

# NEA ASSESSMENT

February 28

# 2018

---

Sponsored Walk App.

My cool  
project  
doode.

## CONTENTS

Analysis .....	3
Description of the problem .....	3
Initial research .....	3
The Current System .....	3
Research .....	4
Solution Outline .....	5
End user .....	5
Staff .....	5
Students .....	6
Parents (indirectly) .....	6
Specific measurable objectives .....	6
Documented Design .....	7
High-level overview .....	7
Web App .....	8
Android App .....	10
Description of algorithms .....	12
Oval-Plotter .....	12
‘Intelligent’ Search Box .....	15
Inviting a new user .....	17
Google OAuth2 Flow .....	18
Creating a RESTful API for data updates and queries. ....	22
Use of external APIs .....	24
api.qrserver.com .....	25
Description of data structures .....	26
Database design .....	26
Samples of planned SQL .....	28
Android App Internal Data .....	29
Design of user interface .....	30
Mobile Tracking App UI Draft Design .....	30
Web App – Tracking and Admin UI Draft Design .....	32
System security and integrity of data .....	35
SQL Prepared Statements .....	35
UAC (User Account Control) .....	36
Tracking data reliability .....	37
Student data accessibility .....	38

Testing .....	38
Test Plan .....	38
Evaluation .....	38
End-user feedback.....	38
Appendix A - Code .....	0
Web App.....	0
Front end pages .....	0
AJAX & API backend pages.....	73
Tools, Resources and Configuration Files .....	100
Android App .....	0
Screen1 .....	0
Welcome .....	1
Check_Status.....	4
Not_Live.....	5
Register .....	8
Confirm .....	10

## ANALYSIS

### DESCRIPTION OF THE PROBLEM

Each year, my school holds a sponsored walk fundraiser. Students walk 10 miles, and at regular intervals must register with a member of staff to ensure their safety, and to track the progress of the student. The current system uses a combination of paper registers and student cards. A student will approach the teacher holding a register, spend 5 minutes finding their name, and then mark both the paper register and the student card with their initials. This process occurs at multiple locations along the walk, and registers will correspond to the specific locations. Here is an example of the current student card →

As seen to the right, there are four locations where registration occurs, where staff will write their initials to confirm a student's whereabouts. However, within the current system there are multiple flaws, which contribute to reduced speed and accuracy when tracking students on the walk. Paper registers often get wet and unusable, causing registration to slow. Some students may even skip registration as the waiting time exceeds 20 minutes, causing big safety issues. Sponsored walk administrators will only know if students register at locations, and hence their whereabouts, when the registers are returned to school late in the day.

So, to have increased efficiency of registration, and live student progress reviews available, a new system must be put in place. A system of electronic registration with database interaction and a suitable web interface for live tracking could serve as a safe and reliable alternative.

**WHAT TO DO IF ANYTHING GOES WRONG**

If you realise you have wandered off the route or got lost:

- If possible, phone school immediately. Give your names and mobile number, explain what the problem is, and give details of any landmarks close by so that we can locate you. Follow instructions given by school (we may need to call you back).
- If you don't have a phone, re-trace your route if you are able to. If not, find a safe place and stay where you are. We will come to find you.

SCHOOL PHONE NUMBER:  
(01423) 566358

If someone is ill or hurts themselves to the extent it is difficult for them to carry on:

- Try to locate a member of staff, or else a parent or Sixth Former.
- If none of the above are nearby, phone school and seek advice. Follow the advice given at the top of this card about phoning school.

Name: \_\_\_\_\_ Form: \_\_\_\_\_

Check-points	Staff Initials
Pot Bank (4.3km)	
Long Liberty (8.4km)	
Beckwithshaw (13.0km)	
Southill Hall (16.3km)	

### INITIAL RESEARCH

#### THE CURRENT SYSTEM

Currently, the registration process involves a three-step process:

1. A student arrives at a checkpoint and searches for the register with their name on it using a range of characters from the alphabet e.g. a student called Harry would visit the register with the character range [A-H].
2. The student waits in a large queue.
3. The student is registered.

This process has many negative aspects, which impact on the security and reliability of registration. Some of these points include:

- Paper registers are susceptible to damage due to weather conditions. This means records can be lost.
- Long waiting times often cause peoples to skip checkpoints.
- Data can only be viewed once the registers are returned to the central hub at Ashville.

There is a cascading negative effect from the current system which results in overall lack of knowledge about student whereabouts. Especially since the walk is such a long distance, and for much it unsupervised, there could be serious consequences. The current system effects the school in the following three ways:

1. Students will not be registered.
2. Teachers will not know where students are.
3. Parents will not be able to be assured by teachers.

---

## RESEARCH

To create a solution which was relevant to those who would be using it, I reached out to the current organiser of the sponsored walk, Mr. Daniels, to ask if we could talk and build an idea of the kind of system he would like to use:

-----Original Message-----  
From: 11WHA <11WHA@ashville.co.uk>  
To: Matthew Daniels <MD@ashville.co.uk>  
Sent: Mon, Nov 6, 2017 12:40 pm  
Subject: My Computing Project - A Proposal

Dear Mr. Daniels,

I am writing to ask you a big favor!

As you know, I am currently taking Computing as an A-level subject and as part of this I must write an application of some description.

I know that, for many years, you have been the organiser of the Sponsored Walk. As the organiser, I understand from Mr. Watson that there have been some problems with the current system. The paper registration system, from my understanding, often leaves a lot to be desired. If the register isn't soaked through or has blown away, usually there are huge queues leading to a group of different people searching for names.

From this, I proposed to write a digital tracking application that would use student's devices to register them at each checkpoint. I was wondering if you had any interest in this system, and if so, whether we may be able to take some time to discuss some points that you would see beneficial in such a system.

Thank you very much for your help!

Best Regards,

William Hall  
U6ER



He responded:



Matthew Daniels <MD@ashville.co.uk>

To: 11WHA; ✉

Hi Will,

Sounds like an interesting project! I am free to talk Tuesdays & Thursdays at 12:50 most weeks. Just pop in when you can.

Regards,

Mr. Daniels  
Maths & Sports



So, I went and spoke to Mr. Daniels and outlined some key points. From our conversation, it was clear that a new system of digital tracking was a welcome idea.

However, for the system to be useful, it was imperative to meet some main specifications. The tracking had to be above all “reliable” with no down time on the day. It was clear that if the school were to put their trust in an electronic system that it had to be a robust one. After all, even with its faults, a paper register can never lose its connection to the servers.

The system also had to be “easy for staff” to use, with minimal learning curve or complex UIs. If the staff were to adopt a new system, they must be able to feel comfortable using it. Similarly, the system must be “easy for me [Mr. Daniels]” and any other administrator staff to track students, noting that that must be the primary function of any “administrator side of things”. It must be easy to find out a student’s whereabouts and noted that an “overview of things” might be helpful too. I asked what an overview might look like, and he remarked that a “general idea about how things were getting along” might help to know when student groups were likely to finish their walk.

---

## SOLUTION OUTLINE

So, from conversation and consolidation, the following key ideas must be met in the system:

1. **Easy to use** – No complex UIs or multi-step processes. It must be easy to adopt.
2. **Easy to read** – Get information about the walk quickly, with no complexity.
3. **Easy to find** – Find a specific student and their whereabouts in seconds.
4. **Easy to rely on** – A robust solution, with no false positives or negatives.

## END USER

---

### STAFF

---

#### REGISTRARS (IN THE FIELD)

These users will be the ones registering students out in the elements on the day of the walk. They are likely to be:

- **Busy** – There are a lot of people to register at a lot of places.
- **Unforgiving** – If their app breaks, they will not be happy.
- **Apprehensive** – They want to be sure the new system will work, otherwise they can’t do their job.

So, to counter such features of a user, the following points must be met in the Android app:

- **Fast** – Registration should take no longer than a few seconds.
- **Reliable** – It must be made very clear if a student has been registered or not.
- **Simple** – In order for staff to adopt the system, it must be intuitive.

---

#### ADMINISTRATORS (BACK AT BASE)

These users will be the ones viewing the tracking data, monitoring the progress of the walk, and finding students if required. They are likely to be:

- **Worried** – There are a lot of students about the leave the campus.
- **Accountable** – If someone is lost, they need to find them immediately.
- **Under fire** – They need to answer to any parents request quickly.

So, to ensure operation for the administrators runs smoothly, the following points must be met:

- **Realtime** – The system needs to relay live data without clicking ‘refresh’.
- **Intuitive** – It must be very simple to find a student.
- **Reliable** – If a parent needs an answer, the staff need to provide the correct response.

---

## STUDENTS

Students will need a way to get their QR code, and maybe see how they are doing on the walk. They are likely to be:

- **Confused** – There will be questions about how the system works.
- **Mischievous** – They will want to try and *break* the system at all costs.

So, to ensure that students get the best possible experience, the following points must be met:

- **Straight forward** – They must have minimal clicks to get their QR code.
- **Secure** – It must be difficult the break/misuse the system.

---

## PARENTS (INDIRECTLY)

Whilst parents will not have any direct access to the system, the system must still act to provide them with information indirectly. They are likely to be:

- **Concerned** – Their child is somewhere in the countryside.
- **Insistent** – Because their child is in the countryside, they will want to know where. Quickly.
- **Uncertain** – Because they want to know quickly, they are likely to doubt any new system.

To counteract this, a few points must be met:

- **Relevant** – Only providing data to the parents which is about their child.
- **Quick** – The lookup of a student must be fast.
- **Assuring** – The data must be presented in a professional and simple way.

## SPECIFIC MEASURABLE OBJECTIVES

To properly track students, and relay that information to an administrator, the following goals must be met:

- ✓ **The Android app must be easy to use, with a clear and concise UI** – The app must be clear to increase the efficiency of registration, and easy to use so that the user is never confused or unsure that registration was successful.
- ✓ **The Android app must be able to scan and decode a QR code** – The app must scan, decode and present QR Code data to the staff member. This allows for human verification.
- ✓ **The Android app must be able to communicate with a database** – The app must be able to read and write to a database from anywhere in the country. This communication must be fast and reliable, so that tracking data can easily be written.
- ✓ **The Android app must be able to be configured for each checkpoint** – The app must be able to accept information about each location such as:
  - **Staff Initials** – Who is registering?
  - **Checkpoint** – Where are they registering?
- ✓ **The Android app must never register a student in error** – This means that at every step, there must be in-depth error checking and multi-part verification to ensure that a student is not falsely registered.

For the administration backend:

- ✓ **The Web app must have a secure login** – There must be measures to ensure that only verified users many view tracking data.
- ✓ **The Web app must display tracking data in a concise way** – It must be easy to view the progress of the walk and see key statistics about the walk progress.
- ✓ **The Web app must be able to track individual students** – It must be easy to view the progress of the walk on a student by student basis, so that individuals can be located.
- ✓ **The Web app must provide administrator tasks** – These tasks include the changing of the walk state, user account control and printing QR codes. These tasks must only be accessed by those who are authorised to do so.
- ✓ **The Web app must never show false data** – In terms of pure reliability on the day of the walk, it is imperative that only data which is accurate is shown, and hence there must be checks ensure that the right data is shown in the right way.

## DOCUMENTED DESIGN

### HIGH-LEVEL OVERVIEW

The project will be comprised of two components: An Android app for the collection of tracking data, and a Web app for the viewing and organising the collected data. The main objective is that tracking speed and efficiency of the sponsored walk is increased, and hence both user interfaces must be easy to use and easy to read. There must be the fewest possible steps to accomplish an action as possible.

Tracking data must be collected and stored so that it may be easily accessible to many different clients and programs. In this way, a cloud hosted MySQL database can be used to store all the tracking data, student data, staff data and location data for the sponsored walk. This ensures that the database can be accessed from any client with an internet connection and the correct login credentials, so that someone out in the field can as easily write tracking data as an administrator read it back at base.

It is also important that tracking 'sessions' are kept discreet, so that data cannot be shared amongst multiple walks in multiple years, preventing incorrectly listed tracking data. Because of this, we need to ensure that the system can be in a number of 'states', where data is encapsulated to only one session of tracking.

The whole system can be in three states:

1. Active – Tracking is live, and students can be registered.
2. Waiting – Tracking has not started, but a start-time has been confirmed.
3. Inactive – Tracking has not started, and no start-time has been confirmed.

Similarly, on the database backend, the data from previous walks must be purged, so that new tracking data can be written to the database without risk of combining new and old data. When you change state, all database data MUST be purged, with the option of capturing the database beforehand to record tracking data for statistical analysis.

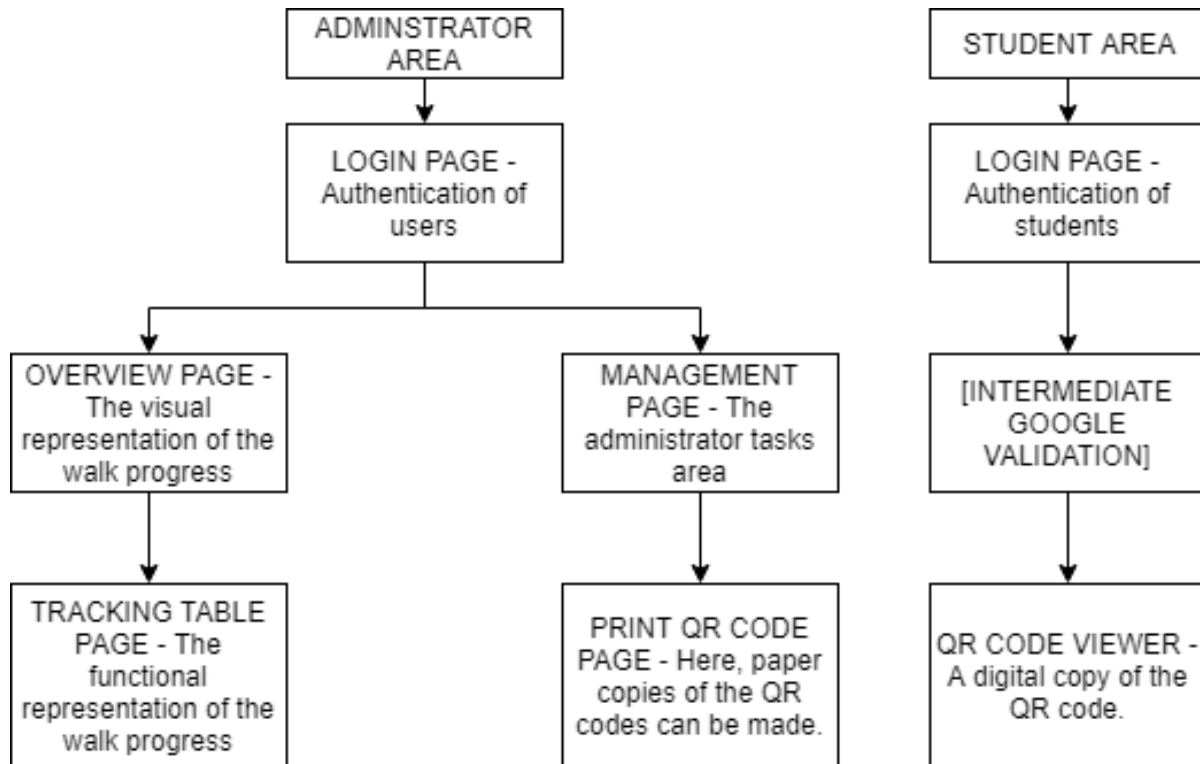
Students must be able to easily access their QR code, with access to both a paper and digital copy of their QR code. It is imperative that students can only access their own QR codes, so that students cannot impersonate other students. To do this, utilising the schools Google Account system and OAuth2 to build in a secure student login using a student-unique password would work.



---

## WEB APP

The web app will be comprised of two sections: Administrator and Student areas. All areas of the web app are non-public, meaning some sort of login must be made to access any data. Below is the proposed structure of the site:



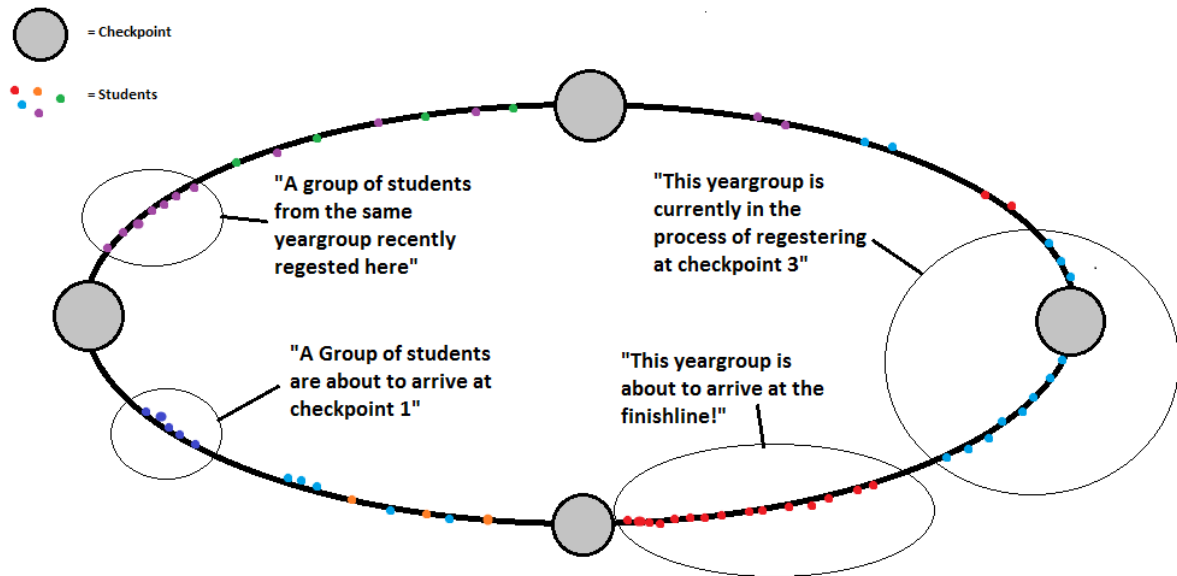
---

### ADMINISTRATOR VIEW – TRACKING AND MANAGEMENT

The web app will allow administrators to carry out a range of tasks. These include:

1. Starting and stopping tracking i.e. changing the 'state' of the system.
2. Viewing the tracking data.
3. Inviting new users.
4. Removing some users.

Tracking data must be presented in an easy to view way – the administrator must be able, at a glance, understand the current whereabouts of students, and progress of the overall walk. To accomplish this, a large oval with coloured dots could be used to display the estimated positions of the students on the walk, based on tracking data and walking-pace calculations. As shown by the below image, some deductions of the walks progress can be made based on the visual aid only:



By clicking on one of the grey 'checkpoints', the user can then view information about that specific checkpoint. They can view which staff member is registering there, and how many students have passed through this checkpoint. They can also see a google maps integration of the location, to benefit from a geological understanding of the checkpoint area, and asses any potential hazards.

The user must also be able to view the tracking data on a student-by-student basis, so that individual students can be located, and any missing persons can easily be traced back to their last known whereabouts. This tracking data will be presented less visually and more functionally, using a table to display an ordered list of all students who are on the walk, including their registration times and locations. This data-table will be geared towards the presentation of facts and statistics, compared to the graphical overview shown above. Individual students can easily be located, and their tracking history fully viewed. It must be quick and easy to find a student, so a search box to type names must be present, and able to dissect names and search both surname and forename on the student database.

As a secondary element to the main tracking data, the web app must also be able to carry out tasks such as controlling access to the tracking software, printing QR codes and controlling the state of the tracking system. There must be an easy to use form to invite new users to gain access to the tracking software, and with similar ease a form to update the permissions of, or remove users, so that there is controlled access over the tracking data. This means that only those with permission can view data, improving security. There must also be an easy way to print the paper versions of the students QR codes, and allow either single cards, or entire sets, to be printed. Finally, it is important that the user be able to control the aforementioned 'state' or the walk, and hence enforce proper sessions and prevent data-sharing. The user must be easily able to commence the tracking state-change (i.e. changing the system from Active to Inactive) but be repeatedly queried to ensure that this destructive action is wanted and not carried out by mistake.

It is important that users who access the web app that they have the correct permissions to carry out the tasks they are supposed to do. For example, consider a teacher who has been granted access to the tracking software but has limited knowledge of its function. Perhaps this teacher may erroneously erase all the tracking data by accidentally changing the state of the system whilst trying to log out. We could give this user reduced permissions so that they cannot access hi-level features but may still view the tracking data. This means that

only the executive administrators can have access to functions such as changing system state or inviting new users and choosing their permission level.

---

#### STUDENT VIEW - QR CODES AND PROGRESS

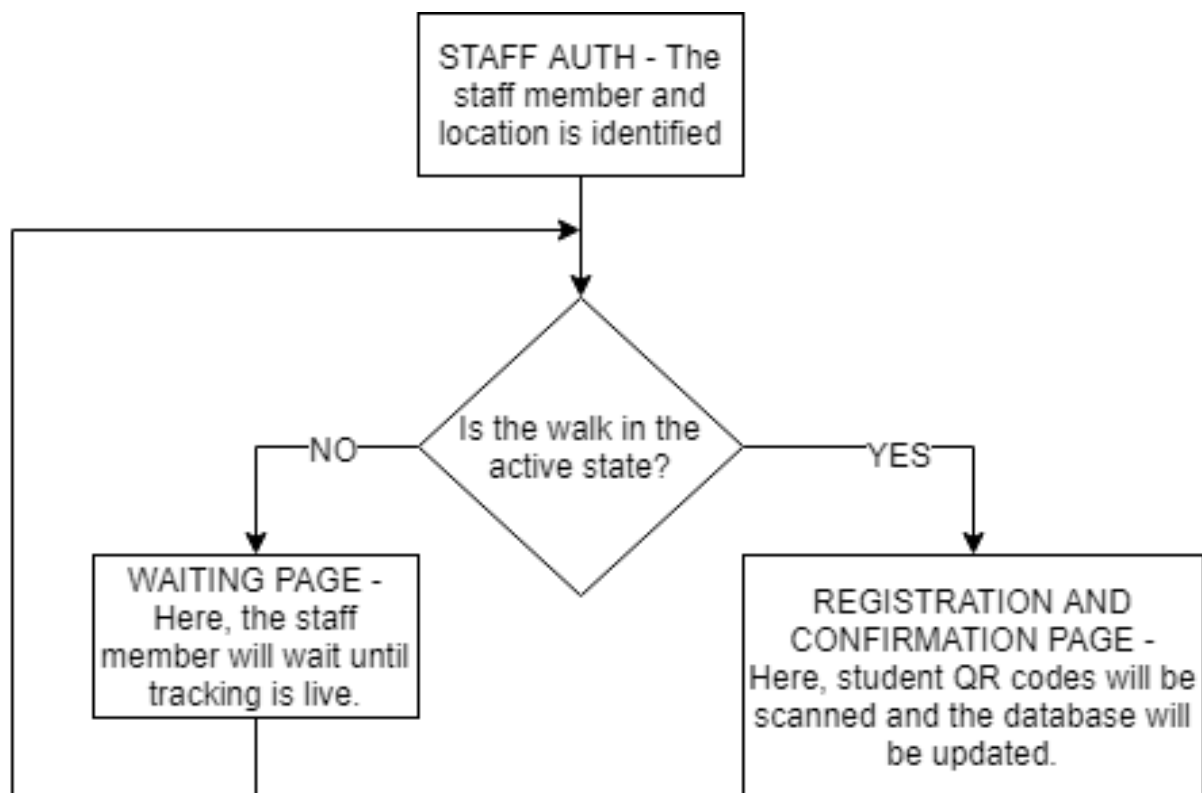
Students must be able to access their 'QR code' with ease, whilst also ensuring that students cannot impersonate each-other, a serious security issue which may lead to false tracking data and a reduced confidence in tracking reliability (More on this in the security section). To solve this, a smaller web app can be built using Google's OAuth2 flow to authenticate students by using their existing Google Account which has been setup by the school. This means that students can only access their own personal QR code and can make use of an existing account which has passwords that automatically update when changed at school. After successful authentication, students can view their QR code and use it to register at the checkpoint locations.

After registration, students can refresh the page to see a small table containing their tracking data and compare it amongst friends and track their progress. This small feature means that student can keep track of themselves and hurry up if needed.

---

#### ANDROID APP

This app will be used on the phones of the teachers/staff that are positioned at the locations around the walk. The basic flow of the app is as follows:



They must be able to easily enter their staff initials and their tracking location. Then, they must be able to view the 'state' of the walk i.e. actively tracking or waiting to begin tracking. When the tracking is active, the teacher/staff must be able to:

1. Scan a QR code
2. View the name of the student who is carrying the QR code.
3. Use human judgment to compare the displayed name and the actual student.
4. Confirm or Decline the registration and update the tracking database.

If the tracking is waiting to begin, the teacher/staff must be able to view a countdown timer to whatever start time has been confirmed by an administrator on the web app. To get this information, the app will make HTTP requests to a database API that I write to allow access to specific data from the database. This will allow clients to access data about system 'state', get the countdown timer and compare student codes to student full names.

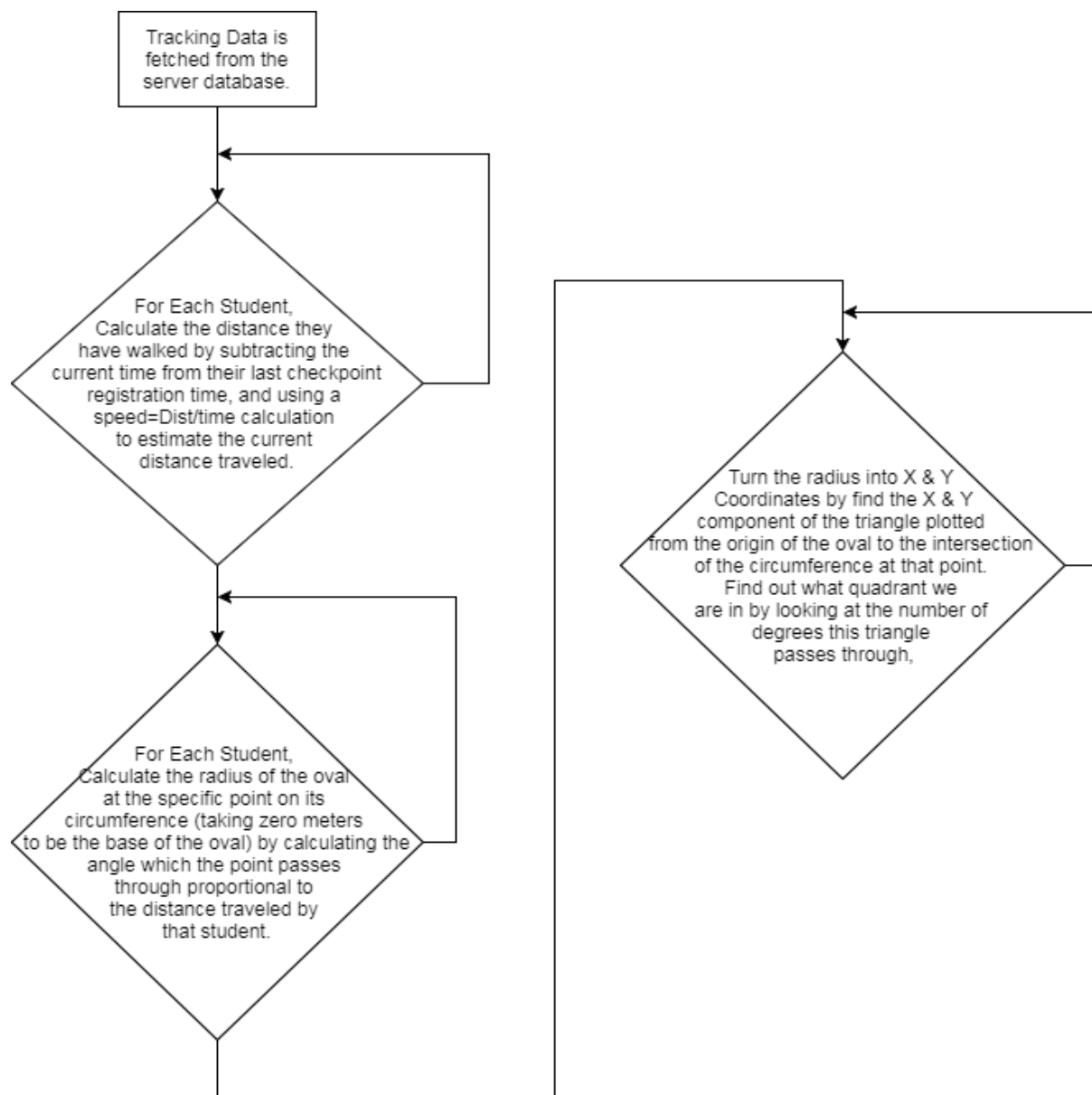
For development, an IDE such as Android Studio could be used. This would grant full control over the functionality of the application, but from initial research proved too much of a learning curve (Learning Java, the complex multithreaded working of an Android app, building and launching an app manually with manifests and config files etc.) for what really should be the secondary addition to the main project of writing the web app. Therefore, I will abstract away some of the complexity of the app development process by using the MIT App Inventor 2 suite. This way, I can quickly create a reliable app to collect data which feeds the main web app.

## OVAL-PLOTTER

A challenge was faced when it came to creating the visual display of the tracking data. I had to take an input of tracking data and output a set of co-ordinates that plot to an oval shape relative to the distance they have travelled on the walk.

## BASIC ALGORITHM

The following flowchart outlines the basic flow of the algorithm – Taking a table of tracking data and returning a table of X & Y coordinates to plot on a HTML Canvas:

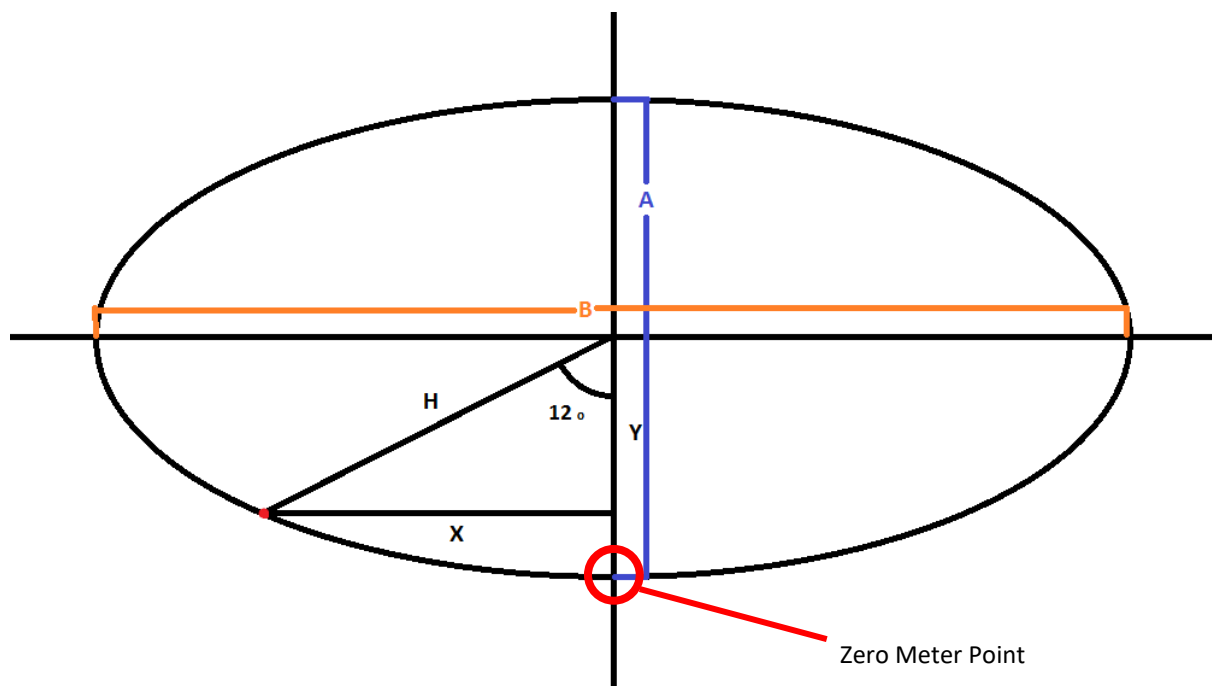


An oval (or Ellipse) has an equation like that of a circle. In fact, a circle is just a special case of an ellipse. The equation is:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

We could rotate 360 degrees about the origin of the ellipse and plot a line to the circumference of the ellipse and, by doing this, we could in fact plot points on the ellipse itself. So, if our sponsored walk were 30000 meters long, we could divide 360 degrees by 30000 meters to get the number of degrees we must turn through to move around the circumference of the ellipse proportional to the distance of one meter travelled. From that, we can calculate the number of degrees we must turn through to plot the point on the circumference proportional to the number of meters we predict the student has travelled through. E.g., If our student travelled 1000 meters:

$$\frac{360^\circ}{30000m} \times 1000m = 12^\circ$$



So, we would draw a line  $12^\circ$  from the bottom centre of the ellipse, (or zero meters):

However, to calculate X & Y, we first need to calculate H so that we can use basic trigonometry find the values of X & Y. H is the Radius at any given point. We can calculate this by using the polar form of the above ellipse equation over the page:

$$r(\theta) = \frac{ab}{\sqrt{(b \cos \theta)^2 + (a \sin \theta)^2}}$$

Where A = Height of the ellipse, B = Width of the ellipse and Theta = the angle we calculated on the previous step. Now we can simply take the X and Y components of the triangle formed from the centre to the circumference, with H as the hypotenuse.

#### DRY RUN

To test the function of the algorithm, let trace out a dry run, and see if the algorithm can run without any errors, and identify if there are any areas for improvement:

<b>Current Time</b>	15:00	<b>Walk Distance</b>	30000 meters	<b>Walk Speed</b>	5000 meters / hour
<b>Checkpoint 0</b>		<b>Checkpoint 1</b>		<b>Checkpoint 2</b>	
0 meters		15000 meters		30000 meters	

Last Location	Time of Registration	Distance Travelled Since Last Location	Total Distance Travelled
0	12:00	15:00 – 12:00 = 3:00, 3:00 * 5000 = <b>15000 meters</b>	15000 + 0 = <b>15000 meters</b>
1	13:00	15:00 – 13:00 = 2:00, 2:00 * 5000 = <b>10000 meters</b>	10000 + 15000 = <b>25000 meters</b>
2	14:00	15:00 – 14:00 = 1:00, 1:00 * 5000 = <b>5000 meters</b>	5000 + 30000 = <b>35000 meters</b>

Error!

The first problem has arisen – since it is impossible to have travelled a greater distance than the overall distance of the walk. Similarly, say that the student travels slower than the algorithm predicts, and hence it appears that the student will have registered when actually they have not. Hence, the algorithm must predict the distance travelled to be less or equal to the distance of the next available checkpoint. Then, after we see that the student has registered at that location, we can begin to estimate travel distance again. That will ensure that the visual representation of the tracking data as accurate as possible, but it must be made clear to the user that the visual representation approximates the progress of each student.

---

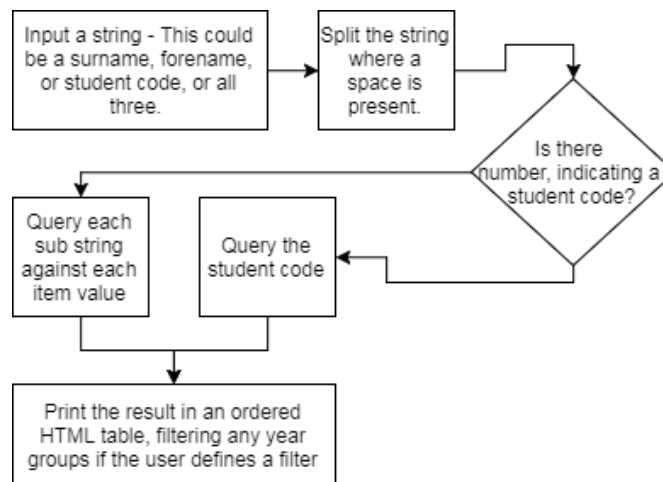
## 'INTELLIGENT' SEARCH BOX

The goal of this algorithm was to ensure that any student's tracking data could be found as quickly as possible through surname, forename, student code or all three in combination. Similarly, I wanted to be able to use a single search-box to return matching students, rather than separate search boxes for each item. Finally, I wanted the algorithm to return near-match results as well as full match results, so that even just the initials of the student could be used to find that students tracking data.

---

### BASIC ALGORITHM

Below is a flowchart of the basic algorithm:



After splitting the string into several substring 'queries', the MySQL database must then be queried to return rows that match the query of many fields. This must include the non-exact matches too. To accomplish this, SQL must be built dynamically to allow for any number of query 'substrings' to be searched against – no easy feat, since to improve security, prepared statements will be used (More on this in the security section). Usually, when using a prepared statement, we know how many values we will query against. But since we are using dynamically generated statements, we need a way to also dynamically bind 'query' strings to the saved statement when we are ready to query. So, to do this, we need to know what exactly happens when we bind a parameter to a prepared statement.

An example of a prepared statement could be:

```
1. $query = $conn->prepare("SELECT name FROM tblNames WHERE age = ?");
```

Then, the binding code in PHP:

```
1. $query->bind_param("i", 18);
```

Here, we bound 18 to the age field. `bind_param` is a method of `$query`, since we defined `$query` as our prepared statement – we are binding parameters to our statement. The `bind_param` method accepts two parameters in this instance. "i" represents the data type we are looking to bind (i = int, s = string), and 18 is the actual value we are binding. From this we can deduce that the method `binds_param` can accept an array, whereby the head is a string of characters defining what datatypes to expect, and the tail is the actual values to bind.

So, we need to build a dynamic array of 'type' and 'values' to then use as the input to our `bind_param` method. To do this, we could use an instance of an object to store all our types and values as we go through the program building our SQL dynamically. Then, when we are ready to call `bind_param`, we can write a 'get' method for the object which will build up and output our array of 'type' and 'values' to pass over to `bind_param`. For example:



```

1. class BindParam
2. {
3.     //Declare Private Variables to store our 'type' and 'Values';
4.     private $values = array(), $types = '';
5.
6.     //Method to append a new 'value' to array, and new 'type' to string.
7.     public function add( $type, $value )
8.     {
9.         $this->values[] = $value;
10.        $this->types .= $type;
11.    }
12.
13.    //Return a new array, which contains references to all the values we stored whilst
    running through the program, and our 'type' string.
14.    public function get()
15.    {
16.        $refs = array();
17.
18.        foreach($this->values as $key => $value)
19.            $refs[$key] = &$this->values[$key];
20.
21.        return array_merge(array($this->types), $refs);
22.    }
23. }

```

To allow for the near-string matches to be returned, we could use the SQL 'LIKE' operator, which acts similarly to the '=' operator, but instead will return all rows matching a string pattern ending with '%' as a delimiter, rather than an exact match.

