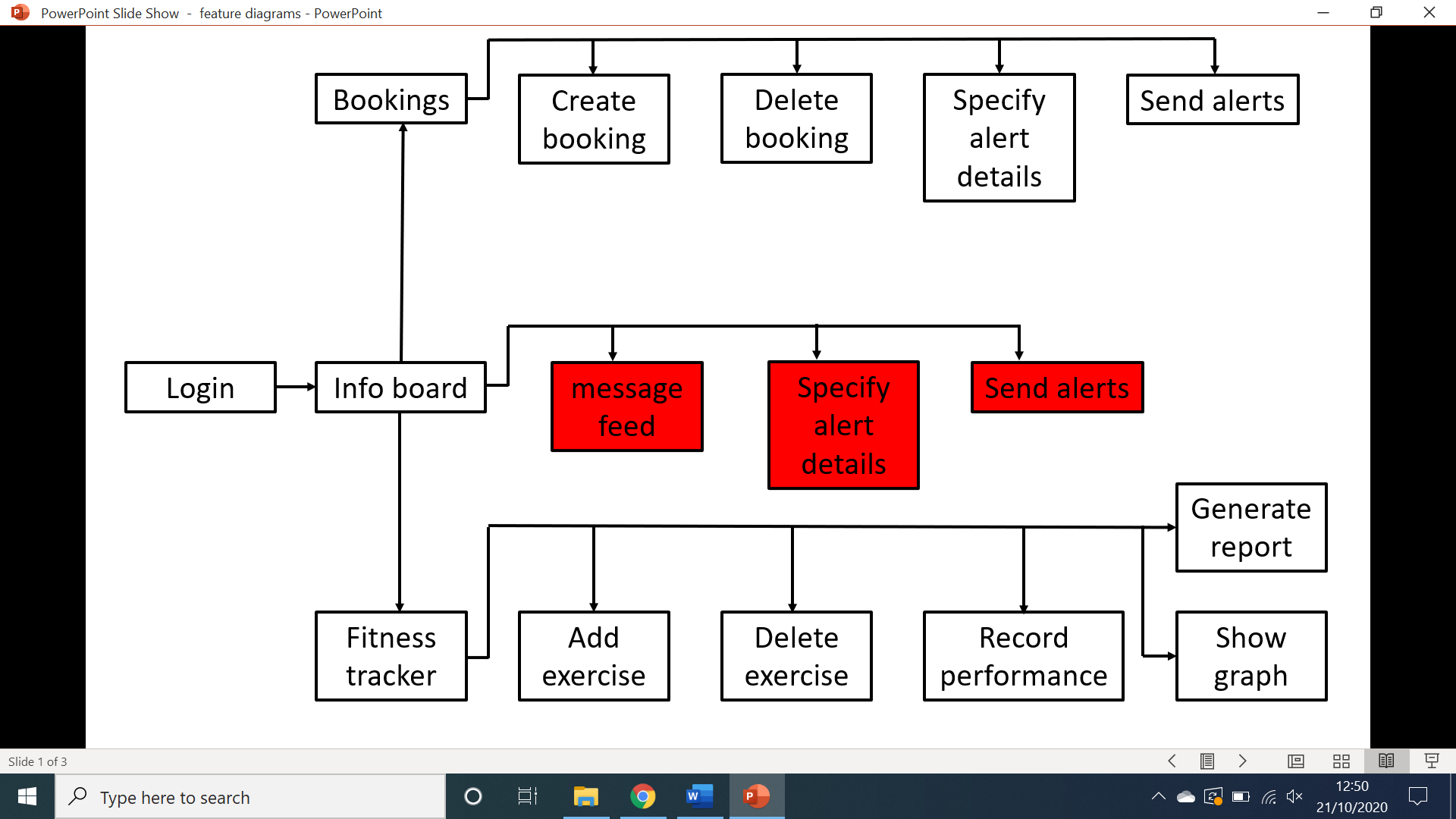
Computer Science

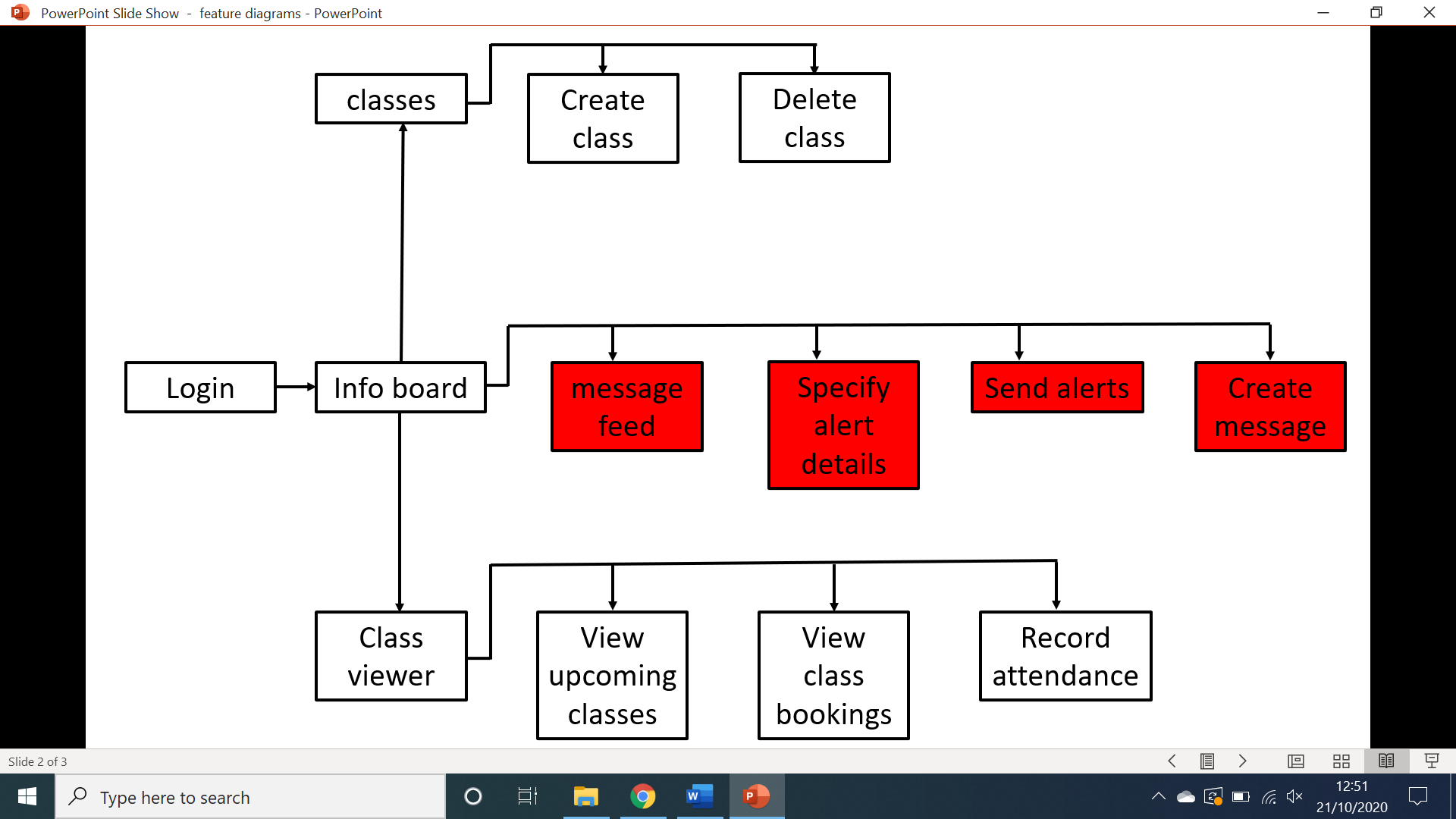
A2 Coursework

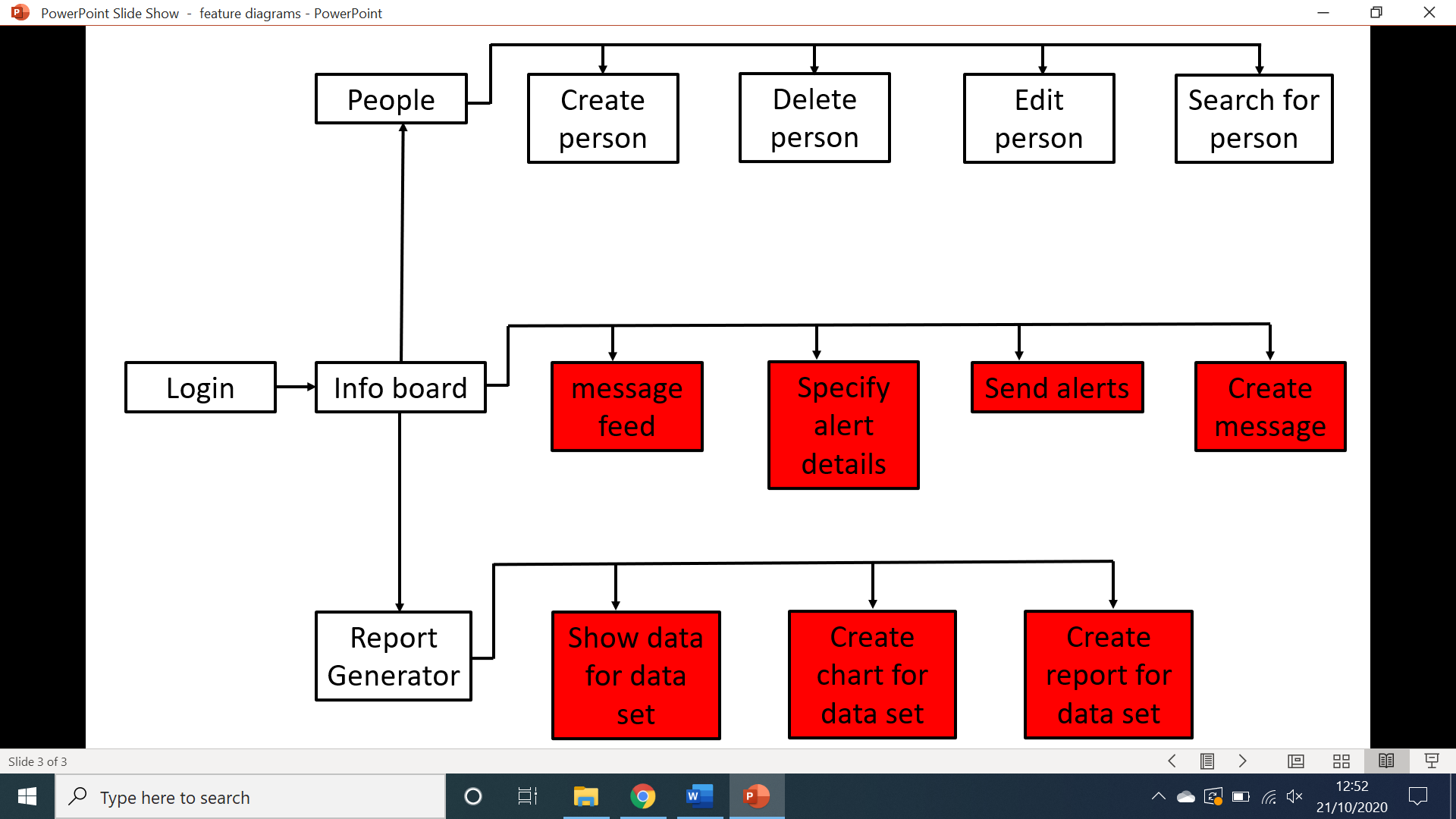
Callum Reid

**Decomposition diagrams:**

Here I will show the decomposition diagrams for every stakeholder I have. The boxes marked in red will not be included in the prototype.

**Client:**

**Trainer:**

**Manager:**

**Description of features:**

Here I will give a description of each of the features that are included in the system.

