation section of my GCE Computer Science Unit 5 NEA, this questionnaire exists for me to gather some basic information about the current system used for planning and organising DofE expeditions.

* Required

1. How large a role does the current eDofE system play in managing the award scheme? *

   Mark only one oval.

	1	2	3	4	5	
It's not really used	◯	◯	◯	◯	◯	It's vital to keeping the scheme running well

2. Other than eDofE, what (computerised) systems are to manage the DofE scheme used? *

   May be none.

3. How easy is the current system to use? *

   Mark only one oval.

	1	2	3	4	5	
Not easy at all - lots of training and experience required	◯	◯	◯	◯	◯	Very easy - no training required

4. What is the number one issue with the current system, if any? *

5. Are there any other particular 'bugbears' with the system?

6. Is it possible for you to upload any screenshots of what your current eDofE interface looks like?

   Obviously this is not possible if there is sensitive information visible. This question is only so I can potentially get an idea of what sort of tools are currently available to you and how they are presented.

   Files submitted:

7. To what extent are paper records still used? *

Paper records might include consent forms and sign up sheets.

Mark only one oval.

	1	2	3	4	5	
Not used at all	◯	◯	◯	◯	◯	Heavily used

8. If not 1, what is the main use of/reason for paper records?

_____

9. Who would need to use any potential comprehensive DofE system? *

Include any additional parties under 'Other'

Check all that apply.

- ☐ Students
- ☐ Teachers
- ☐ Reception/Admin
- ☐ External DofE moderators

Other: ☐ _____

Student data and processing

10. How is data currently entered into the system? *

Mark only one oval.

- ◯ By students themselves
- ◯ Manually transferred from paper records by teachers
- ◯ By scanning in documents
- ◯ Other: _____

11. What 3 pieces of student data would be most important to see at a glance/in overview? *

If I've forgotten a potential piece/s of information, enter it/them under 'Other'.

Check all that apply.

- [ ] Current award level (i.e. Bronze/Silver/Gold)
- [ ] Current progress per section (i.e. proposal/in progress/completed)
- [ ] Projected completion date
- [ ] Past training/teaching sessions attendance
- [ ] Date of next expedition/key event
- [ ] Address/Contact details
- [ ] Other: _____

12. How are you able to process data already in the system? *

Check all that apply.

- [ ] Edit student-entered data
- [ ] Send messages to students
- [ ] Send 'todo lists' to students
- [ ] Search student data/records
- [ ] Export records (as PDFs or to print) of student data
- [ ] Other: _____

13. Is there anything else you wish you could do in addition to the above?

It could be one of the options above that you didn't select above.

_____

Thank you for your time!

Document 2 Questionnaire response

DofE Scheme Management - Current system

Forming the basis for the Investigation section of my GCE Computer Science Unit 5 NEA, this questionnaire exists for me to gather some basic information about the current system used for planning and organising DofE expeditions.

How large a role does the current eDofE system play in managing the award scheme? *

	1	2	3	4	5	
It's not really used	○	○	○	○	◉	It's vital to keeping the scheme running well

Other than eDofE, what (computerised) systems are to manage the DofE scheme used? *

May be none.

none

How easy is the current system to use? *

	1	2	3	4	5	
Not easy at all - lots of training and experience required	○	○	◉	○	○	Very easy - no training required

What is the number one issue with the current system, if any? *

difficult to find information

Are there any other particular 'bugbears' with the system?

.....

Is it possible for you to upload any screenshots of what your current eDofE interface looks like?

Obviously this is not possible if there is sensitive information visible. This question is only so I can potentially get an idea of what sort of tools are currently available to you and how they are presented.

To what extent are paper records still used? *

Paper records might include consent forms and sign up sheets.

	1	2	3	4	5	
Not used at all	○	◉	○	○	○	Heavily used

If not 1, what is the main use of/reason for paper records?

ease of use for parents

Who would need to use any potential comprehensive DofE system? *

Include any additional parties under 'Other'

- [x] Students
- [x] Teachers
- [x] Reception/Admin
- [x] External DofE moderators
- [ ] Other: .....

Student data and processing

How is data currently entered into the system? *

- (●) By students themselves
- ( ) Manually transferred from paper records by teachers
- ( ) By scanning in documents
- ( ) Other: .....

What 3 pieces of student data would be most important to see at a glance/in overview? *

If I've forgotten a potential piece/s of information, enter it/them under 'Other'.

- [x] Current award level (i.e. Bronze/Silver/Gold)
- [x] Current progress per section (i.e. proposal/in progress/completed)
- [ ] Projected completion date
- [x] Past training/teaching sessions attendance
- [ ] Date of next expedition/key event
- [ ] Address/Contact details
- [ ] Other: .....

How are you able to process data already in the system? *

- [x] Edit student-entered data
- [x] Send messages to students
- [ ] Send 'todo lists' to students
- [ ] Search student data/records
- [x] Export records (as PDFs or to print) of student data
- [ ] Other: .....