#### DRY RUN

To better illustrate the generated SQL, a few examples are listed below:

Search String	SQL Condition	Results	Explained
Joe Bloggs	(forename LIKE 'Joe%' OR surname LIKE 'Joe%') AND (forename LIKE 'Bloggs%' OR surname LIKE 'Bloggs%')	Joe Bloggs	Joe Bloggs matches all the conditions.
Bloggs Joe	(forename LIKE 'Bloggs%' OR surname LIKE 'Bloggs%') AND (forename LIKE 'Joe%' OR surname LIKE 'Joe%')	Joe Bloggs	Even if the name is inverted, he will match the conditions
J Blog	(forename LIKE 'J%' OR surname LIKE 'J%') AND forename LIKE 'Blog%' OR surname LIKE 'Blog%')	Joe Bloggs Jane Blogger	Now, some other close matches are returned too.
J B	(forename LIKE 'J%' OR surname LIKE 'J%') AND (forename LIKE 'B%' OR surname LIKE 'B%')	Joe Bloggs Jane Blogger Barry Jones	Now, matching initials in any order are returned.
J B 11JB	studentCode = 11JB	Joe Bloggs	Since a number is present, we must have used the student code, and so we can search just the code.

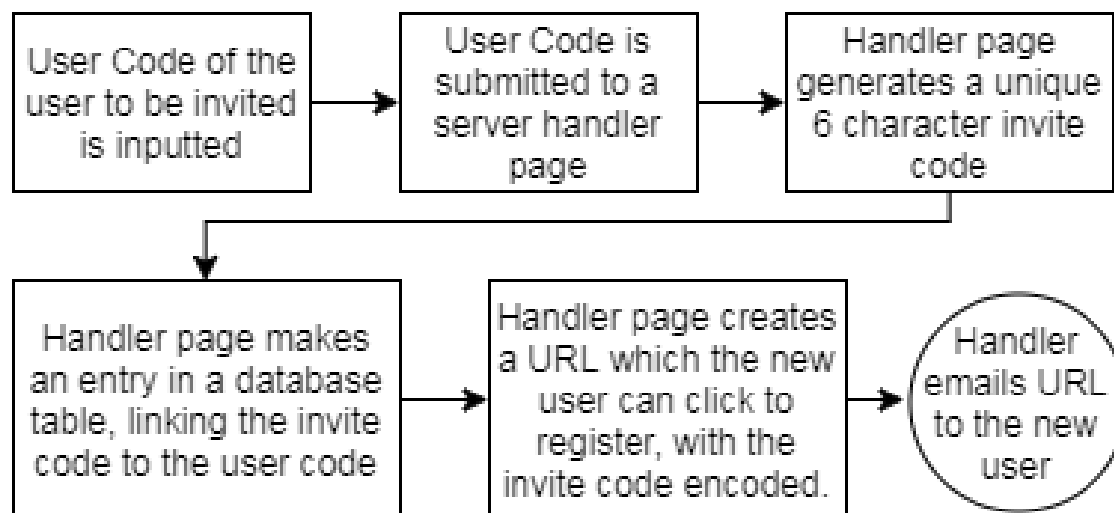
---

## INVITING A NEW USER

Inviting a new user to view the tracking suite requires a secure invite to be sent, that cannot be used by anyone else, and that is destroyed after registration. To do this, unique invite system can be implemented, whereby each invite is identified by a unique invite code, which references the user code of the new user to be invited. That means that only users with an invite code can register, and only the intended person can use the registration form to register. This is better explained below:

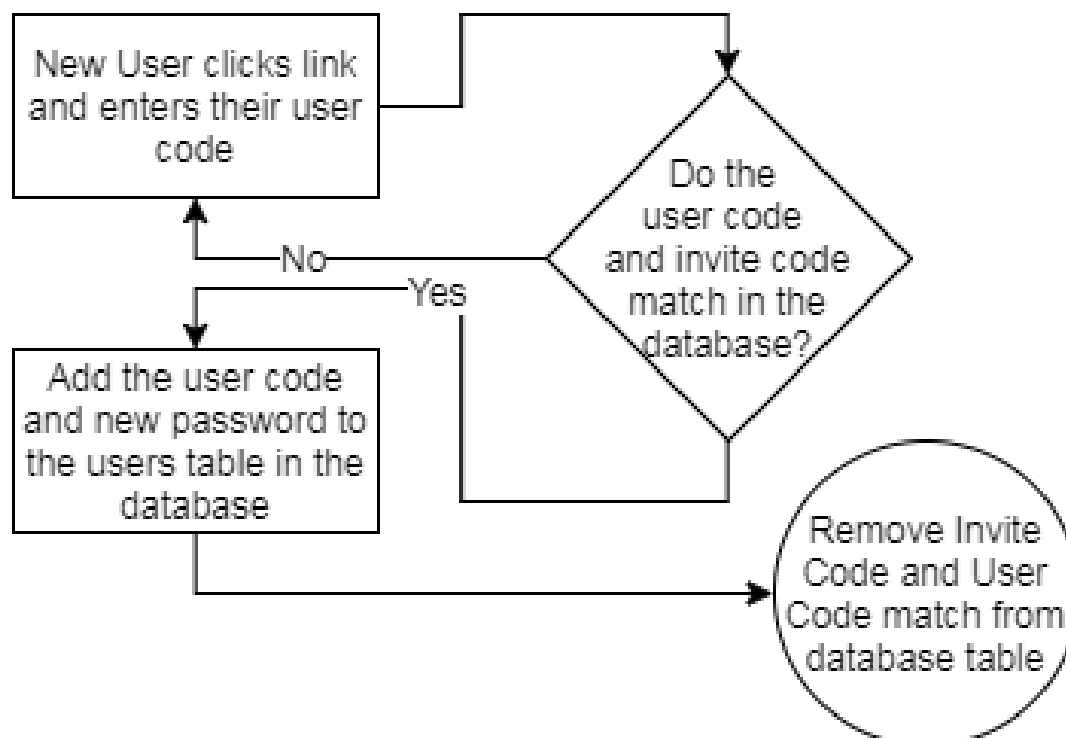
---

### CREATING THE INVITE



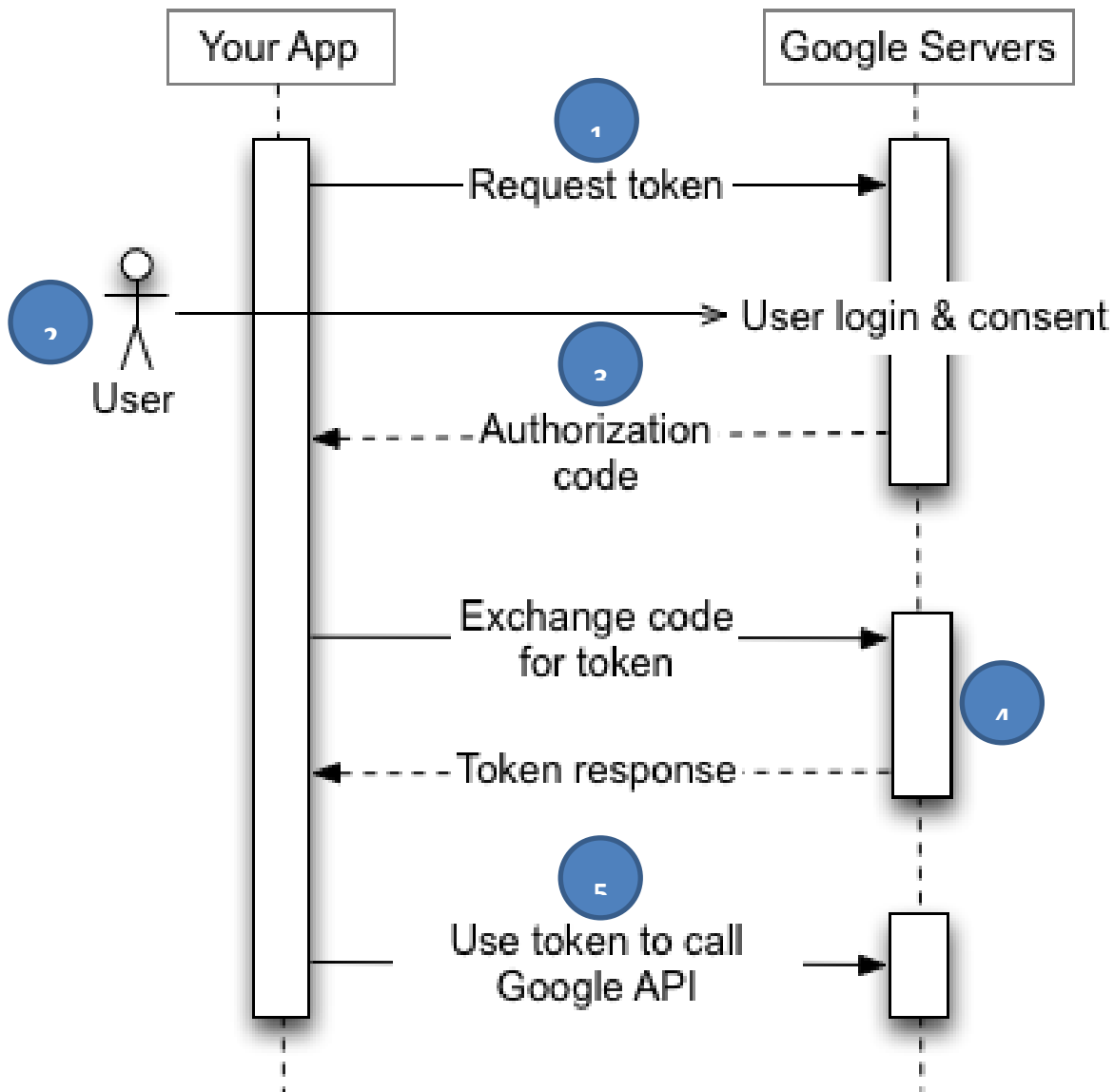
---

### RECEIVING THE INVITE



## OVERVIEW

To utilise the Google authentication flow, first we must understand how exactly we utilise Google's servers to gain login and credential information. The following diagram outlines the process:



Firstly, let's take each step and explore it further:

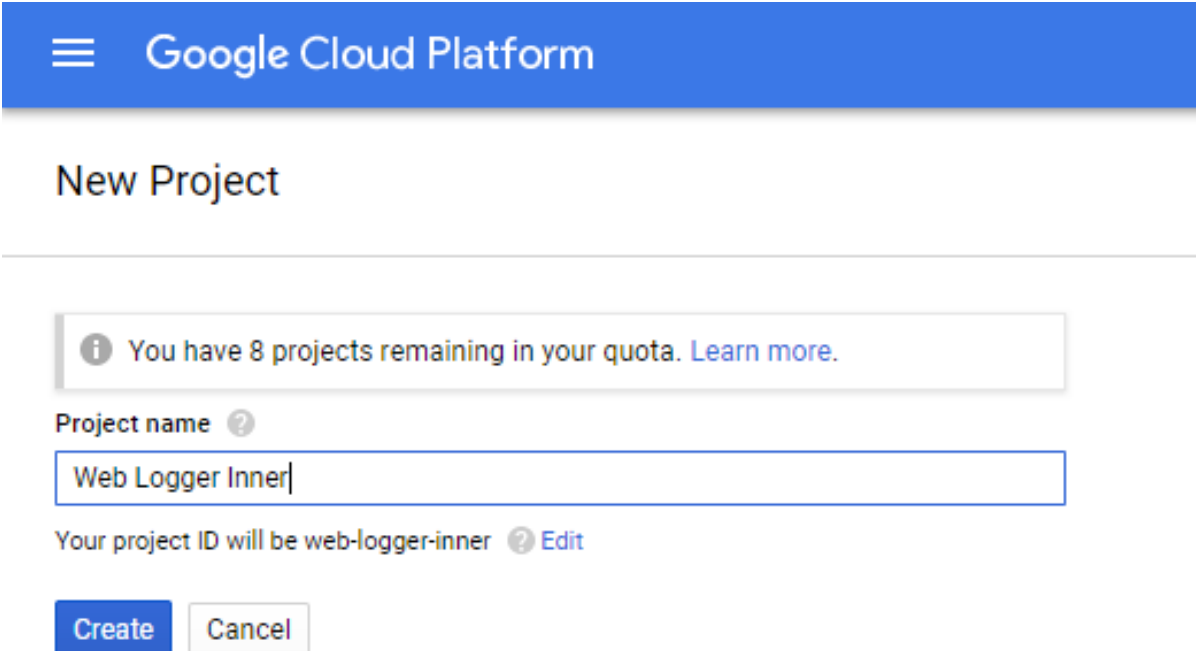
- 1) **Send a Request Token to the server** – This starts the authorisation process. We obtain our request token by first contacting Google and requesting new credentials to access the Google servers. These credentials include a client-specific ID (like a username), and a client-specific secret (like a password). We only must request these credentials once, and then we can access the Google servers indefinitely. The combination of these client-specific credentials acts as a request token, and we use that to initialise the authorisation process.

- 2) **The user submits their login credentials** – This step links a user to the Google servers to submit their login details. This step links the request token (Client specific) to the user whose details we will be given access too (User specific). In this way, we have linked the client and user together, so that only the right client will be given access to the right user information.
- 3) **The client receives an authorisation code** – After the user verification is successful, the client will receive a large complex string from the Google servers which is unique for each authorisation. This authorisation code acts as a symbol that user login was successful, but it also acts as a client-to-user identification token. In this way, we will be able to request the user's information later using this AUTH code.
- 4) **The client and Google server exchange authorisation codes for access tokens** – The client will submit their previously gained authorisation code, where the Google servers will verify its authenticity. If the AUTH code is cleared for use, then an access token will be returned. Whilst an AUTH code tells us that we have permission to access the user's data, the access token acts as the key to unlock and 'access' the data itself. Access tokens are short lived and must be frequently refreshed on the Google server for prolonged use of the user's data.
- 5) **The client submits the access token and receives the user data** – The Google server will return a JSON string containing all the data we requested, using the access token as a unique identifier, and returns it to the client.

---

## IMPLEMENTATION

When it comes to putting into practice such a system, there are a few steps to follow. Firstly, to gain those credentials as mentioned in the first step, we must define a new credential set in the Google developer console. Firstly, we must make a new 'project':



Google Cloud Platform

## New Project

*i* You have 8 projects remaining in your quota. [Learn more.](#)

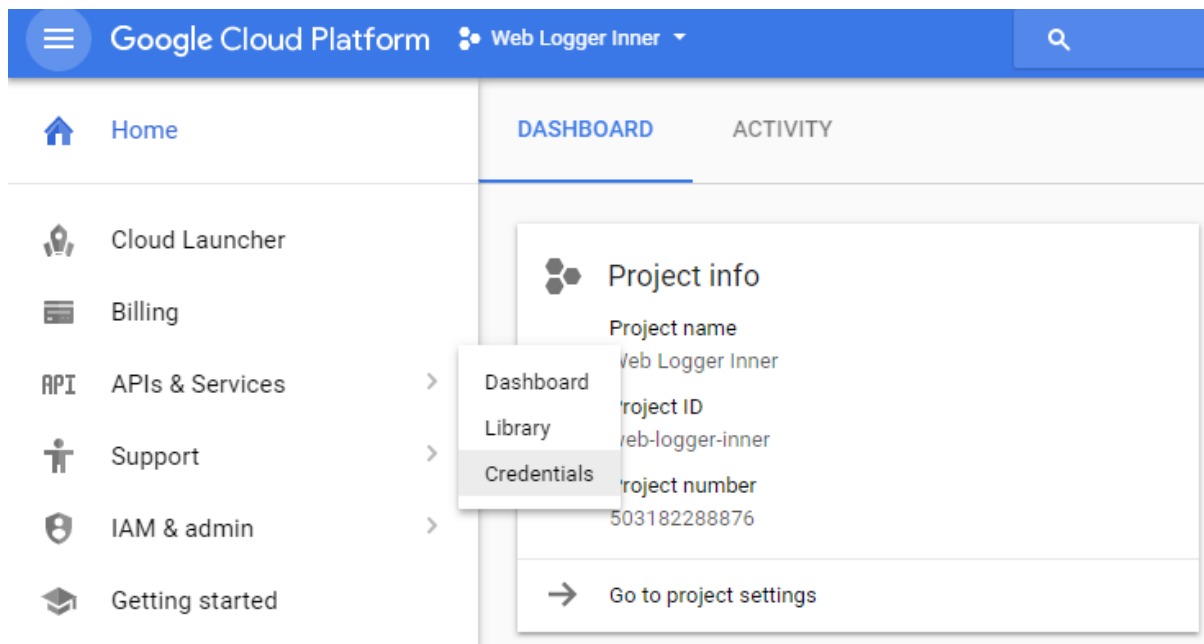
Project name *?*

Web Logger Inner

Your project ID will be web-logger-inner *?* [Edit](#)

Create Cancel

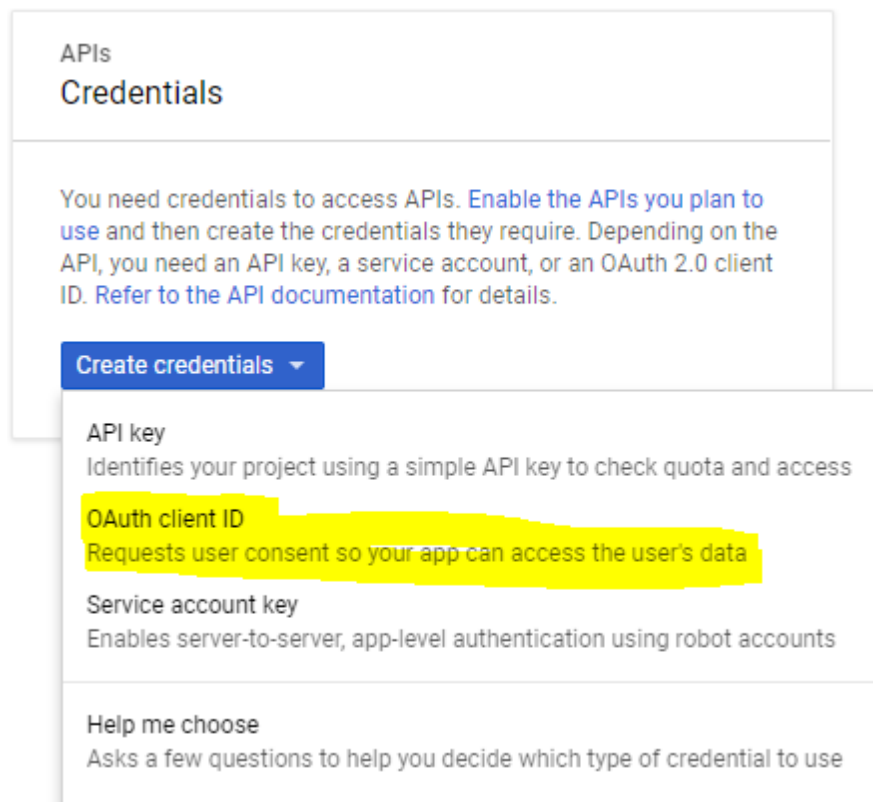
Next, at the dashboard, browse to the credential creation page:



Then, we need to define how our OAuth authentication screen will look like. This is the page that will be presented to the user to gain their login information:

The image shows the 'Credentials' page in the Google Cloud Platform console. The left sidebar is expanded to show 'APIs & Services' with sub-links for Dashboard, Library, and Credentials. The main area is titled 'Credentials' and has tabs for 'Credentials', 'OAuth consent screen', and 'Domain verification'. The 'OAuth consent screen' tab is active, showing a form to configure the OAuth consent screen. The form includes fields for 'Email address' (williamhall1324@gmail.com), 'Product name shown to users' (Product name), 'Homepage URL' (Optional), 'Product logo URL' (Optional), 'Privacy policy URL' (Optional), and 'Terms of service URL' (Optional). There is also a section for the product logo with a placeholder image and a note about the max size (120x120 px). A 'Save' button and a 'Cancel' button are at the bottom. On the right side, there is an illustration of a laptop and a smartphone displaying the consent screen, along with explanatory text about the consent screen and the requirement for an email address and product name.

After clicking save, we can now get our credentials, starting by requesting an OAuth Client ID (This is our 'username'):



Then, we can define a login name, and a call-back URL:

Application type

- ☒ Web application
- ☐ Android [Learn more](#)
- ☐ Chrome App [Learn more](#)
- ☐ iOS [Learn more](#)
- ☐ PlayStation 4
- ☐ Other

Name

Web Login 1

Restrictions

Enter JavaScript origins, redirect URIs, or both

**Authorized JavaScript origins**  
For use with requests from a browser. This is the origin URI of the client application. It can't contain a wildcard (https://\*.example.com) or a path (https://example.com/subdir). If you're using a nonstandard port, you must include it in the origin URI.

https://www.example.com

**Authorized redirect URIs**  
For use with requests from a web server. This is the path in your application that users are redirected to after they have authenticated with Google. The path will be appended with the authorization code for access. Must have a protocol. Cannot contain URL fragments or relative paths. Cannot be a public IP address.

https://11wha.ashvillecomputing.co.uk/Project/googleAuth/Callback ×

https://www.example.com/oauth2callback

Create Cancel

A call-back URL is the page that Google will return to after the authentication has occurred. They pass the authorisation code as a 'GET' variable on the end of this URL.

Now, by clicking create, we can now view our ClientID (username) and Client Secret (Password):

## OAuth client

Here is your client ID

503182288876-9hb1b3vrak64un9633hr9cc4nveba1p9.apps.googleusercontent.com 

Here is your client secret

Now, we have the login details, we need a way to use them. We could just use the pre-build Google PHP and JavaScript tools, which have been written by Google to access the OAuth flow. However, these pre-written snippets require a large amount of overhead and are more complex than required to simply gain authentication. The snippets by Google are built for gaining access to user's Google drives, giving read/write permissions or granting a persistent authorisation.

That is why I will create my own authentication code to simplify the process tremendously (See Appendix A > Web App > Tools, Resources and Configuration files > ../Project/googleAuth/googleLoginHelper.php). This login helper will abstract the process of building the URL which the user will visit to view the OAuth screen we setup earlier, gaining an authorisation token, passing it back to the Google server to turn into an access token and finally retrieving the user data from the Google servers using that access token.

---

### CREATING A RESTFUL API FOR DATA UPDATES AND QUERIES.

There are certain aspects of the PHP and MySQL backend that must be accessible from a public client without the need for logging in beforehand. Certain processes and functions must be executable on the server without first requiring a longwinded authorisation process. Examples of this include the scanning app, where it must be made easy to add a new piece of tracking data to the tracking logs database from any number of clients out in the field. Similarly, in the tracking app, it must be made easy to ascertain the status of the walk, so that the app may reflect the status and start time of the walk. To do this, we must create a set of HTTP pages that can serve us specific data from a query, but also accept data in the form of GET parameters when we are updating data on the server.

---

### DESIGN PRINCIPALS

Creating a web-based API means that we must first understand what an API is. An API or Application Programming Interface is defined as a 'set of functions or procedures that allow the creation of applications which access the features or data of another service'. From this we can deduce that an API is just a set of functions that help us to write other functions by abstracting or simplifying some procedures within it. A web-based API just means that those functions are accessible over the internet, and usually take on the form of database operations and tasks. To make a web-based API, we must first adhere to a set of rules that define how a database should be accessed over a local network. The acronym goes CRUD!

C – Create   R – Read   U – Update   D – Destroy

We should be able to create a new record, read existing records, update existing records and destroy records on the database, all over the local network. This would give us full, abstracted control of the database with the power to execute everyday operations, but without the power to execute too many high-level tasks such as deleting entire databases or servers.

---

## MAKING THE API ACCESSIBLE OVER THE INTERNET

Now, if we wanted to access the API online, we must take CRUD one step further by using a set of server-defined operations to provide CRUD over the internet. To create a new record, we could use a POST operation. To read a record, a GET operation. Updating would be a PUT operation, and finally a deletion would be a DELETE operation. The action of accessing a database over the internet using the hypertext transfer protocol and POST, GET, PUT & DELETE operations means that our API can be described as a RESTful API, where REST stands for the Representational State Transfer methodology of API design.

There also must be a consistent way of accessing the API, by visiting a common root folder on the server which will contain all the unique API pages. Since we will want to access many aspects of the database, including many tables, we will need many unique pages that will abstract access to specific information.

An example what URL might be constructed to access the RESTful API:



---

## STRUCTURED DATA RESPONSES

For all the API pages, it is important to return to the user information about the status of the API call, if it was successful and to gather any data that may be returned via the API. To ensure that any data returned is usable, we must present it to the user in a usable way. It would be a bad idea to simply print out the phrase 'OK' or 'Complete' in string form to the page. This would be an example of an unstructured response, where it is not made clear exactly what the data means/can be used for.

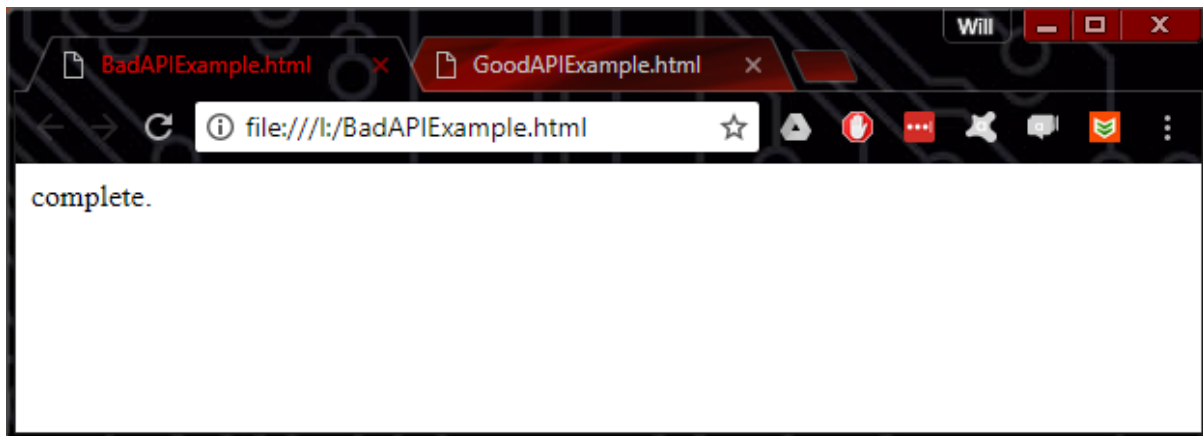
To maintain a consistent API, structured responses must be employed. A structured response is more than a simple string with a message/data within it. They instead come in the form of a JSON (JavaScript Object Notation) string which is a structured data set with variable names and data encoded directly into the string. This allows for easy parsing and processing by any client code. The following outlines these two examples in more detail:



## UNSTRUCTURED RESPONSE EXAMPLE

---

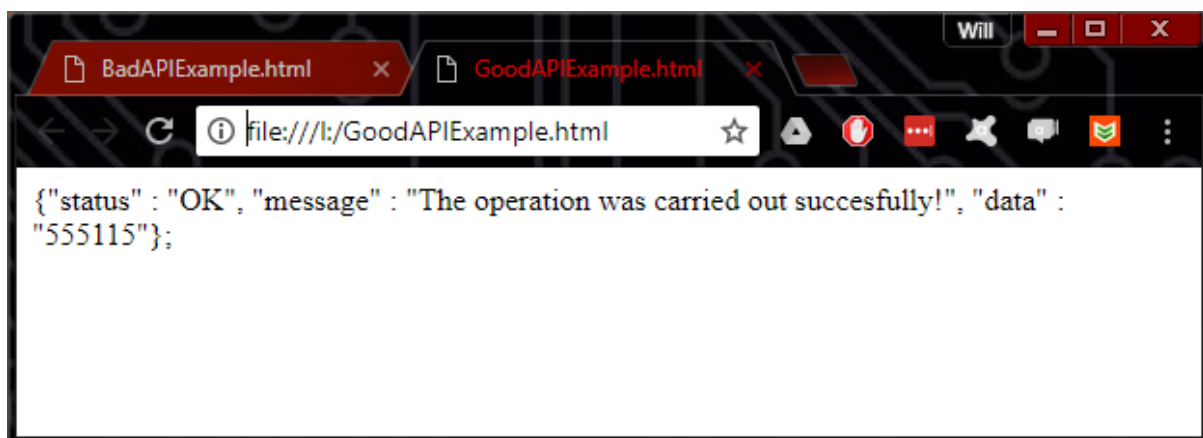
The following example illustrates a bad example of an API response. The data is a simple string without any form of structure:



## STRUCTURED RESPONSE EXAMPLE

---

In contrast, the following is an example of a good structured API response. The data is formatted in a way which is easily readable, and more importantly, more easily parsed/understood by any client code that may receive it:



## USE OF EXTERNAL APIS

---

As written above, an API is a set of functions that helps you to write applications.

To improve the speed and ease of writing the code for the web-app front-end UI, two Javascript APIs were used. These APIs are written in vanilla Javascript and use the built-in Javascript function creation tools to make functions for specific tasks/operations. We can then access these functions in any `<script>` tag on the page.

Similarly, to improve the ease of writing code for the PHP backend server, a PHP Emler API was used, to give the ability to send more complex emails straight from the server. Contrary to the Javascript APIs, PHP has no global function creation tools, so instead classes are made which encapsulate methods that make up the API.

---

## JQUERY

jQuery is a HTML DOM & Javascript abstraction API, meaning removes the complexity of operations such as changing the properties of DOM elements, handling complex HTML selectors, handling mouse and keyboard events and making AJAX calls. It removes some of the complexity of vanilla Javascript operations by making it easier to select and effect items on the screen. To use jQuery, you must include a link to it like how you might link a CSS file. In this link, we are fetching the jQuery script from an external Google CDN:

```
1. <link href = "cdn.google.com/jquery/jquery.js.min" />
```

---

## EASEL.JS

Easel.js is a HTML canvas API that abstracts some of the more difficult canvas operations, such as plotting ellipses, drawing circles of many colours and creating clickable areas on the canvas itself. It makes it much easier to update the canvas, make additions or subtractions, and quickly draw complex shapes without the need to do additional complex maths. Just like all Javascript APIs, Easel.js must be linked:

```
1. <link href = "cdn.smartjs.com/API/Easel/Easel.js.min" />
```

---

## PHPMAILER

PHPMailer is an email creation and sending tool for PHP. It abstracts the process of creating an email with all the correct headers and parameters, allows for direct HTML emails and provides the option to send directly from the server it is running on, or through a separate discreet SMTP server remotely. PHPMailer works as an object that encapsulates the methods of the API. To user it, you must first download the source code of the PHP mailer object, then you can instantiate a new PHP mailer object from within the code:

```
1. require_once '../Path/To/PHPMailer_Source_Code';
2.
3. $mailer = new PHPMailer();
4.
5. $mailer->setContent($textStringHere);
6.
7. $mailer->setSubjectLink($subjectStringHere);
8.
9. $mailer->send();
```

---

## API.QRSERVER.COM

api.arServer.com is a restful API for QR Code creation. By including some GET parameters, the API returns a PNG image of the QR code. This API will be used for the creation of all the QR codes throughout the project.

Below are the GET variables you can include in the request:

Variable	Description	Example
size	Define the pixel width of the QR code PNG image upon return.	350x350
data	A string of data to be encoded in the QR code	11wha

## DESCRIPTION OF DATA STRUCTURES

### DATABASE DESIGN

Database design was a crucial element to ensuring the smooth and efficient operation of the sponsored walk tracking app. Since all aspects of the web app and android app lie in a PHP and MySQL backend, it is important that any data is stored efficiently so that it may be accessed efficiently.

Firstly, the whole database must conform to first normal form – that there should be no non-key dependencies and hence no duplicate data. Secondly, there should be a relationship between tables to ensure that there are no inter-table duplicates. Since in MySQL, there is limited relational database design tools, I instead chose to use link tables to create pseudo relationships between tables.

Below is the basic database structure defined in DDL:

```
tblStudents(studentCode, forename, surname, yearGroup)
```

```
tblStaff(staffCode, forename, surname)
```

```
tblCheckpoints(checkpointID, fullName, distanceFromLast)
```

```
tblStudentStaffCheckpointLink(studentCode, staffCode, checkpointID, walkID, registrationTime)
```

```
tblWalks (walkID, startTime, status)
```

Now, let's explore the specific structure of each table:

#### TBLSTUDENTS

The purpose of this table is to store information about all the students who could be tracked through software.

NAME	TYPE	SIZE	PURPOSE	EXAMPLE
<b>STUDENTCODE</b>	VARCHAR	8	PRIMARY KEY – The unique identifier of every student.	11ch
<b>FORENAME</b>	STRING	128	The first name of each student.	Connor
<b>SURNAME</b>	STRING	128	The second name of each student.	Hartwell
<b>YEARGROUP</b>	INTEGER	2	The year group of each student.	13

---

#### TBLSTAFF

The purpose of this table is to store information about all the staff members who might wish to register students at each checkpoint.

NAME	TYPE	SIZE	PURPOSE	EXAMPLE
STAFFCODE	VARCHAR	8	PRIMARY KEY – The unique identifier of every staff member.	SWA
FORENAME	STRING	128	The first name of each staff member.	Stuart
SURNAME	STRING	128	The second name of each staff member.	Watson

---

#### TBLCHECKPOINTS

The purpose of this table is to store information about each checkpoint, including its name and distance values.

NAME	TYPE	SIZE	PURPOSE	EXAMPLE
CHECKPOINTID	INTEGER	2	PRIMARY KEY – The unique identifier of each checkpoint	1
FULLNAME	STRING	128	The full name of each checkpoint i.e. human readable names	Stuart
DISTANCEFROMLAST	INTEGER	128	The distance from the previous checkpoint to this checkpoint (i.e. Leg length).	3400

---

#### TBLSTUDENTSTAFFCHECKPOINTLINK

The purpose of this table is to act as a link table, storing records about each registration event.

NAME	TYPE	SIZE	PURPOSE	EXAMPLE
STUDENTCODE	INTEGER	2	COMPOUND KEY – The unique identifier of every student.	1
STAFFCODE	STRING	128	COMPOUND KEY – The unique identifier of every staff member.	Stuart
CHECKPOINTID	STRING	128	COMPOUND KEY - The unique identifier of each checkpoint	Watson
WALKID	INTEGER	2	COMPOUND KEY – The unique identifier of each walk that has occurred.	32
REGISTRATIONTIME	DATE-TIME		A Date-Time stamp outlining when the student finished.	2018-02-21 17:16:40

---

## SAMPLES OF PLANNED SQL

Throughout the entirety of the project, and with almost every PHP backend operation, some SQL must be created and run to fetch some data from the many tables as defined in the previous section.

The following is some examples of SQL that will be run to fetch more complex data sets within the project. (Note, this does not include the dynamically generated SQL as it has been mentioned and explored in the 'Description of algorithms' section):

---

### SELECTING ALL THE STUDENTS AND THEIR AFFILIATED TRACKING DATA

The following SQL selects the 'studentCode', 'locationID' and 'regTime', fields for each student within the table 'tblStuStaffLocLink', with an inner join used to fetch 'yearGroup' from the table 'tblStudents'. The use of the inner join enforces the 1<sup>st</sup> normal form of the database by ensuring there is no repeated data, using existing data from another table to fill in the blanks:

```
SELECT tblStuStaffLocLink.studentCode,
tblStuStaffLocLink.locationID, tblStuStaffLocLink.regTime,
tblStudents.yearGroup

FROM tblStuStaffLocLink

INNER JOIN tblStudents

ON tblStuStaffLocLink.studentCode = tblStudents.studentCode

ORDER BY tblStuStaffLocLink.studentCode
```

---

### SELECTING ALL THE STUDENTS AND SOME TRACKING DATA IF IT EXISTS

Like the first example, this SQL statement fetches all the students and their corresponding fields. However, instead of using an inner join, this statement uses a left join. This means that every student will be returned, regardless of whether they have any records in the table 'tblStuStaffLocLink', but if there is a link then that data about the registration event will be returned too:

```
SELECT tblStudents.studentCode, tblStudents.surname,
tblStudents.forename, tblStudents.yearGroup,
tblStuStaffLocLink.locationID, tblStuStaffLocLink.regTime

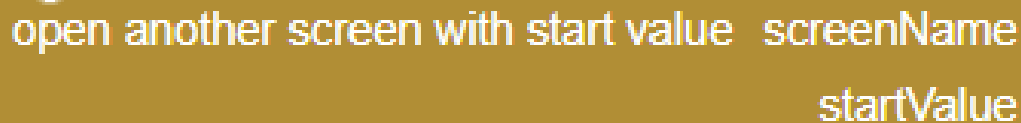
FROM tblStudents

LEFT JOIN tblStuStaffLocLink

ON tblStudents.studentCode = tblStuStaffLocLink.studentCode
```

## ANDROID APP INTERNAL DATA

An Android app works through a series of encapsulated pages, that are separate from any other page. This includes any data, which can only be shared by pages during the initialisation step. This means that, if we want to share data, we must keep it in a format which is universal to all pages. MIT app inventor uses 'lists' which are essentially just associative arrays. These arrays can be passed between pages in the 'open another screen with start value' block:

A yellow block from MIT App Inventor with the text 'open another screen with start value' and two input fields labeled 'screenName' and 'startValue'.

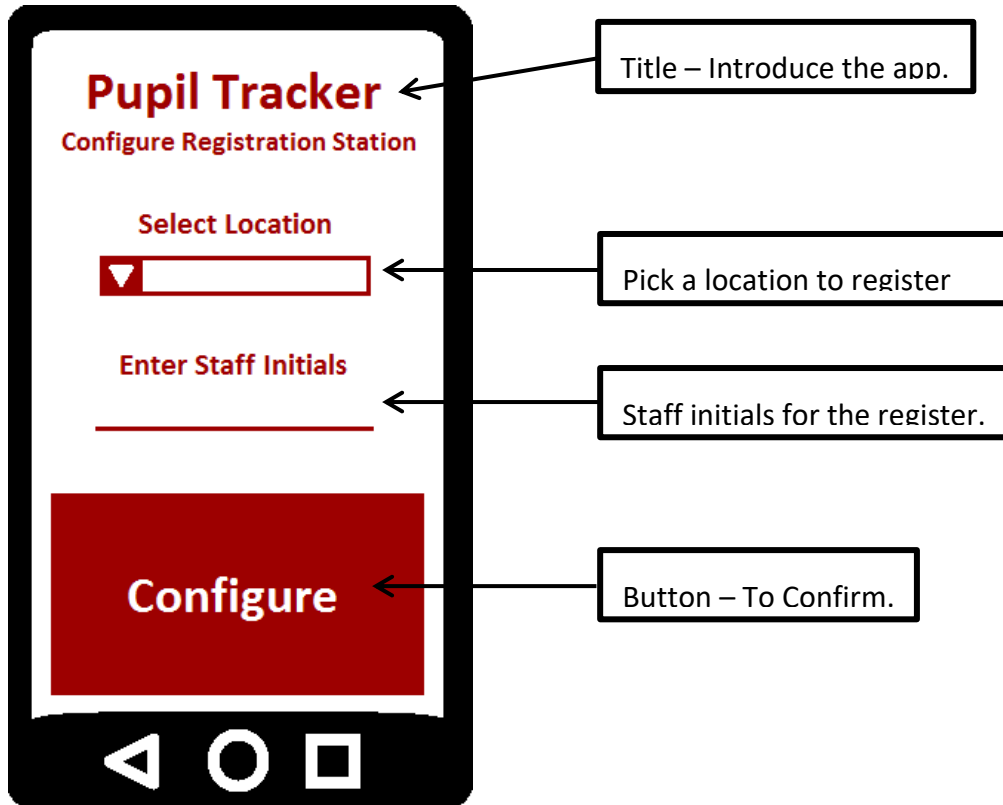
This data can then be accessed on the next page by using the 'get start value' block:

A yellow block from MIT App Inventor with the text 'get start value' and a small yellow tab on the left side.