Is there anything else you wish you could do in addition to the above?

It could be one of the options above that you didn't select above.

.....

Thank you for your time!

This form was created inside of Rougemont School.

Google Forms

Document 3 Enrolment Form

DofE Participant Enrolment Form

Please print clearly in CAPITALS or type your details in. You must complete all of the questions.

DofE Centre and group details (if you know them):

DofE Centre:	DofE group:

DofE level:

Bronze ☐ Silver ☐ Gold ☐
Have you registered for any previous levels of the DofE? No ☐ Yes ☐
If YES – please give the name of the DofE Centre you were registered at: *e*DofE ID number (if known) :

Personal details:

First name:	Last name:
Gender: Male ☐ Female ☐	Date of Birth: / /
Primary language English ☐ Welsh ☐ Other ☐	
Date you wish to start your DofE programme if known (enrolment date): / /	

When you first sign in to *e*DofE you will be asked to record some personal details such as your contact details, ethnicity and personal circumstances along with details of any medical needs you may have. This data is used to enable your Leaders to support you doing your DofE programme and for the DofE's statistical and reporting purposes. You will always have a 'prefer not to say' option.

Contact details:

Email address:	
Address (line1):	
Address (line 2):	
Town/City:	
County:	Postcode:
Telephone:	Mobile number:

Emergency contact details:

Emergency Contact name:	Relationship to you:
Emergency contact telephone number(s):	

P.T.O.

eDofE participant enrolment form 2012 Version 1

DofE Participant Enrolment Form

Declaration:
I agree to enrol as a participant on a DofE programme. I understand that I will be managing my programme using the online *e*DofE system. I acknowledge that this system has a set of terms and conditions that I agree to. These terms and conditions are available at www.eDofE.org

Print Name	Signature	Date
		/ /

Consent to enrol from parent or guardian (if applicant is under 18 years old).

I agree to my son / daughter / ward doing a DofE programme. I note that it is my responsibility to check that any activity my son / daughter / ward undertakes for their DofE programme is appropriately managed and insured, unless the activity is directly managed or organised by their DofE group, centre or Licensed Organisation.

Print Name	Signature	Date
		/ /

Note:
Data supplied on this form and in *e*DofE and information about DofE activities recorded in *e*DofE will be used by the DofE Charity, the Licensed Organisation and DofE centre to monitor and manage DofE participation and progress by young people and manage and support Leaders.

The DofE Charity will use personal data to communicate useful and relevant information to either help participants complete a DofE programme, Leaders/LOs to run DofE programmes more effectively or help the DofE Charity to improve the quality and breadth of its programmes.

Occasionally the DofE Charity may send you information relating to commercial offers. If you do not wish to receive commercial information from the DofE Charity you can choose not to by amending your contact preferences in your eDofE profile at any time.

For Licensed Organisation/Centre administration only:

Date registered onto *e*DofE	/ /
Expected start date	/ /
Participant Fee received	Yes ☐ No ☐
Username	
User ID number	

Document 4 Parental Consent Form

FORM 03 – Parental Consent (RESIDENTIAL)

Educational Visits and School Trips

Parental Consent Form

The purpose of this form is to obtain your consent for your child to take part in the proposed outdoor / off-site educational activity set out below. It is designed so that all information relating to your child's health and fitness can be assessed, prior to the trip, allowing for any necessary arrangements to be made.

This document may be completed electronically and returned via email or printed and completed by hand. Please click in the relevant boxes to add text or check the box to provide your answer. Once completed, please return by email to seniorschool@rsch.co.uk. or you may print and return the form to the School Reception in hard copy form.

SECTION A - DETAILS OF PROPOSED ACTIVITY *(to be completed by the Party Leader)*

Main travel destination: **"Brecon Beacons"**

Main activities included during the visit: **"Silver Assessed DofE Walk"**

Date of Trip/Visit: **"03 April 2020 – 05 April 2020"**

Any additional information: **"Three days walking with two nights camping"**

SECTION B – PUPIL INFORMATION *(to be completed by the parent or guardian)*

Pupil's full name: Click here to enter text. Pupil's Tutor Group: Click here to enter text.

Date of Birth: Click here to enter a date.

Home address: Click here to enter text.

Home telephone number: Click here to enter text.

Emergency Contact Details

Please provide primary and secondary contact names and details for use in an emergency. Please note: The designated primary contact person must be contactable for the whole duration of the trip/visit.

Primary contact:

Name: Click here to enter text. Telephone number: Click here to enter text.

Secondary contact:

Name: Click here to enter text. Telephone number: Click here to enter text.

MEDICAL INFORMATION - *Please provide as much detail as possible or enter **NO** as required.*

Does your child suffer from any **allergies**? No ☐ Yes: Click here to enter text.

Does your child have any specific **medical** or **dietary** requirements? No ☐ Yes: Click here to enter text.

Does your child have any specific **visual** or **auditory** medical conditions? No ☐ Yes: Click here to enter text.

Is your child currently receiving treatment for any medical condition? No ☐ Yes: Click here to enter text.

Is your child required to take any prescribed medication? No ☐ Yes: Click here to enter text.

Has your child had a tetanus injection in the last five years? No ☐ Yes: Click here to enter text.

FORM 03 – Parental Consent (RESIDENTIAL)

Does your child suffer from motion/travel sickness? No ☐ Yes: Click here to enter text.

Does your child suffer from any phobias? No ☐ Yes: Click here to enter text.

In the past four weeks, has your child had…

- A physical injury? No ☐ Yes: Click here to enter text.
- An infectious disease? No ☐ Yes: Click here to enter text.
- Diarrhoea or vomiting? No ☐ Yes: Click here to enter text.

Name of family doctor: Click here to enter text.

Name of doctor's practice: Click here to enter text.

ADDITIONAL INFORMATION

Please indicate your child's swimming ability

Non swimmer ☐ Water confident ☐ can swim more than 50 meters ☐

Please provide any other information that you consider the School/Party Leader should know prior to departure. Click here to enter text.

SECTION C - ACKNOWLEDGEMENT OF RISK *(to be completed by the Party Leader)*

A risk assessment has been completed by the Party Leader to identify any additional risks beyond those encountered in a normal school day. These may include:

Access to a balcony☐	Farm machinery☐	Play areas☐
Altitude Sickness ☐	Fire (and smoke inhalation)☐	Road Traffic Accident☒
Animal Bite ☒	Flight induced DVT☐	Separation from party ☒
Assault☐	Food Poisoning☒	Sickness/Diarrhoea☒
Avalanche☐	Fracture/dislocation☒	Slip/trip or fall☒
Burns☒	Head injury☐	Snow blindness☐
Climbing and heights☒	Heat stroke☒	Sun Burn☒
Cuts/Bruising☒	Hyperthermia☒	Terrorist attack☐
Dehydration☒	Hypothermia☒	Travel sickness☐
Drowning and open water☐	Infection from animal☐	Vulnerable children☐
Electrocution☐	Members of the general public☒	
Exhaustion☒	Motion sickness☒	

Please be aware that the above risks may exist. However, every care will be taken to effectively minimise these and ensure the safety of all pupils.

CONDUCT DURING THE TRIP

All pupils will be made aware that the reputation of the School is paramount during offsite activities. This reputation can either be enhanced or damaged by their conduct. **All** school rules and policies therefore apply for the whole duration of any visit or trip.

All pupils are expected to behave in a responsible manner at all times during the educational activity. They must take direction from the tutor or nominated leader and follow all instructions or guidance given by activity instructors. When instructed, they must wear any clothing or protective equipment issued to them by instructors/tutors and not interfere with any of this. All local rules must be followed, e.g. those of the activity

FORM 03 – Parental Consent (RESIDENTIAL)

centre, country code, piste etiquette, etc. If the activity involves field work, then participants must remember not to harm flora and fauna, leave litter, damage footpaths, walls, hedges, etc.

SECTION E - CONSENT DECLARATION

I, being the parent / guardian of the child named at the head of this form, give consent for him / her to attend the proposed educational visit or trip. I give consent for them to receive emergency medical treatment, including anaesthetic, as considered necessary by any medical doctor present, should the need arise. I have informed the School and Party Leader of all medical conditions that my child suffers from or treatments required to maintain their health.

Signed by Parent or Guardian: Please type your name or sign here Date: Click here to enter a date.

Relationship to pupil: Click here to enter text.

ADDITIONAL INFORMATION

DATA PROTECTION

Rougemont School is a Data Controller for the purposes of the GDPR 2018. This Act regulates how we obtain, use and retain information about individuals. When you sign this form, you are consenting to Rougemont School holding your personal information for this purpose. This information is used only for the purposes for which it is given and is not passed on to a third party.