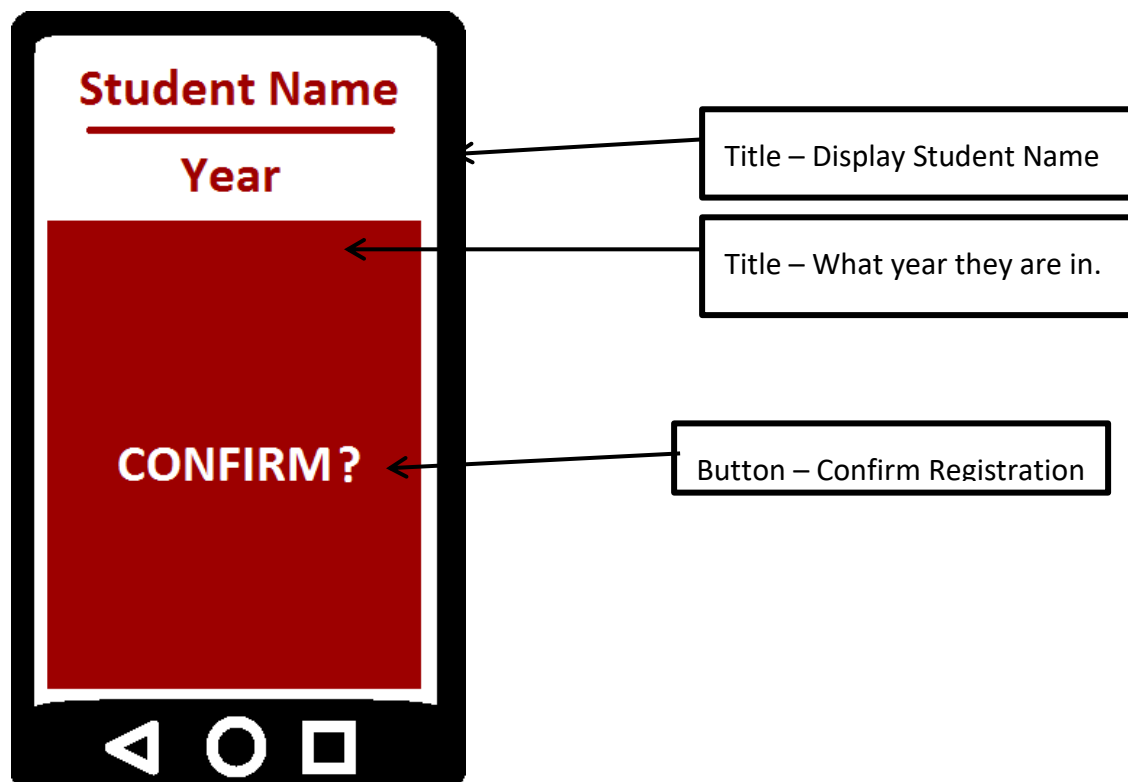
So, to ensure continuity throughout each page, a data structure that does not change must be passed amongst the pages so that data from previous pages can be viewed on every future page. The following data outlines the data that will need to be passed between the pages, and the index that data will have within the 'universal list' (Not that lists are not zero indexed):

INDEX	DATA	DESCRIPTION
1	Staff Code	Unique ID for each staff member.
2	Staff Full Name	Full name of each staff member.
3	Checkpoint Name	The full name of each checkpoint.
4	Checkpoint ID	The unique ID for each checkpoint.
5	WalkStatus	The status of the walk.
6	Notices	Any notices to present to the user.
7	Student Name	The full name of each student.
8	Student Code	Unique ID for each student.
9	Student YearGroup	The year group of the student.

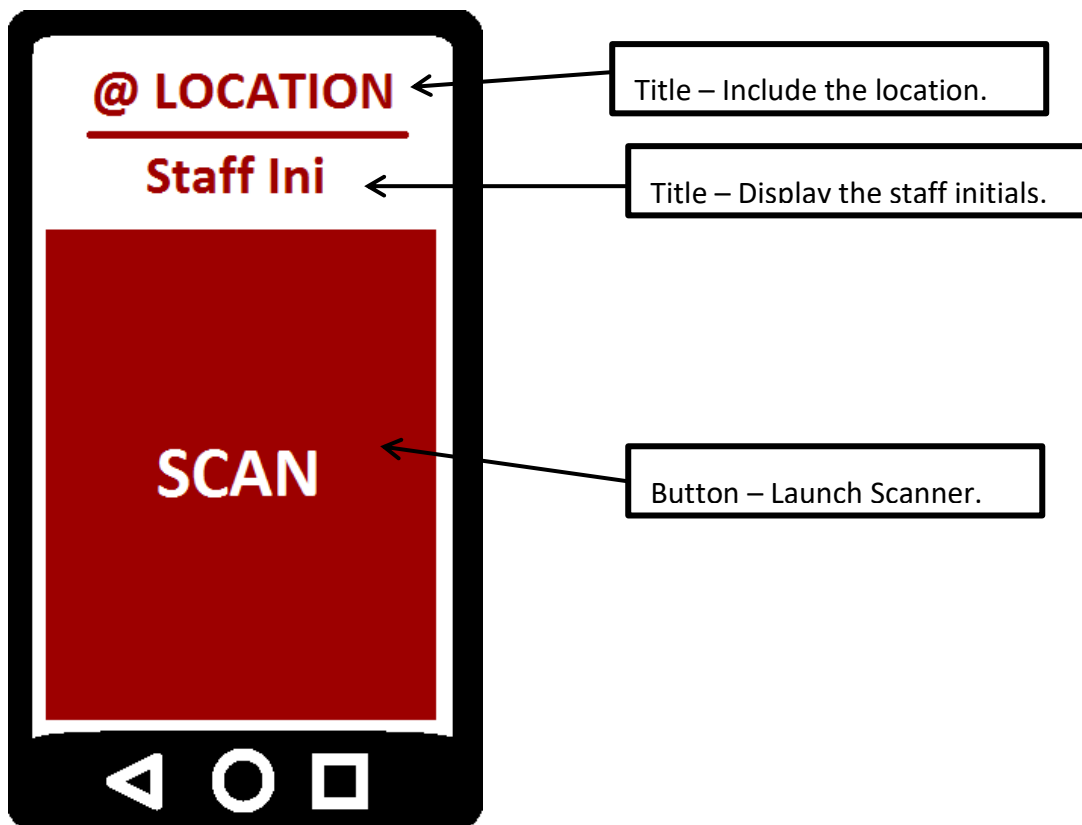
LANDING & CONFIG SCREEN



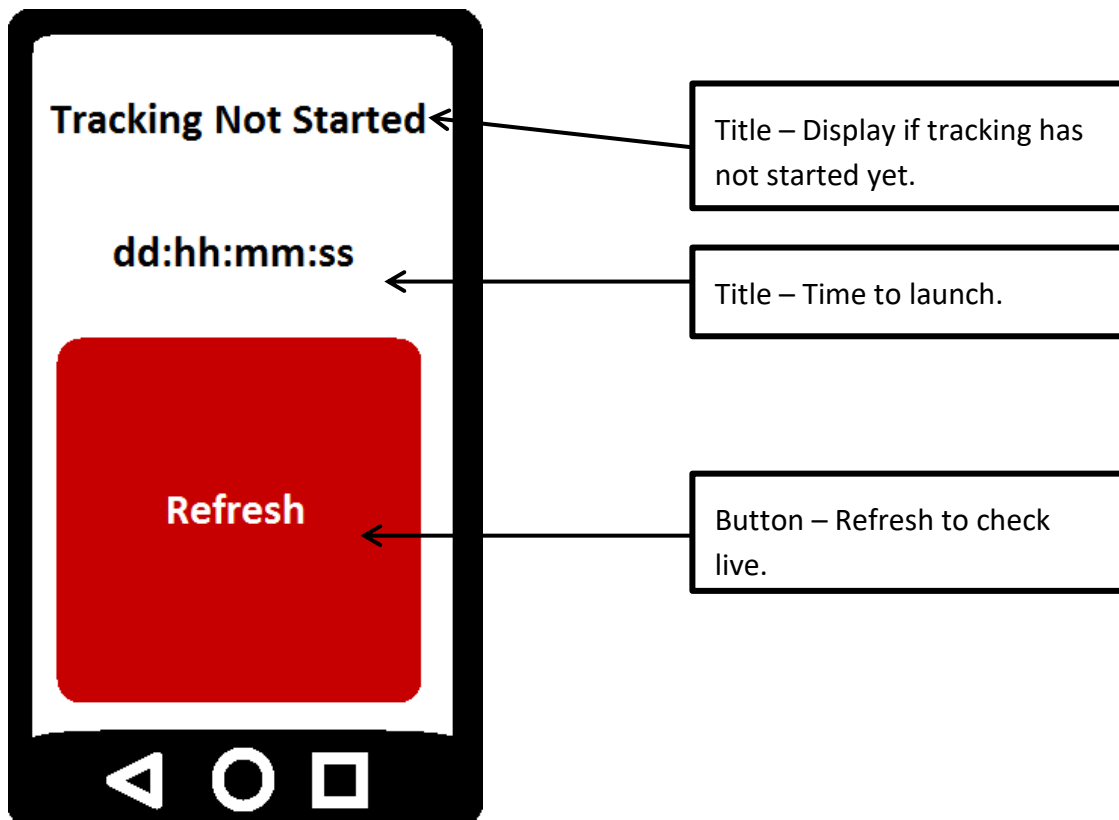
CONFIRM STUDENT REGISTRATION SCREEN



#### SCAN NEW STUDENT SCREEN

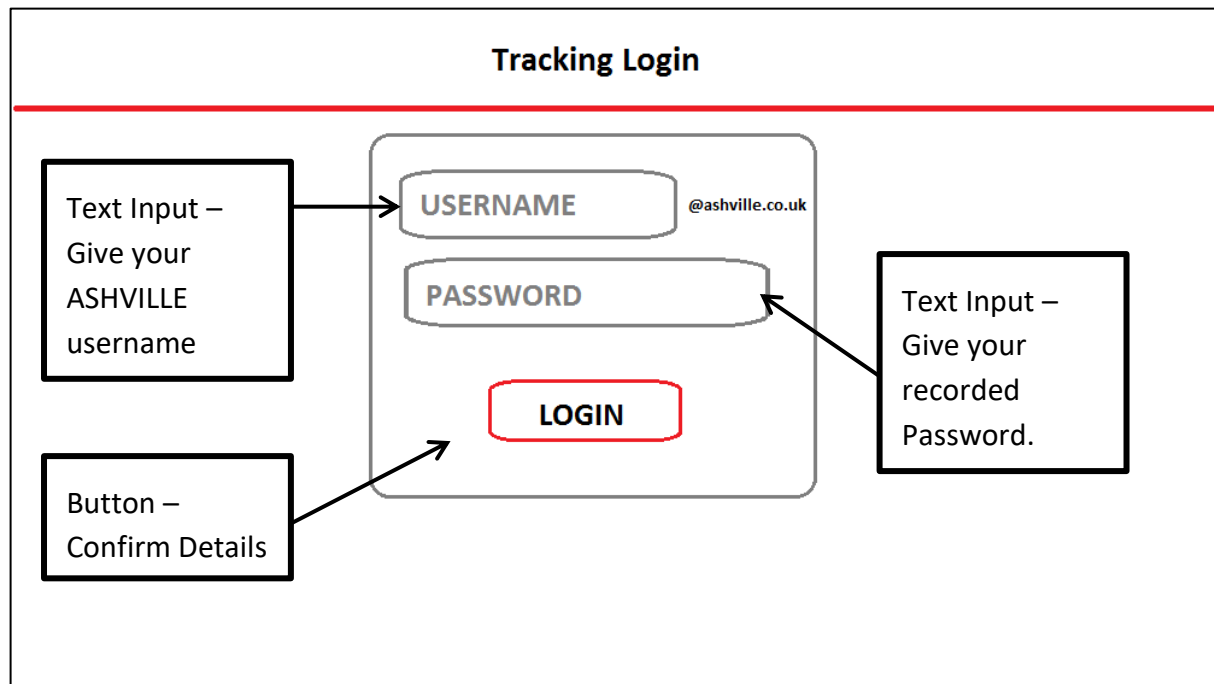


#### WAITING SCREEN

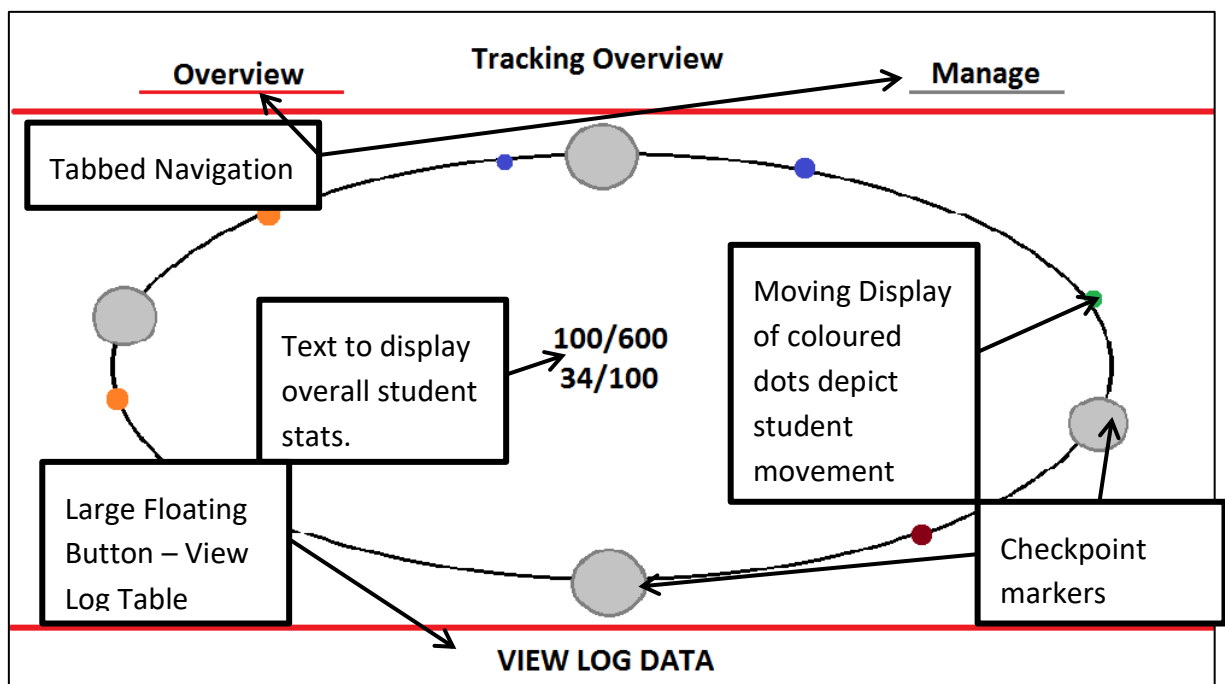




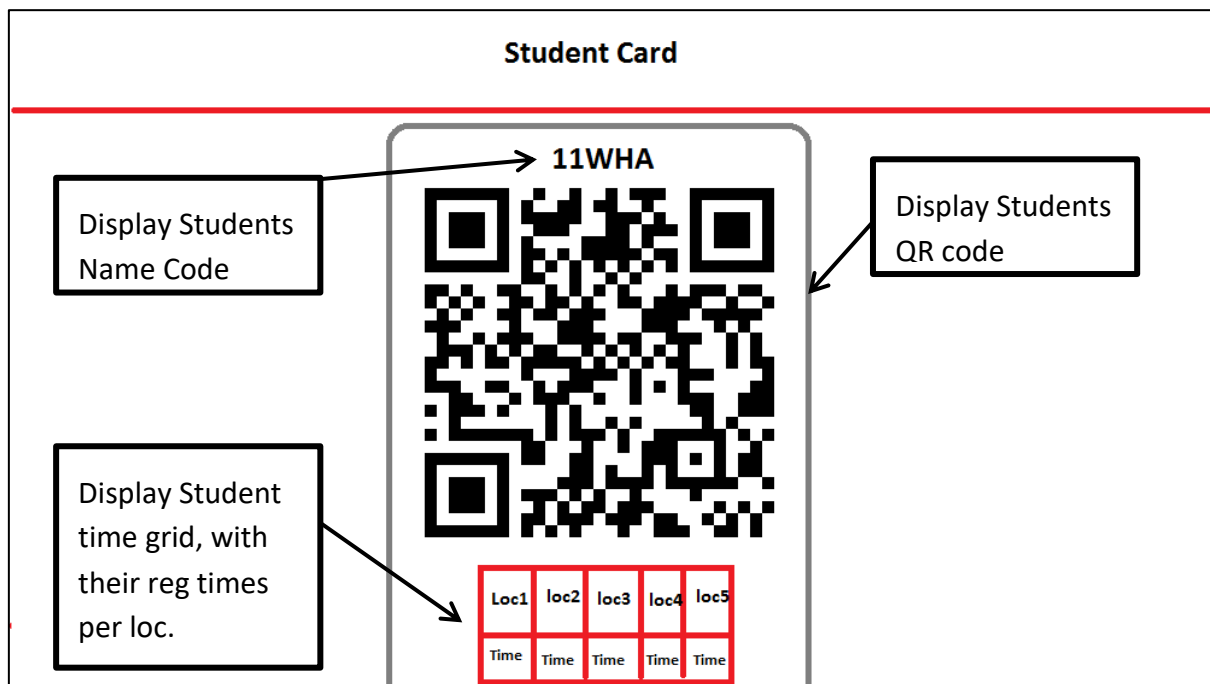
LOGIN SCREEN



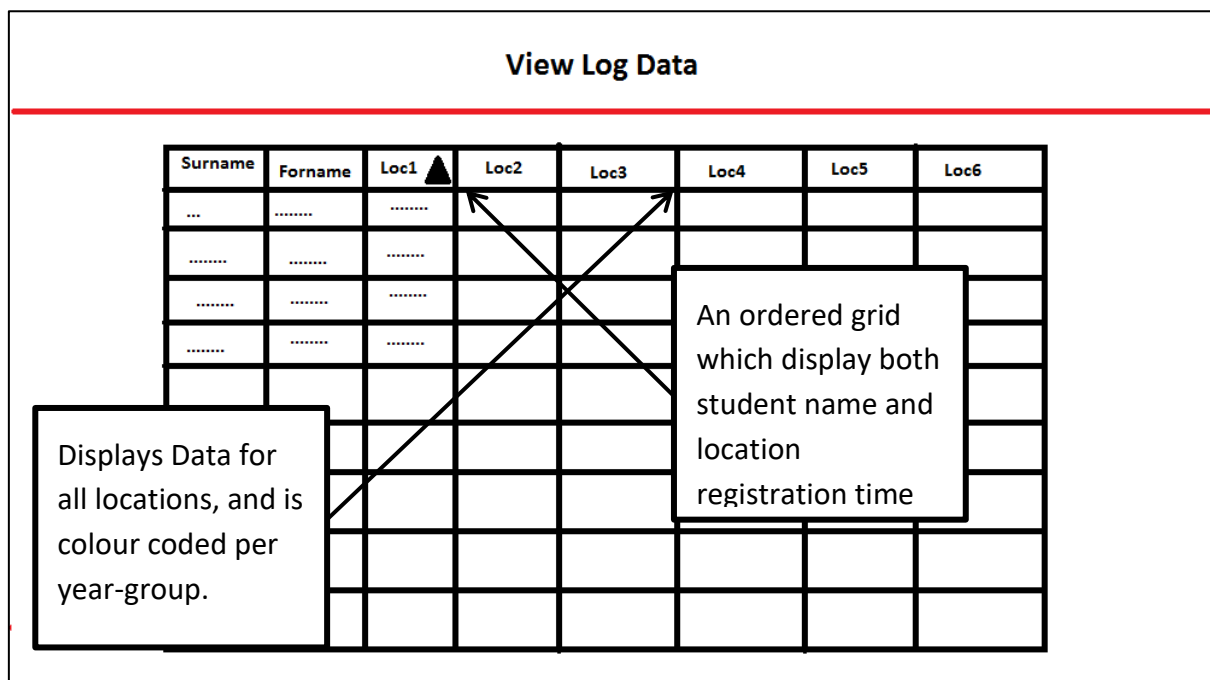
OVERVIEW SCREEN



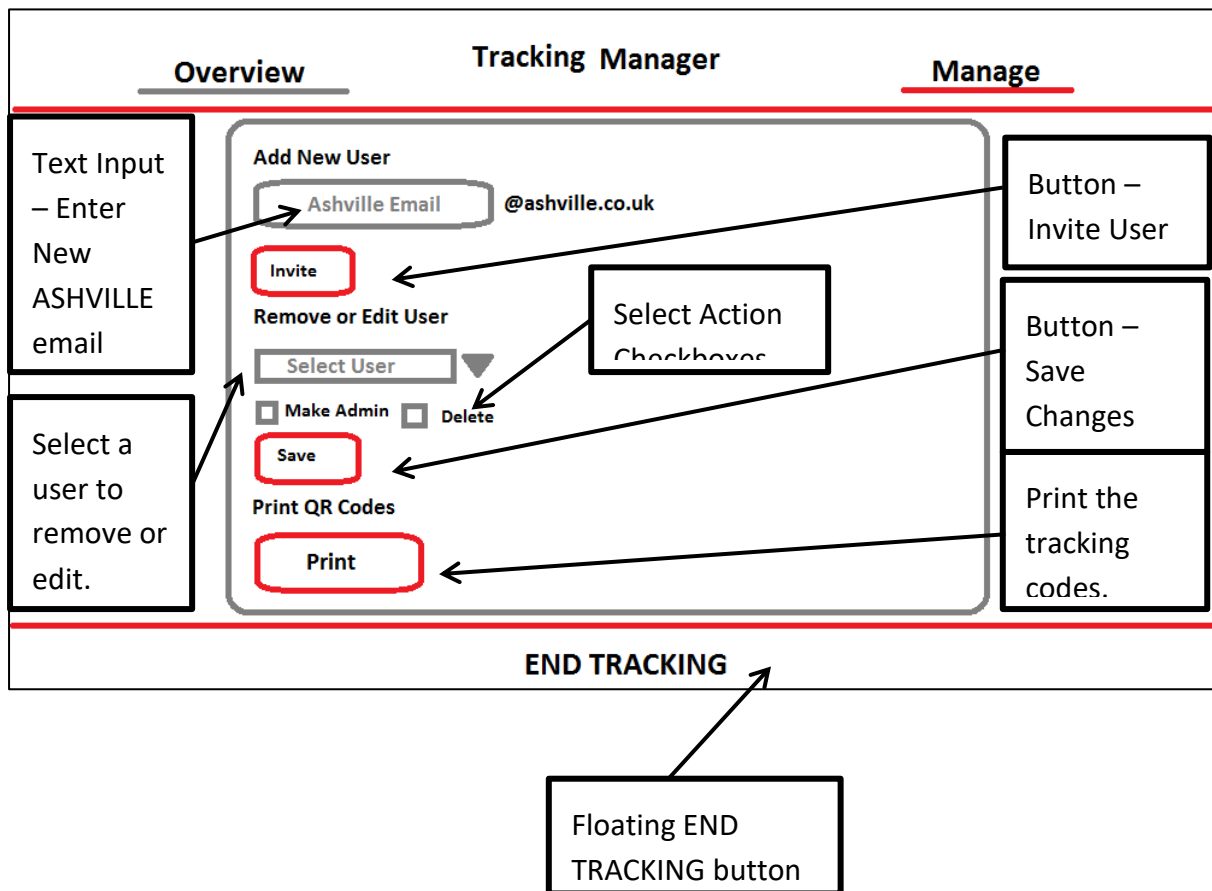
## STUDENT SCREEN



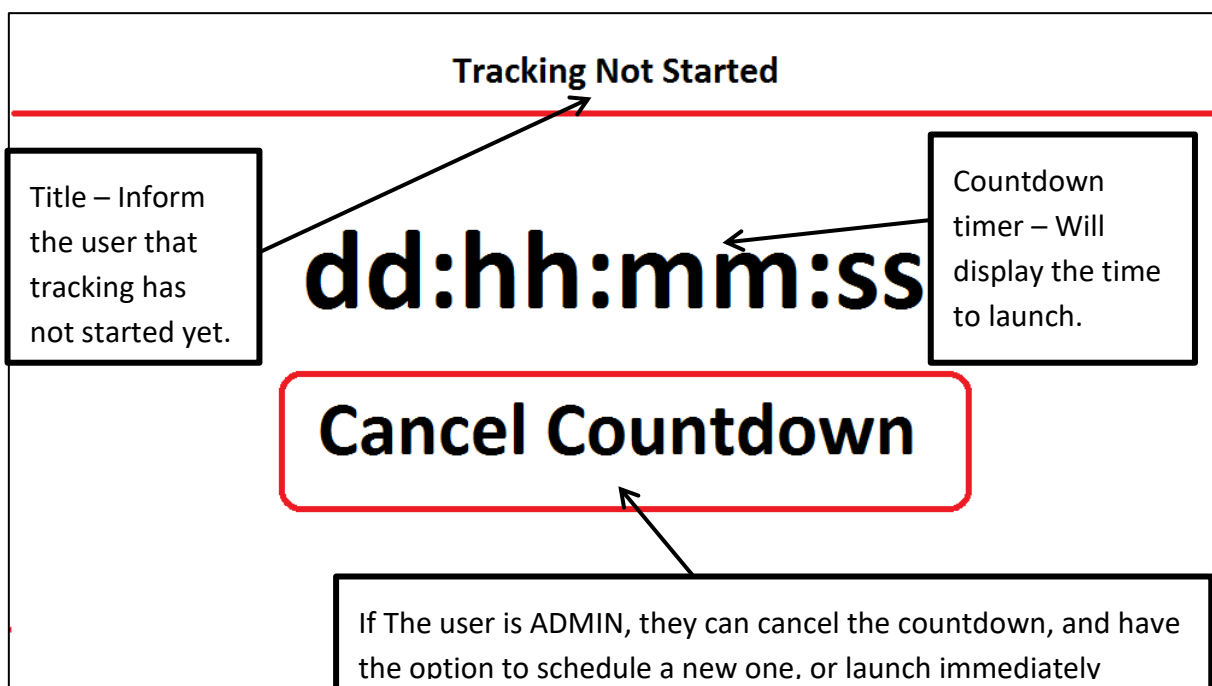
## LOG TABLE SCREEN



## MANAGEMENT SCREEN



## WAITING SCREEN



## SQL PREPARED STATEMENTS

Suppose we had an email input box on a HTML page. Any user could type in their email address, and then we would query a database to see if that email address already exists in a table. Say the user entered the email address of *11wha@ashville.co.uk*. The email string would be passed to a PHP script, and we might create an SQL string that looks like this:

```
1. SELECT COUNT(*) FROM tblEmails WHERE email = "11wha@ashville.co.uk";
```

When we run the SQL, we get a response containing the number of rows for which `email = "11wha@ashville.co.uk"` is true. The user could enter any value as the value of email. Even if our HTML page has lots of validation on what makes up an email address, someone could always bypass it. Say we entered an email address of *coolCats";"@ashville.com*. This email would be seen as valid by the HTML page, but crucially it contains a semicolon and some extra quotation marks. Whilst it may not seem so bad, it is important to note that the presence of a semicolon spells bad news for an SQL statement. A semicolon represents the delimiting factor of an SQL statement – a character that marks the end of one SQL string and the beginning of the next. So, our SQL string may look like this:

```
1. SELECT COUNT(*) FROM tblEmails WHERE email = "coolCats";
2. "@ashville.com";
```

When our SQL server runs this SQL string, it will see two SQL strings to run, since the semicolon marks the end of one string and the start of the next. So, firstly, we will return the number of rows for which `email = "coolCats"`, and then try run the now invalid statement of `@ashville.com`; Clearly, this is an invalid SQL command and our server will throw an error.

The worst outcome so far has been that we have thrown an SQL error, because we included the delimiting factor (";") in an input string. However, someone with a bad attitude could come along and really cause some damage to our server. Let's suppose that this bad actor enters the following string into our email address input box: *1@ashville.co.uk";DROP TABLE (\*)*; The HTML validation passes since we have all the parts of an email address we would expect. Now let's look at our SQL string(s):

```
1. SELECT COUNT(*) FROM tblEmails WHERE email = "1@ashville.co.uk";
2. DROP TABLE (*);
```

When we run our SQL, two things happen. Firstly, we get a count of all the rows where `email = "1@ashville.co.uk"`, which is not too bad. However, since our SQL server has noticed the delimiting factor, it now sees another SQL string to run. This string is `DROP TABLE (*)`; , a command which **deletes all the tables and their content in the database**. That would spell disaster! We would lose all our data and table structure and must rebuild from scratch. This could cost lots of money and time for big companies, but for the Sponsored Walk app, it could result in the loss of student tracking information, rendering the entire system broken.

This type of attack is called an SQL Injection Attack, and has been used to render entire websites useless, sometimes with many weeks of downtime. However, it became clear that an alternative method for creating SQL and running it on a server was required. It is too dangerous to simply splice user-entered strings into pre-written SQL strings on the server. Instead, the *prepared statement* was created. A prepared statement is one which is defined in the code and run on the SQL server *before* any user-entered values are included. Instead,

placeholders are included via the character “?” in the SQL string, and have parameters bound to them after the initial string has been run on the server. An example of a prepared statement may look like this:

```
1. SELECT COUNT(*) FROM tblEmails WHERE email = ?;
```

Now, when we run the SQL, the server identifies the “?”, and identifies that as a placeholder value. Now, when our bad actor comes along, and enters the string `1@ashville.co.uk";DROP TABLE (*);` the SQL server already knows to expect a string and not to look for any further SQL statements within the string – we told the server that the “?” represents a string and a string only. So, now we will be selecting all the rows where `email = "1@ashville.co.uk";DROP TABLE (*);"`.

In essence, the prepared statement runs and stores SQL on the server prior to the inclusion of any user-entered data. Any user-entered data is then included but is treated as a full string and a string only. Of course, it is possible to include integers and date-time objects too. So, to recap, the following is some PHP pseudocode to illustrate the process of running a prepared statement:















```
1. $statement = "SELECT COUNT(*) FROM tblEmails WHERE email = ?;";
2.
3. $SQLStatment = $databaseConnection->prepare($statement);
4.
5. $userData = $_POST['someDataFromHTML'];
6.
7. $SQLStatement->bindParamToPlaceholder($userData);
8.
9. $SQLStatement->execute();
```

---

## UAC (USER ACCOUNT CONTROL)

For the web-app, it is important to ensure that only those with the correct permissions have access to the correct data for their permission level. For example, it is important that those without proper permission levels have no access to the overriding functions of the database, such as changing the state of the walk, erasing tracking data or changing settings related to the walk. There will be two permission levels: High and Low. To enforce the permission levels, each user will have an associated level of permission stored alongside their username and password. Then, when each page is created on the server, certain areas/pages and scripts will be dynamically loaded into the final HTML document before being sent over HTTP to the browser.

For example, for a low-level user, the ‘end walk button’ will not be included. Similarly, for a high-level user, the scripts for updating other user’s permissions will be included. The following table gives information of the proposed areas of access/permissions of each type of user:

ABILITY	LOW-LEVEL	HIGH-LEVEL
VIEW TRACKING DATA.		
PRINT QR CODES.		
INVITE NEW USERS.		
UPDATE USER PERMISSIONS.		
REMOVE USERS.		
CHANGE THE WALK STATE.		
DEFINE A START TIME FOR WALK LAUNCH.		

#### TRACKING DATA RELIABILITY

The ultimate purpose of the Sponsored Walk App is to track and present student tracking data. It must be possible to search for individual students and view their progress. However, even with the best data-processing possible, it is impossible to track students if they misuse or share QR tracking codes with other students. In this way, students could have mismatched data, and tracking of individual students would be unreliable. Hence, it must be made difficult to gain access to other students QR codes so that students cannot impersonate each other, and there must be a final step of human verification before a registration occurs at each checkpoint.

Below are the two major points explored:

- 1) **QR Code Access Moderation** – In order to make it more difficult to access the QR codes of other students, a Google account login will be used. The Google login is associated with each school account and incorporates password updates and account deletion/changes. In this way, we can make it more difficult to view and save other students QR codes, since a password is required to view the QR code.
- 2) **Human Verification** – Even if the students were to share passwords to their Google accounts or distribute fraudulent QR codes, there must be one final level of verification to eliminate the ability to falsely register at a location. Whilst a computer may be tricked by a misused QR code, a human can more accurately determine if the student registering matches the student QR code. To accomplish this, each checkpoint terminal (Android tracking app) will have a two-step registration process:
  - a. The registrar will scan the students QR code. This will display all the information about the QR code – Including the student name, year group and DOB.
  - b. The registrar can choose to confirm the student registration by clicking a confirmation button on screen. If needed, the registrar can challenge the prospective student using the presented data associated with the scanned QR code.

---

## STUDENT DATA ACCESSIBILITY

To uphold the upmost data security, and for safeguarding issues, it is critical to ensure that only those people who are authorised can access the personal data of students. This means that any portion of any app that may provide student data is done so only after sufficient authorisation. Below is a description of measures takes to improve data security:

### Student Data: Points of Risk

### Measure

*Name & Location - Web app tracking data table*

Secure user account system and full control of user accounts and permissions from an administrator perspective.

*Name & Location - Student QR codes*

Google oAuth verification of each student before access is granted.

*Name, Year Group, DOB – Confirmation stage of the android tracking app.*

Google oAuth verification of each staff member, and requirement for valid Ashville email address before access is granted.

## TESTING

### TEST PLAN

## EVALUATION

### END-USER FEEDBACK.

## APPENDIX A - CODE

### WEB APP

#### FRONT END PAGES

The following pages will be presented to users.

---

##### ../PROJECT/LOGIN.PHP

This page handles the login of every user who wishes to access the tracking software. It used PHP sessions to track who is logged in and includes validation to ensure that sessions are discreet.

```
1. <?php
2. //Start the session
3. session_start();
4.
5. //Enable Error reporting for debug
6. error_reporting( E_ALL );
7. ini_set( 'display_errors', 1 );
8.
9. //Defining the secure access parameter means that any config or functional files
10. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
11. define( 'secureAccessParameter', true );
12. require '../Project/Tools/config.php';
13. require '../Project/Tools/walkFunctions.php';
14.
15. //Make a new database connection.
16. $conn = dbConnect();
17.
18. //Reset the user information string.
19. $msg = "";
20.
21. //If we are requesting a logout, or if we are coming to login from the registration page, then first
22. //destroy the session and any variables to ensure that any user that might be logged in do not remain
23. //logged in. This is a major security action, to prevent shared sessions.
24. if ( isset( $_GET[ 'logout' ] ) or isset( $_GET[ 'fromReg' ] ) ) {
25.     unset( $_SESSION[ 'login' ] );
26.     unset( $_SESSION[ 'level' ] );
```



```

27.     session_destroy();
28.
29.     //Update the user message.
30.     if ( isset( $_GET[ 'fromReg' ] ) ) {
31.         $msg = "Thank You For Registering - Now Login!";
32.     } else {
33.         $msg = "YOU SUCCESSFULLY LOGGED OUT";
34.     }
35. }
36.
37. //If the user clicked the submit button, then begin the verification flow.
38. if ( isset( $_POST[ 'subBtn' ] ) ) {
39.
40.     //Prepare a statement to select the user from the admin users table.
41.     $sqlQuery = $conn->prepare( "SELECT * FROM tblAdminUsers WHERE uName = ?" );
42.     $sqlQuery->bind_param( "s", $uName );
43.
44.     //Bind params and execute.
45.     $uName = strtoupper( $_POST[ 'uName' ] );
46.     $sqlQuery->execute();
47.
48.     //Get the SQL return object.
49.     $result = $sqlQuery->get_result();
50.
51.     //Check if the user exists in the table
52.     if ( $result->num_rows > 0 ) {
53.
54.         //Convert the SQL return objec to an accoc. array.
55.         $dataRow = $result->fetch_assoc();
56.
57.         //Check if the password matches through a hashing algoirithm.
58.         if ( password_verify( $_POST[ 'uPass' ], $dataRow[ 'pass' ] ) ) {
59.
60.             //Is succesfull, update the session variables and regenerate the
61.             //session id to prevent session hijacking.
62.             $_SESSION[ 'login' ] = $dataRow[ 'uName' ];
63.             $_SESSION[ 'level' ] = $dataRow[ 'level' ];
64.             session_regenerate_id();
65.
66.             //Else, the password was incorrect. Update the message.
67.         } else {
68.             $msg = "INCORRECT PASSWORD.";

```

```

69.     }
70.     //Else, the username was incorrect. Update the message.
71. } else {
72.     $msg = "INCORRECT USERNAME.";
73. }
74.
75. //Close connections.
76. $sqlQuery->close();
77. dbClose();
78.
79. }
80.
81. //If the user is already logged in, then redirect.
82. if ( isset( $_SESSION[ "login" ] ) ) {
83.     header( 'Location:http://11wha.ashvillecomputing.co.uk/Project/Overview' );
84.     exit();
85. }
86.
87. ?>
88.
89. <!doctype html>
90. <html>
91.
92. <head>
93.     <meta charset="utf-8">
94.     <title>Login</title>
95.     <link rel="stylesheet" type="text/css" href="main.css">
96.     <link rel="icon" href="../Project/Resources/favicon.png" type="image/x-icon"/>
97.     <link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro" rel="stylesheet">
98.     <meta name="viewport" content="width=device-width, initial-scale=1.0">
99.     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
100.    <script>
101.        $( document ).ready( function () {
102.
103.            //Fade in the main body division.
104.            $( "#bodyDiv" ).fadeIn( 500 );
105.
106.            //Array to store the text input DOM references.
107.            var textDom = [];
108.            textDom[ 0 ] = $( "#uNameText" );
109.            textDom[ 1 ] = $( "#uPassText" );
110.

```

```

111.         //If the user clicks submit, run checks.
112.         $( "#submitBtn" ).click( function () {
113.
114.             var valid = true;
115.
116.             //For each text input, check if they entered some text. Update CSS to reflect
117.             //the state of each box (i.e. Red for invalid, Black for valid).
118.             for ( var i = 0; i < 2; i++ ) {
119.                 if ( textDom[ i ].val().length == 0 ) {
120.
121.                     textDom[ i ].css( "borderColor", "#d81015" );
122.                     valid = false;
123.                 } else {
124.                     textDom[ i ].css( "borderColor", "black" );
125.                 }
126.             }
127.
128.             //If checkes passed, submit the form. This refreshes the page.
129.             if ( valid ) {
130.                 $( "#bodyDiv" ).fadeOut( 150 );
131.                 setTimeout(
132.                     function () {
133.                         $( '#loginForm' ).submit();
134.                     }, 150 );
135.             }
136.         } );
137.     } );
138. </script>
139.</head>
140.
141.<header>
142.    <h1 style="padding-top: 32px;">Tracker Login!</h1>
143.</header>
144.
145.<body>
146.    <div id="bodyDiv">
147.        <div class="loginWrap">
148.            <form action="../../Project/Login.php" method="post" id="loginForm">
149.                <div class="sbsWrap">
150.                    <input type="text" name="uName" id="uNameText" class="inText short" placeholder="USERNAME" value="<?php if(isset($_GET['fromReg']))
151.                        ) echo $_GET['fromReg']; ?>">
                    <p>@ashville.co.uk</p>

```

```

152.         </div>
153.         <input type="password" name="uPass" id="uPassText" class="inText" placeholder="PASSWORD">
154.         <input type="hidden" name="subBtn">
155.     </form>
156.     <div class="btn" id="submitBtn">Login</div>
157.     <p class="msg">
158.         <?php echo $msg;?>
159.     </p>
160. </div>
161. </div>
162.</body>

```

---

## ../PROJECT/OVERVIEW.PHP

This page contains the main overview of the sponsored walk. The content of the page is dynamic – meaning it will change depending on the current state of the walk. When the walk is active, the page quietly refreshes every 10 seconds to produce a ‘live’ view.

```

1. <?php
2. //Start the session
3. session_start();
4.
5. //Enable Error reporting for debug
6. error_reporting( E_ALL );
7. ini_set( 'display_errors', 1 );
8.
9. //Defining the secure access parameter means that any config or functional files
10. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
11. define( 'secureAccessParameter', true );
12. require '../Project/Tools/config.php';
13. require '../Project/Tools/walkFunctions.php';
14.
15. //If the user has not logged in, then the session variable of 'login' will not be set,
16. //so redirect to the Login page.
17. if ( !isset( $_SESSION[ 'login' ] ) ) {
18.     header( 'Location:http://11wha.ashvillecomputing.co.uk/Project/Login' );
19.     exit();
20. }
21.
22. //Call the get walk status function. Assign a variable to its returned value.
23. $walkStatus;
24. $responseObject = json_decode( getWalkStatus(), 1 );

```

```

25. if ( $responseObject[ 'status' ] == "OK" )$walkStatus = $responseObject[ 'walkStatus' ];
26. ?>
27.
28. <!DOCTYPE html">
29. <html xmlns="http://www.w3.org/1999/xhtml">
30.
31. <head>
32.     <title>Overview</title>
33.     <link rel="stylesheet" type="text/css" href="main.css">
34.     <link rel="icon" href="../Project/Resources/favicon.png" type="image/x-icon"/>
35.     <link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro" rel="stylesheet">
36.     <meta name="viewport" content="width=device-width, initial-scale=1.0">
37.     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
38.
39.     <!-- Including the EASEL.JS API - this abstracts such processes as adding clickable areas and
40. complex graphics on the canvas element. -->
41.     <script src="https://code.createjs.com/easeljs-0.8.2.min.js"></script>
42.
43.     <script>
44.         $( document )
45.             .ready( function () {
46.
47.                 //When the div with the 'toManage' ID is clicked, redirect to the manage page.
48.                 $( "#toManage" )
49.                     .click( function () {
50.                         $( "#bodyDiv" )
51.                             .fadeOut( 150 );
52.                         setTimeout(
53.                             function () {
54.                                 window.location.href = "../Project/Manage";
55.                             }, 150 );
56.                     } );
57.
58.                 //When the div with the 'btnLogOut' ID is clicked, redirect to the Login/Logout page.
59.                 $( "#btnLogOut" )
60.                     .click( function () {
61.                         $( "#bodyDiv" )
62.                             .fadeOut( 150 );
63.                         setTimeout(
64.                             function () {
65.                                 window.location.href = "../Project/Login?logOut";
66.                             }, 150 );

```

```

67.         } );
68.     } );
69. </script>
70.
71. <!-- If the user is low level and the walk is status 'zero' i.e. not active,
72. then load JavaScript to show just the main div. -->
73. <?php if($_SESSION['level'] == 0 && $walkStatus == 0) {?>
74. <script>
75.     $( document )
76.     .ready( function () {
77.         $( "#bodyDiv" ).fadeIn( 500 );
78.     } );
79. </script>
80. <?php } ?>
81.
82. <!-- If the user is high level and the walk is status 'zero' i.e. not active,
83. then load JavaScript to show the scheduling screen -->
84. <?php if($_SESSION['level'] == 1 && $walkStatus == 0) {?>
85. <script>
86.     $( document )
87.     .ready( function () {
88.
89.         $( "#bodyDiv" )
90.         .fadeIn( 150 );
91.
92.         //When the scheduling button is clicked, run function.
93.         $( "#btnScheduleWalk" )
94.         .click( function () {
95.
96.             //Get the start time value entered by the user.
97.             var startTrackTime = $( "#startTrackTime" )
98.             .val();
99.
100.            //Create a comparable date object with the start time value.
101.            var dateStart = new Date( startTrackTime );
102.
103.            //Set the minimum date to be entered to now.
104.            var dateMin = new Date();
105.
106.            //Reset the info tag so that we don't show multiple errors in one tag.
107.            $( "#info" )
108.            .html( " " );

```

```

109.
110.         //If chosen start time is in the future, then proceed.
111.         if ( startTrackTime != "" && dateStart > dateMin ) {
112.
113.             //Define AJAX call to a PHP handler page. This page updates the walks current status for all clients.
114.             //Here, we are setting the status to one, with a time in the future. Since we have to wait for the countdown, we
115.             //now enter the Waiting state.
116.             $.ajax( {
117.                 url: "../Project/AJAX/updateWalkStatus",
118.                 method: "POST",
119.                 data: {
120.                     status: 1,
121.                     startTime: startTrackTime
122.                 },
123.                 success: function ( result ) {
124.                     //Parse the returned JSON response.
125.                     var responseObject = JSON.parse( result );
126.
127.                     //If the state change was made succesfully, then reload page.
128.                     if ( responseObject.status == "OK" ) {
129.                         $( "#bodyDiv" )
130.                             .fadeOut( 150 );
131.                         setTimeout(
132.                             function () {
133.                                 window.location.href = "../Project/Overview";
134.                             }, 150 );
135.                     } else {
136.                         $( "#info" )
137.                             .html( "Error: Could not launch walk. Please refresh and try again." );
138.                     }
139.                 }
140.             } );
141.         } else {
142.             //Display the error - the time they chose was not in the future!
143.             $( "#info" )
144.                 .html( "Please select a valid start time! Ensure it is a date and time in the future..." );
145.         }
146.     } );
147.
148.     //Launch the walk immidiately when this button is clicked.
149.     $( "#btnLaunchWalk" )
150.         .click( function () {

```

```

151.
152.         //Reset the info tag so that we don't show multiple errors in one tag.
153.         $( "#info" )
154.             .html( " " );
155.
156.         //Define AJAX call to a PHP handler page. This page updates the walks current status for all clients.
157.         //Here, we are setting the status to one, with the current time. Since we don't have any time to wait, we
158.         // now enter the Active State.
159.         $.ajax( {
160.             url: "../Project/AJAX/updateWalkStatus",
161.             method: "POST",
162.             data: {
163.                 status: 1,
164.                 startTime: new Date()
165.                     .toISOString()
166.             },
167.             success: function ( result ) {
168.                 //Parse the returned JSON response.
169.                 var responseObject = JSON.parse( result );
170.
171.                 //If the state change was made succesfully, then reload page.
172.                 if ( responseObject.status == "OK" ) {
173.
174.                     $( "#bodyDiv" )
175.                         .fadeOut( 150 );
176.                     setTimeout(
177.                         function () {
178.                             window.location.href = "../Project/Overview";
179.                         }, 150 );
180.                 } else {
181.                     $( "#info" )
182.                         .html( "Error: Could not launch walk. Please refresh and try again." );
183.                 }
184.             }
185.         } );
186.     } );
187. } );
188. </script>
189.
190. <!-- If user is high level, and the walk is in Waiting State, then load the Javascript to show the countdown controls -->
191. <?php } if($_SESSION['level'] == 1 && $walkStatus == 1){?>
192. <script>

```



```

193.     $( document )
194.         .ready( function () {
195.
196.             $( "#bodyDiv" )
197.                 .fadeIn( 150 );
198.
199.             //When the cancel button is clicked, run function.
200.             $( "#btnCancel" )
201.                 .click( function () {
202.
203.                     //Define AJAX call to a PHP handler page. This page updates the walks current status for all clients.
204.                     //Here, we are setting the status to zero, with a null string for the time. The system now enters the
205.                     //Inactive state.
206.                     $.ajax( {
207.                         url: "../Project/AJAX/updateWalkStatus",
208.                         method: "POST",
209.                         data: {
210.                             status: 0,
211.                             startTime: ''
212.                         },
213.                         success: function ( result ) {
214.                             //On success, reload the page.
215.                             $( "#bodyDiv" )
216.                                 .fadeOut( 150 );
217.                             setTimeout(
218.                                 function () {
219.                                     location.reload( true );
220.                                 }, 150 );
221.                         }
222.                     } );
223.                 } );
224.         } );
225.     </script>
226.
227.     <!-- If user is any level, and the walk is in Waiting State, then load the Javascript to show the countdown timer -->
228.     <?php } if($walkStatus == 1){?>
229.     <script>
230.         $( document )
231.             .ready( function () {
232.
233.                 $( "#bodyDiv" )
234.                     .fadeIn( 150 );

```

```

235.
236.     var timeToLaunch;
237.     var lastTime;
238.
239.     //Define an AJAX call to a PHP handler page. This page fetches the UNIX seconds time to launch.
240.     $.ajax( {
241.         url: "../Project/API/getTimeToStart?",
242.         success: function ( result ) {
243.
244.             //Parse the returned JSON response.
245.             var responseObject = JSON.parse( result );
246.
247.             //If the return status is OK, then...
248.             if ( responseObject.status == "OK" ) {
249.                 //On success, set timeToLaunch to the result.
250.                 timeToLaunch = responseObject.timeToStart;
251.
252.                 //Set lastTime to equal the current time in seconds.
253.                 lastTime = Math.floor( new Date()
254.                     .getTime() / 1000 );
255.
256.                 //Call the ticker function.
257.                 ticker();
258.             }
259.         }
260.     } );
261.
262.     function ticker() {
263.
264.         //Set currTime to the current time in seconds.
265.         var currTime = Math.floor( new Date()
266.             .getTime() / 1000 );
267.
268.
269.         //If current time is greater than last time, it means that one second has elapsed.
270.         if ( currTime > lastTime ) {
271.
272.             //Set lastTime to equal the current time in seconds.
273.             lastTime = Math.floor( new Date()
274.                 .getTime() / 1000 );
275.
276.             //Remove one second from the fetched UNIX start time.

```

```

277.         timeToLaunch--;
278.
279.         //Using modulus calculations, we can find the number of days, hours, months and seconds to launch
280.         var d = Math.floor( timeToLaunch / ( 60 * 60 * 24 ) );
281.         var h = Math.floor( ( timeToLaunch % ( 60 * 60 * 24 ) ) / ( 60 * 60 ) );
282.         var m = Math.floor( ( timeToLaunch % ( 60 * 60 ) ) / ( 60 ) );
283.         var s = Math.floor( timeToLaunch % ( 60 ) );
284.
285.         //Print the remaining time to the screen.
286.         $( "#timer" )
287.             .html( d + "d" + h + "h" + m + "m" + s + "s" );
288.
289.     }
290.
291.     //If the time to launch is zero seconds, then reload the page.
292.     if ( timeToLaunch == 0 ) {
293.         $( "#bodyDiv" )
294.             .fadeOut( 1500 );
295.         setTimeout(
296.             function () {
297.                 window.location.href = "../Project/Overview";
298.             }, 1500 );
299.     } else {
300.         //Else, call the ticker function in another 333 milliseconds to check if a second has elapsed.
301.         setTimeout( ticker, 333 );
302.     }
303. }
304. } );
305. </script>
306.
307. <!-- If user is any level, and the walk is in the Active state, the load the JavaScript code to show the graphical
308. overview of the tracking data. -->
309. <?php } if($walkStatus == 2){?>
310. <script>
311.     $( document )
312.         .ready( function () {
313.
314.             //Show the loading spinner element.
315.             $( "#mainSpin" )
316.                 .fadeIn( 100 );
317.
318.             //If the 'toTracking' button is clicked, then load the Tracking Page.

```

```

319.      $( "#toTracking" )
320.      .click( function () {
321.          $( "#bodyDiv" )
322.          .fadeOut( 150 );
323.          setTimeout(
324.              function () {
325.                  window.location.href = "../Project/Tracking";
326.              }, 150 );
327.      } );
328.
329.      //Instantiate an instace of an object which will hold the information about the checkpoints
330.      var checkpoints = new Object( {
331.          distance: new Array( 0, 4300, 8400, 13000, 16300 ),
332.          initials: new Array( "SF", "PB", "LL", "BW" )
333.      } );
334.
335.      //Define an accociative array to store HEX colour information for each yeargroup.
336.      var colours = [ "#FFBF00", "#FF7300", "#FF0D00", "#00FF37", "#0037FF", "#FF01DD", "#8000FF" ];
337.
338.      //Define the Easel.js Stage - A HTML canvas API. The stage will be where we print our graphic data display.
339.      var stage = new createjs.Stage( "trackingCanvas" );
340.      stage.enableMouseOver( 50 );
341.
342.      //Define a variable that can be used to track wether this is the first time fetching tracking data from the server.
343.      var first = true;
344.
345.      //Call the update function.
346.      update();
347.
348.      //Bind click event listerers to each of the accordion pannels toggle switches
349.      //- these panels will show checkpoint data.
350.      for ( let i = 1; i < 5; i++ ) {
351.          $( "#tog" + i.toString() )
352.          .click( function () {
353.              //Pass 'i' to the panelSwitch function.
354.              panelSwitch( i );
355.          } );
356.      }
357.
358.      //When the div with the 'launchInfo' class is clicked, open the information modal.
359.      $( ".launchInfo" )
360.      .click( function () {

```

```

361.         $( "#infoModal" )
362.             .fadeIn( 150 );
363.     } );
364.
365.     //When the div with the infoModalClose' ID is clicked, close the information modal.
366.     $( "#infoModalClose" )
367.         .click( function () {
368.             $( "#infoModal" )
369.                 .fadeOut( 150 );
370.         } );
371.
372.     function update() {
373.
374.         //Call the getLogData function, which accepts checkpoints and an anonymous callback function
375.         //as parameters. Since the getLogData function contains an asynchronous AJAX function, we need a way
376.         //to execute more code after the AJAX function has finished. We cannot write inline code here since it will
377.         //run before the AJAX has finished its call. To get around this, we can pass a function to the parent function,
378.         //which will be called from within the parent function whenever the AJAX is successful.
379.
380.         getLogData( checkpoints, function ( logData ) {
381.
382.             //Clear the stage and prepare for graphics.
383.             stage.removeAllChildren();
384.             stage.update();
385.
386.             //Run the 'plotTimeline' function with param 'logData'
387.             plotTimeline( logData );
388.
389.             //Run the 'plotStudents' function with param 'logData'
390.             plotStudents( logData );
391.
392.             //Run the 'plotCheckpoints' function with param 'logData'
393.             plotCheckpoints( checkpoints );
394.
395.             //Run the 'populatePanels' function with param 'logData'
396.             populatePanels( logData );
397.
398.             //If this is the first time getting some data, then we have to first fade out the
399.             //CSS loading spinner element, so that the page below will be displayed.
400.             //By using a loading spinner, we can show the user that an asynchronous process is running.
401.             if ( first ) {
402.                 first = false;

```

```

403.         $( "#mainSpin" )
404.             .fadeOut( 150 );
405.         setTimeout(
406.             function () {
407.                 $( "#bodyDiv" )
408.                     .fadeIn( 500 );
409.                 }, 150 );
410.         }
411.
412.         //Call the update function in 5 seconds, which contains the 'getLogData' function
413.         //since params cannot be passed through a setTimeout call.
414.         setTimeout( update, 5000 );
415.
416.     } );
417. }
418.
419. //A function which will open the sliding panel that contains the checkpoint information. It
420. //takes a target panel, opening that panel and closing all other sliding panels that do not match the target.
421.
422. function panelSwitch( target ) {
423.
424.     //For Each Panel
425.     for ( let j = 1; j < 5; j++ ) {
426.
427.         //If panel is not the target
428.         if ( target != j ) {
429.
430.             //Close that panel, using a jQuery selector to match the panel object, over a time of 75 milliseconds.
431.             $( "#pan" + j.toString() )
432.                 .slideUp( 75 );
433.         }
434.     }
435.
436.     //Set a timeout to the same length as the closing panel time.
437.     setTimeout(
438.         function () {
439.
440.             //Open (or close) the target panel on click over a time of 75 milliseconds/
441.             $( "#pan" + target.toString() )
442.                 .slideToggle( 75 );
443.
444.             //Scroll the page to center the newly opened panel on the screen, over the course of 500 milliseconds.

```

```

445.         $( 'body, html' )
446.             .animate( {
447.                 scrollTop: $( "#panelContainer" )
448.                     .offset()
449.                     .top - 170
450.             }, 500 );
451.     }, 75 );
452. }
453.
454.     //A function that will plot the ellipse on the EASEL.js stage, using tracking data to generate dynamic text statistics.
455.     function plotTimeline( trackingData ) {
456.
457.         //Instantiate an EASEL.js shape.
458.         var timeline = new createjs.Shape();
459.
460.         //Instantiate an EASEL.js text element. Generate the text string based on how many people have started the walk.
461.         var startCountText = new createjs.Text( "Started: " + trackingData[ 'startCount' ] + "/" + trackingData[ 'stuCount' ], "50px Source Sans Pro", "black" );
462.
463.         //Instantiate an EASEL.js text element. Generate the text string based on how many people has finished against how many people started.
464.         var finishCountText = new createjs.Text( "Finished: " + trackingData[ 'checkCount' ][ 4 ] + "/" + trackingData[ 'startCount' ], "50px Source Sans Pro", "black" );
465.
466.         //Set the graphic drawing properties. Then, draw the ellipse.
467.         timeline.graphics
468.             .setStrokeStyle( 2 )
469.             .beginStroke( "#000" )
470.             .beginFill( "#fff" )
471.             .drawEllipse( 0, 0, stage.canvas.width - 50, stage.canvas.height - 50 );
472.
473.         //Set the top-left-corner position of the ellipse - 25px indented here.
474.         timeline.y = 25;
475.         timeline.x = 25;
476.
477.         //Set the start and finish count text based on the canvas width and the pixel-width of the generated text.
478.         startCountText.x = ( stage.canvas.width - startCountText.getMeasuredWidth() ) / 2;
479.         finishCountText.x = ( stage.canvas.width - finishCountText.getMeasuredWidth() ) / 2;
480.
481.         //Position the text elements.
482.         startCountText.y = ( stage.canvas.height - 80 ) / 2 - 25;
483.         finishCountText.y = ( stage.canvas.height - 80 ) / 2 + 25;

```

```

484.
485.         //Prepare the stage for a graphics update.
486.         stage.addChild( timeLine, startCountText, finishCountText );
487.
488.         //Update the stage to the new graphics.
489.         stage.update();
490.     }
491.
492.     //A function to plot each student on the timeline.
493.     function plotStudents( trackingData ) {
494.
495.         //Instantiate an EASEL.js shape.
496.         var circle = new createjs.Shape();
497.
498.         //For each student distance, plot the student.
499.         for ( var i = 0; i < trackingData[ 'distances' ].length; i++ ) {
500.
501.             //Make a clone of the EASEL.js shape, copying the graphics instance too.
502.             var cloneCirc = circle.clone( true );
503.
504.             //Draw the circle, with the colour as defined by the colour array and the yeargroup of the student.
505.             cloneCirc.graphics.beginFill( colours[ trackingData[ 'distances' ][ i ][ 'year' ] - 7 ] )
506.                 .drawCircle( 0, 0, 6 );
507.
508.             //Calculate the coordinates of the student by calling the 'CalculateCoords' function, accepting the
509.             //distance travelled, ellipse width and height as the parameters.
510.             var pointCoords = CalculateCoords(
511.                 trackingData[ 'distances' ][ i ][ 'meters' ], ( stage.canvas.width - 50 ) / 2, ( stage.canvas.height - 50 ) / 2
512.             );
513.
514.             //Set the students coordinates.
515.             cloneCirc.x = pointCoords[ "x" ] + stage.canvas.width / 2;
516.             cloneCirc.y = pointCoords[ "y" ] + stage.canvas.height / 2;
517.
518.             //Prepare the stage for a graphics update.
519.             stage.addChild( cloneCirc );
520.
521.             //Update the stage to the new graphics.
522.             stage.update();
523.         }
524.     }
525.

```



```

526.         //A function to plot the clickable checkpoints at their locations on the timeline.
527.         function plotCheckpoints( checkpoints ) {
528.
529.             //Instantiate an EASEL.js shape.
530.             var checkPoint = new createjs.Shape();
531.
532.             //Instantiate an EASEL.js text element with an empty text string.
533.             var checkText = new createjs.Text( "", "24px Source Sans Pro", "black" );
534.
535.             //For Each Checkpoint, plot a clickable circle.
536.             for ( let i = 0; i < 4; i++ ) {
537.
538.                 //Make a clone of the EASEL.js shape and text element, copying the graphics instance too.
539.                 var cloneCheck = checkPoint.clone( true );
540.                 var cloneText = checkText.clone( true );
541.
542.                 //Draw a grey circle.
543.                 cloneCheck.graphics.beginFill( "#b2b2b2" )
544.                     .drawCircle( 0, 0, 20 );
545.
546.                 //Set the text string to the checkpoint initials.
547.                 cloneText.text = checkpoints[ 'initials' ][ i ];
548.
549.                 //Calculate the coordinates of the checkpoint circle by calling the 'CalculateCoords' function, accepting the
550.                 //checkpoint distance, ellipse width and height as the parameters.
551.                 var pointCoords = CalculateCoords(
552.                     checkpoints[ 'distance' ][ i ], ( stage.canvas.width - 50 ) / 2, ( stage.canvas.height - 50 ) / 2
553.                 );
554.
555.                 //Set the checkpoint initials text based on the canvas width, top-left-corner coords and the
556.                 //pixel-width of the generated text.
557.                 cloneText.x = pointCoords[ "x" ] + ( stage.canvas.width - cloneText.getMeasuredWidth() ) / 2;
558.                 cloneText.y = pointCoords[ "y" ] + ( stage.canvas.height - cloneText.getMeasuredHeight() ) / 2 - 6;
559.
560.                 //Set the top-left-corner circle coords based on the canvas width.
561.                 cloneCheck.x = pointCoords[ "x" ] + stage.canvas.width / 2;
562.                 cloneCheck.y = pointCoords[ "y" ] + stage.canvas.height / 2;
563.
564.                 //Set the cursor to the pointer when hovering over the clickable element.
565.                 cloneCheck.cursor = "pointer";
566.
567.                 //Prepare the stage for a graphics update.

```

```

568.         stage.addChild( cloneCheck, cloneText );
569.
570.         //Update the stage to the new graphics.
571.         stage.update();
572.
573.         //Add a new event which handles the click event, by switching to the relevant checkpoint info panel.
574.         cloneCheck.addEventListener( "click", function () {
575.             panelSwitch( i + 1 );
576.         } );
577.     }
578. }
579.
580. //A function to populate the checkpoint information panels.
581. function populatePanels( trackingData ) {
582.
583.     //For each checkpoint panel, update the information.
584.     for ( var i = 0; i < 5; i++ ) {
585.
586.         //Select the checkpoint information element in our panel, and update the text to display the checkpoint student count.
587.         $( "#check" + ( i + 1 ) )
588.             .html( trackingData[ 'checkCount' ][ i ] + "/" + trackingData[ 'startCount' ] );
589.
590.         //Define a variable to store which staff are here.
591.         var staffString = "";
592.
593.         //If there is more than zero staff members at the checkpoint, then add them to the staff string, with a clickable link to the
594.
595.         //staff information card viewer.
596.         if ( trackingData[ 'checkStaff' ][ i ].length > 0 ) {
597.
598.             //For each staff member, add them to the staff string, within a HTML 'a' tag.
599.             for ( var j = 0; j < trackingData[ 'checkStaff' ][ i ].length; j++ ) {
600.                 staffString += '<a href = "../Project/staffCard?staffCode=' +
601.                     trackingData[ 'checkStaff' ][ i ][ j ] + '>' +
602.                     trackingData[ 'checkStaff' ][ i ][ j ] + '</a> ';
603.             }
604.         } else {
605.             //If no-one is here, let the user know.
606.             staffString = "No-One Here Yet!";
607.         }
608.
609.         ///Select the staff displayer element in our panel, and update the html to show who is here with a clickable link.

```

```

609.         $( "#staffHere" + ( i + 1 ) )
610.             .html( staffString );
611.     }
612. }
613.
614. //A function to get and format some log data from the server.
615. function getLogData( checkpoints, callBack ) {
616.
617.     //Outline a new object to populate with the log data.
618.     var logData = {
619.         distances: [],
620.         checkCount: [ 0, 0, 0, 0, 0, 0 ],
621.         checkStaff: [
622.             [ '' ],
623.             [ '' ],
624.             [ '' ],
625.             [ '' ],
626.             [ '' ]
627.         ],
628.         stuCount: 0,
629.         startCount: 0
630.     };
631.
632.     //Define an AJAX call to a PHP handler page. This page fetches the log data from the server.
633.     $.ajax( {
634.         url: "../Project/AJAX/getLogData",
635.         success: function ( result ) {
636.
637.             //Turn the response into a usable Javascript object.
638.             var returnObj = JSON.parse( result );
639.
640.             //The number of students is written to the data object.
641.             logData[ 'stuCount' ] = returnObj[ 'stuCount' ];
642.
643.             //Make an array containing each server time element (day, hours, mins ect.).
644.             var serverTimeArray = returnObj[ 'serverTime' ].split( /[-T:]/ );
645.
646.             //Make a javascript date object with the server time elements.
647.             var serverTime = new Date(
648.                 serverTimeArray[ 0 ],
649.                 serverTimeArray[ 1 ] - 1,
650.                 serverTimeArray[ 2 ],

```

```

651.             serverTimeArray[ 3 ],
652.             serverTimeArray[ 4 ],
653.             serverTimeArray[ 5 ] );
654.
655.             //The average human walking pace.
656.             const METERSPERSEC = 1.389;
657.
658.             //For each returned log, decode the data and build up the usable log object.
659.             for ( var i = 0; i < returnObj[ 'logs' ].length; i++ ) {
660.
661.                 //Increment the student start count by one.
662.                 logData[ 'startCount' ]++;
663.
664.                 //If the log states that the student last registered at the last checkpoint (i.e. finish point) then increment the che
ckpoint
665.                 //student counter by one, and break. Else, we need to create a new student log to print.
666.                 if ( returnObj[ 'logs' ][ i ][ 'locID' ] == 4 ) {
667.
668.                     //Increment the student checkpoint counter by one.
669.                     logData[ 'checkCount' ][ 4 ]++;
670.
671.                 } else {
672.
673.                     //Make an array containing each registration time element (day, hours, mins ect.).
674.                     var timeStamp = returnObj[ 'logs' ][ i ][ 'regTime' ].split( /[- :]/ );
675.
676.                     //Make a javascript date object with the registration time elements.
677.                     var regTime = new Date(
678.                         timeStamp[ 0 ],
679.                         timeStamp[ 1 ] - 1,
680.                         timeStamp[ 2 ],
681.                         timeStamp[ 3 ],
682.                         timeStamp[ 4 ],
683.                         timeStamp[ 5 ] );
684.
685.                     //Get the number of seconds since the student registered, according the the current server time.
686.                     var secSinceReg = Math.round( ( serverTime - regTime ) / 1000 );
687.
688.                     //Calculate the distance the student has travelled since registering.
689.                     var distance = Math.round( METERSPERSEC * secSinceReg );
690.
691.                     //Since the current server time may be ahead of the database timestamp time, we must account for this

```

```

692.          //by setting any negative distance to zero meters.
693.          if ( distance < 0 ) distance = 0;
694.
695.          //Add distance travelled since last registration to the distance of the last registration checkpoint from the
696.          //start line to get a total distance travelled.
697.          distance += checkpoints[ 'distance' ][ returnObj[ 'logs' ][ i ][ 'locID' ] ];
698.
699.          //Since the student has to have travelled through checkpoint zero, increment the student checkpoint counter by one
700.          logData[ 'checkCount' ][ 0 ]++;
701.
702.          //The following 'if' block will increment student checkpoint counters by one, depending on what checkpoints have b
703.          //been passed
704.          //through. Each checkpoint counter will go up by one depending on the number of students who have passed through t
705.          //hat
706.          //checkpoint.
707.          if ( returnObj[ 'logs' ][ i ][ 'locID' ] >= 1 ) {
708.              logData[ 'checkCount' ][ 1 ]++;
709.          }
710.          if ( returnObj[ 'logs' ][ i ][ 'locID' ] >= 2 ) {
711.              logData[ 'checkCount' ][ 2 ]++;
712.          }
713.          if ( returnObj[ 'logs' ][ i ][ 'locID' ] >= 3 ) {
714.              logData[ 'checkCount' ][ 3 ]++;
715.          }
716.          //End If Block
717.
718.          //If the calculated distance is greater than the distance to the next checkpoint, set the distance to that value.
719.
720.          //This accounts for the error where the distance travelled may be calculated to be greater than the distance to th
721.          //e next
722.          //checkpoint, possibly leading to innacurate tracking data if the student is walking slower than expected.
723.          if ( distance > checkpoints[ 'distance' ][ returnObj[ 'logs' ][ i ][ 'locID' ] + 1 ] ) {
724.              distance = checkpoints[ 'distance' ][ returnObj[ 'logs' ][ i ][ 'locID' ] + 1 ];
725.          }
726.          //Push the calculated distance and student information to the log data distances array within the log data object.
727.          logData[ 'distances' ].push( new Object( {

```

```

728.             stuCode: returnObj[ 'logs' ][ i ][ 'stuCode' ],
729.             meters: distance,
730.             year: returnObj[ 'logs' ][ i ][ 'year' ]
731.         } ) );
732.
733.     }
734.
735. }
736.
737.     //Add the staff member and checkpoint location to log data object.
738.     logData[ 'checkStaff' ] = returnObj[ 'checkStaff' ];
739.
740.     //After the asynchronous ajax call and processing has finished, continue to the next part of the code by calling the
741.     //passed callback function with the log data object as the parameter.
742.     callBack( logData );
743. }
744.
745. } );
746. }
747.
748. function CalculateCoords( meters, yRad, xRad ) {
749.     //for a 16300 meter circuit
750.     const DEGPERMETER = 0.0220858896;
751.     //Nice Conversion Constant between deg and rad
752.     const DEGTORADS = Math.PI / 180;
753.     //Calculate the number of degrees through which the point moves
754.     var fullDeg = meters * DEGPERMETER;
755.
756.     //'If' statement block to find out which quad we are in. Set a multiplier to adjust
757.     //coords accordingly. Factor in the goofy co-ordinate system too. Calculate our adjusted quadrant
758.     //specific angle, measured from the x-axis.
759.     var xMultiplier = 0;
760.     var yMultiplier = 0;
761.
762.     var quadDeg = 0;
763.
764.     if ( fullDeg <= 90 ) {
765.         //adjusted 1st Quad
766.         xMultiplier = -1;
767.         yMultiplier = 1;
768.         //Get adjusted angle
769.         quadDeg = 90 - fullDeg;

```

```

770.         } else if ( fullDeg > 90 && fullDeg <= 180 ) {
771.             //adjusted 2nd Quad
772.             xMultiplier = -1;
773.             yMultiplier = -1;
774.             //Get adjusted angle
775.             quadDeg = fullDeg - 90;
776.         } else if ( fullDeg > 180 && fullDeg <= 240 ) {
777.             //adjusted 3rd Quad
778.             xMultiplier = 1;
779.             yMultiplier = -1;
780.             //Get adjusted angle
781.             quadDeg = 270 - fullDeg;
782.         } else {
783.             //adjusted 4th Quad
784.             xMultiplier = 1;
785.             yMultiplier = 1;
786.             //Get adjusted angle
787.             quadDeg = fullDeg - 270;
788.         }
789.
790.         var quadRad = quadDeg * DEGTORADS;
791.
792.         //Radius at point equation, which is the polar form of the ellipse equation.
793.         var radiusAtPoint =
794.             xRad *
795.             yRad /
796.             Math.sqrt(
797.                 Math.pow( yRad, 2 ) * Math.pow( Math.sin( quadRad ), 2 ) +
798.                 Math.pow( xRad, 2 ) * Math.pow( Math.cos( quadRad ), 2 )
799.             );
800.
801.         //Get are final x-coord using some Sine action!
802.         var xCoord = radiusAtPoint * Math.cos( quadRad );
803.         xCoord *= xMultiplier;
804.
805.         //Get are final y-coord using some Sine action!
806.         var yCoord = radiusAtPoint * Math.sin( quadRad );
807.         yCoord *= yMultiplier;
808.
809.         //Return a packaged object with our data inside.
810.         var returnObj = new Object();
811.         returnObj[ "x" ] = xCoord;

```

```

812.         returnObj[ "y" ] = yCoord;
813.         return returnObj;
814.     }
815.
816.     } );
817. </script>
818. <?php } ?>
819.</head>
820.
821.<header>
822.    <h1>Sponsored Walk!</h1>
823.    <nav>
824.        <ul>
825.            <li class="active">Overview</li>
826.            <li id="toManage">Manage</li>
827.        </ul>
828.    </nav>
829.</header>
830.
831.<body>
832.    <!-- If user is any level, and the walk is in the Active state, the load the HTML to show the graphical
833.    overview of the tracking data. -->
834.    <?php if($walkStatus == 2) {?>
835.
836.    <div id="bodyDiv">
837.        <p style="text-align: center;">Welcome,
838.        <!-- Print the user name into the HTML title -->
839.        <?php echo $_SESSION['login'];?>
840.    </p>
841.    <div class="canvasWrap">
842.        <canvas id="trackingCanvas" width="1024" height="512" style="background-color:white;"></canvas>
843.        <div class="center sbsWrap">
844.            <table>
845.                <tr>
846.                    <td>
847.                        <div class="colourBox" style="background-color:#8000FF"> </div>
848.                    </td>
849.                    <td>
850.                        <div>Year 13</div>
851.                    </td>
852.                    <td>
853.                        <div class="colourBox" style="background-color:#FF01DD"> </div>

```



```

854.         </td>
855.         <td>
856.             <div>Year 12</div>
857.         </td>
858.         <td>
859.             <div class="colourBox" style="background-color:#0037FF"> </div>
860.         </td>
861.         <td>
862.             <div>Year 11</div>
863.         </td>
864.         <td>
865.             <div class="colourBox" style="background-color:#00FF37"> </div>
866.         </td>
867.         <td>
868.             <div>Year 10</div>
869.         </td>
870.         <td>
871.             <div class="colourBox" style="background-color:#FF0D00"> </div>
872.         </td>
873.         <td>
874.             <div>Year 9</div>
875.         </td>
876.         <td>
877.             <div class="colourBox" style="background-color:#FF7300"> </div>
878.         </td>
879.         <td>
880.             <div>Year 8</div>
881.         </td>
882.         <td>
883.             <div class="colourBox" style="background-color:#FFBF00"> </div>
884.         </td>
885.         <td>
886.             <div>Year 7</div>
887.         </td>
888.     </tr>
889. </table>
890.     
891. </div>
892. </div>
893. <h2>View Leg Information:</h2>
894. <div id="panelContainer">
895.     <div class="togglePan top" id="tog1">

```

```

896.         <p><b>1. Soothill Foyer</b>
897.         </p>
898.     </div>
899.     <div class="panel" id="pan1">
900.         <div class="sbsWrap">
901.             <iframe width="45%" height="250" frameborder="0" style="border:0" src="https://www.google.com/maps/embed/v1/place?q=place_id:ChIJK
bkmr09ReUgRiI-gV2HYJkc
902.             &key=AIzaSyAezm79hi0hcOadd1PPAuMdbpZAhi-tRPU
903.             &maptype=satellite" allowfullscreen>
904.             </iframe>
905.
906.             <div style="vertical-align: top;width:50%;">
907.                 <h3 class="omitFromWrap"><b>Checkpoint Stats</b></h3>
908.                 <p class="omitFromWrap"><b>Students Registered Here:</b>
909.                 </p>
910.                 <p class="omitFromWrap" id="check1">0/0</p>
911.                 <p class="omitFromWrap"><b>Staff Registering Here:</b>
912.                 </p>
913.                 <p class="omitFromWrap" id="staffHere1">No-one Yet!</p>
914.             </div>
915.         </div>
916.     </div>
917.     <div class="togglePan" id="tog2">
918.         <p><b>2. Pot Bank</b>
919.         </p>
920.     </div>
921.     <div class="panel" id="pan2">
922.         <div class="sbsWrap">
923.             <iframe width="45%" height="250" frameborder="0" style="border:0" src="https://www.google.com/maps/embed/v1/place?q=place_id:ChIJr
__hH5lWeUgRF6Bj3ufqWuc
924.             &key=AIzaSyAezm79hi0hcOadd1PPAuMdbpZAhi-tRPU
925.             &maptype=satellite" allowfullscreen>
926.             </iframe>
927.
928.             <div style="vertical-align: top;width:50%;">
929.                 <h3 class="omitFromWrap"><b>Checkpoint Stats</b></h3>
930.                 <p class="omitFromWrap"><b>Students Registered Here:</b>
931.                 </p>
932.                 <p class="omitFromWrap" id="check2">0/0</p>
933.                 <p class="omitFromWrap"><b>Staff Registering Here:</b>
934.                 </p>
935.                 <p class="omitFromWrap" id="staffHere2">No-one Yet!</p>

```

```

936.         </div>
937.     </div>
938. </div>
939. <div class="togglePan" id="tog3">
940.     <p><b>3. Long Liberty</b></p>
941. </div>
942. <div class="panel" id="pan3">
943.     <div class="sbsWrap">
944.         <iframe width="45%" height="250" frameborder="0" style="border:0" src="https://www.google.com/maps/embed/v1/place?q=place_id:ChIJU
aggM-JVeUgRldxGHXtqtmY
945.         &key=AIzaSyAezm79hi0hcOadd1PPAuMdbpZAhi-tRPU
946.         &motype=satellite" allowfullscreen>
947.     </iframe>
948.
949.     <div style="vertical-align: top;width:50%;">
950.         <h3 class="omitFromWrap"><b>Checkpoint Stats</b></h3>
951.         <p class="omitFromWrap"><b>Students Registered Here:</b>
952.         <p class="omitFromWrap" id="check3">0/0</p>
953.         <p class="omitFromWrap"><b>Staff Registering Here:</b>
954.         <p class="omitFromWrap" id="staffHere3">No-one Yet!</p>
955.     </div>
956. </div>
957. <div class="togglePan" id="tog4">
958.     <p><b>4. Beckwithshaw</b></p>
959. </div>
960. <div class="panel" id="pan4">
961.     <div class="sbsWrap">
962.         <iframe width="45%" height="250" frameborder="0" style="border:0" src="https://www.google.com/maps/embed/v1/place?q=place_id:ChIJ0
1Tok45WeUgRtACgrbu_cdc
963.         &key=AIzaSyAezm79hi0hcOadd1PPAuMdbpZAhi-tRPU
964.         &motype=satellite" allowfullscreen>
965.     </iframe>
966.
967.     <div style="vertical-align: top;width:50%;">
968.         <h3 class="omitFromWrap"><b>Checkpoint Stats</b></h3>
969.         <p class="omitFromWrap"><b>Students Registered Here:</b>
970.     </div>
971. </div>

```

```

976.         <p class="omitFromWrap" id="check4">0/0</p>
977.         <p class="omitFromWrap"><b>Staff Registering Here:</b>
978.         </p>
979.         <p class="omitFromWrap" id="staffHere4">No-one Yet!</p>
980.     </div>
981. </div>
982. </div>
983. </div>
984.
985.     <div id="btnLogOut" class="btn center">
986.         Log Out
987.     </div>
988.     <div class="lowerFloat" id="toTracking">
989.         View Tracking Data
990.     </div>
991. </div>
992.
993. <!-- If user is any level, and the walk is in the Waiting state, then load the HTML to show the timer. -->
994. <?php } if($walkStatus == 1) {?>
995.
996. <div id="bodyDiv">
997.     <h1 id="timer" class="center largeText">Timer Here...</h1>
998.
999.     <!-- If user is high level, load the HTML to show the cancel button. -->
1000.     <?php if($_SESSION['level'] == 1) {?>
1001.
1002.     <div id="btnCancel" class="btn large center">Cancel Countdown</div>
1003.
1004.     <?php } ?>
1005.
1006.     <div id="btnLogOut" class="btn center">
1007.         Log Out
1008.     </div>
1009. </div>
1010.
1011. <!-- If user is high level, and the walk is in the inactive state, the load the HTML to show the countdown start tools. -->
1012. <?php } if($_SESSION['level'] == 1 && $walkStatus == 0){?>
1013.
1014. <div id="bodyDiv">
1015.     <div class="wrapper">
1016.         <h2>Schedule Tracking Start</h2>

```

```

1017.         <p>Select Launch Time. This is the time that the tracking will go live. It must be a date and time in the future. A countdown will
commence to this time when you click 'Launch'.</p>
1018.         <div class="sbsWrap">
1019.             <h2>Launch Time: </h2>
1020.             <input type="datetime-local" class="padInput" id="startTrackTime" min="<?php echo date(" Y-m-
d\TH:i ");//Echo the server time as a minimum selction value.?)>">
1021.         </div>
1022.         <p id="info" class="msg"> </p>
1023.         <div id="btnScheduleWalk" class="btn large center">
1024.             Schedule Launch!
1025.         </div>
1026.         <h2>Or</h2>
1027.         <div id="btnLaunchWalk" class="btn large center">
1028.             Launch Now!
1029.         </div>
1030.         <div id="btnLogOut" class="btn center">
1031.             Log Out
1032.         </div>
1033.     </div>
1034. </div>
1035.
1036. <!-- If user is low level, and the walk is in the inactive state, the load the HTML to show the friendly message. -->
1037. <?php } if($_SESSION['level'] == 0 && $walkStatus == 0) {?>
1038.
1039.     <div id="bodyDiv">
1040.         <h1>Check Back Later...</h1>
1041.         <h2>Looks Like Tracking Hasnt Begun Just Yet. Come By Later For The Countdown!</h2>
1042.         <div id="btnLogOut" class="btn center">
1043.             Log Out
1044.         </div>
1045.     </div>
1046.
1047. <?php } ?>
1048.
1049.     <div class="modal" id="infoModal">
1050.         <div class="content">
1051.             <h2>Tracking System Information</h2>
1052.             <p>OVERVIEW PAGE -
Here you can find a visual overview of the status of the walk. You can see an approximate position of everyone on the walk by observing the coloured
dots making their way clockwise around the graphical representation of the walk. You can also explore each checkpoint by clicking on its name or the gr
ey bubble on the oval.</p>

```

```

1053.         <p>TRACKING PAGE -
    By clicking on the 'View Tracking Data' button at the bottom of this page, you can see a table with all the exact registration events of each student
    on the walk. You can search for individual students or order the table to see who still to register at certain checkpoints.</p>
1054.         <p>MANAGE PAGE -
    Here, you can print tracking QR codes, and if you have administrator permissions, make changes to who can view the sponsored walk tracker.</p>
1055.         <div class="btn center" id="infoModalClose">Got It!</div>
1056.         </div>
1057.     </div>
1058.
1059.     <div class="spinner" id="mainSpin" style="display:none;"></div>
1060.
1061. </body>
1062.
1063. </html>

```

---

## ../PROJECT/MANAGE.PHP

This page contains all the administrator tasks that can be carried out by the users of the web app. These include adding and removing users, printing QR codes and ending the tracking session (if they are high-level).

```

1.  <?php
2.  //Start the session
3.  session_start();
4.
5.  //Enable Error reporting for debug
6.  error_reporting( E_ALL );
7.  ini_set( 'display_errors', 1 );
8.
9.  //Defining the secure access parameter means that any config or functional files
10. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
11. define( 'secureAccessParameter', true );
12. require '../Project/Tools/config.php';
13. require '../Project/Tools/walkFunctions.php';
14.
15. //If the user has not logged in, then the session variable of 'login' will not be set,
16. //so redirect to the Login page.
17. if ( !isset( $_SESSION[ 'login' ] ) ) {
18.     header( 'Location:http://11wha.ashvillecomputing.co.uk/Project/Login' );
19.     exit();
20. }
21.