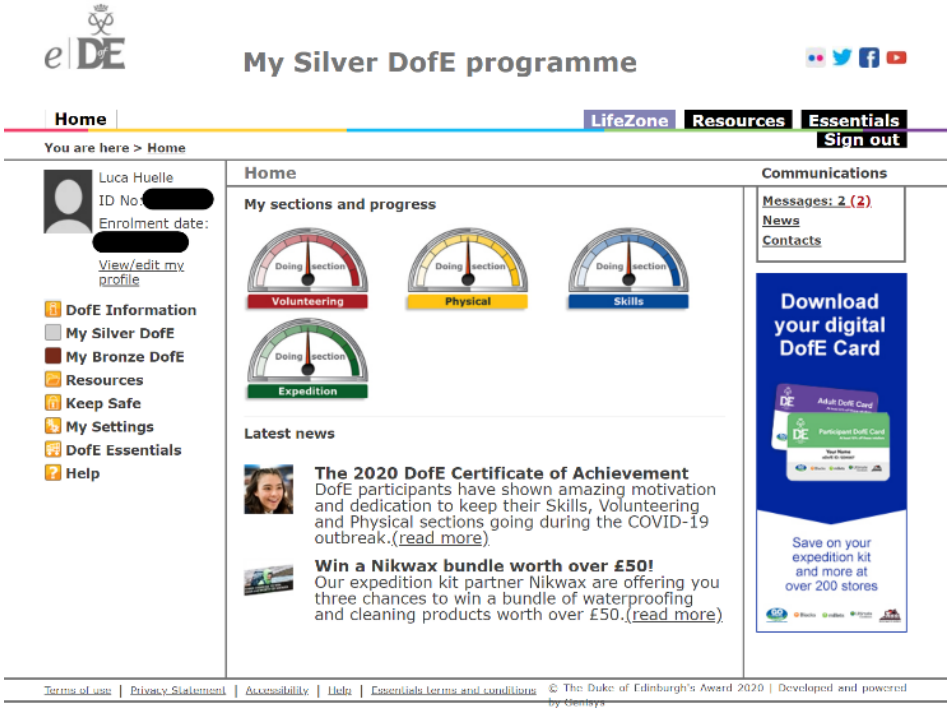
INSURANCE

Your child is insured by the school, with additional insurance cover provided by Endsleigh Insurance. Should you wish, for any reason, to increase the cover provided as standard, you may do so at your own cost.