```

```

22. //Call the get walk status function. Assign a variable to its returned value.
23. $walkStatus;
24. $returnObject = json_decode( getWalkStatus(), 1 );
25. if ( $returnObject[ 'status' ] == "OK" )$walkStatus = $returnObject[ 'walkStatus' ];
26.
27. ?>
28.
29. <!DOCTYPE html">
30. <html xmlns="http://www.w3.org/1999/xhtml">
31.
32. <head>
33.     <title>Manage Walk</title>
34.     <link rel="stylesheet" type="text/css" href="main.css">
35.     <link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro" rel="stylesheet">
36.     <link rel="icon" href="../../Project/Resources/favicon.png" type="image/x-icon"/>
37.     <meta name="viewport" content="width=device-width, initial-scale=1.0">
38.     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
39.     <script>
40.         $( document )
41.             .ready( function () {
42.
43.                 $( "#bodyDiv" )
44.                     .fadeIn( 500 );
45.
46.                 //When the div with the 'toOverview' ID is clicked, redirect to the overview page.
47.                 $( "#toOverview" )
48.                     .click( function () {
49.                         $( "#bodyDiv" )
50.                             .fadeOut( 150 );
51.                         setTimeout(
52.                             function () {
53.                                 window.location.href = "../../Project/Overview";
54.                             }, 150 );
55.                     } );
56.
57.                 //When the div with the 'btnLogOut' ID is clicked, redirect to the Login/Logout page.
58.                 $( "#btnLogOut" )
59.                     .click( function () {
60.                         $( "#bodyDiv" )
61.                             .fadeOut( 150 );
62.                         setTimeout(
63.                             function () {

```

```

64.             window.location.href = "../Project/Login?logOut";
65.         }, 150 );
66.     } );
67.
68.     //When the div with the 'btnPrintCodes' ID is clicked, begin the print code function.
69.     $( "#btnPrintCodes" )
70.         .click( function () {
71.
72.             //Reset information text
73.             $( "#info4" )
74.                 .html( " " );
75.
76.             //Display the printCodes modal
77.             $( "#printCodesModal" )
78.                 .fadeIn( 150 );
79.
80.             //When the print all checkbox is toggles, toggle the print specific student code text box
81.             //to either be enabled or disabled for text entry.
82.             $( "#printAll" )
83.                 .on( 'click', function () {
84.
85.                 //If checked, then...
86.                 if ( $( this )
87.                     .is( ':checked' ) ) {
88.
89.                     //Disable text-box.
90.                     $( "#printCodeName" )
91.                         .prop( 'disabled', true );
92.
93.                 } else {
94.
95.                     //Enable text-box.
96.                     $( "#printCodeName" )
97.                         .prop( 'disabled', false );
98.                 }
99.             } );
100.
101.             //When the div with the 'printCodesModalConfirm' ID is clicked, begin the print launcher.
102.             $( "#printCodesModalConfirm" )
103.                 .click( function () {
104.
105.                 //Define Variables to hold print request data.

```



```

106.         var params = "";
107.         var valid = true;
108.
109.         //If the specific student code text box is populated and the print all box is not checked, then...
110.         if ( $( "#printCodeName" )
111.             .val() != "" && !$( "#printAll" )
112.             .is( ':checked' ) ) {
113.
114.             //Define AJAX call to a PHP handler page. This handler checks to see if the user code we entered is valid/exists.
115.
116.             $.ajax( {
117.                 url: "../Project/API/studentExists?stuCode=" + $( "#printCodeName" )
118.                 .val(),
119.                 success: function ( result ) {
120.
121.                     //Parse the returned JSON response.
122.                     var responseObject = JSON.parse( result );
123.
124.                     //If the return status is OK, then...
125.                     if ( responseObject.status == "OK" ) {
126.                         params = "?stuCode=" + $( "#printCodeName" )
127.                             .val();
128.
129.                         //Try to launch the print page.
130.                         tryPrint();
131.                     } else {
132.
133.                         //Set valid boolean to false, then display an error to let the user know that the user code was invalid
134.
135.                         valid = false;
136.                         $( "#info4" )
137.                             .html( "Error: Student " + $( "#printCodeName" )
138.                                 .val() + " does not exist! Check the student code and try again." );
139.                     }
140.                 }
141.             } );
142.
143.             //Else if the print-all checkbox is not checked and no code has been entered.
144.             } else if ( !$( "#printAll" )
145.                 .is( ':checked' ) ) {

```

```

146.         //Set valid boolean to false, then diplay an error to let the user know that they need to enter a user code.
147.         valid = false;
148.         $( "#info4" )
149.             .html( "Please Enter A Student Code!" );
150.
151.     } else {
152.
153.         //Try to launch the print page.
154.         tryPrint();
155.     }
156.
157.     //A function to launch the print code page, by first checking to see if all validation checkes have been passed.
158.     function tryPrint() {
159.         if ( valid ) window.location.href = "../Project/printCodes" + params;
160.     }
161. } );
162.
163. //When the div with the 'printCodesModalCancel' ID is clicked, close the print code modal.
164. $( "#printCodesModalCancel" )
165.     .click( function () {
166.         $( "#printCodesModal" )
167.             .fadeOut( 150 );
168.     } );
169.
170. } );
171. } );
172. </script>
173. <?php if($_SESSION['level'] == 1) { ?>
174. <!--If user is high-level, include hi-level scripts-->
175. <script>
176.     $( document )
177.         .ready( function () {
178.
179.             //When the div with the 'btnEndWalk' ID is clicked, begin the end walk confirmation sequence.
180.             $( "#btnEndWalk" )
181.                 .click( function () {
182.
183.                     //Reset information text
184.                     $( "#info5" )
185.                         .html( " " );
186.
187.                     //Reset the end walk confirmation password value.

```

```

188.         $( "#endWalkPass" )
189.             .val( "" );
190.
191.         //Display the end walk modal.
192.         $( "#endModal" )
193.             .fadeIn( 150 );
194.
195.         //When the div with the 'btnEndWalk' ID is clicked, begin the password checks.
196.         $( "#endModalConfirm" )
197.             .click( function () {
198.
199.                 //Capture the password value from the textbox.
200.                 var userPass = $( "#endWalkPass" )
201.                     .val();
202.
203.                 //Define AJAX call to a PHP handler page. This handler checks to see if the admin password we entered was correct.
204.                 $.ajax( {
205.                     url: "../Project/AJAX/passwordCheck",
206.                     method: "POST",
207.                     data: {
208.                         pass: userPass
209.                     },
210.                     success: function ( result ) {
211.
212.                         //Parse the returned JSON response.
213.                         var responseObject = JSON.parse( result );
214.
215.                         //If the return status is OK, then...
216.                         if ( responseObject.status == "OK" ) {
217.
218.                             //Fade out current modal elements, to make space for the loading spinner.
219.                             $( "#endWalkPass" )
220.                                 .fadeOut( 150 );
221.                             $( "#endModalConfirm" )
222.                                 .fadeOut( 150 );
223.                             $( "#endModalCancel" )
224.                                 .fadeOut( 150 );
225.
226.                             //Display the loading spinner.
227.                             $( "#endModalSpin" )
228.                                 .fadeIn( 150 );
229.

```

```

230.         //Update the information text to let the user know what the current process is.
231.         $( "#info5" )
232.             .html( "Ending Tracking Session..." );
233.
234.         //Define AJAX call to a PHP handler page. This page updates the walks current status for all clients.
235.         //Here, we are setting the status to zero, with a null string for the time. The system now enters the
236.         //Inactive state.
237.         $.ajax( {
238.             url: "../Project/AJAX/updateWalkStatus",
239.             method: "POST",
240.             data: {
241.                 status: 0,
242.                 startTime: ''
243.             },
244.             success: function ( result ) {
245.
246.                 //Parse the returned JSON response.
247.                 var responseObject = JSON.parse( result );
248.
249.                 //If the state change was made succesfully, then redirect to the overview page.
250.                 if ( responseObject.status == "OK" ) {
251.                     setTimeout(
252.                         function () {
253.                             $( "#bodyDiv" )
254.                                 .fadeOut( 150 );
255.                             setTimeout(
256.                                 function () {
257.                                     window.location.href = "../Project/Overview";
258.                                 }, 150 );
259.                             }, 1000 );
260.
261.                     //Else, if the state change failed, change the modal elements.
262.                 } else {
263.
264.                     //Fade out the loading spinner.
265.                     $( "#endModalSpin" )
266.                         .fadeOut( 150 );
267.
268.                     //Fade in the cancel button.
269.                     $( "#endModalCancel" )
270.                         .fadeIn( 150 );
271.

```

```

272.                                     //Update the information text to display an error.
273.                                     $( "#info5" )
274.                                     .html( "Error: Could Not End Tracking. Please Try Again." );
275.                                     }
276.                                 }
277.                            } );
278.
279.                            //Else, if the password entered was incorrect, tell the user.
280.                        } else {
281.
282.                            //Reset the password value.
283.                            $( "#endWalkPass" )
284.                            .val( "" );
285.
286.                            //Update the information text to display an error telling the user there password was incorrect.
287.                            $( "#info5" )
288.                            .html( responseObject.message );
289.                        }
290.                    }
291.                } );
292.            } );
293.
294.            //When the div with the 'endModalCancel' ID is clicked, close the end walk modal.
295.            $( "#endModalCancel" )
296.            .click( function () {
297.                $( "#endModal" )
298.                .fadeOut( 150 );
299.            } );
300.        } );
301.
302.        //When the div with the 'btnNewUser' ID is clicked, begin the add new user sequence.
303.        $( "#btnNewUser" )
304.        .click( function () {
305.
306.            //If the new user code textbox is populated, and the permissions have been selected, then proceed.
307.            if ( $( "#newUserCode" )
308.                .val() != "" && $( "#newPermSelect" )
309.                .val() != "" ) {
310.
311.                //Set variables to contain the new user information.
312.                var user = $( "#newUserCode" )
313.                .val();

```

```

314.         var perm = $( "#newPermSelect" )
315.             .val();
316.
317.         //Reset the information text.
318.         $( "#info" )
319.             .html( " " );
320.
321.         //Display a message to confirm the user to be added.
322.         $( "#addModalName" )
323.             .html( "Are you sure you want to add: '" + user + "'" );
324.
325.         //Show the add user modal buttons.
326.         $( "#addModalConfirm" )
327.             .show();
328.         $( "#addModalCancel" )
329.             .show();
330.
331.         //Hide the modal loading spinner.
332.         $( "#addModalSpin" )
333.             .hide();
334.
335.         //Set the user code entry textbox to have a black border, since through validation checks, it may be set to
336.         //red to indicate a lack of text entry.
337.         $( "#newUserCode" )
338.             .css( "border-color", "black" );
339.
340.         //Fade in the add user modal.
341.         $( "#addModal" )
342.             .fadeIn( 150 );
343.
344.         //When the div with the 'addModalConfirm' ID is clicked, begin the new user checks.
345.         $( "#addModalConfirm" )
346.             .click( function () {
347.
348.                 //Fade out the add user modal buttons.
349.                 $( "#addModalConfirm" )
350.                     .fadeOut( 150 );
351.                 $( "#addModalCancel" )
352.                     .fadeOut( 150 );
353.
354.                 //Fade in the loading spinner.
355.                 $( "#addModalSpin" )

```

```

356.         .fadeIn( 150 );
357.
358.         //Display a confirmation message of user invitation.
359.         $( "#addModalName" )
360.             .html( "Inviting " + user + "!" );
361.
362.         //Define AJAX call to a PHP handler page. This handler checks to see if the new user already exists on the databas
e, and
363.         //if not, then invites the user to the tracking system.
364.         $.ajax( {
365.             url: "../Project/AJAX/inviteNewUser",
366.             method: "POST",
367.             data: {
368.                 newUname: user,
369.                 newLevel: perm
370.             },
371.             success: function ( result ) {
372.                 //Parse the returned JSON response.
373.                 var responseObject = JSON.parse( result );
374.
375.                 //If the return status is OK, then...
376.                 if ( responseObject.status == "OK" ) {
377.
378.                     $( "#addModalName" )
379.                         .html( "Success!" );
380.                     setTimeout(
381.                         function () {
382.                             $( "#addModal" )
383.                                 .fadeOut( 150 );
384.                         }, 1000 );
385.
386.                     //Else, display an error letting the user know that they are trying to add a user that already exists.
387.                 } else {
388.                     $( "#addModalName" )
389.                         .html( " Error: " + user + " already exists! Please try a different user!" );
390.                     $( "#addModalCancel" )
391.                         .fadeIn( 150 );
392.                     $( "#addModalSpin" )
393.                         .fadeOut( 150 );
394.
395.                 }

```

```

396.         }
397.     } );
398. } );
399.
400.     //When the div with the 'addModalCancel' ID is clicked, close the add new user modal.
401.     $( "#addModalCancel" )
402.         .click( function () {
403.             $( "#addModal" )
404.                 .fadeOut( 150 );
405.         } );
406.
407.     //Else, if the user has not specified a usercode or permission, display a message to prompt the user to try again.
408. } else {
409.
410.     //Set the textbox border colour to red to indicate faliure.
411.     $( "#newUserCode" )
412.         .css( "border-color", "red" );
413.
414.     //Display the error message.
415.     $( "#info" )
416.         .html( "Please Enter User and Permission" );
417. }
418. } );
419.
420. //When the div with the 'btnUpUser' ID is clicked, begin the update user permissions sequence.
421. $( "#btnUpUser" )
422.     .click( function () {
423.
424.         //If the user has selected a user, and a new permission for that user, the proceed.
425.         if ( $( "#userSelect" )
426.             .val() != "" && $( "#permissionSelect" )
427.             .val() != "" ) {
428.
429.             //Set variables to contain the new user information.
430.             var user = $( "#userSelect" )
431.                 .val();
432.             var perm = $( "#permissionSelect" )
433.                 .val();
434.
435.             //Define AJAX call to a PHP handler page. This handler updates the user permissions, by recieving a
436.             //user and new permission, and updating the entry on the database.
437.             $.ajax( {

```



```

438.         url: "../Project/AJAX/updateUserPerms",
439.         method: "POST",
440.         data: {
441.             uName: user,
442.             uLevel: perm
443.         },
444.         success: function ( result ) {
445.
446.             //Parse the returned JSON response.
447.             var responseObject = JSON.parse( result );
448.
449.             //If the return status is OK, then...
450.             if ( responseObject.status == "OK" ) {
451.                 $( "#info2" )
452.                     .html( "Updated Permissions For " + user );
453.
454.                 //Else, display an error message.
455.             } else {
456.                 $( "#info2" )
457.                     .html( "Error: Could Not Update Permissions..." );
458.             }
459.         }
460.     } );
461.
462.     //Else, if the selection boxes are empty, display an error message to let the user know what to do.
463. } else {
464.     $( "#info2" )
465.         .html( "Please Select User and Permission" );
466. }
467. } );
468.
469. //When the div with the 'btnRemoveUser' ID is clicked, begin the remove user sequence.
470. $( "#btnRemoveUser" )
471.     .click( function () {
472.
473.         //If the selection is not populated.
474.         if ( $( "#removeUserSelect" )
475.             .val() != "" ) {
476.
477.             //Set variable to capture the selection value.
478.             var user = $( "#removeUserSelect" )
479.                 .val();

```

```

480.
481.         //Reset the information text.
482.         $( "#info3" )
483.             .html( " " );
484.
485.         //Display a message to confirm the removal of the user.
486.         $( "#removeModalName" )
487.             .html( "Are you sure you want to remove: '" + user + "' ?" );
488.
489.         //Fade in the remove user modal.
490.         $( "#removeModal" )
491.             .fadeIn( 150 );
492.
493.         //When the div with the 'removeModalConfirm' ID is clicked, remove the user.
494.         $( "#removeModalConfirm" )
495.             .click( function () {
496.
497.                 //Fade out the confirmation button.
498.                 $( "#removeModalConfirm" )
499.                     .fadeOut( 150 );
500.                 $( "#removeModalCancel" )
501.                     .fadeOut( 150 );
502.
503.                 //Fade in the loading spinner.
504.                 $( "#removeModalSpin" )
505.                     .fadeIn( 150 );
506.
507.                 //Define AJAX call to a PHP handler page. This handler removes the selected user from the users database table.
508.                 $.ajax( {
509.                     url: "../Project/AJAX/removeUser",
510.                     method: "POST",
511.                     data: {
512.                         uName: user
513.                     },
514.                     success: function ( result ) {
515.
516.                         //Parse the returned JSON response.
517.                         var responseObject = JSON.parse( result );
518.
519.                         //If the return status is OK, then...
520.                         if ( responseObject.status == "OK" ) {
521.

```

```

522.             $( "#removeModalName" )
523.                 .html( "Removed " + user + ". Refreshing... Please Wait!" );
524.             setTimeout(
525.                 function () {
526.                     $( "#removeModal" )
527.                         .fadeOut( 150 );
528.                     $( "#bodyDiv" )
529.                         .fadeOut( 150 );
530.                     location.reload( true );
531.                 }, 1000 );
532.
533.             //Else, display an error message telling the user to try again.
534.         } else {
535.
536.             //Display the error message.
537.             $( "#removeModalName" )
538.                 .html( "Error: Could Not Remove User..." );
539.
540.             //Fade out the loading spinner.
541.             $( "#removeModalSpin" )
542.                 .fadeOut( 150 );
543.
544.             //Fade in the modal cancel button.
545.             $( "#removeModalCancel" )
546.                 .fadeIn( 150 );
547.         }
548.     }
549. } );
550.
551.
552. //When the div with the 'removeModalCancel' ID is clicked, close the remove user modal.
553. $( "#removeModalCancel" )
554.     .click( function () {
555.         $( "#removeModal" )
556.             .fadeOut( 150 );
557.     } );
558.
559. //Else, if the selection was null, display an error message.
560. } else {
561.     $( "#info3" )
562.         .html( "Please Select User." );
563. }

```

```

564.         } );
565.
566.         //When the div with the 'launchInfo' class is clicked, open the information modal.
567.         $( ".launchInfo" )
568.             .click( function () {
569.                 $( "#infoModal" )
570.                     .fadeIn( 150 );
571.             } );
572.
573.         //When the div with the infoModalClose' ID is clicked, close the information modal.
574.         $( "#infoModalClose" )
575.             .click( function () {
576.                 $( "#infoModal" )
577.                     .fadeOut( 150 );
578.             } );
579.     } );
580. </script>
581. <?php } ?>
582.</head>
583.
584.<header>
585.    <h1>Sponsored Walk!</h1>
586.    <nav>
587.        <ul>
588.            <li id="toOverview">Overview</li>
589.            <li class="active">Manage</li>
590.        </ul>
591.    </nav>
592.</header>
593.
594.<body>
595.    <div id="bodyDiv">
596.
597.        <h2>Student Tracking Codes</h2>
598.        <div class="wrapper">
599.            <h3>Print QR Code For Student</h3>
600.            <div class="btn" id="btnPrintCodes">Print!</div>
601.        </div>
602.
603.        <!-- If the user is high level, include the user account management -->
604.        <?php if($_SESSION['level'] == 1) { ?>
605.

```

```

606.     <h2>User Controls</h2>
607.     <div class="wrapper">
608.         <h3>Add New User:</h3>
609.         <p>Invite anyone with an Ashville Email Address to access the Tracking Suite:</p>
610.         <h4>Email Address</h4>
611.         <div class="sbsWrap">
612.             <input type="text" class="inText short" id="newUserCode" placeholder="USER">
613.             <p>@ashville.co.uk</p>
614.         </div>
615.
616.         <h4>Permissions:</h4>
617.         <div class="sbsWrap">
618.             <select id="newPermSelect">
619.                 <option value="">Select Permission...</option>
620.                 <option value="0">View and Track</option>
621.                 <option value="1">Full Administrator Access</option>
622.             </select>
623.             
624.         </div>
625.
626.         <div id="btnNewUser" class="btn">
627.             Invite User!
628.         </div>
629.
630.         <p id="info" class="msg"> </p>
631.
632.         <hr>
633.
634.         <h3>Update Current User Permissions:</h3>
635.
636.         <div class="sbsWrap">
637.             <p>Change </p>
638.             <select id="userSelect">
639.                 <option value="">Select User...</option>
640.
641.                 <?php
642.                     //Define and create a database connection.
643.                     $conn = dbConnect();
644.
645.                     //Prepare an SQL statement to select all usernames from the admin user table.
646.                     $sqlQuery = $conn->prepare( "SELECT uName FROM tblAdminUsers");
647.

```

```

648.         //Run the SQL query.
649.         $sqlQuery->execute();
650.
651.         //Set variable to the SQL result object.
652.         $result = $sqlQuery->get_result();
653.
654.         //If the result has more than zero rows
655.         if ( $result->num_rows > 0 )
656.         {
657.             //Set a variable to store the data rows as an array.
658.             $data = $result->fetch_all();
659.
660.             //For each row of the data
661.             foreach($data as $dataRow)
662.             {
663.                 //If the user is not the currently logged in user, then echo an option into the HTML
664.                 //document to contain that user.
665.                 if($dataRow[0] != $_SESSION['login'])
666.                 {?>
667.
668.                 <option value="<?php echo $dataRow[0]; ?>">
669.                     <?php echo $dataRow[0]; ?>
670.                 </option>
671.
672.                 <?php
673.                 }
674.                 }
675.                 }
676.
677.             //Close the prepared statement query.
678.             $sqlQuery->close();
679.
680.             //Close the database connection.
681.             dbClose();
682.             ?>
683.
684.         </select>
685.
686.         <p> to </p>
687.
688.
689.         <select id="permissionSelect">

```

```

690.         <option value="">Select Permission...</option>
691.         <option value="0">View and Track</option>
692.         <option value="1">Full Administrator Access</option>
693.     </select>
694.
695.     
696.
697. </div>
698.
699. <div id="btnUpUser" class="btn">
700.     Update User
701. </div>
702.
703. <p id="info2" class="msg"> </p>
704.
705. <hr>
706.
707. <h3>Remove User:</h3>
708. <select id="removeUserSelect">
709.     <option value="">Select User...</option>
710.     <?php
711.
712.         //For each row of the data
713.         foreach($data as $dataRow)
714.         {
715.             //If the user is not the currently logged in user, then echo an option into the HTML
716.             //document to contain that user.
717.             if($dataRow[0] != $_SESSION['login'])
718.             {?>
719.
720.                 <option value="<?php echo $dataRow[0]; ?>">
721.                     <?php echo $dataRow[0]; ?>
722.                 </option>
723.
724.             <?php
725.             }
726.             }
727.             ?>
728.         </select>
729.     <div id="btnRemoveUser" class="btn">
730.         Remove!
731.     </div>

```

```

732.         <p id="info3" class="msg"> </p>
733.     </div>
734.
735.     <?php ??>
736.
737.     <div id="btnLogOut" class="btn center">
738.         Log Out
739.     </div>
740. </div>
741.
742. <!-- If the user is high level, and the walk is the Active State, include the walk ending button -->
743. <?php if($walkStatus == 2 && $_SESSION['level'] == 1) {?>
744.
745.     <div id="btnEndWalk" class="lowerFloat">
746.         End Tracking!
747.     </div>
748.
749. <?php } ?>
750.
751. <div class="spinner" id="mainSpin" style="display:none;"></div>
752.
753. <!-- If the user is high level, include the user account management modals -->
754. <?php if($_SESSION['level'] == 1) {?>
755.
756.     <div class="modal" id="removeModal">
757.         <div class="content">
758.             <h2>Confirm User Removal?</h2>
759.             <p id="removeModalName"> </p>
760.             <div class="miniSpin center" id="removeModalSpin" style="display:none;"></div>
761.             <div class="sbsWrap">
762.                 <div class="btn center" id="removeModalConfirm">Confirm?</div>
763.                 <div class="btn center" id="removeModalCancel">Cancel?</div>
764.             </div>
765.         </div>
766.     </div>
767.     <div class="modal" id="addModal">
768.         <div class="content">
769.             <h2>Confirm Invite User?</h2>
770.             <p id="addModalName"> </p>
771.             <div class="miniSpin center" id="addModalSpin" style="display:none;"></div>
772.             <div class="sbsWrap">
773.                 <div class="btn center" id="addModalConfirm">Confirm?</div>

```



```

774.         <div class="btn center" id="addModalCancel">Cancel?</div>
775.     </div>
776. </div>
777. </div>
778. <div class="modal" id="infoModal">
779.     <div class="content">
780.         <h2>Permission Info</h2>
781.         <p>View and Track - Users can View tracking data and search for students.</p>
782.         <p>Full Administrator Access - All permissions, including starting and stopping tracking, and adding new users.</p>
783.         <div class="btn center" id="infoModalClose">Got It!</div>
784.     </div>
785. </div>
786. <div class="modal" id="endModal">
787.     <div class="content">
788.         <h2>End Sponsored Walk Tracking?</h2>
789.         <p id="endModalText">Are You Sure You Want To Stop Tracking? This Action Will De-
activate all tracking stations, delete any tracking logs, and reset the sponsored walk tracking suite. Only end tracking when ALL STUDENTS have comple
ted the walk.</p>
790.         <input type="password" class="inText" id="endWalkPass" placeholder="PASSWORD">
791.         <div class="miniSpin center" id="endModalSpin" style="display:none;"></div>
792.         <div class="sbsWrap">
793.             <div class="btn center" id="endModalConfirm">Confirm?</div>
794.             <div class="btn center" id="endModalCancel">Cancel?</div>
795.         </div>
796.         <p id="info5" class="msg"> </p>
797.     </div>
798. </div>
799.
800. <?php } ?>
801.
802. <div class="modal" id="printCodesModal">
803.     <div class="content">
804.         <h2>Print QR Code For Student</h2>
805.         <p id="printCodesModalText">Choose Your QR Code Printing Options Here:</p>
806.         <input type="checkbox" id="printAll" unchecked>Print QR Code For All Students?</input>
807.         <input type="text" class="inText" id="printCodeName" placeholder="STUDENT CODE">
808.         <div class="miniSpin center" id="printCodesModalSpin" style="display:none;"></div>
809.         <div class="sbsWrap">
810.             <div class="btn center" id="printCodesModalConfirm">Confirm?</div>
811.             <div class="btn center" id="printCodesModalCancel">Cancel?</div>
812.         </div>
813.         <p id="info4" class="msg"> </p>

```

```

814.      </div>
815.    </div>
816.</body>
817.
818.</html>

```

---

[../PROJECT/TRACKING.PHP](#)

This page contains the functional table which will hold the table of tracking data for each student. It polls the server to receive a new copy of the table every 10 seconds, so that the data is 'live'.

```

1. <?php
2. //Start the session
3. session_start();
4.
5. //Enable Error reporting for debug
6. error_reporting( E_ALL );
7. ini_set( 'display_errors', 1 );
8.
9. //Defining the secure access parameter means that any config or functional files
10. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
11. define( 'secureAccessParameter', true );
12. require '../Project/Tools/config.php';
13. require '../Project/Tools/walkFunctions.php';
14.
15. //If the user has not logged in, then the session variable of 'login' will not be set,
16. //so redirect to the Login page.
17. if ( !isset( $_SESSION[ 'login' ] ) ) {
18.     header( 'Location:http://11wha.ashvillecomputing.co.uk/Project/Login' );
19.     exit();
20. }
21.
22. //Call the get walk status function. Assign a variable to its returned value.
23. $walkStatus;
24. $returnObject = json_decode( getWalkStatus(), 1 );
25. if ( $returnObject[ 'status' ] == "OK" )$walkStatus = $returnObject[ 'walkStatus' ];
26.
27. //If the walk is not in the Active state, then redirect to the overview page.
28. if($walkStatus != 2)
29. {
30.     header( 'Location:http://www.11wha.ashvillecomputing.co.uk/Project/Overview' );

```

```

31.     exit();
32. }
33. ?>
34.
35. <!DOCTYPE html">
36. <html xmlns="http://www.w3.org/1999/xhtml">
37.
38. <head>
39. <title>Tracking</title>
40. <link rel="stylesheet" type="text/css" href="main.css">
41. <link rel="icon" href="../../Project/Resources/favicon.png" type="image/x-icon" />
42. <link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro" rel="stylesheet">
43. <meta name="viewport" content="width=device-width, initial-scale=1.0">
44. <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
45. <script>
46. $(document)
47.     .ready(function() {
48.
49.         //Fade in the main loading spinner.
50.         $("#mainSpin")
51.             .fadeIn(150);
52.
53.         //When the div with the 'toOverview' ID is clicked, redirect to the overview page.
54.         $("#toOverview")
55.             .click(function() {
56.                 $("#bodyDiv")
57.                     .fadeOut(150);
58.                 setTimeout(
59.                     function() {
60.                         window.location.href = "../Project/Overview";
61.                     }, 150);
62.             });
63.
64.         //When the div with the 'btnLogOut' ID is clicked, redirect to the Login/Logout page.
65.         $("#btnLogOut")
66.             .click(function() {
67.                 $("#bodyDiv")
68.                     .fadeOut(150);
69.                 setTimeout(
70.                     function() {
71.                         window.location.href = "../Project/Login?logOut";
72.                     }, 150);

```

```

73.         });
74.
75.         //Variable to track wether this is the first time running through the program.
76.         var first = true;
77.
78.         //Variable to store the display order of the tracking table.
79.         var orderCode = "";
80.
81.         //Call the ticker update function.
82.         ticker();
83.
84.         //When anything in the table container division is clicked, then select the
85.         //enclosed element that was clicked.
86.         $("#tblContainer")
87.             .on('click', '#item', function() {
88.
89.                 //If the clicked items 'data' is set to A (Ascending).
90.                 if ($(this)
91.                     .data("sort") == "A") {
92.
93.                     //Change the sort direction to D (Decending).
94.                     $(this)
95.                         .data("sort", "D");
96.
97.                     //Construct the orderCode with the new sort and location information.
98.                     orderCode =
99.                         $(this)
100.                            .data("loc") +
101.                            $(this)
102.                                .data("sort");
103.                 }
104.                 //Else, if the clicked items 'data' is set to D (Decending).
105.                 else if ($(this)
106.                     .data("sort") == "D") {
107.                     //Change the sort direction to null (There will be no sorting).
108.                     $(this)
109.                         .data("sort", "");
110.
111.                     //Construct a null order code.
112.                     orderCode = "";
113.                 }
114.                 //Else, if the clicked items 'data' is set to Null (No sorting).

```

```

115.         else if ($(this)
116.             .data("sort") == "") {
117.
118.             //Change the sort direction to A (Ascending).
119.             $(this)
120.                 .data("sort", "A");
121.
122.             //Construct the orderCode with the new sort and location information.
123.             orderCode =
124.                 $(this)
125.                     .data("loc") +
126.                 $(this)
127.                     .data("sort");
128.         }
129.
130.         //Call the print table function.
131.         printTable();
132.     });
133.
134.     //If any on the year select radio buttons are toggles, then call the print table function.
135.     $("input[name='yearSelect']")
136.         .change(function() {
137.             printTable();
138.         });
139.
140.     //If the name search box is typed in, then call the print table function.
141.     $("#nameSearch")
142.         .keyup(function() {
143.             printTable();
144.         });
145.
146.     //Define a function that will call the print table function every 10 seconds. This will ensure that the table is
147.     //updated automatically, without the need for user refreshing.
148.     function ticker() {
149.         printTable();
150.         setTimeout(ticker, 10000);
151.     }
152.
153.     //A function that will gather options selected by the user, and print a table with the content returned
154.     //from an AJAX call.
155.     function printTable() {
156.

```

```

157.         //Get the selected year value.
158.         var yearData = $("input:radio[name='yearSelect']:checked")
159.             .val();
160.
161.         //Get the search term from the name searchbox.
162.         var nameSearch = $("#nameSearch")
163.             .val();
164.
165.         //Define the base URL which points to the PHP handler.
166.         var url = "../Project/AJAX/printLogTable?";
167.
168.         //Add the order code to the URL.
169.         url += ("order=" + orderCode);
170.
171.         //If the selected year is not 'all', then add the year data to the URL.
172.         if (yearData != "all") url += ("%year=" + yearData);
173.
174.         //If the selected name is not null, then add the name data to the URL.
175.         if (nameSearch != "") url += ("%query=" + nameSearch);
176.
177.         //Define AJAX call to a PHP handler page. This handler will accept some data from the user, such as a
178.         //yeargroup or name search terms, and returns a constructed HTML table, printed from the database.
179.         $.ajax({
180.             url: url,
181.             success: function(result) {
182.
183.                 //Print the table to the screen. This will add the returned HTML from the server to the page.
184.                 $("#tblContainer")
185.                     .html(result);
186.
187.                 //If this is the first time we have printed the table, then remove the loading spinner and show
188.                 //the table and the main body.
189.                 if (first) {
190.                     $("#mainSpin")
191.                         .fadeOut(150);
192.                     setTimeout(
193.                         function() {
194.                             $("#bodyDiv")
195.                                 .fadeIn(150);
196.                         }, 150);
197.                     first = false;
198.                 }

```

```

199.
200.         //For each column, bind mouse event handlers. We will then bind the cell highlighting to the table.
201.         for (let i = 1; i < 8; i++) {
202.             //Bind a mouse enter event to each column.
203.             $(".data")
204.                 .on('mouseenter', '.col' + i, function() {
205.
206.                 //Add the highlight CSS class to the entire column.
207.                 $(".col" + i)
208.                     .addClass("highlight");
209.
210.                 //Get the row which the cell is on.
211.                 var row = $(this)
212.                     .data("row");
213.
214.                 //Add the highlight CSS class to the entire row.
215.                 $(".row" + row)
216.                     .addClass("highlight");
217.             });
218.
219.             //Bind a mouse leave event to each column.
220.             $(".data")
221.                 .on('mouseleave', '.col' + i, function() {
222.
223.                 //Remove the highlight CSS class from the entire column.
224.                 $(".col" + i)
225.                     .removeClass("highlight");
226.
227.                 //Get the row which the cell is on.
228.                 var row = $(this)
229.                     .data("row");
230.
231.                 //Remove the highlight CSS class from the entire row.
232.                 $(".row" + row)
233.                     .removeClass("highlight");
234.             });
235.         }
236.     }
237. });
238. }
239.
240. //When a div with the class 'launchInfo' is clicked, display the information modal.