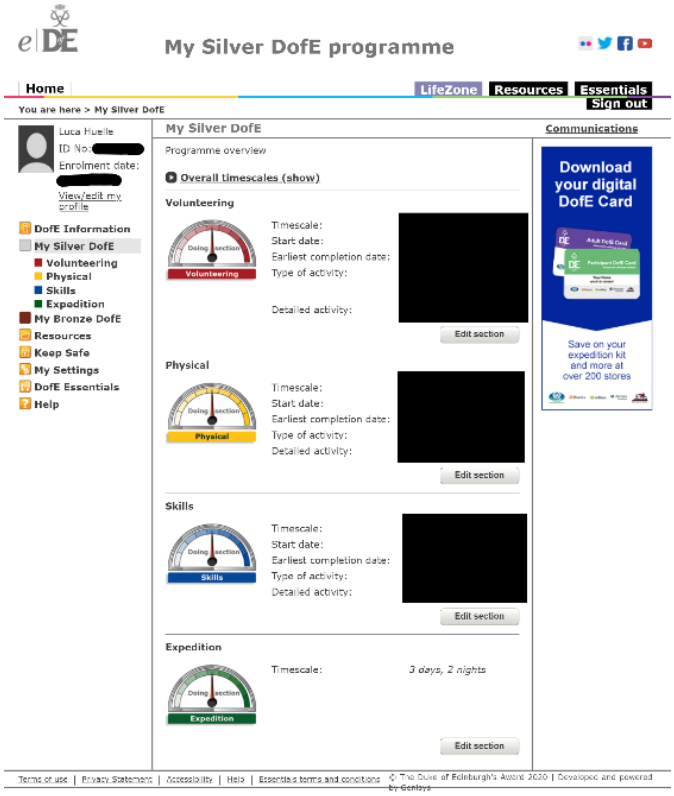
Document 5 Timescales for each level

APPENDIX B – SCREENSHOTS

Screenshot 1 eDofE – student home screen



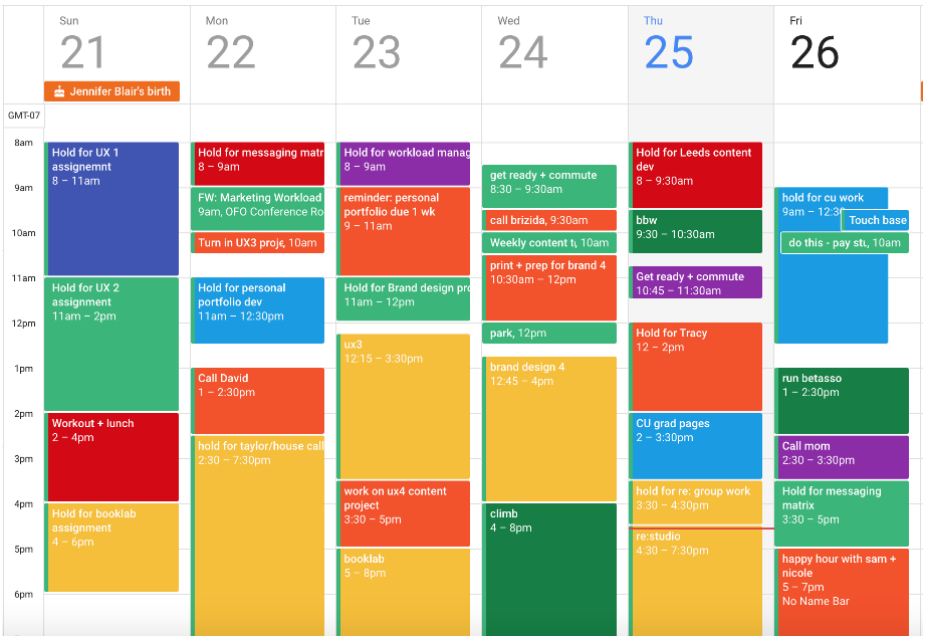
Screenshot 2 eDofE – section overview



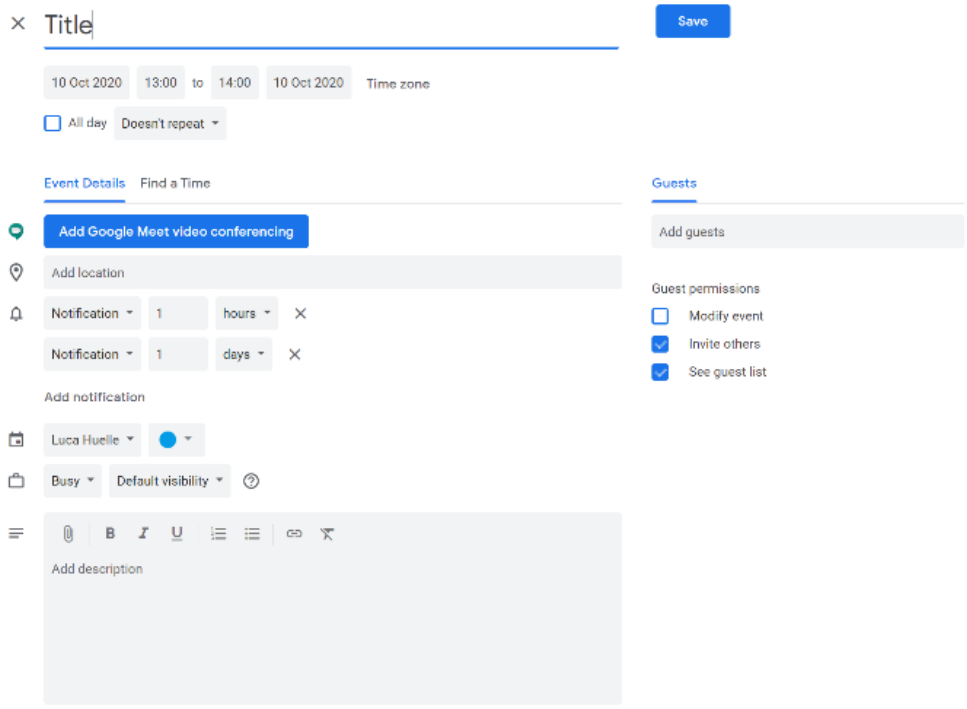
Screenshot 3 eDofE – completed section information

Screenshot 4 Google Calendar – week view

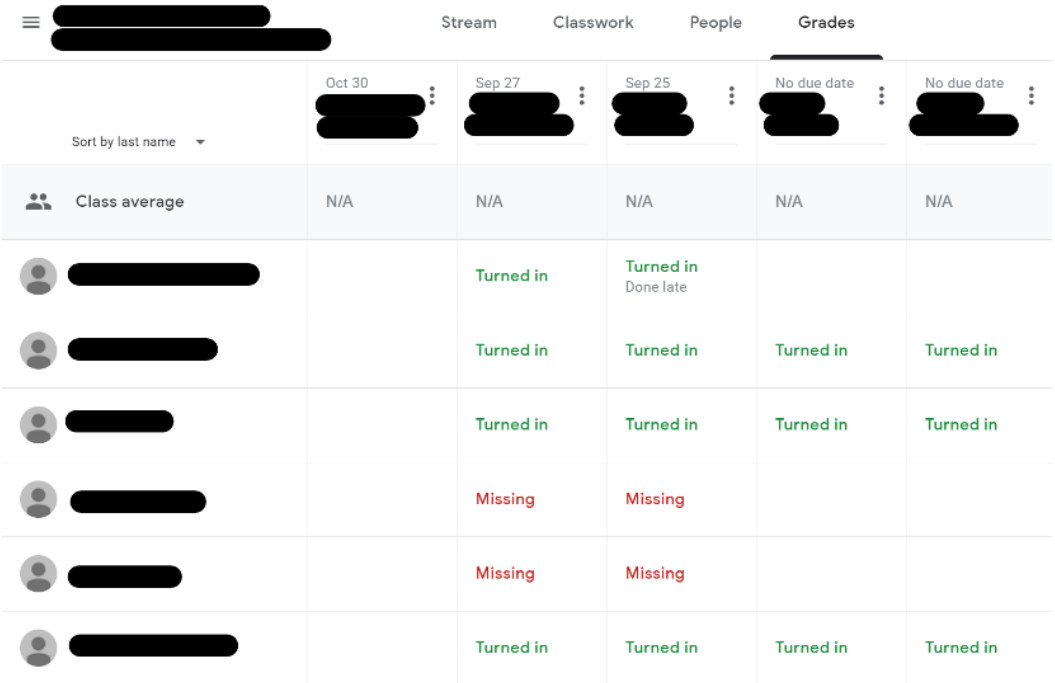
*Source: https://miro.medium.com/max/3920/1*MVePFPo19hol8zrINy5K0w.png*