```

```

241.     $(".launchInfo")
242.     .click(function() {
243.         $("#infoModal")
244.         .fadeIn(150);
245.     });
246.
247.     //When a div with the id 'infoModalClose' is clicked, close the information modal.
248.     $("#infoModalClose")
249.     .click(function() {
250.         $("#infoModal")
251.         .fadeOut(150);
252.     });
253. });
254.</script>
255.</head>
256.
257.
258.<header>
259.    <h1>Student Search!</h1>
260.    <nav>
261.        <ul>
262.            <li id="toOverview" >Return To Overview</li>
263.        </ul>
264.    </nav>
265.</header>
266.
267.<body>
268.    <div id="bodyDiv">
269.        <table class = "yearContainer center">
270.            <tr>
271.                <td>
272.                    <label>
273.                        <input type="radio" name="yearSelect" value="all" checked>
274.                        All
275.                    </label>
276.                </td>
277.                <td>
278.                    <label class = "y7">
279.                        <input type="radio" name="yearSelect" value="7">
280.                        Year 7
281.                    </label>
282.                </td>

```



```

283.         </td>
284.         <td>
285.             <label class = "y8">
286.                 <input type="radio" name="yearSelect" value="8">
287.                     Year 8
288.             </label>
289.         </td>
290.         <td>
291.             <label class = "y9">
292.                 <input type="radio" name="yearSelect" value="9">
293.                     Year 9
294.             </label>
295.         </td>
296.         <td>
297.             <label class = "y10">
298.                 <input type="radio" name="yearSelect" value="10">
299.                     Year 10
300.             </label>
301.         </td>
302.         <td>
303.             <label class = "y11">
304.                 <input type="radio" name="yearSelect" value="11">
305.                     Year 11
306.             </label>
307.         </td>
308.         <td>
309.             <label class = "y12">
310.                 <input type="radio" name="yearSelect" value="12">
311.                     Year 12
312.             </label>
313.         </td>
314.         <td>
315.             <label class = "y13">
316.                 <input type="radio" name="yearSelect" value="13">
317.                     Year 13
318.             </label>
319.         </td>
320.         <td>
321.     </tr>
322. </table>
323.
324. <div class = "short center">

```

```

325.     <div class="sbsWrap">
326.         <input type="text" id = "nameSearch" class ="inText short" placeholder="Student Search">
327.         
328.     </div>
329. </div>
330.
331.     <div id = "tblContainer">
332.         Loading...
333.     </div>
334.
335.     <div id="btnLogOut" class="btn center">
336.         Log Out
337.     </div>
338.
339. </div>
340. <div class="spinner" id="mainSpin" style="display:none;"></div>
341.     <div class="modal" id="infoModal">
342.         <div class="content">
343.             <h2>Search For Any Student</h2>
344.             <p>From here, you can search for any student, using their surname, forename or Ashville student code.</p>
345.             <p>The search will use all your data to help find a student. Don't worry about the order or format of your search.</p>
346.             <div class="btn center" id="infoModalClose">Got It!</div>
347.         </div>
348.     </div>
349.</body>

```

---

../PROJECT/STUDENT.PHP

This page allows the students to begin the login process, and ultimately access the Google OAuth flow, to later access only their own QR code.

```

1. <?php
2. //Start the session
3. session_start();
4.
5. //Enable Error reporting for debug
6. error_reporting( E_ALL );
7. ini_set( 'display_errors', 1 );
8.
9. //Defining the secure access parameter means that any config or functional files
10. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
11. define( 'secureAccessParameter', true );
12. require_once '../Project/googleAuth/gconfig.php';

```

```

13. require_once '../Project/googleAuth/googleLoginHelper.php';
14.
15. //If the user has not logged in, then the session variable of 'login' will not be set,
16. //so redirect to the Login page.
17. if(isset($_SESSION['stuLogin']))
18. {
19.     header( 'Location:http://11wha.ashvillecomputing.co.uk/Project/studentCard?stuCode=' . $emailSegments[0] );
20.     exit();
21. }
22.
23. //Define an array of scopes which we will use to authorise access to the users information.
24. $scopes = array('https://www.googleapis.com/auth/userinfo.profile' , 'https://www.googleapis.com/auth/userinfo.email', 'https://www.googleapis.com/auth/plus.me');
25.
26. //Instantiate a GoogleLoginHelper class. This helper gives us some methods to help with the authorization process.
27. $gHelper = new GoogleLoginHelper();
28.
29. ?>
30.
31. <!doctype html>
32. <html>
33.     <head>
34.     <title><?php if(isset($_GET['stuCode'])) echo strtoupper($_GET['stuCode']) . " - "; ?>Student Card</title>
35.     <link rel="stylesheet" type="text/css" href="main.css">
36.     <link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro" rel="stylesheet">
37.     <link rel="icon" href="../Project/Resources/favicon.png" type="image/x-icon"/>
38.     <meta name="viewport" content="width=device-width, initial-scale=1.0">
39.     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
40.     <script>
41.         $( document ).ready( function () {
42.             //Fade in the main body division.
43.             $( "#bodyDiv" ).fadeIn( 150 );
44.
45.         } );
46.     </script>
47. </head>
48.
49. <header style="padding-top: 32px;">
50.     <h1>Welcome, Student!</h1>
51. </header>
52.
53. <body>

```

```

54.     <div id = "bodyDiv">
55.         <div class = "loginWrap">
56.             <a class = "center" href = "
57.                 <?php
58.                     //Echo the URL created by the gHelper object into the href of the link tag.
59.                     echo $gHelper->GetAuthURL(CLIENT_ID, CLIENT_REDIRECT_URL, $scopes);
60.                 ?>
61.             ">
62.             </a>
63.             <p class = "msg">
64.                 <?php
65.                     //If we are returning to this page because of a bad domain error, print a message to let the user know to try again.
66.                     if(isset($_GET['badDomain'])) echo "You must use your Ashville email address! Please try again..."
67.                 ?>
68.             </p>
69.         </div>
70.     </div>
71. </body>
72. </html>

```

---

## ../PROJECT/REGISTER.PHP

This page is where prospective users of the tracking system will have the ability to define a new password and then become a full user of the system.

```

1. <?php
2. //Start the session
3. session_start();
4.
5. //Enable Error reporting for debug
6. error_reporting( E_ALL );
7. ini_set( 'display_errors', 1 );
8.
9. //Defining the secure access parameter means that any config or functional files
10. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
11. define( 'secureAccessParameter', true );
12. require '../Project/Tools/config.php';
13. require '../Project/Tools/walkFunctions.php';
14.
15. //Make a new database connection.
16. $conn = dbConnect();
17.
18. //Reset the user information string.

```

```

19. $msg = "";
20.
21. //Boolean to track if the invite expired.
22. $expired = false;
23.
24. //Prepare a statement to select the prospective user's info, matching the inviteID from the GET
25. //variable encoded in the URL.
26. $sqlQuery = $conn->prepare( "SELECT userCode, level FROM tblInvites WHERE inviteID = ?" );
27. $sqlQuery->bind_param( "s", $inviteCode );
28.
29. //Bind params and execute.
30. $inviteCode = $_GET[ 'inviteCode' ];
31. $sqlQuery->execute();
32.
33. //Get the SQL return object.
34. $result = $sqlQuery->get_result();
35.
36. //Check if the user exists in the table.
37. if ( $result->num_rows > 0 ) {
38.
39.     //If the user clicked the submit button, then proceed to add the user to the database.
40.     if ( isset( $_POST[ 'subBtn' ] ) ) {
41.
42.         //Variable to contain the username.
43.         $uName = strtoupper( $_POST[ 'uName' ] );
44.
45.         //Convert the SQL return object into an assoc. array.
46.         $dataRow = $result->fetch_assoc();
47.
48.         //If the user entered the right user code into the text input, then continue. This is a
49.         //security step to ensure that only the user who was send the email can get access to the
50.         //tracking suite.
51.         if ( $uName == $dataRow[ 'userCode' ] ) {
52.
53.             //Prepare a statement to insert the user into the user table.
54.             $sqlQuery = $conn->prepare( "INSERT INTO tblAdminUsers (uName,pass,level) VALUES (?,?,?)" );
55.             $sqlQuery->bind_param( "ssi", $uName, $pass, $level );
56.
57.             //Hash the plaintext password.
58.             $pass = password_hash( $_POST[ 'uPass' ], PASSWORD_DEFAULT );
59.
60.             //Bind params and execute.

```

```

61.         $level = $dataRow[ 'level' ];
62.         $sqlQuery->execute();
63.
64.         //Remove the entry for this user in the invite table.
65.         $sqlQuery = $conn->prepare( "DELETE FROM tblInvites WHERE userCode = ?" );
66.         $sqlQuery->bind_param( "s", $uName );
67.         $sqlQuery->execute();
68.
69.         //Redirect to the login page with the fromReg GET variable set.
70.         header( "Location:http://www.11wba.ashvillecomputing.co.uk/Project/Login.php?fromReg=$uName");
71.         exit();
72.     } else {
73.
74.         //If the usernames don't match, then alert the user.
75.         $msg = "Username Invalid. Please use the same Ashville Username that your invitation was sent to.";
76.     }
77. }
78. //Else, the invite has expired.
79. } else {
80.
81.     $msg = "Invite Expired.";
82.     $expired = true;
83. }
84.
85. //Close connections.
86. $sqlQuery->close();
87. dbClose();
88. ?>
89.
90. <!doctype html>
91. <html>
92.
93. <head>
94.     <meta charset="utf-8">
95.     <title>Register</title>
96.     <link rel="stylesheet" type="text/css" href="main.css">
97.     <link rel="icon" href="../Project/Resources/favicon.png" type="image/x-icon" />
98.     <link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro" rel="stylesheet">
99.     <meta name="viewport" content="width=device-width, initial-scale=1.0">
100.    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
101.    <script>
102.        $( document ).ready( function () {

```

```

103.
104.     //Fade in the main body division.
105.     $( "#bodyDiv" ).fadeIn( 500 );
106.
107.     //Array to store the text input DOM references.
108.     var textDom = [];
109.     textDom[ 0 ] = $( "#uNameText" );
110.     textDom[ 1 ] = $( "#uPassText" );
111.     textDom[ 2 ] = $( "#conPassText" );
112.
113.     //If the user clicks submit, run checks.
114.     $( "#submitBtn" ).click( function () {
115.
116.         var valid = true;
117.
118.         //For each text input, check if they entered some text. Update CSS to reflect
119.         //the state of each box (i.e. Red for invalid, Black for valid).
120.         for ( var i = 0; i < 3; i++ ) {
121.             if ( textDom[ i ].val().length == 0 ) {
122.
123.                 textDom[ i ].css( "borderColor", "#d81015" );
124.                 valid = false;
125.             } else {
126.                 textDom[ i ].css( "borderColor", "black" );
127.             }
128.         }
129.
130.         //If the new password entry is less than 8 characters, alert user with CSS and a message.
131.         if(textDom[ 1 ].val().length < 8)
132.         {
133.             valid = false;
134.             $( "#info" ).html( "Password Must Be At Least 8 Characters Long!" );
135.             textDom[ 1 ].css( "borderColor", "#d81015" )
136.         }
137.         //Else, if the two passwords don't match, then alert user with CSS and a message.
138.         else if(textDom[ 1 ].val() != textDom[ 2 ].val())
139.         {
140.             valid = false;
141.             $( "#info" ).html( "Passwords Do Not Match!" );
142.             textDom[ 1 ].css( "borderColor", "#d81015" )
143.             textDom[ 2 ].css( "borderColor", "#d81015" )
144.         }

```

```

145.
146.         //If checkes passed, submit the form. This refreshes the page.
147.         if ( valid ) {
148.             $( "#bodyDiv" ).fadeOut( 150 );
149.             setTimeout(
150.                 function () {
151.                     $( '#regForm' ).submit();
152.                 }, 150 );
153.
154.             }
155.         } );
156.     } );
157. </script>
158.</head>
159.
160.<header>
161.    <h1 style="padding-top: 32px;">Tracker Register!</h1>
162.</header>
163.
164.<body>
165.    <div id="bodyDiv">
166.        <div class="loginWrap">
167.            <form action="../../Project/Register.php?inviteCode=<?php echo $_GET['inviteCode']; ?>" method="post" id="regForm" >
168.                <div class = "sbsWrap">
169.                    <input type="text" name="uName" id="uNameText" class="inText short" placeholder="USERNAME" <?php if($expired) echo "disabled"; ?>>
170.
171.                    <p>@ashville.co.uk</p>
172.                    <input type="password" name="uPass" id="uPassText" class="inText" placeholder="NEW PASSWORD" <?php if($expired) echo "disabled"; ?>>
173.                    <input type="password" name="conPass" id="conPassText" class="inText" placeholder="RE-
174.                    TYPE PASSWORD" <?php if($expired) echo "disabled"; ?>>
175.                    <input type="hidden" name="subBtn" <?php if($expired) echo "disabled"; ?>>
176.                </form>
177.                <div class="btn <?php if($expired) echo "disabled"; ?>" id = "submitBtn">Register</div>
178.                <p class="msg" id="info">
179.                    <?php echo $msg;?>
180.                </p>
181.            </div>
182.</div>

```

../PROJECT/STUDENTCARD.PHP



This page is the QR code viewer for each student's QR code. It also contains a small table to show their tracking data.

```
1. <?php
2. //Start the session
3. session_start();
4.
5. //Enable Error reporting for debug
6. error_reporting( E_ALL );
7. ini_set( 'display_errors', 1 );
8.
9. //Defining the secure access parameter means that any config or functional files
10. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
11. define( 'secureAccessParameter', true );
12. require '../Project/Tools/config.php';
13. require '../Project/Tools/walkFunctions.php';
14.
15. //If the student is not logged in, or if a student code has not been defined, then redirect to the
16. //login area.
17. if(!isset($_SESSION['stuLogin']) or !isset($_GET['stuCode']))
18. {
19.     header( 'Location:http://11wha.ashvillecomputing.co.uk/Project/Student' );
20.     exit();
21. }
22. else
23. {
24.     //Get the segments of the email address.
25.     $emailSegments = explode("@", $_SESSION['stuLogin']['email']);
26.
27.     //If the email address from google does not match the GET variable (tamper detection of GET
28.     //variables), then redirect.
29.     if(strtoupper($emailSegments[0]) != strtoupper($_GET['stuCode']))
30.     {
31.         header( 'Location:http://11wha.ashvillecomputing.co.uk/Project/googleAuth/Logout' );
32.         exit();
33.     }
34. }
35.
36. ?>
37.
38. <!doctype html>
39. <html>
40.     <head>
```

```

41. <title><?php if(isset($_GET['stuCode'])) echo strtoupper($_GET['stuCode']) . " - "; ?>Student Card</title>
42. <link rel="stylesheet" type="text/css" href="main.css">
43. <link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro" rel="stylesheet">
44. <link rel="icon" href="../../Project/Resources/favicon.png" type="image/x-icon"/>
45. <meta name="viewport" content="width=device-width, initial-scale=1.0">
46. <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
47. <script>
48.     $( document ).ready( function () {
49.
50.         //Fade in the main division.
51.         $( "#bodyDiv" ).fadeIn( 150 );
52.
53.         //If the logout button is clicked, then visit the login page with the logout variable set.
54.         $( "#btnLogOut" ).click( function () {
55.             $( "#bodyDiv" ).fadeOut( 150 );
56.             setTimeout(
57.                 function () {
58.                     window.location.href = "../../Project/googleAuth/LogOut.php";
59.                 }, 150 );
60.             } );
61.         } );
62.     </script>
63. </head>
64.
65. <header style="padding-top: 32px;">
66.     <h1><?php if(isset($_GET['stuCode'])) echo strtoupper($_GET['stuCode']) . " - "; ?>Student Card</h1>
67. </header>
68.
69. <body>
70.     <div id = "bodyDiv">
71.     <?php
72.
73.         //Make a new database connection.
74.         $conn = dbConnect();
75.
76.         //Prepare and execute a statement to select the user with the matching student code from the database
77.         $sqlQuery = $conn->prepare( "SELECT surname, forename FROM tblStudents WHERE studentCode = ?" );
78.         $sqlQuery->bind_param("s", $stuCode);
79.         $stuCode = strtoupper($_GET['stuCode']);
80.         $sqlQuery->execute();
81.
82.         //Fetch the SQL return object.

```

```

83.     $result = $sqlQuery->get_result();
84.
85.     //Convert the SQL return object to an assoc. array.
86.     $dataRow = $result->fetch_assoc();
87.
88.     //Then, print out the QR code, using the QR code creation API from qrserver.com.
89.     ?>
90.
91.     <div class = "loginWrap">
92.         <h1><?php echo $stuCode . ": " . $dataRow['forename'] . " " . $dataRow['surname'];?></h1>
93.         <a href = "https://api.qrserver.com/v1/create-qr-code/?size=350x350&data=<?php echo $stuCode;?>" download>
94.             <img src = "https://api.qrserver.com/v1/create-qr-code/?size=350x350&data=<?php echo $stuCode;?>"
95.                 class = "downloadQuery" width = 350px height = 350px>
96.             <p class = "btn center">DOWNLOAD</p>
97.         </a>
98.         <h2>Tracking Data:</h2>
99.         <div>
100.            <table class = "data center">
101.                <?php
102.
103.                //Prepare and execute a statement to fetch the tracking information from the tracking table,
104.                //where the student code matches that of the currently logged in user.
105.                $sqlQuery = $conn->prepare( "SELECT locationID, regTime FROM tblStuStaffLocLink WHERE studentCode = ? ORDER BY locationID" );
106.                $sqlQuery->bind_param("s", $stuCode);
107.                $stuCode = strtoupper($_GET['stuCode']);
108.                $sqlQuery->execute();
109.
110.                //Fetch and convert the SQL return object into an assoc. array.
111.                $result = $sqlQuery->get_result();
112.                $data = $result->fetch_all();
113.
114.                //Define an array of the location names.
115.                $locNames = array('Soothill Foyer (S)', 'Pot Bank', 'Long Liberty', 'Beckwithshaw', 'Soothill Foyer (F)');
116.
117.                //A Looping counter to track our progress through the printing process.
118.                $count = 0;
119.
120.                //For each location, print out the corresponding registration time. If no time
121.                //exists yet, then print a dash.
122.                for($i = 0; $i < 5; $i++)
123.                {
124.                    //If we have not yet printed all the locations that we have records for, then print

```

```

125.         //out the registration time in the row.
126.         if($count < count($data))
127.         {
128.             //If the location matches $i (Since it may be possible that registrations //occoured out of order)
129.             if($data[$count][0] == $i)
130.             { ?>
131.
132.                 <tr>
133.                     <td><?php echo $locNames[$i];?></td>
134.                     <td><?php echo substr($data[$count][1],11,5); ?></td>
135.                 </tr>
136.
137.                 <?php
138.                 $count++;
139.             }
140.         }
141.         //Else, we don't have data for this location yet, so just print a dash.
142.         else
143.         { ?>
144.
145.             <tr>
146.                 <td><?php echo $locNames[$i];?></td>
147.                 <td><?php echo " - " ?></td>
148.             </tr>
149.
150.             <?php }
151.         }
152.     ?>
153. </table>
154. </div>
155.
156. </div>
157.
158. <div class = "btn center" id = "btnLogOut">
159.     Log Out
160. </div>
161. </div>
162.</body>
163.</html>

```

---

../PROJECT/STAFFCARD.PHP

This page provides further detail about each staff member.

```
1. <?php
2. //Start the session
3. session_start();
4.
5. //Enable Error reporting for debug
6. error_reporting( E_ALL );
7. ini_set( 'display_errors', 1 );
8.
9. //Defining the secure access parameter means that any config or functional files
10. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
11. define( 'secureAccessParameter', true );
12. require '../Project/Tools/config.php';
13. require '../Project/Tools/walkFunctions.php';
14.
15. //If the student is not logged in, or if a student code has not been defined, then redirect to the
16. //login area.
17. if(!isset($_SESSION['stuLogin']) or !isset($_GET['stuCode']))
18. {
19.     header( 'Location:http://11wha.ashvillecomputing.co.uk/Project/Student' );
20.     exit();
21. }
22. else
23. {
24.     //Get the segments of the email address.
25.     $emailSegments = explode("@", $_SESSION['stuLogin']['email']);
26.
27.     //If the email address from google does not match the GET variable (tamper detection of GET
28.     //variables), then redirect.
29.     if(strtoupper($emailSegments[0]) != strtoupper($_GET['stuCode']))
30.     {
31.         header( 'Location:http://11wha.ashvillecomputing.co.uk/Project/googleAuth/Logout' );
32.         exit();
33.     }
34. }
35.
36. ?>
37.
38. <!doctype html>
```

```

39. <html>
40.   <head>
41. <title><?php if(isset($_GET['stuCode'])) echo strtoupper($_GET['stuCode']) . " - "; ?>Student Card</title>
42.   <link rel="stylesheet" type="text/css" href="main.css">
43.   <link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro" rel="stylesheet">
44.   <link rel="icon" href="../../Project/Resources/favicon.png" type="image/x-icon"/>
45.   <meta name="viewport" content="width=device-width, initial-scale=1.0">
46.   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
47.   <script>
48.     $( document ).ready( function () {
49.
50.       //Fade in the main division.
51.       $( "#bodyDiv" ).fadeIn( 150 );
52.
53.       //If the logout button is clicked, then visit the login page with the logout variable set.
54.       $( "#btnLogOut" ).click( function () {
55.         $( "#bodyDiv" ).fadeOut( 150 );
56.         setTimeout(
57.           function () {
58.             window.location.href = "../../Project/googleAuth/LogOut.php";
59.           }, 150 );
60.       } );
61.     } );
62.   </script>
63. </head>
64.
65. <header style="padding-top: 32px;">
66.   <h1><?php if(isset($_GET['stuCode'])) echo strtoupper($_GET['stuCode']) . " - "; ?>Student Card</h1>
67. </header>
68.
69. <body>
70.   <div id = "bodyDiv">
71.     <?php
72.
73.     //Make a new database connection.
74.     $conn = dbConnect();
75.
76.     //Prepare and execute a statement to select the user with the matching student code from the database
77.     $sqlQuery = $conn->prepare( "SELECT surname, forename FROM tblStudents WHERE studentCode = ?" );
78.     $sqlQuery->bind_param("s", $stuCode);
79.     $stuCode = strtoupper($_GET['stuCode']);
80.     $sqlQuery->execute();

```

```

81.
82. //Fetch the SQL return object.
83. $result = $sqlQuery->get_result();
84.
85. //Convert the SQL return object to an assoc. array.
86. $dataRow = $result->fetch_assoc();
87.
88. //Then, print out the QR code, using the QR code creation API from qrserver.com.
89. ?>
90.
91. <div class = "loginWrap">
92.     <h1><?php echo $stuCode . ": " . $dataRow['forename'] . " " . $dataRow['surname'];?></h1>
93.     <a href = "https://api.qrserver.com/v1/create-qr-code/?size=350x350&data=<?php echo $stuCode;?>" download>
94.         <img src = "https://api.qrserver.com/v1/create-qr-code/?size=350x350&data=<?php echo $stuCode;?>"
95.             class = "downloadQuery" width = 350px height = 350px>
96.         <p class = "btn center">DOWNLOAD</p>
97.     </a>
98.     <h2>Tracking Data:</h2>
99.     <div>
100.         <table class = "data center">
101.             <?php
102.
103.                 //Prepare and execute a statement to fetch the tracking information from the tracking table,
104.                 //where the student code matches that of the currently logged in user.
105.                 $sqlQuery = $conn->prepare( "SELECT locationID, regTime FROM tblStuStaffLocLink WHERE studentCode = ? ORDER BY locationID" );
106.                 $sqlQuery->bind_param("s", $stuCode);
107.                 $stuCode = strtoupper($_GET['stuCode']);
108.                 $sqlQuery->execute();
109.
110.                 //Fetch and convert the SQL return object into an assoc. array.
111.                 $result = $sqlQuery->get_result();
112.                 $data = $result->fetch_all();
113.
114.                 //Define an array of the location names.
115.                 $locNames = array('Soothill Foyer (S)', 'Pot Bank', 'Long Liberty', 'Beckwithshaw', 'Soothill Foyer (F)');
116.
117.                 //A Looping counter to track our progress through the printing process.
118.                 $count = 0;
119.
120.                 //For each location, print out the corresponding registration time. If no time
121.                 //exists yet, then print a dash.
122.                 for($i = 0; $i < 5; $i++)

```

```

123.         {
124.             //If we have not yet printed all the locations that we have records for, then print
125.             //out the registration time in the row.
126.             if($count < count($data))
127.             {
128.                 //If the location matches $i (Since it may be possible that registrations occurred out of order)
129.                 if($data[$count][0] == $i)
130.                 { ?>
131.
132.                     <tr>
133.                         <td><?php echo $locNames[$i];?></td>
134.                         <td><?php echo substr($data[$count][1],11,5); ?></td>
135.                     </tr>
136.
137.                     <?php
138.                     $count++;
139.                 }
140.             }
141.             //Else, we don't have data for this location yet, so just print a dash.
142.             else
143.             { ?>
144.
145.                 <tr>
146.                     <td><?php echo $locNames[$i];?></td>
147.                     <td><?php echo " - " ?></td>
148.                 </tr>
149.
150.                 <?php }
151.             }
152.             ?>
153.         </table>
154.     </div>
155.
156. </div>
157.
158. <div class = "btn center" id = "btnLogout">
159.     Log Out
160. </div>
161. </div>
162. </body>
163. </html>

```



---

## AJAX & API BACKEND PAGES

The following pages carry out functions on the server, mainly database orientated action, that return a structured response.

.....  
[../PROJECT/AJAX/GETLOGDATA.PHP](#)

Returns a JSON object containing the formatted content of the database table containing tracking data.

```
1. <?php
2. //Start the session
3. session_start();
4.
5. //Enable Error reporting for debug
6. error_reporting( E_ALL );
7. ini_set( 'display_errors', 1 );
8.
9. //Defining the secure access parameter means that any config or functional files
10. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
11. define( 'secureAccessParameter', true );
12. require '../Project/Tools/config.php';
13.
14. //If the session variable for login is set, it means a user has been authenticated.
15. if ( isset( $_SESSION[ 'login' ] ) ) {
16.
17.     //Create a new database connection. The dbConnect function is defined in the config file.
18.     $conn = dbConnect();
19.
20.     //Define an associative array which will be used to store an organised store of all tracking
21.     //log data. This array will be returned later as a JSON string.
22.     $returnArray = array("logs" => array(array("stuCode" => '', "locID" => 0, "year" => 0, "regTime" => "")), "checkStaff" => array(a
rray(), array(), array(), array(), array()), "stuCount" => 0, "serverTime" => '');
23.
24.     //Define a pointer to track our position in the send array we will return.
25.     $returnArrayIndex = 0;
26.
27.     //Prepare a statement to select all our tracking data from two tables in our database. An inner join
28.     //will be performed on two tables - 'tblStudents' and a link table 'tblStuStaffLocLink'. This inner join
29.     //will return all students who have registered at a location, including that location and the registering
30.     //staff member. This will ensure that the database remains in normal form, for max efficiency.
31.     $sqlQuery = $conn-
>prepare( "SELECT tblStuStaffLocLink.studentCode, tblStuStaffLocLink.locationID,tblStuStaffLocLink.regTime, tblStudents.yearGroup FRO
```

```

M tblStuStaffLocLink INNER JOIN tblStudents ON tblStuStaffLocLink.studentCode = tblStudents.studentCode ORDER BY tblStuStaffLocLink.s
tudentCode" );
32.
33.     //Execute the statment.
34.     $sqlQuery->execute();
35.
36.     //Fetch the SQL return object.
37.     $result = $sqlQuery->get_result();
38.
39.     //If there is more than one returned row, then...
40.     if ( $result->num_rows > 0 ) {
41.
42.
43.         while ( $dataRow = $result->fetch_assoc() ) {
44.
45.             //If the student code from the SQL data row does is not equal to the student code stored
46.             //in the current index of our return array, it means that we are taking into consideration a new student.
47.             //Since many students can register at many locations, there will be many data rows where the student
48.             //code is the same for different locations. Since our returned SQL object is ordered by
49.             //student code, it means we can simply check if the new row's student code matches the last
50.             //index of the return array to determine wether we need to make a new entry to the array.
51.             if ( $dataRow[ 'studentCode' ] != $returnArray['logs'][$returnArrayIndex][ 'stuCode' ] )
52.             {
53.                 //Increment the index pointer by one. This will make a new 'entry' to the array.
54.                 $returnArrayIndex++;
55.
56.                 //Write the student details and location information to the return array.
57.                 $returnArray['logs'][$returnArrayIndex][ 'stuCode' ] = $dataRow[ 'studentCode' ];
58.                 $returnArray['logs'][$returnArrayIndex][ 'year' ] = $dataRow[ 'yearGroup' ];
59.                 $returnArray['logs'][$returnArrayIndex][ 'locID' ] = $dataRow[ 'locationID' ];
60.                 $returnArray['logs'][$returnArrayIndex][ 'regTime' ] = $dataRow[ 'regTime' ];
61.             }
62.
63.             //If the registration location ID currently stored for that student is less than the new SQL
64.             //data row's location ID, then overwrite the location ID with the new SQL data row's value.
65.             //This ensures that only the most current location ID is returned for each student in the array.
66.             if($returnArray['logs'][$returnArrayIndex][ 'locID' ] < $dataRow[ 'locationID' ])
67.             {
68.                 //Overwrite location ID and registration time.
69.                 $returnArray['logs'][$returnArrayIndex][ 'locID' ] = $dataRow[ 'locationID' ];
70.                 $returnArray['logs'][$returnArrayIndex][ 'regTime' ] = $dataRow[ 'regTime' ];
71.             }

```

```

72.     }
73. }
74.
75. //Since the array always has an empty entry at index 0, remove the first entry.
76. array_shift($returnArray['logs']);
77.
78. //Prepare a statement to select the staffCodes and location IDs from the link table.
79. $sqlQuery = $conn->prepare( "SELECT staffCode, locationID FROM tblStuStaffLocLink ORDER BY staffCode" );
80.
81. //Execute the statment.
82. $sqlQuery->execute();
83.
84. //Fetch the SQL return object.
85. $result = $sqlQuery->get_result();
86.
87. //Define a tracking array. This will store information about wether we
88. //have written any information to the checkstaff array, based on the index
89. //of Location ID. This prevents zero-index errors in the comparison later on,
90. //where we must check the previous index of the checkstaff array with the current
91. //index. Without first knowing if the array is empty, we cannot know wether we can
92. //carry out a comparison operation.
93. $first = array(true,true,true,true,true);
94.
95. //If there is more than one returned row, then...
96. if($result->num_rows > 0)
97. {
98.     //While a new row can be read, loop.
99.     while($dataRow = $result->fetch_assoc())
100.     {
101.         //If this is the first time any staff code will be written to the array, then...
102.         if($first[$dataRow['locationID']])
103.         {
104.             //Push that staff code to the staff information array.
105.             array_push($returnArray['checkStaff'][$dataRow['locationID']], $dataRow['staffCode']);
106.
107.             //Now, the location has been written to! Set the first pointer to false.
108.             $first[$dataRow['locationID']] = false;
109.         }
110.
111.         //Else, if the new staff code from the SQL data row does not equal the previous index of the
112.         //staff information array, then push that staff member to the array. This ensures that each
113.         //staff code is only written to the array once, since our SQL return object is sorted by staff code.

```

```

114.         else if($returnArray['checkStaff'][$dataRow['locationID']][count($returnArray['checkStaff'][$dataRow['locationID']
115.         ]) - 1] != $dataRow['staffCode'])
116.         {
117.             //Push that staff code to the staff information array.
118.             array_push($returnArray['checkStaff'][$dataRow['locationID']], $dataRow['staffCode']);
119.         }
120.     }
121.
122.     //Prepare a statment to select the number of students from the student table.
123.     $sqlQuery = $conn->prepare( "SELECT COUNT(*) studentCode FROM tblStudents" );
124.
125.     //Execute the statment.
126.     $sqlQuery->execute();
127.
128.     //Fetch the SQL return object.
129.     $result = $sqlQuery->get_result();
130.
131.     //Convert the SQL return object into an associative array.
132.     $data = $result->fetch_all();
133.
134.     //Write the student count to the return array.
135.     $returnArray[ 'stuCount' ] = $data[ 0 ][ 0 ];
136.
137.     //Write the current server time to the return array.
138.     $returnArray[ 'serverTime' ] = date('Y-m-d\TH:i:s');
139.
140.     //Echo/Return the encoded JSON string.
141.     echo json_encode( $returnArray );
142.
143.     //Close the SQL query.
144.     $sqlQuery->close();
145.
146.     //Close the database connection.
147.     dbClose();
148.
149.     //Else, if the user has not logged in, then display an error message and exit.
150. } else {
151.     echo( "Improper Access" );
152.     exit();
153. }
154.

```

155.      ?>

---

../PROJECT/AJAX/PRINTLOGTABLE.PHP

Returns a HTML string which contains the formatted, ordered and search query specific content of the database table containing tracking data.

```
1. <?php
2. //Start the session
3. session_start();
4.
5. //Enable Error reporting for debug
6. error_reporting( E_ALL );
7. ini_set( 'display_errors', 1 );
8.
9. //Defining the secure access parameter means that any config or functional files
10. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
11. define( 'secureAccessParameter', true );
12. require '.../Project/Tools/config.php';
13.
14. //If the session variable for login is set, it means a user has been authenticated.
15. if ( isset( $_SESSION[ 'login' ] ) ) {
16.
17.     //Define a database connection
18.     $conn = dbConnect();
19.
20.     //Define variables for use during sorting, printing and searching.
21.     $prevStu = null;
22.     $counter = 0;
23.     $logArray = array();
24.     $yearGroup = null;
25.     $numPointer = null;
26.     $tableHeaderArray = array( "Soothill Foyer", "Pot Bank", "Long Liberty", "Beckwithshaw", "Soothill Foyer" );
27.
28.     //Instantiate a new instance of the DynamicSQLHelper object - an SQL prepared statement parameter helper class.
29.     $dynamicSQLHelper = new DynamicSQLHelper();
30.
31.     //Store the column sort option from the GET param ['order']
32.     $columnSortOptions = $_GET[ 'order' ];
33.
34.     //Define a number of template SQL strands from which we will build our final SQL string.
35.     $sqlName = "(tblStudents.forename LIKE ? OR tblStudents.surname LIKE ?)";
```

```

36. $sqlStuCode = "tblStudents.studentCode LIKE ?";
37. $sqlYear = "tblStudents.yearGroup = ?";
38.
39. //Define the main SQL statement to select student and tracking data from the database.
40. $sql = "SELECT tblStudents.studentCode, tblStudents.surname, tblStudents.forename, tblStudents.yearGroup, tblStuStaffLocLink.locationID, tblStuStaffLocLink.regTime FROM tblStudents LEFT JOIN tblStuStaffLocLink ON tblStudents.studentCode = tblStuStaffLocLink.studentCode";
41.
42. //If a year has been defined or a name search term has been defined, add to the main SQL.
43. if ( isset( $_GET[ 'year' ] ) or isset( $_GET[ 'query' ] ) ) $sql .= " WHERE ";
44.
45. //If a year has been defined, then add an SQL strand to query against yearGroup, and
46. //add a param to hold the yearGroup value.
47. if ( isset( $_GET[ 'year' ] ) ) {
48.     $dynamicSQLHelper->addStrand( $sqlYear );
49.     $dynamicSQLHelper->addParam( 's', $_GET[ 'year' ] );
50. }
51.
52. //If a name search term is defined, then...
53. if ( isset( $_GET[ 'query' ] ) ) {
54.
55.     //Get each individual string, split at each space.
56.     $queryStrings = explode( " ", $_GET[ 'query' ] );
57.
58.     //For each query string, if it contains a number, then set numPointer to equal the index of the
59.     //number-containing-string in the query string array.
60.     for ( $i = 0; $i < count( $queryStrings ); $i++ ) {
61.         if ( strpos( $queryStrings[ $i ], "0123456789" ) ) $numPointer = $i;
62.     }
63.
64.     //If a number was entered, then override the name search as we know that the user
65.     //has entered a studentCode directly instead. A student code has a more definite result
66.     //that a name search.
67.     if ( !is_null( $numPointer ) ) {
68.         $dynamicSQLHelper->addStrand( $sqlStuCode );
69.         $dynamicSQLHelper->addParam( 's', $dynamicSQLHelper->buildLike( $queryStrings[ $numPointer ] ) );
70.     } else {
71.         //For every term, add strand and term values to the SQL helper object.
72.         foreach ( $queryStrings as $term ) {
73.             $dynamicSQLHelper->addStrand( $sqlName );
74.             $dynamicSQLHelper->addParam( 's', $dynamicSQLHelper->buildLike( $term ) );
75.             $dynamicSQLHelper->addParam( 's', $dynamicSQLHelper->buildLike( $term ) );

```

```

76.     }
77.     }
78. }
79.
80. //Get all strands.
81. $sql .= $dynamicSQLHelper->getStrands();
82.
83. //No matter what, order by studentCode.
84. $sql .= " ORDER BY studentCode";
85.
86. //Prepare the SQL server with our statement.
87. $sqlQuery = $conn->prepare( $sql );
88.
89. //If a year or name search term has been defined, then call 'call_user_func_array' inbuild function.
90. //This function passes an array to, and then calls, a user defined method. In this case, we are passing the array of
91. //parameters stored within the dynamicSQLHelper object to the inbuild sqlQuery 'bind_param' method. This method
92. //normally accepts a range of parameters in the form of string, which we could not pass to it directly since we do not
93. //know the number of parameters we will pass at the beginning of each run. By using the 'call_user_func_array' inbuild //function
94. //, we can have any number of params, since an array we define ACTS as the params themselves.
95. if ( isset( $_GET[ 'year' ] ) or isset( $_GET[ 'query' ] ) ) call_user_func_array( array( $sqlQuery, 'bind_param' ), $dynamicSQLHel
per->getParams() );
96.
97. //Execute the query and get the result object. Convert the result object to an accociative array.
98. $sqlQuery->execute();
99. $result = $sqlQuery->get_result();
100. $data = $result->fetch_all();
101.
102. //For each row of data returned, build an array to store the log data.
103. for ( $i = 0; $i < $result->num_rows; $i++ ) {
104.     //Since the same student could have multiple rows retuned for them (the nature of a left join), we must
105.     //check if we are viewing a new student, or a more recent registration event for the same student, by
106.     //comparing the unique key of 'studentCode'
107.     if ( $data[ $i ][ 0 ] != $prevStu ) {
108.
109.         //Set the previous studentCode to the current studentCode.
110.         $prevStu = $data[ $i ][ 0 ];
111.
112.         //Increment loop counter by one. This reference the next index of the array.
113.         $counter++;
114.
115.         //Add a new entry to the log array.