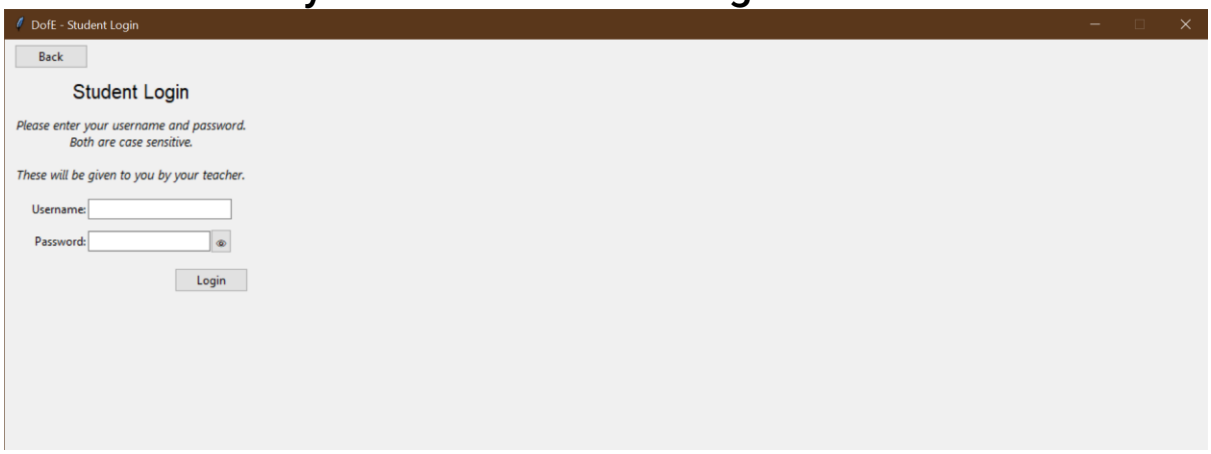
Screenshot 5 Google Calendar – adding an event



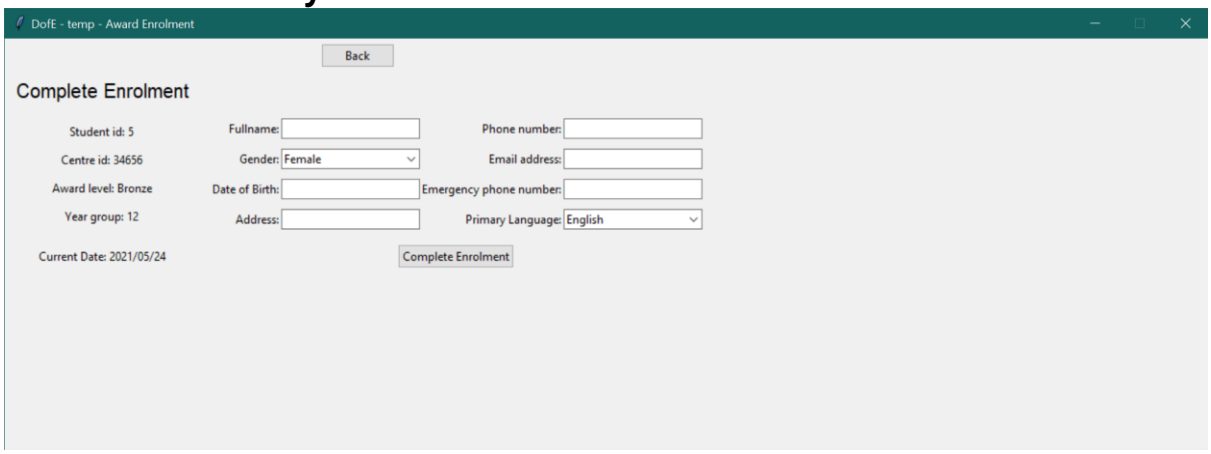
Screenshot 6 Google Classroom – Student grade overview



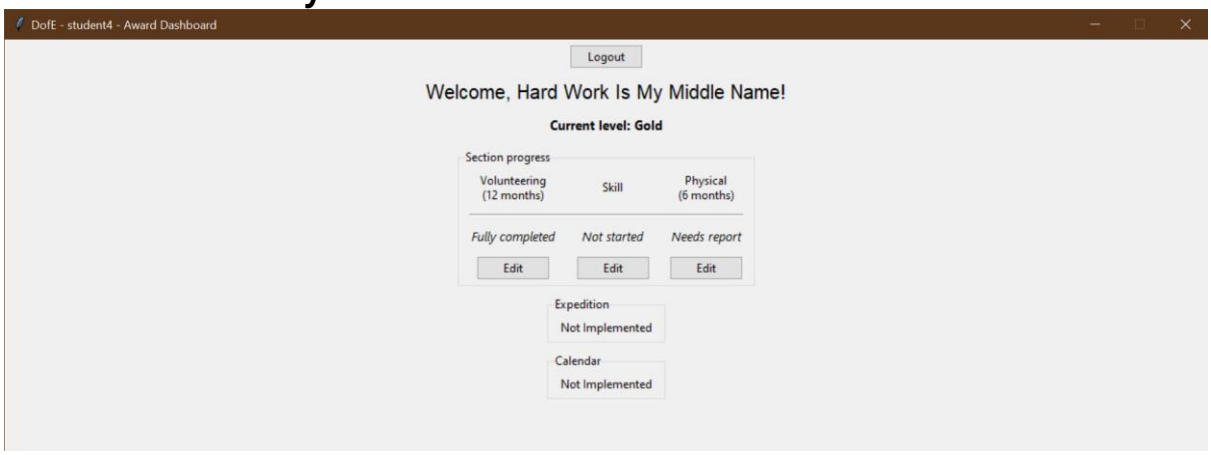
Screenshot 7 My solution – student login



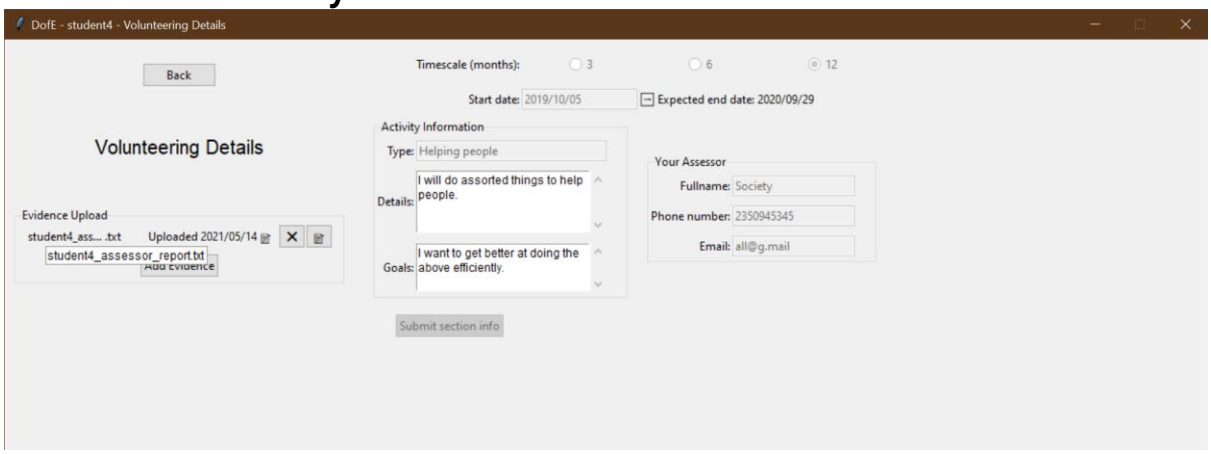
Screenshot 8 My solution – student enrolment



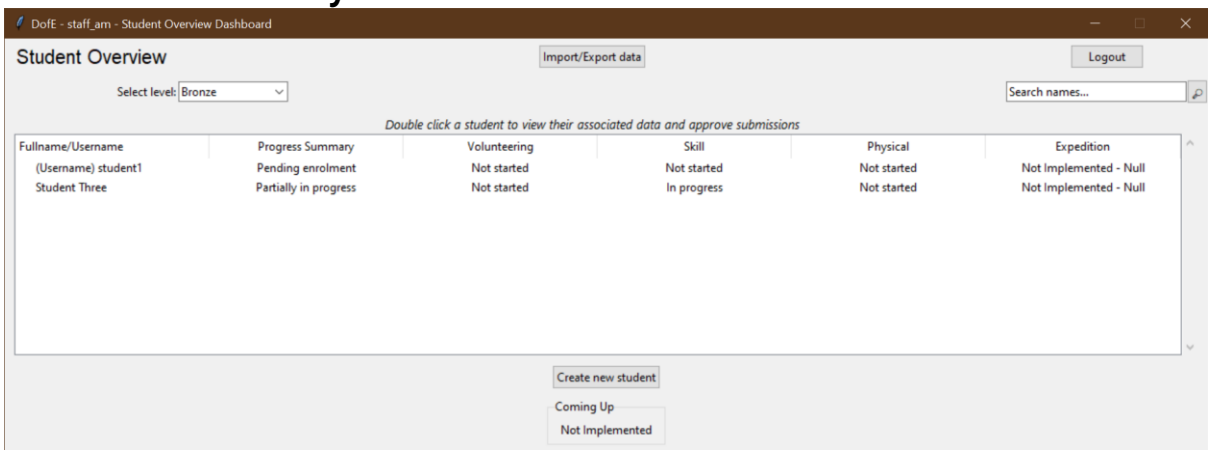
Screenshot 9 My solution – student dashboard