```

```

116.         $logArray[ $counter ] = array( "stuCode" => '', "surname" => '', "forename" => '', "year" => '', "logData" => arra
y( '', '', '', '', '' ) );
117.
118.         //Add the registration event details.
119.         $logArray[ $counter ][ 'stuCode' ] = $data[ $i ][ 0 ];
120.         $logArray[ $counter ][ 'surname' ] = $data[ $i ][ 1 ];
121.         $logArray[ $counter ][ 'forename' ] = $data[ $i ][ 2 ];
122.         $logArray[ $counter ][ 'year' ] = $data[ $i ][ 3 ];
123.     }
124.
125.         //Format and add the registration time to the log array, in the format [HH:MM].
126.         $logArray[ $counter ][ 'logData' ][ $data[ $i ][ 4 ] ] = substr( $data[ $i ][ 5 ], 11, 5 );
127.     }
128.     ?>
129.
130.     <table class="data center">
131.         <tr class="row1">
132.             <td class="col1" data-row="1">Surname</td>
133.             <td class="col2 thickRight" data-row="1">Forename</td>
134.
135.             <?php
136.             //For each column (since there are five locations), add the location name to each header. Then, if any
137.             //column sort options are defined, echo those options back into the table html 'data' params and
138.             //print a unicode arrow to signify the sort direction.
139.             for ( $i = 0; $i < 5; $i++ ) {
140.                 ?>
141.                 <td id="item" class="col<?php echo $i + 3; ?>" data-row="1" data-loc="<?php echo $i; ?>" data-sort="
142.                 <?php
143.                     if(!empty($columnSortOptions)) echo $columnSortOptions[1];
144.                 ?>
145.                 ">
146.                 <?php
147.
148.                     echo $tableHeaderArray[ $i ];
149.
150.                     if ( !empty($columnSortOptions) && $columnSortOptions[ 0 ] == $i ) {
151.                         if ( strtoupper( $columnSortOptions[ 1 ] ) == "A" )echo " ↑";
152.                         else echo " ↓";
153.                     }
154.
155.                 ?>
156.

```



```

157.         </td>
158.
159.         <?php } ?>
160.
161.     </tr>
162.
163. <?php
164.
165.     //Define an array of saved rows.
166.     $savedArray = array();
167.
168.     //The following block takes each log and print it according to the column sorting options.
169.     //This allows for the ordering of students based on wether they have or have not registered
170.     //at a specific location.
171.     for($i = 1; $i <= count($logArray); $i++)
172.     {
173.         //If there are sorting options defined...
174.         if(!empty($columnSortOptions))
175.         {
176.             //If the student has not registered at the checkpoint id stated in the sort options...
177.             if(empty($logArray[$i]['logData'][$columnSortOptions[0]]))
178.             {
179.                 //If the column sorting is set to descending, then Print the log as a HTML table row.
180.                 //Else, save the log for printing later.
181.                 if($columnSortOptions[1] == "D") PrintLog($i, $logArray);
182.                 else SaveLog($i, $savedArray);
183.             }
184.             else
185.             {
186.                 //If the column sorting is set to ascending, then Print the log as a HTML table row.
187.                 //Else, save the log for printing later.
188.                 if($columnSortOptions[1] == "D") SaveLog($i, $savedArray);
189.                 else PrintLog($i, $logArray);
190.             }
191.         }
192.         //Else, print the log as no sorting has been defined.
193.         else
194.         {
195.             PrintLog($i, $logArray);
196.         }
197.     }
198.

```

```

199.         //For each saved log, print it out.
200.         for($i = 0; $i < count($savedArray); $i++)
201.         {
202.             PrintLog($savedArray[$i], $logArray);
203.         }
204.
205.         ?>
206.
207.
208.
209.         </table>
210.         <?php
211.
212.         //Else, the user must be accessing improperly, so deny access...
213.         } else {
214.             echo( "Improper Access" );
215.             exit();
216.         }
217.
218.
219.         //A function to print out a HTML table row from a tracking log array.
220.         function PrintLog( $logRef, $logArray ) {
221.             ?>
222.             <tr class="row"<?php echo $logRef + 1;?>">
223.                 <td class="col1 y"<?php echo $logArray[$logRef]['year'];?>" data-row="<?php echo $logRef + 1;?>">
224.                     <?php echo $logArray[$logRef]['surname'];?>
225.                 </td>
226.                 <td class="col2 thickRight y"<?php echo $logArray[$logRef]['year'];?>" data-row="<?php echo $logRef + 1;?>">
227.                     <?php echo $logArray[$logRef]['forename'];?>
228.                 </td>
229.
230.             <?php
231.
232.             //For each location, print out the registration time if log data exists in the log array. Else print
233.             //a red dash.
234.             for($j = 0; $j < 5; $j++) {
235.
236.                 if(empty($logArray[$logRef]['logData'][$j]))
237.                 {?>
238.
239.                 <td class="col"<?php echo ($j + 3);?>" red" data-row="<?php echo $logRef + 1;?>">-</td>
240.

```

```

241.         <?php } else { ?>
242.
243.         <td class="col<?php echo ($j + 3);?> green" data-row="<?php echo $logRef + 1;?>">
244.             <?php echo $logArray[$logRef]['logData'][$j];?>
245.         </td>
246.
247.         <?php } } ?>
248.
249.     </tr>
250.
251.     <?php
252.     }
253.
254.     //Function to push a log array index reference to the saved log array.
255.     function SaveLog( $logRef, & $savedLogs ) {
256.         array_push( $savedLogs, $logRef );
257.     }
258.
259.     class DynamicSQLHelper {
260.         //Declare Private Variables to store our 'type' and 'Values';
261.         private $queryStrands = array(), $values = array(), $types = '';
262.
263.         //Method to append a new 'value' to array and new 'type' to string.
264.         public
265.         function addParam( $type, $value ) {
266.             $this->values[] = $value;
267.             $this->types .= $type;
268.         }
269.
270.
271.         //Method to add a new SQL strand to 'queryStrands'.
272.         public
273.         function addStrand( $strand ) {
274.             $this->queryStrands[] = $strand;
275.         }
276.
277.         //Return a new array, which contains references to all the values we stored whilst running through the program, and our 't
ype' string.
278.         public
279.         function getParams() {
280.             $refs = array();
281.

```

```

282.         foreach ( $this->values as $key => $value )
283.             $refs[ $key ] = & $this->values[ $key ];
284.
285.         return array_merge( array( $this->types ), $refs );
286.     }
287.
288.     //Method to return all the SQL strands in a formtted manor, with an AND between each strand.
289.     public
290.     function getStrands() {
291.         return implode( " AND ", $this->queryStrands );
292.     }
293.
294.     //Method to build the SQL like statement from a search param.
295.     public
296.     function buildLike( $string ) {
297.         return $string . '%';
298.     }
299. }
300. ?>

```

---

../PROJECT/AJAX/UPDATEWALKSTATUS.PHP

Updates the status of the sponsored walk.

```

1. <?php
2. session_start();
3.
4. error_reporting( E_ALL );
5. ini_set( 'display_errors', 1 );
6.
7. //Make an accociative array to store our return object.
8. $returnJSON = array( "status" => "", "message" => "" );
9.
10. if(isset($_SESSION['login']) && $_SESSION['level'] == 1 && isset($_POST[ 'status' ]) && isset($_POST[ 'startTime' ]))
11. {
12.     //Defining the secure access parameter means that any config or functional files
13.     //can only be accessed via a 'require' rather than directly through URL and HTTP, improving saftey.
14.     define( 'secureAccessParameter', true );
15.     require '../Project/Tools/config.php';
16.
17.     //Define a database connection.

```

```

18. $conn = dbConnect();
19.
20. $sqlQuery = $conn->prepare("UPDATE tblWalkStatus SET status = ?, startTime = ? WHERE walkID = 1");
21. $sqlQuery->bind_param( "is", $status, $startString);
22.
23. $status =$_POST[ 'status' ];
24. $startString =$_POST[ 'startTime' ];
25. $sqlQuery->execute();
26.
27. if($status == 0)
28. {
29.     $sqlQuery = $conn->prepare( "TRUNCATE tblStuStaffLocLink" );
30.     $sqlQuery->execute();
31. }
32.
33. $sqlQuery->close();
34. dbClose();
35.
36. //Write messages to validate the success of the operation.
37. $returnJSON[ 'status' ] = "OK";
38. $returnJSON[ 'message' ] = "Status Updated sucessfully!";
39. } else {
40.     //Write error messages.
41.     $returnJSON[ 'status' ] = "ERROR";
42.     $returnJSON[ 'message' ] = "Access Denied!";
43. }
44. //Return the JSON response.
45. echo json_encode( $returnJSON );
46. ?>

```

---

../PROJECT/AJAX/INVITENEWUSER.PHP

Invites a new user to view the web app, by sending them an email containing a link which is appending with a unique ID for each invite.

```

1. <?php
2. //Start the session
3. session_start();
4.
5. //Enable Error reporting for debug
6. error_reporting( E_ALL );
7. ini_set( 'display_errors', 1 );

```

```

8.
9. //Defining the secure access parameter means that any config or functional files
10. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
11. define( 'secureAccessParameter', true );
12. require '../Project/Tools/config.php';
13. require '../Project/Tools/PHPMailer/src/Exception.php';
14. require '../Project/Tools/PHPMailer/src/PHPMailer.php';
15. require '../Project/Tools/PHPMailer/src/SMTP.php';
16.
17. //Create an alias to the PHP mailer namespace.
18. use PHPMailer\ PHPMailer\ PHPMailer;
19. use PHPMailer\ PHPMailer\ Exception;
20.
21. //Make an associative array to store our return object.
22. $returnJSON = array( "status" => "", "message" => "" );
23.
24. //If the user is logged in, and the user is high level, and the PHP POST params have been set.
25. if ( isset( $_SESSION[ 'login' ] ) && $_SESSION[ 'level' ] == 1 && isset( $_POST[ 'newUsername' ] ) && isset( $_POST[ 'newLevel' ] ) ) {
26.
27.     //Create a new database connection. The dbConnect function is defined in the config file.
28.     $conn = dbConnect();
29.
30.     //Set variables to contain the POST params.
31.     $newUsername = strtoupper( $_POST[ 'newUsername' ] );
32.     $newLevel = $_POST[ 'newLevel' ];
33.
34.     //Define a boolean to flag if the user already exists.
35.     $exists = false;
36.
37.     //Prepare a statement to select all the usernames from the current admin user table.
38.     $sqlQuery = $conn->prepare( "SELECT uName FROM tblAdminUsers" );
39.
40.     //Execute the statement.
41.     $sqlQuery->execute();
42.
43.     //Fetch the SQL return object.
44.     $result = $sqlQuery->get_result();
45.
46.     //If there is more than one returned row, then...
47.     if ( $result->num_rows > 0 ) {
48.

```

```

49.      //While a new row can be read, loop.
50.      while ( $dataRow = $result->fetch_assoc() ) {
51.
52.          //If the username we are trying to invite already exists in the user table, then set $exists to true;
53.          if ( $dataRow[ 'uName' ] == $newUname ) {
54.
55.              $exists = true;
56.          }
57.      }
58.  }
59.
60.      //If the user does not already exist, then...
61.      if ( !$exists ) {
62.
63.          //Declare an empty variable to hold our invite code.
64.          $inviteCode = "";
65.
66.          //Declare an array of characters to use in the invite code generation.
67.          $characters = str_split( "0123456789abcdefghijklmnopqrstuvwxyz" );
68.
69.          //Select 10 characters at random and append them to the empty variable.
70.          for ( $i = 0; $i < 10; $i++ )$inviteCode .= $characters[ mt_rand( 0, count( $characters ) - 1 ) ];
71.
72.          //Declare a variable to hold the HTML email we will send with the invite instructions and
73.          //links to the registration page.
74.          $txt = '
75. <html>
76. <head>
77. <title>HTML email</title>
78. </head>
79. <body>
80. <h2>You Have Been Invited To Access The Sponsored Walk Tracker!</h2>
81. <br>
82. <p>Dear ' . $newUname . ',</p>
83. <br>
84. <p>You have been invited to access the sponsored walk tracking suite. From here, you can track the progress of the sponsored walk, an
85. <a href="http://11wha.ashvillecomputing.co.uk/Project/Register.php?inviteCode=' . $inviteCode . '">Click To Register!</a>
86. <br>
87. <p>Best Regards,</p>
88. <p>Sponsored Walk Admin</p>
89. <br>

```

```

90. <p>If your having problems with the button above, Copy and paste this URL: http://11wha.ashvillecomputing.co.uk/Project/Register.php?
    inviteCode=" . \$inviteCode . '</p>
91. </body>
92. </html>';
93.
94.     //Instantiate a new instance of the PHP mailer class. This is an API to send emails
95.     //through a simple PHP server.
96.     $mail = new PHPMailer( true );
97.
98.     //Try to send the email - since there are so many things that could go wrong...
99.     try {
100.
101.         //Declare the parameters for the email, such as the email of the new user, and the
102.         //subject line and body of the email.
103.         $mail->setFrom( 'sponsoredWalkTracking@ashville.co.uk', 'Tracking Invite' );
104.         $mail->addAddress( "$newUname@ashville.co.uk", $newUname );
105.         $mail->isHTML( true );
106.         $mail->Subject = "Invite To Access Sponsored Walk Tracking";
107.         $mail->Body = $txt;
108.
109.         //Send the email.
110.         $mail->send();
111.
112.         //Prepare a statment to insert this new invite code into the database along with the
113.         //new user code and the level of access they will have.
114.         $sqlQuery = $conn->prepare( "INSERT INTO tblInvites (inviteID, userCode, level) VALUES (?,?)" );
115.
116.         //Bind the params to the query.
117.         $sqlQuery->bind_param( "ssi", $inviteCode, $newUname, $newLevel );
118.
119.         //Execute the statment.
120.         $sqlQuery->execute();
121.
122.         //Write messages to validate the success of the operation.
123.         $returnJSON[ 'status' ] = "OK";
124.         $returnJSON[ 'message' ] = "Complete!";
125.
126.         //If an error is thrown, then let the user know what went wrong.
127.     } catch ( Exception $e ) {
128.
129.         //Write error messages.
130.         $returnJSON[ 'status' ] = "ERROR";

```



```

131.             $returnJSON[ 'message' ] = "Mailer Error: " . $mail->ErrorInfo;
132.         }
133.
134.     }
135.     //Else, the user exists.
136.     else {
137.         //Write error messages.
138.         $returnJSON[ 'status' ] = "ERROR";
139.         $returnJSON[ 'message' ] = "The user you are trying to invite already exists!";
140.     }
141.
142.     //Close the SQL query.
143.     $sqlQuery->close();
144.
145.     //Close the database connection.
146.     dbClose();
147.
148.
149.     //Else, if the conditions for operation are not met, then give an error and exit.
150. } else {
151.     //Write error messages.
152.     $returnJSON[ 'status' ] = "ERROR";
153.     $returnJSON[ 'message' ] = "Access Denied!";
154. }
155.
156. //Return the JSON response.
157. echo json_encode( $returnJSON );
158. ?>

```

---

[../PROJECT/AJAX/REMOVEUSER.PHP](#)

Removes a user's ability to access the tracking suite.

```

1. <?php
2. session_start();
3.
4. error_reporting( E_ALL );
5. ini_set( 'display_errors', 1 );
6.
7. //Make an accociative array to store our return object.
8. $returnJSON = array( "status" => "", "message" => "" );

```

```

9.
10. //If the user is logged in, and the user is an admin, and the user has POSTed us a username, then...
11. if ( isset( $_SESSION[ 'login' ] ) && $_SESSION[ 'level' ] == 1 && isset($_POST[ 'uName' ])) {
12.
13.     //Defining the secure access parameter means that any config or functional files
14.     //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
15.     define( 'secureAccessParameter', true );
16.     require '../Project/Tools/config.php';
17.
18.     //Define a database connection.
19.     $conn = dbConnect();
20.
21.     //Define a statement to delete a user from the tracking users table. Bind the username as a param.
22.     $sqlQuery = $conn->prepare( "DELETE FROM tblAdminUsers WHERE uName = ?" );
23.     $sqlQuery->bind_param( "s", $uName );
24.     $uName = $_POST[ 'uName' ];
25.
26.     //Execute the command.
27.     $sqlQuery->execute();
28.
29.     //Close connections.
30.     $sqlQuery->close();
31.     dbClose();
32.
33.     //Write messages to validate the success of the operation.
34.     $returnJSON[ 'status' ] = "OK";
35.     $returnJSON[ 'message' ] = "User was removed sucessfully!";
36. } else {
37.     //Write error messages.
38.     $returnJSON[ 'status' ] = "ERROR";
39.     $returnJSON[ 'message' ] = "Access Denied!";
40. }
41.
42. //Return the JSON response.
43. echo json_encode( $returnJSON );
44. ?>

```

---

../PROJECT/AJAX/UPDATEUSERPERMS.PHP

Updates a user's permissions on the tracking suite.

```
1. <?php
2. session_start();
3.
4. error_reporting( E_ALL );
5. ini_set( 'display_errors', 1 );
6.
7. //Make an accociative array to store our return object.
8. $returnJSON = array( "status" => "", "message" => "" );
9.
10. //If the user is logged in, and the user is an admin, and the user has POSTed us a username and level, then...
11. if ( isset( $_SESSION[ 'login' ] ) && $_SESSION[ 'level' ] == 1 && isset( $_POST[ 'uLevel' ] ) && isset( $_POST[ 'uName' ] ) ) {
12.
13.     //Defining the secure access parameter means that any config or functional files
14.     //can only be accessed via a 'require' rather than directly through URL and HTTP, improving saftey.
15.     define( 'secureAccessParameter', true );
16.     require '../Project/Tools/config.php';
17.
18.     //Define a database connection.
19.     $conn = dbConnect();
20.
21.     //Define a query to update the current level of the user. Bind params for the new level of the
22.     //existing user.
23.     $sqlQuery = $conn->prepare( "UPDATE tblAdminUsers SET level = ? WHERE uName = ?" );
24.     $sqlQuery->bind_param( "is", $level, $uName );
25.     $level = $_POST[ 'uLevel' ];
26.     $uName = $_POST[ 'uName' ];
27.
28.     //Execute the command.
29.     $sqlQuery->execute();
30.
31.     //Close connections.
32.     $sqlQuery->close();
33.     dbClose();
34.
35.     //Write messages to validate the success of the operation.
36.     $returnJSON[ 'status' ] = "OK";
37.     $returnJSON[ 'message' ] = "User was removed sucessfully!";
38. } else {
```

```

39.     //Write error messages.
40.     $returnJSON[ 'status' ] = "ERROR";
41.     $returnJSON[ 'message' ] = "Access Denied!";
42. }
43. //Return the JSON response.
44. echo json_encode( $returnJSON );
45. ?>

```

---

#### ../PROJECT/AJAX/PASSWORDCHECK.PHP

Checks the currently logged in user's password against a string passed to the function. Returns comparison result.

```

1. <?php
2. //Start the session
3. session_start();
4.
5. //Enable Error reporting for debug
6. error_reporting( E_ALL );
7. ini_set( 'display_errors', 1 );
8.
9. //Defining the secure access parameter means that any config or functional files
10. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
11. define( 'secureAccessParameter', true );
12. require '../Project/Tools/config.php';
13.
14. //Make an associative array to store our return object.
15. $returnJSON = array( "status" => "", "message" => "" );
16.
17. //If the user is logged in, and the POST data has been set, then...
18. if (isset( $_SESSION[ 'login' ] ) && isset($_POST[ 'pass' ])) {
19.
20.     //Make a database connection.
21.     $conn = dbConnect();
22.
23.     //Define a query to select the hashed password for the currently logged in user. This
24.     //means that it is only possible to check the password of the currently logged in user, rather
25.     //than having the ability to check the password for any user, a potential security hole.
26.     $sqlQuery = $conn->prepare( "SELECT pass FROM tblAdminUsers WHERE uName = ?" );
27.
28.     //Bind params
29.     $sqlQuery->bind_param( "s", $uName );

```

```

30. $uName = $_SESSION[ 'login' ];
31.
32. //Execute the query.
33. $sqlQuery->execute();
34. $result = $sqlQuery->get_result();
35.
36. //If there was a response
37. if($result->num_rows > 0)
38. {
39.     //Fetch the data from the SQL return object.
40.     $dataRow = $result->fetch_assoc();
41.
42.     //If the hashed version of the password entered matches the hashed version from the
43.     //database, then...
44.     if(password_verify($_POST[ 'pass' ], $dataRow['pass']))
45.     {
46.         //Write messages to validate the success of the operation.
47.         $returnJSON[ 'status' ] = "OK";
48.         $returnJSON[ 'message' ] = "Password is correct!";
49.     }
50.     else
51.     {
52.         //Write error messages.
53.         $returnJSON[ 'status' ] = "ERROR";
54.         $returnJSON[ 'message' ] = "Password is incorrect!";
55.     }
56. }
57.
58. //Close the query and database connection.
59. $sqlQuery->close();
60. dbClose();
61.
62. } else {
63.     //Write error messages.
64.     $returnJSON[ 'status' ] = "ERROR";
65.     $returnJSON[ 'message' ] = "Access Denied!";
66. }
67.
68. //Return the JSON response.
69. echo json_encode( $returnJSON );
70. ?>

```

---

../PROJECT/API/GETTIMETOSTART.PHP

Return the number of seconds until the walk begins.

```
1. <?php
2. //Enable Error reporting for debug
3. error_reporting( E_ALL );
4. ini_set( 'display_errors', 1 );
5.
6. //Defining the secure access parameter means that any config or functional files
7. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
8. define( 'secureAccessParameter', true );
9. require '../Project/Tools/config.php';
10.
11. //Create a new database connection. The dbConnect function is defined in the config file.
12. $conn = dbConnect();
13.
14. //Prepare a statement to select the start time from the walk status table. The start time was
15. //defined by an administrator user on the overview page.
16. $sqlQuery = $conn->prepare( "SELECT startTime FROM tblWalkStatus WHERE walkID = 1" );
17.
18. //Execute the statement.
19. $sqlQuery->execute();
20.
21. //Fetch the SQL return object.
22. $result = $sqlQuery->get_result();
23.
24. //Fetch the first row of data from the SQL return object.
25. $dataRow = $result->fetch_assoc();
26.
27. //Convert the string time to a PHP DateTime object. Store in a variable.
28. $startDateTime = strtotime($dataRow['startTime']);
29.
30. //Make an associative array to store our return object.
31. $returnJSON = array("status" => "OK", "timeToStart" => $startDateTime - time());
32.
33. //Return the start time minus the current time to get the time to start.
34. echo json_encode($returnJSON);
35. ?>
```

---

../PROJECT/API/GETWALKSTATUS.PHP

Returns the status of the walk (Active? Waiting? Inactive?). This uses the existing walkFunctions file to return the walk status.

```
1. <?php
2. error_reporting( E_ALL );
3. ini_set( 'display_errors', 1 );
4.
5. define( 'secureAccessParameter', true );
6. require '.../Project/Tools/config.php';
7. require '.../Project/Tools/walkFunctions.php';
8.
9. echo getWalkStatus();
10. ?>
```

---

../PROJECT/API/NEWDATA.PHP

Adds a new registration event to the tracking data table.

```
1. <?php
2. //Enable Error reporting for debug
3. error_reporting( E_ALL );
4. ini_set( 'display_errors', 1 );
5.
6. //Defining the secure access parameter means that any config or functional files
7. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
8. define( 'secureAccessParameter', true );
9. require '.../Project/Tools/config.php';
10.
11. //Make an associative array to store our return object.
12. $returnJSON = array("status" => "", "message" => "");
13.
14. //Define a database connection.
15. $conn = dbConnect();
16.
17. //If all the GET params are set, then run.
18. if(isset($_GET[ 'stuCode' ]) && isset($_GET[ 'staffCode' ]) && isset($_GET[ 'locID' ]))
19. {
20.     //Prepare a statement to select the student code from the student table. This is a check exists statement. Bind params too.
21.     $sqlQuery = $conn->prepare( "SELECT studentCode FROM tblStudents WHERE studentCode = ?" );
22.     $sqlQuery->bind_param( "s", $stuCode );
```

```

23.     $stuCode = $_GET[ 'stuCode' ];
24.     $staffCode = $_GET[ 'staffCode' ];
25.     $locID = $_GET[ 'locID' ];
26.
27.     //Execute the statement and fetch the SQL return object.
28.     $sqlQuery->execute();
29.     $result = $sqlQuery->get_result();
30.
31.     //If the student exists, then insert a registration event for them.
32.     if ( $result->num_rows > 0 ) {
33.
34.         //Statement to insert tracking data
35.         $sqlQuery = $conn->prepare( "INSERT INTO tblStuStaffLocLink (studentCode, staffCode, locationID) VALUES (?,?,?)" );
36.         $sqlQuery->bind_param( "sss", $stuCode, $staffCode, $locID );
37.         $sqlQuery->execute();
38.
39.         //Write messages.
40.         $returnJSON['status'] = "OK";
41.         $returnJSON['message'] = "Complete!";
42.     }
43.     else
44.     {
45.         //Write messages.
46.         $returnJSON['status'] = "ERROR";
47.         $returnJSON['message'] = "The student you are trying to add new data for does not exist.";
48.     }
49.
50. //Close connections.
51. $sqlQuery->close();
52. dbClose();
53.
54. }
55. else
56. {
57.     //Write messages.
58.     $returnJSON['status'] = "ERROR";
59.     $returnJSON['message'] = "Please ensure you have included all GET variables: 'stuCode', 'staffCode', 'locID'";
60. }
61.
62. //echo the return object as a JSON string .
63. echo json_encode($returnJSON);
64.

```



65. ?>

../PROJECT/API/STAFFEXISTS.PHP

Returns true if a staff member exists in the staff table.

```
1. <?php
2. //Enable Error reporting for debug
3. error_reporting( E_ALL );
4. ini_set( 'display_errors', 1 );
5.
6. //Defining the secure access parameter means that any config or functional files
7. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
8. define( 'secureAccessParameter', true );
9. require '../Project/Tools/config.php';
10.
11. //Make an associative array to store our return object.
12. $returnJSON = array("status" => "", "message" => "", "fullName" => "");
13.
14. //If the GET param is set, then run.
15. if(isset($_GET[ 'staffCode' ]))
16. {
17.     //Make a database connection.
18.     $conn = dbConnect();
19.
20.     //Prepare and execute a statement which returns the staff information which matches the given staff code.
21.     $sqlQuery = $conn->prepare( "SELECT staffCode, forename, surname FROM tblStaff WHERE staffCode = ?" );
22.     $sqlQuery->bind_param( "s", $staffCode );
23.     $staffCode = strtoupper($_GET[ 'staffCode' ]);
24.     $sqlQuery->execute();
25.
26.     //Fetch the SQL return object.
27.     $result = $sqlQuery->get_result();
28.
29.     //If there is more than zero rows, the staff member exists.
30.     if($result->num_rows > 0)
31.     {
32.         //Convert SQL return object to assoc. array.
33.         $data = $result->fetch_assoc();
34.
35.         //Write messages and data to the return array.
```

```

36.     $returnJSON['status'] = "OK";
37.     $returnJSON['fullName'] = $data['forename'] . " " . $data['surname'];
38.     $returnJSON['message'] = "The staff member does exists.";
39. }
40. else
41. {
42.     //Write messages.
43.     $returnJSON['status'] = "ERROR";
44.     $returnJSON['fullName'] = "NULL";
45.     $returnJSON['message'] = "The staff member does not exist.";
46. }
47.
48. //Close connections.
49. $sqlQuery->close();
50. dbClose();
51. }
52. else
53. {
54.     //Write messages.
55.     $returnJSON['status'] = "ERROR";
56.     $returnJSON['fullName'] = "NULL";
57.     $returnJSON['message'] = "Please ensure you have included all GET variables: 'staffCode'";
58. }
59.
60. //echo the return object as a JSON string.
61. echo json_encode($returnJSON);
62. ?>

```

---

../PROJECT/API/STUDENTEXISTS.PHP

Returns true is a student exists in the student table.

```

1. <?php
2. //Enable Error reporting for debug
3. error_reporting( E_ALL );
4. ini_set( 'display_errors', 1 );
5.
6. //Defining the secure access parameter means that any config or functional files
7. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.
8. define( 'secureAccessParameter', true );
9. require '../Project/Tools/config.php';

```

```

10.
11. //Make an associative array to store our return object.
12. $returnJSON = array("status" => "", "message" => "", "fullName" => "", "yearGroup" => "");
13.
14. //If the GET param is set, then run.
15. if(isset($_GET[ 'stuCode' ]))
16. {
17.     //Make a database connection.
18.     $conn = dbConnect();
19.
20.     //Prepare and execute a statement which returns the student information which matches the given student code.
21.     $sqlQuery = $conn->prepare( "SELECT studentCode, surname, forename, yearGroup FROM tblStudents WHERE studentCode = ?" );
22.     $sqlQuery->bind_param( "s", $stuCode );
23.     $stuCode = strtoupper($_GET[ 'stuCode' ]);
24.     $sqlQuery->execute();
25.
26.     //Fetch the SQL return object.
27.     $result = $sqlQuery->get_result();
28.
29.     //If there is more than zero rows, the staff member exists.
30.     if($result->num_rows > 0)
31.     {
32.         //Convert SQL return object to assoc. array.
33.         $data = $result->fetch_assoc();
34.
35.         //Write messages and data to the return array.
36.         $returnJSON['status'] = "OK";
37.         $returnJSON['fullName'] = $data['forename'] . " " . $data['surname'];
38.         $returnJSON['yearGroup'] = $data['yearGroup'];
39.         $returnJSON['message'] = "The student does exists.";
40.     }
41.     else
42.     {
43.         //Write messages.
44.         $returnJSON['status'] = "ERROR";
45.         $returnJSON['fullName'] = "NULL";
46.         $returnJSON['yearGroup'] = "NULL";
47.         $returnJSON['message'] = "The student does not exist.";
48.     }
49.
50.     //Close connections.
51.     $sqlQuery->close();

```

```

52.     dbClose();
53. }
54. else
55. {
56.     //Write messages.
57.     $returnJSON['status'] = "ERROR";
58.     $returnJSON['fullName'] = "NULL";
59.     $returnJSON['yearGroup'] = "NULL";
60.     $returnJSON['message'] = "Please ensure you have included all GET variables: 'stuCode'";
61. }
62.
63. //echo the return object as a JSON string.
64. echo json_encode($returnJSON);
65. ?>

```

---

## TOOLS, RESOURCES AND CONFIGURATION FILES

These files help the operation of all the other files. They could be used by either a front end or back end page.

---

### ../PROJECT/TOOLS/CONFIG.PHP

Contains database connection functions and database credentials.

```

1. <?php
2.
3. //If the secure access parameter is not defined, then kill the process.
4. if ( !defined( 'secureAccessParameter' ) ) die( 'Improper Access' );
5.
6. //Define databas connection crededentials.
7. define( 'DB_USER', 'wha-v25-u-145153' );
8. define( 'DB_PASS', 'z4/6Dh/CN' );
9. define( 'DB_SERVER', '10.16.16.1' );
10. define( 'SERVER_SUBDOMAIN', '11wha' );
11. define( 'SERVER_DOMAIN', 'ashvillecomputing.co.uk' );
12.
13. //A function that returns a super-global database connection.
14. function dbConnect( $server = DB_SERVER, $username = DB_USER, $password = DB_PASS, $database = 'wha-v25-u-
    145153', $link = 'conn' ) {
15.     global $$link;
16.
17.     $$link = new mysqli( $server, $username, $password, $database );

```

```

18.
19.     if ( $$link->connect_error ) {
20.         die( "Connection failed: " . $$link->connect_error );
21.     }
22.
23.     return $$link;
24. }
25.
26. //A function that closes a superglobal database connection.
27. function dbClose( $link = 'conn' ) {
28.     global $$link;
29.
30.     return $$link->close();
31. }
32. ?>

```

---

../PROJECT/TOOLS/WALKFUNCTIONS.PHP

Contains a function to get the status of the sponsored walk.

```

1. <?php
2. //If the secure access parameter is not defined, then kill the process.
3. if ( !defined( 'secureAccessParameter' ) ) die( 'Improper Access' );
4.
5. //A function which returns the current status of the walk.
6. function getWalkStatus() {
7.
8.     //Define an associative array to hold our return params.
9.     $returnJSON = array("status" => "OK", "walkStatus" => "");
10.
11.     //Make a database connection.
12.     $conn = dbConnect();
13.
14.     //Prepare and execute a statement to pull the current walk status from the walkStatus table.
15.     $sqlQuery = $conn->prepare( "SELECT status, startTime FROM tblWalkStatus WHERE walkID = 1" );
16.     $sqlQuery->execute();
17.
18.     //Fetch and convert the SQL return object.
19.     $result = $sqlQuery->get_result();
20.     $dataRow = $result->fetch_assoc();
21.

```

```

22. //If the walk is in the WAITING state (1), then run.
23. if($dataRow['status'] == 1)
24. {
25.     //Create a php date-time object from the SQL timestamp.
26.     $startDateTime = DateTime::createFromFormat('Y-m-d\TH:i', $dataRow['startTime']);
27.
28.     //If the start time has already passed, then the walk upgrades to the ACTIVE state (2). Else, hold at WAITING (1).
29.     if($startDateTime < new DateTime('now'))
30.     {
31.         $returnJSON['walkStatus'] = "2";
32.     }
33.     else
34.     {
35.         $returnJSON['walkStatus']= "1";
36.     }
37. }
38. //Else, the walk is in the INACTIVE state (0).
39. else
40. {
41.     $returnJSON['walkStatus'] = "0";
42. }
43.
44. //Close connections.
45. $sqlQuery->close();
46. dbClose();
47.
48. //Return the return object as a JSON string.
49. return json_encode($returnJSON);
50. }
51. ?>

```

---

../PROJECT/GOOGLEAUTH/GCONFIG.PHP

Contains the ClientID and Client Secret for use in the Google OAuth flow.

```

1. <?php
2.
3. //If the secure access parameter is not set, kill the process.
4. if ( !defined( 'secureAccessParameter' ) )die( 'Improper Access' );
5.
6. // Google App Client Id

```

```

7. define('CLIENT_ID', '113699226954-d13ipgtirln8f5na422jrcq2eb9di472.apps.googleusercontent.com');
8.
9. // Google App Client Secret
10. define('CLIENT_SECRET', '*****');
11.
12. // Google App Redirect Url
13. define('CLIENT_REDIRECT_URL', 'http://11wha.ashvillecomputing.co.uk/Project/googleAuth/Callback.php');
14.
15. ?>

```

---

../PROJECT/GOOGLEAUTH/GOOGLELOGINHELPER.PHP

A PHP object that contains methods to abstract Google OAuth processes. These include: Generation of the OAuth URL, fetching authorisation and access tokens, and fetching user data.

```

1. <?php
2.
3. //Define a google login helper class. This class will be used to provide methods to help the google auth
4. //process. The helper class will help by building auth URLs that provide a route to the google login flow,
5. //fetchURLRequesting access tokens by passing an authorisation token to a google server, and fetchURLRequesting user information using
   that access token.
6. class GoogleLoginHelper
7. {
8.     //This function creates an auth URL, a project specific URL that contains information about who is trying to
9.     //accessing the users Google Account. This takes a clientID, redirectURL and a scopes array as parameters.
10.    public function GetAuthURL($clientID, $redirectURL, $scopes = array())
11.    {
12.        //Base Google OAuth2 url
13.        $url = 'https://accounts.google.com/o/oauth2/v2/auth?';
14.
15.        //Define a variable to hold our scope list. The scopes define what data we will be given access to via the
16.        //google auth flow.
17.        $scopeList = "";
18.
19.        //For eachURLRequest scope, add it to the string.
20.        foreach($scopes as $scope) $scopeList .= ($scope . " ");
21.
22.        //Add the scope string to the base URL, by first URL encoding it to prevent escaping errors.
23.        $url .= 'scope=' . urlencode($scopeList);
24.
25.        //Add the redirect URL to the base URL, by first URL encoding it to prevent escaping errors. This defines

```

```

26.     //where google will return to after login is succesfull.
27.     $url .= '&redirect_uri=' . urlencode($redirectURL);
28.
29.     //Add the client ID to the base URL, to tell google who is trying to access the users data.
30.     $url .= '&client_id=' . $clientID;
31.
32.     //Add parameters to create a login flow to fit our project. Here, we will set the login domain to that of
33.     //ashville.co.uk, and tell google to always ask the user to select an account when logging in via google.
34.     $url .= '&access_type=online&response_type=code&hd=ashville.co.uk&prompt=select_account';
35.
36.     //Return the full URL.
37.     return $url;
38. }
39.
40. //This function will excURLRequestange our authorization token for an access token. An authorization token by itself does
41. //not give us any information about the user. We need to first excURLRequestange it using Googles secure POST token server.
42. public function GetAccessToken($client_id, $redirect_uri, $client_secret, $code) {
43.
44.     //Secure POST server URL
45.     $url = 'https://accounts.google.com/o/oauth2/token';
46.
47.     //Define our post parameters, by creating a parametet string containing our client information and auth code.
48.     $curlPost =
49.         'client_id=' . $client_id
50.         . '&redirect_uri='
51.         . $redirect_uri
52.         . '&client_secret='
53.         . $client_secret
54.         . '&code=' . $code
55.         . '&grant_type=authorization_code';
56.
57.     //Initialise a PHP cURL operation. cURL is the method of making POST requests from within an ApacURLRequeste server.
58.     $cURLRequest = curl_init();
59.
60.     //Pass the URL to the cURL URL parameter.
61.     curl_setopt($cURLRequest, CURLOPT_URL, $url);
62.
63.     //Tell the cURL request to output the return value as a string during the exec method.
64.     curl_setopt($cURLRequest, CURLOPT_RETURNTRANSFER, 1);
65.
66.     //Tell the cURL request to use a basic HTTP POST request header.
67.     curl_setopt($cURLRequest, CURLOPT_POST, 1);

```



```

68.
69.     //Tell the cURL request to include our post data we constructed earlier.
70.     curl_setopt($cURLRequest, CURLOPT_POSTFIELDS, $curlPost);
71.
72.     //Save our cURL response as an array, decoded from the JSON string that Google returns. This includes our access token.
73.     $data = json_decode(curl_exec($cURLRequest), true);
74.
75.     //Get the POST HTTP code. This tells us about how the server responded to our request.
76.     $http_code = curl_getinfo($cURLRequest,CURLINFO_HTTP_CODE);
77.
78.     //If the server responded with anything other than a '200' (success) code, then break and tell the user what went wrong.
79.     if($http_code != 200)
80.         throw new Exception('Error : Failed to receive access token');
81.
82.     //Return the data array.
83.     return $data;
84. }
85.
86. //A function to fetch an array of user data. This takes the access token as a parameter.
87. public function GetUserProfileInfo($access_token) {
88.
89.     //Return the file content of the user info URL, using our access token as a unique user identifier.
90.     return file_get_contents('https://www.googleapis.com/oauth2/v1/userinfo?access_token=' . $access_token);
91. }
92. }
93. ?>

```

---

[../PROJECT/GOOGLEAUTH/CALLBACK.PHP](#)

The Google call-back page for use in the OAuth flow. Contains validation to ensure that authentication was successful.

```

1. <?php
2. //Start the session
3. session_start();
4.
5. //Enable Error reporting for debug
6. error_reporting( E_ALL );
7. ini_set( 'display_errors', 1 );
8.
9. //Defining the secure access parameter means that any config or functional files
10. //can only be accessed via a 'require' rather than directly through URL and HTTP, improving safety.