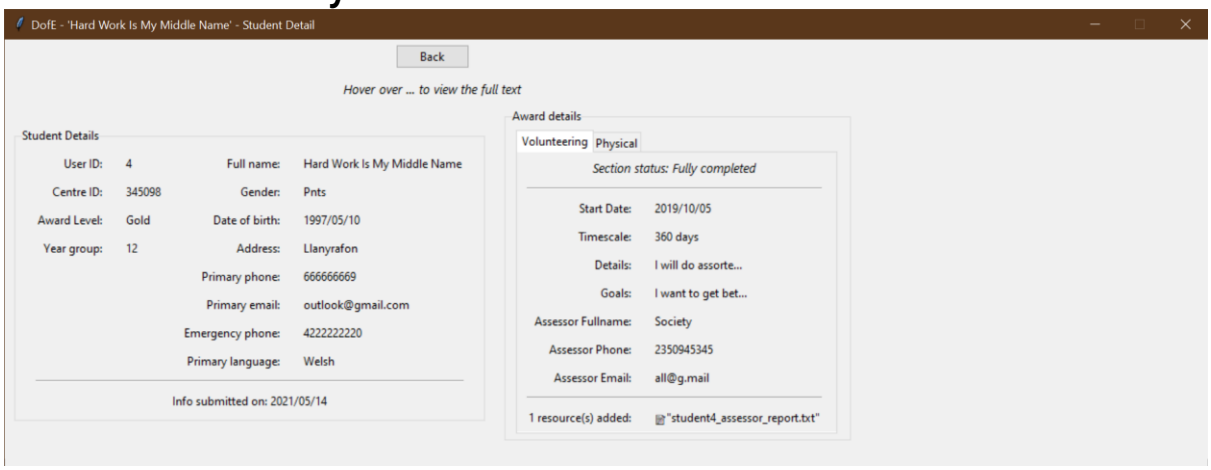
Screenshot 10 My solution – section details



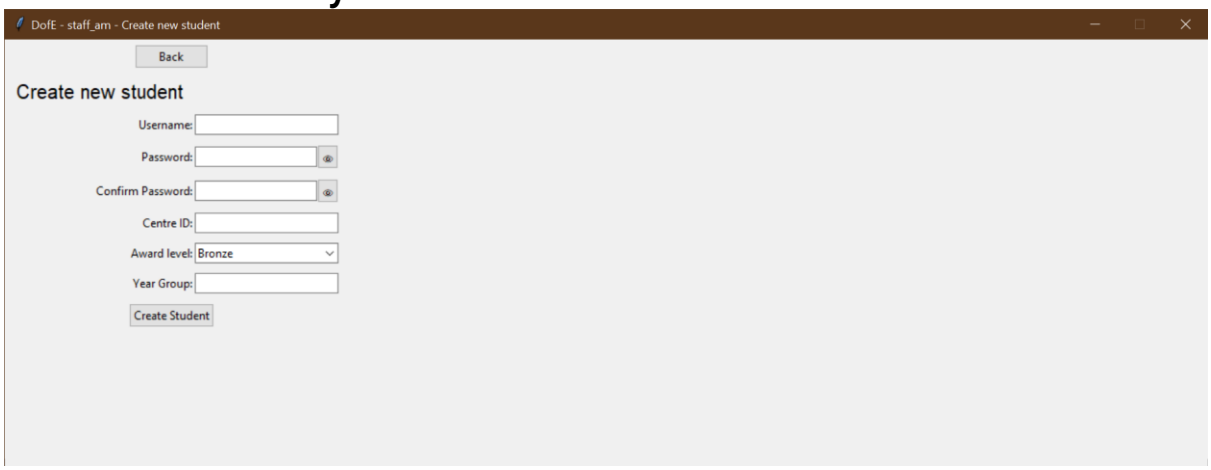
Screenshot 11 My solution – staff student overview



Screenshot 12 My solution – student details



Screenshot 13 My solution – student creation



Screenshot 14 My solution – command line help

```
C:\Users\User\OneDrive\Python Programming\A-Level computing stuff\gce-unit-5>python main.py --help
usage: main.py [-h] [-f SUFFIX] (-g | -s | -p)

optional arguments:
  -h, --help            show this help message and exit
  -f SUFFIX, --file-save-suffix SUFFIX
                        optional suffix to add when loading files (use to load specific tables)
  -g, --show-gui        show GUI to log in to system as staff or student
  -s, --create-staff-account
                        launch interactive command line to create a staff/admin account
  -p, --populate-tables
                        launch interactive command line to create random test student data
```

APPENDIX C – DIAGRAM AND NOTATION KEYS

Data flow diagrams