```

```

11. define( 'secureAccessParameter', true );
12. require_once 'gconfig.php';
13. require_once 'googleLoginHelper.php';
14.
15. //If a code has been set by Google (i.e. authorisation was succesfull), then proceed.
16. if(isset($_GET['code'])) {
17.
18.     //Try the following code.
19.     try {
20.
21.         //Instantiate a new googleloginhelper object.
22.         $gHelper = new GoogleLoginHelper();
23.
24.         //Get the access token from the Google Servers.
25.         $data = $gHelper->GetAccessToken(CLIENT_ID, CLIENT_REDIRECT_URL, CLIENT_SECRET, $_GET['code']);
26.
27.         //Extract the access token string and set to a variable.
28.         $access_token = $data['access_token'];
29.
30.         //Get the user info from the Google Servers.
31.         $user_info = $gHelper->GetUserProfileInfo($access_token);
32.
33.         //Set the student login session variable to equal the user info JSON decoded array.
34.         $_SESSION['stuLogin'] = json_decode($user_info, true);
35.
36.         //Get the segments of the email address.
37.         $emailSegments = explode("@", $_SESSION['stuLogin']['email']);
38.
39.         //If the domain is not 'ashville.co.uk', then redirect.
40.         if($emailSegments[1] != "ashville.co.uk")
41.         {
42.             //Redirect with the 'badDomain' GET variable set.
43.             header( 'Location:http://11wha.ashvillecomputing.co.uk/Project/Student?badDomain' );
44.         }
45.         else
46.         {
47.             //Regenerate the session ID to prevent session hijack attacks.
48.             session_regenerate_id();
49.
50.             //Redirect to the student card viewer page, with the student code email segement.
51.             header( 'Location:http://11wha.ashvillecomputing.co.uk/Project/studentCard?stuCode=' . $emailSegments[0] );
52.

```

```

53.     }
54.
55. }
56. //If there was an error from one of the helper functions, then catch it and display the error.
57. catch(Exception $e) {
58.     echo $e->getMessage();
59.     exit();
60. }
61. }
62. ?>

```

---

## ../PROJECT/GOOGLEAUTH/LOGOUT.PHP

A page which destroys the PHP session for any Google-authenticated user.

```

1. <?php
2. //Start the session
3. session_start();
4.
5. //Enable Error reporting for debug
6. error_reporting( E_ALL );
7. ini_set( 'display_errors', 1 );
8.
9. //Unset the student login details.
10. unset($_SESSION['stuLogin']);
11.
12. //Destroy the session, so that session login checks will return false.
13. session_destroy();
14.
15. //Redirect to the student login page.
16. header( 'Location:http://www.11wha.ashvillecomputing.co.uk/Project/Student' );
17.
18. ?>

```

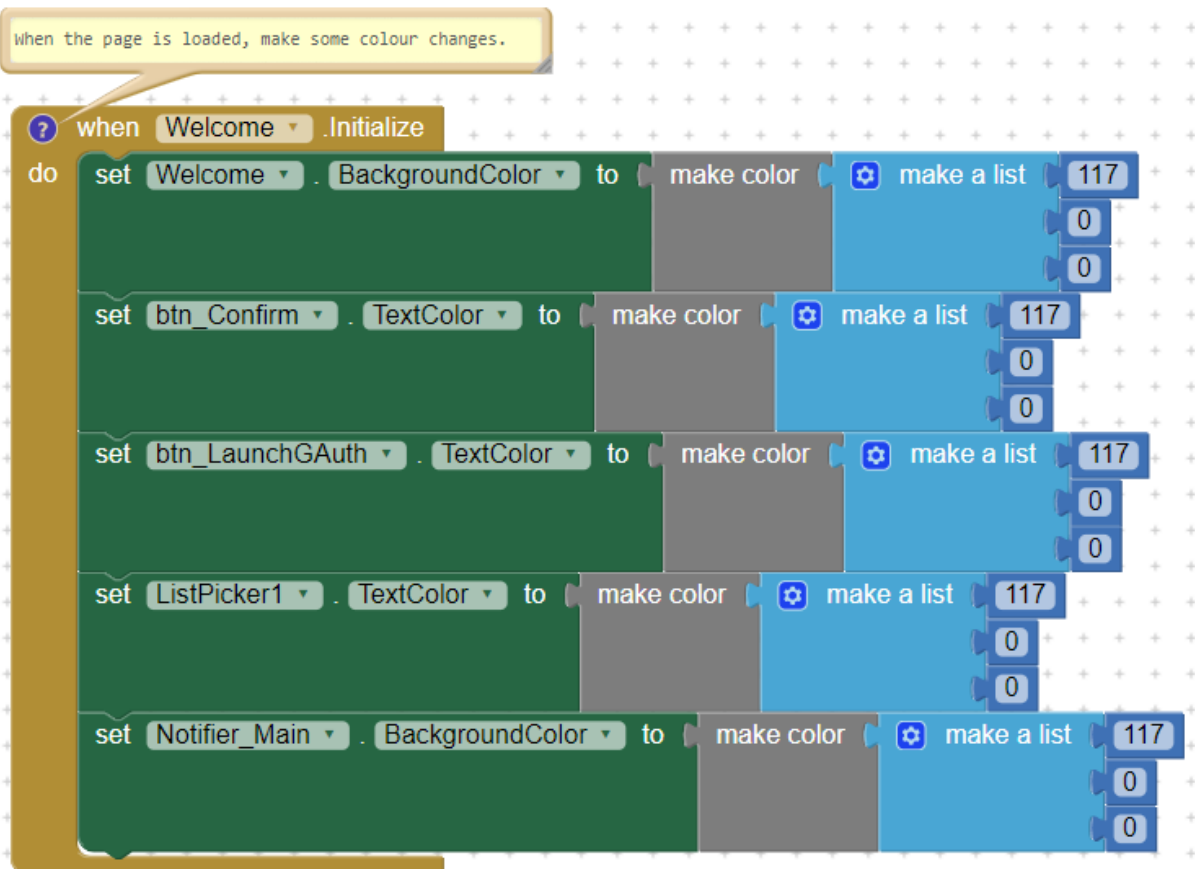
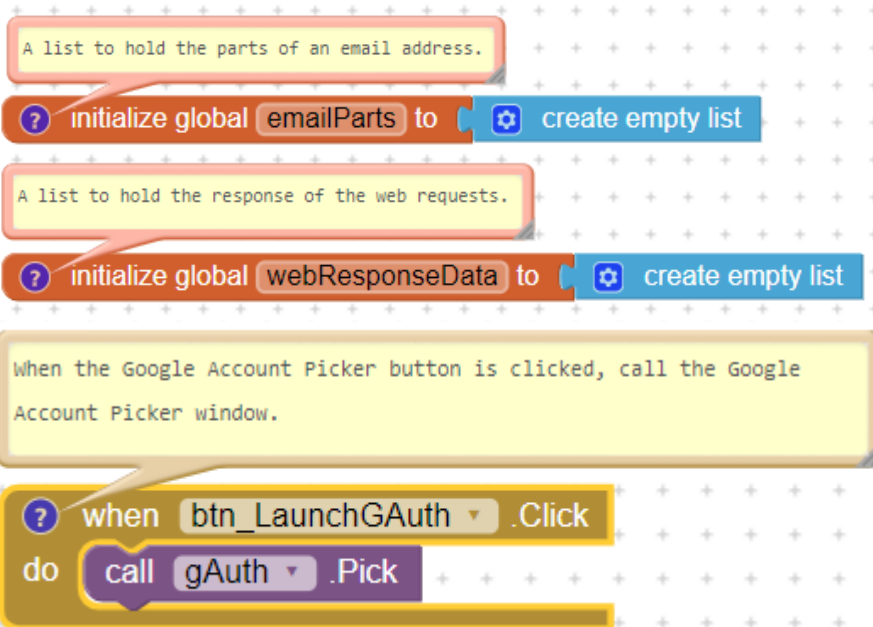
## SCREEN1

Screen1 is a simple redirect screen, since the MITApp inventor requires that the first screen be called Screen1. For use-ability, this page simply points to the welcome page.

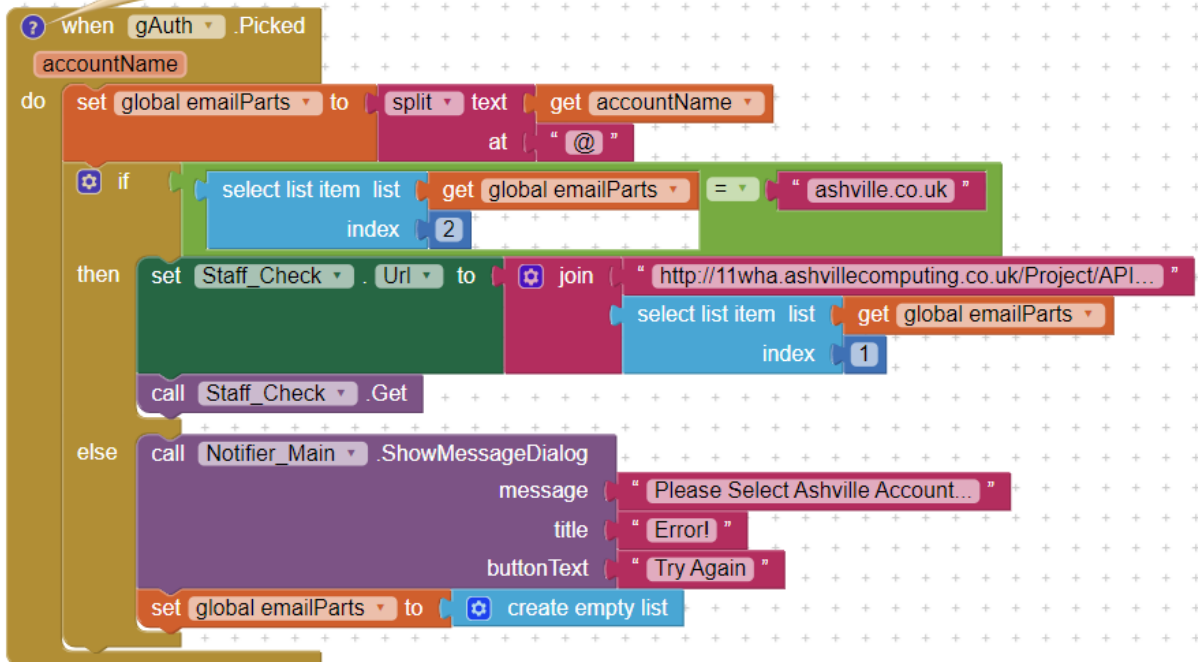


## WELCOME

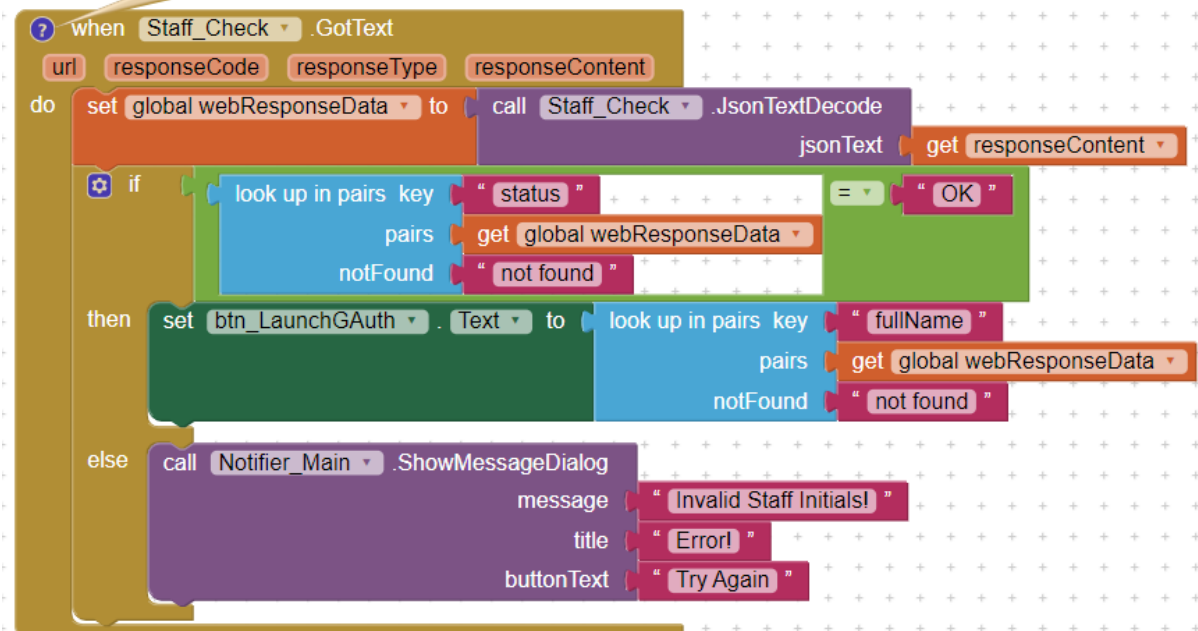
This page authenticates a staff member, by getting their initials (unique staff code) and their registration location. Then, move to the check\_status screen.



When the user has selected an account, check whether that account is an Ashville account, and then call a web check to see if that staff member exists.



When a response is received by from the staff checker, if the check returns true, then update the text on screen to show that user full name as a confirmation step. Else, notify of an error.



When the user has selected a registration location, then update the text on screen to show that location as a validation step.

```
when ListPicker1.AfterPicking
do set ListPicker1.Text to ListPicker1.Selection
```

When the confirm button is clicked, then check if the email address and location have been selected, before constructing the universal 'list' and opening the check\_status page.

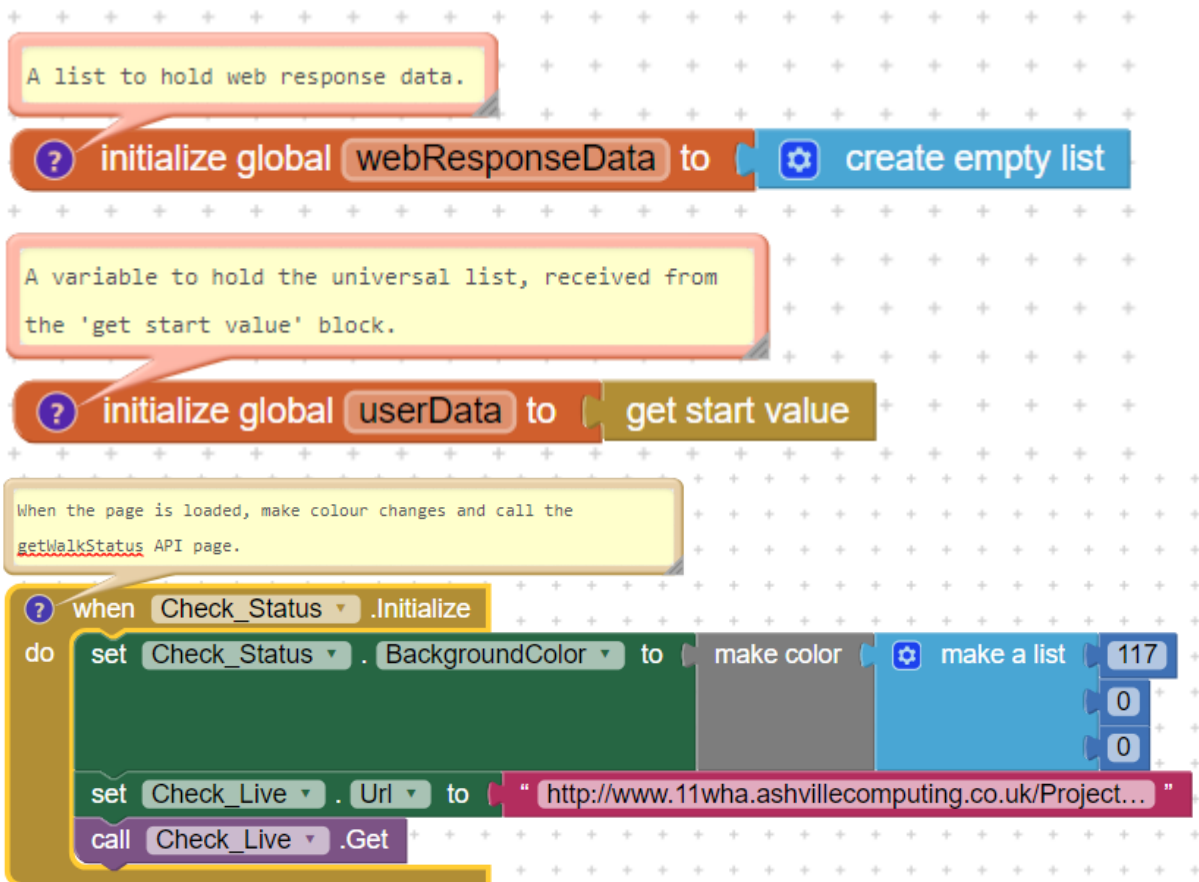
```

when btn_Confirm.Click
do
  if is list empty? list
  then
    call Notifier_Main.ShowDialog
    message "Please Select Ashville Account!"
    title "Error!"
    buttonText "Try Again"
  else if ListPicker1.Selection = ""
  then
    call Notifier_Main.ShowDialog
    message "Please Select Registration Location!"
    title "Error!"
    buttonText "Try Again"
  else
    open another screen with start value screenName "Check_Status"
    startValue
    make a list
    select list item list
    index 1
    btn_LaunchGAuth.Text
    ListPicker1.Selection
    ListPicker1.SelectedIndex
    0
    "Notice: Blimey, it's cold today!"
    ""
    ""
    ""
  end if
end do

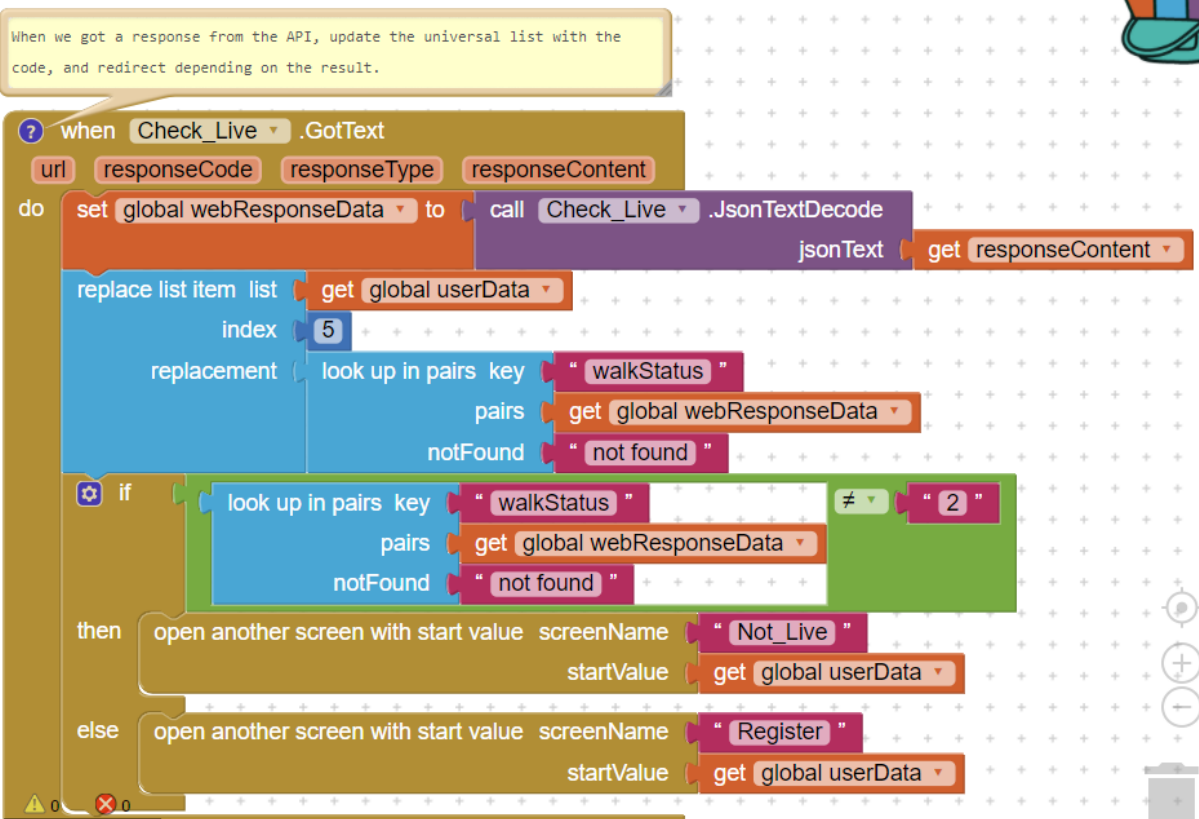
```

## CHECK\_STATUS

This page asks the server if the walk is active, and then redirects as required.

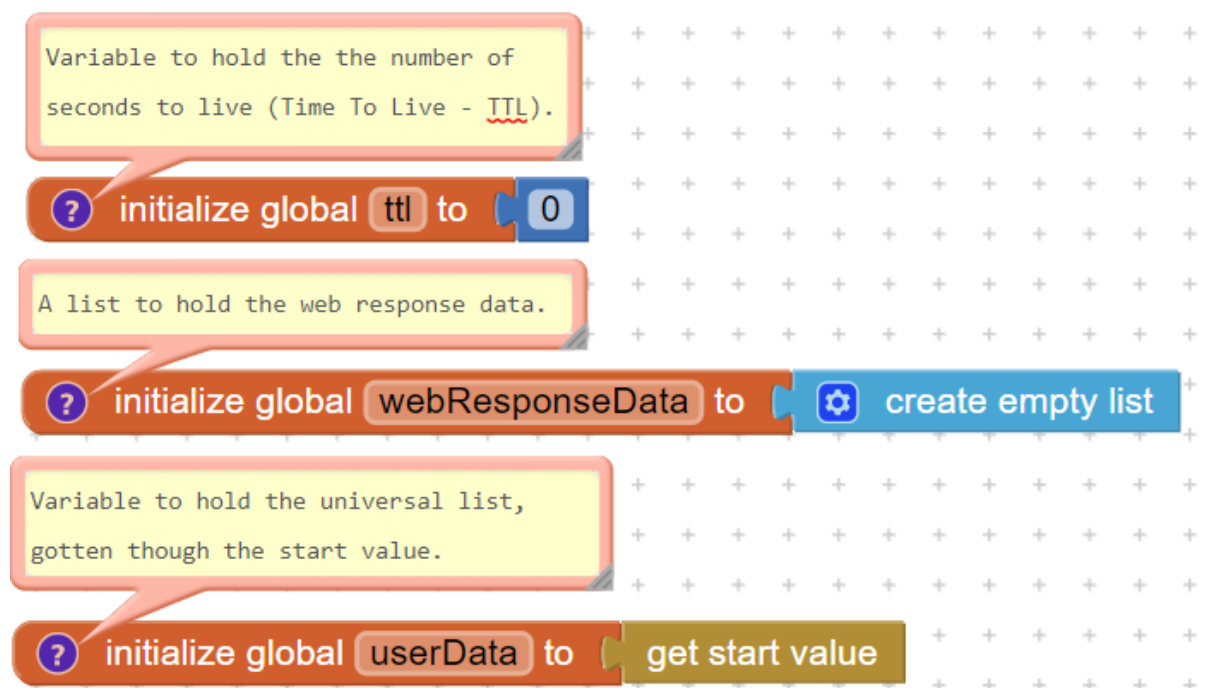






## NOT\_LIVE

This is the 'waiting page' where we display a message and a countdown timer if required. A refresh button allows the user to check if the walk is active yet.



When the page is loaded, make some colour changes and see what status the walk is. If we are waiting, call the TTL API, to get the number of seconds until the walk begins. Else, display a nice message if we are inactive.

```

when Not_Live.Initialize
do
  set Not_Live.BackgroundColor to make color
  make a list 117
  0
  0

  set btn_Refresh.TextColor to make color
  make a list 117
  0
  0

  set lbl_name.Text to select list item list
  get global userData
  index 2

  set lbl_loc.Text to select list item list
  get global userData
  index 3

  if
    select list item list
    get global userData
    index 5
    =
    " 1 "
  then
    set Get_TTL.Url to " http://11wha.ashvillecomputing.co.uk/Project/API..."
    call Get_TTL.Get
  else
    set lbl_info.Text to " Check Back Later For The Countdown! "
    set Ticker.TimerEnabled to false
  
```

Show Warnings

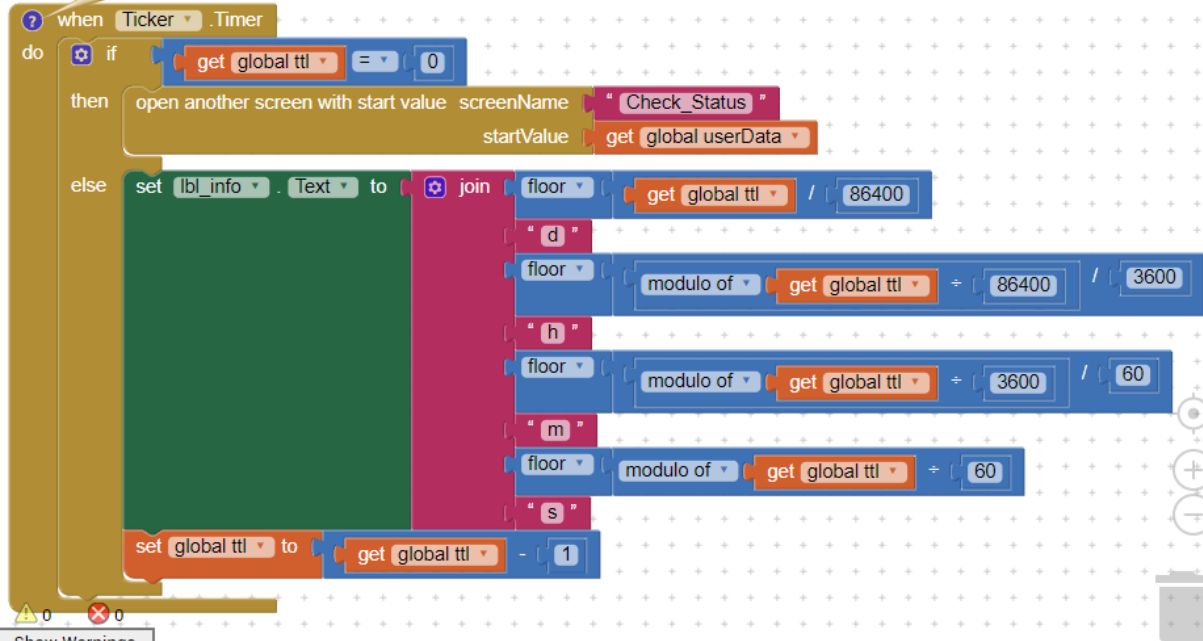
When the API has responded, then update the global variable with the value.

```

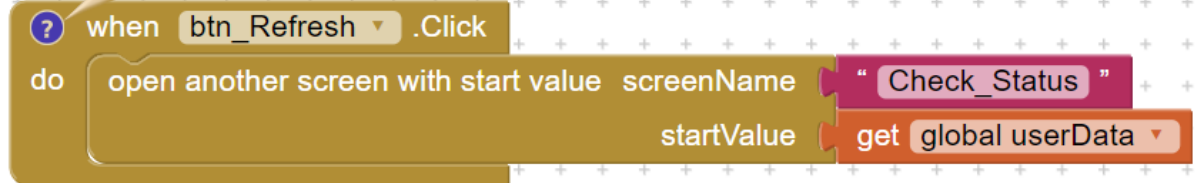
when Get_TTL.GotText
url responseCode responseType responseContent
do
  set global webResponseData to call Get_TTL.JsonTextDecode
  jsonText get responseContent

  set global ttl to look up in pairs key
  " timeToStart "
  pairs get global webResponseData
  notFound " not found "
  
```

A ticker, called each second, to minus one from the `ttl`. Then, format the `ttl` in a string 'ddhhmmss', and print. If the `ttl` ever reaches 0, move to the registration `check_live` page to ensure no state changes have been made.



When refresh is clicked, go back to the `check_status` page.



## REGISTER

This page is where the staff member is prompted to scan a student's QR code.

A List to hold the response of a web request.

? initialize global webResponseData to create empty list

A String to hold the unique student ID.

? initialize global stuCode to ""

A List to hold the universal list.

? initialize global userData to get start value

When the page is loaded, make some colour and text changes.

? when Register.Initialize do

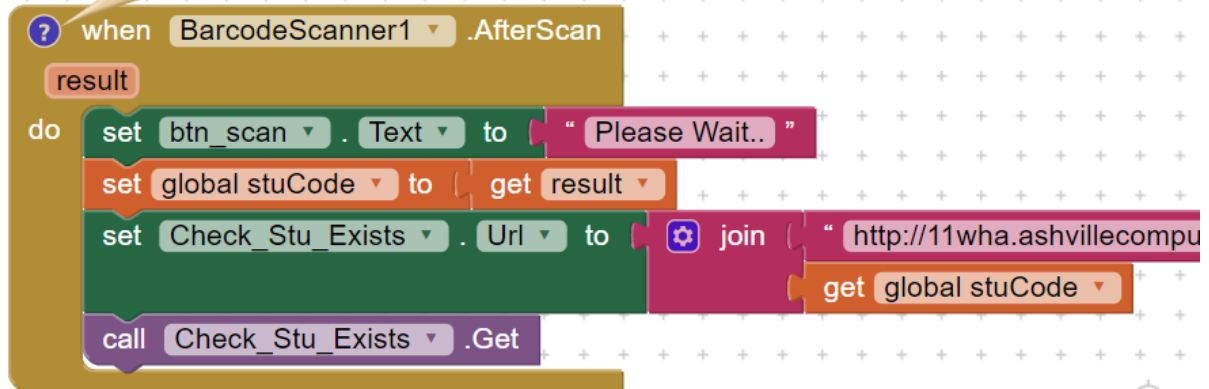
- set Register.BackgroundColor to make color make a list 117 0 0
- set btn\_scan.TextColor to make color make a list 117 0 0
- set lbl\_Loc.Text to select list item list get global userData index 3
- set lblNotice.Text to select list item list get global userData index 6

When the scan launch button is clicked, launch the scanner.

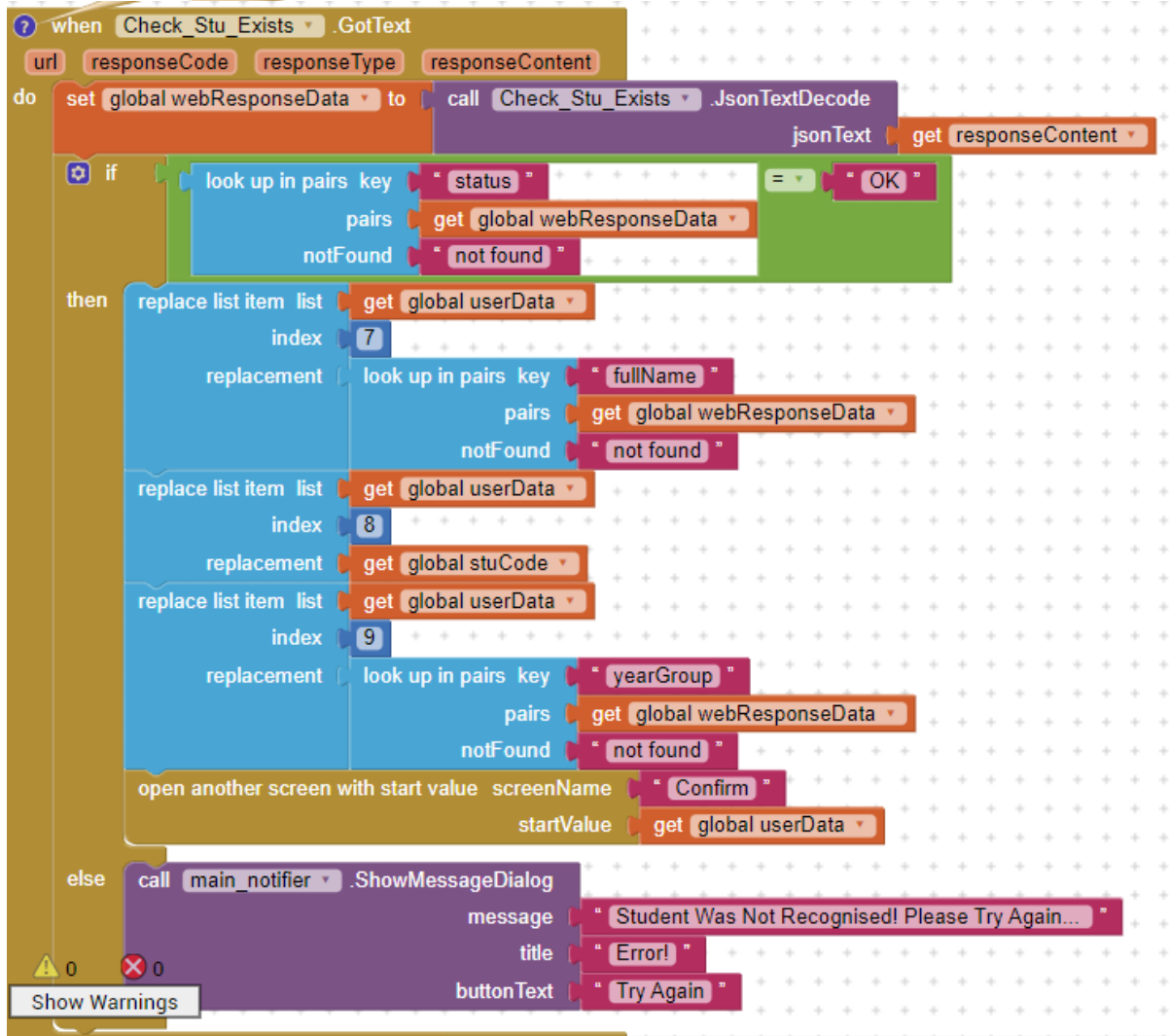
? when btn\_scan.Click do

- set btn\_scan.Text to "Loading..."
- call BarcodeScanner1.DoScan

After the scanner has scanned and parsed the QR code, then see if the student exists in the student table.



When we got a response from the API, see if the student exists and then redirect if true, updating the universal list with the student code, students full name and year group. Else, throw an error.



## CONFIRM

This page is where the human verification of the student occurs before the server is told to register a new registration event.

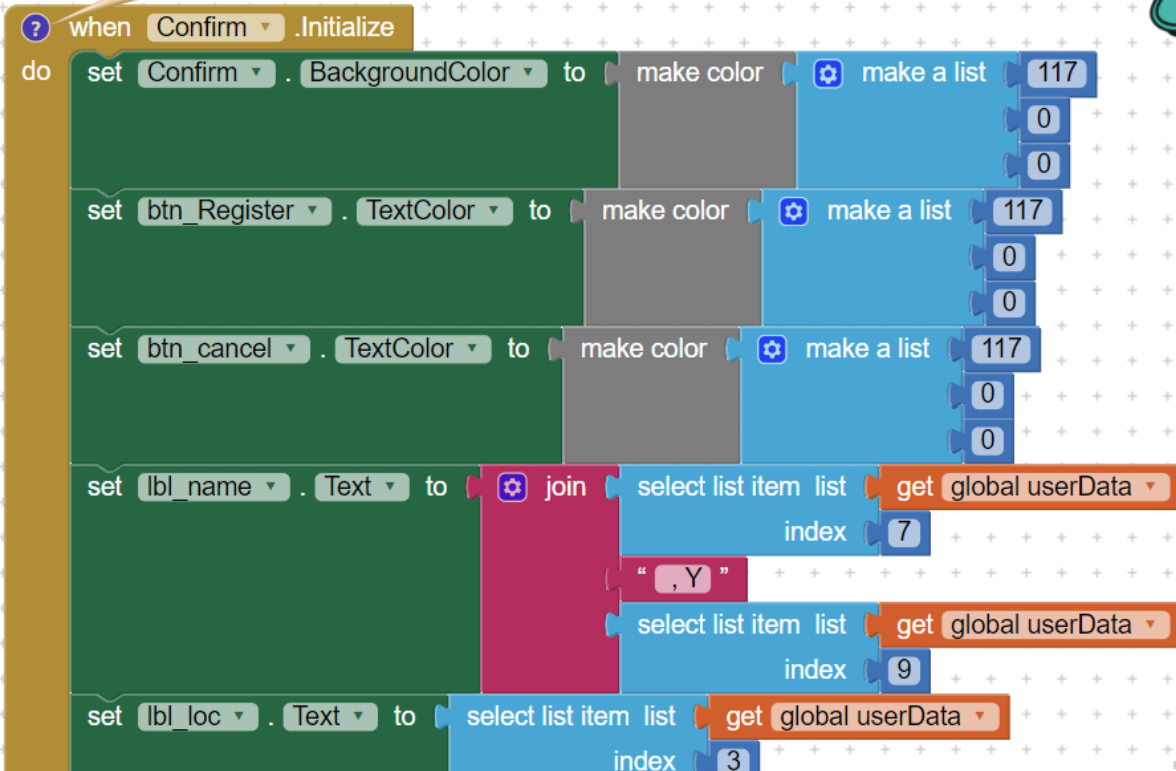
A list to hold the web response data.

? initialize global webResponseData to create empty list

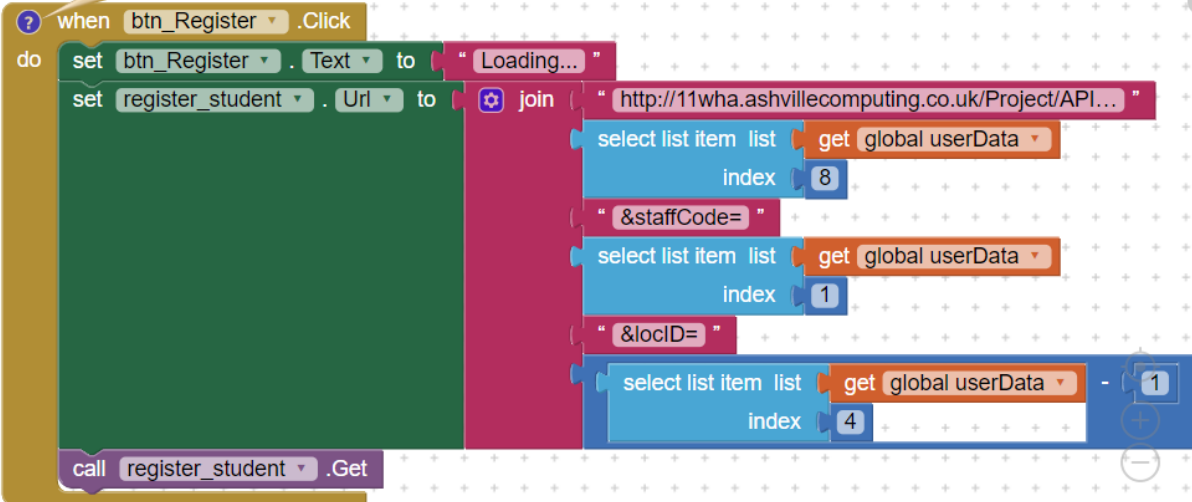
A list to hold the universal list, gotten from the start value.

? initialize global userData to get start value

When the page is loaded, make some colour and text changes.



When the registration button is clicked, build the API URL and then call the newData API.



When we have a response from the API, check for errors and then build suitable responses to let the user know what has happened, error or no error. Then, return to the register page/.