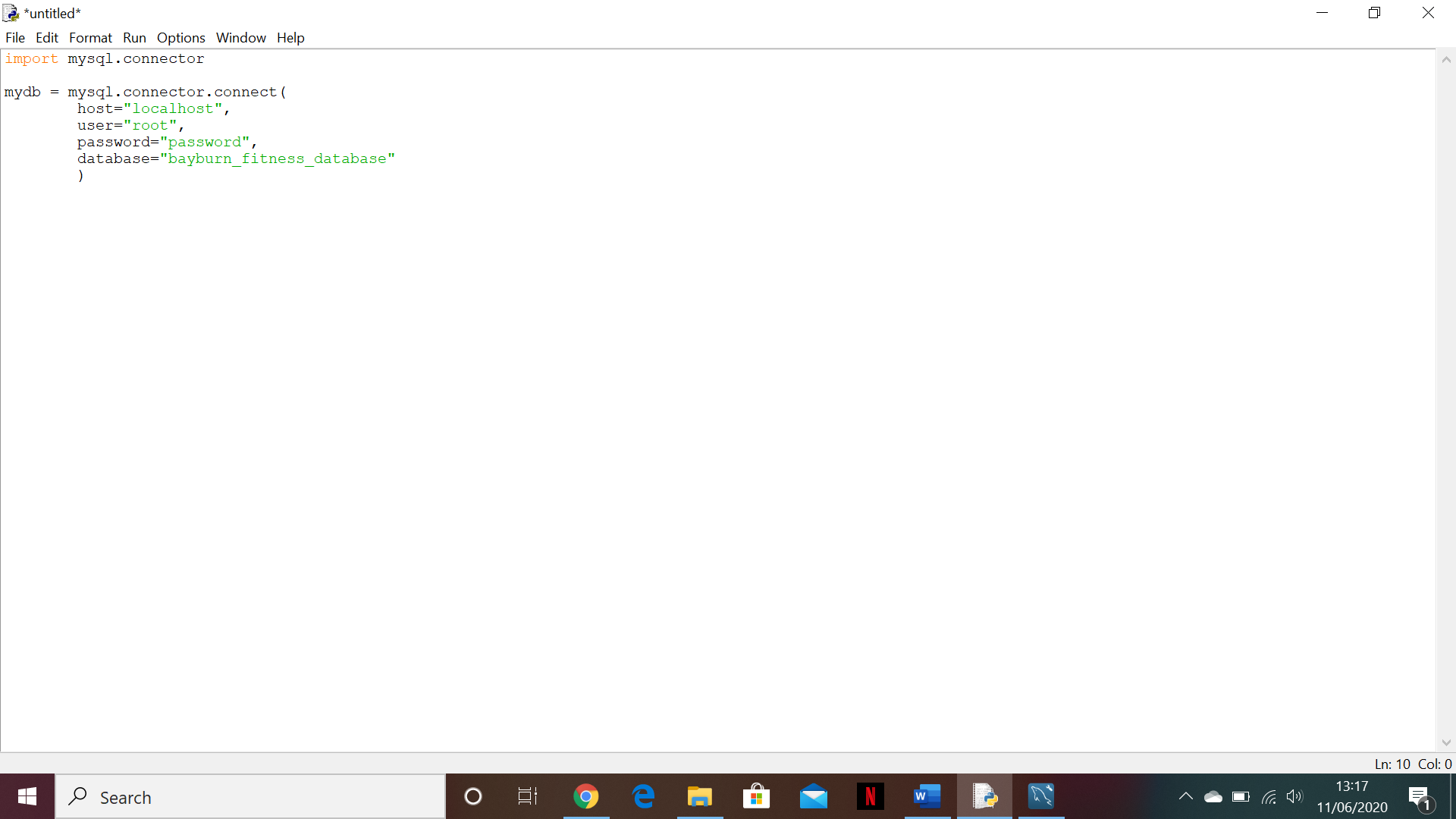
* **Login (Client, Trainer, Manager):** the login page will allow all stakeholders to enter their credentials and be directed to their appropriate screen given that the username and password is correct.
* **Bookings (Client):** the bookings page allows a client to access the full functionality needed to create and delete bookings
* **Create booking (Client):** this feature allows a client to view all possible classes which they can book into using a calendar selector. When a date is selected using the calendar, all possible classes are populated into a tree view and can be clicked on to book. This updates another tree view which shows all of the client’s current bookings.
* **Delete booking (Client):** this feature allows a client to unbook a booking by clicking on it in the tree view which shows all previous bookings
* **Specify alert details (Client):** this feature allows clients to select when they will receive notifications about upcoming classes as well as select how they wish to receive these notifications.
* **Send alerts (Client):** this feature will allow clients to be sent messages via email or SMS automatically in order to inform them of their class using radio buttons.
* **Information board (Client, Trainer, Manager):** this screen gives the user access to the functionality of the information board messaging system.
* **Fitness tracker (client):** this screen gives the user access to the functionality of the fitness tracking system.
* **Add exercise (Client):** this feature allows the client to add an exercise that they wish to track to the system using an entry box.
* **Delete exercise (Client):** this feature allows the client to delete one of their selected exercises from the system using the interactive tree view which is populated with all of their exercises.
* **Record performance (client):** this feature allows a client to record the weight, reps, sets and time for which they completed an exercise using several entry boxes.
* **Show graph (client):** this feature will generate a graph that shows the date on the x axis and various exercise data on the right, such as time, weight, reps and sets.
* **Generate report (client):** this feature allows all the data for a specific exercise to be presented in a word document as well as having the graph be generated and placed in the word document as well.
* **Classes (Trainer):** this screen gives the trainer access to the functionality required to add, edit, delete and view classes.
* **Create class (Trainer):** this feature allows a trainer to create a class for clients to book into, this is done through the use of several entry boxes, tick boxes and a text box.
* **Delete class (Trainer):** this feature allows the trainer to delete a class that has been created, this is done through the use of an interactive tree view that is populated with all previously created classes.
* **Class viewer (Trainer):** this screen allows a trainer to view all of their upcoming classes as well as the people who are attending each class.
* **View class bookings (Trainer):** this feature will allow a trainer to select an upcoming class and see who has been booked into it, this is done through the use of an interactive tree view that is populated with all the classes in the next 30 days.
* **Record attendance (Trainer):** this feature will allow a trainer to mark a person as present to a class, this is done by clicking on the person on the interactive tree view that gets populated when a class is selected.
* **People (Manager):** this screen gives the manager access to the functionality that will allow them to create, delete, update or retrieve information about system users.
* **Create person (Manager):** this feature allows a manager to create an account for someone to use the system, this is done through the use of various text and entry boxes as well as a calendar selector and several selection boxes and radio buttons..
* **Delete person (Manager):** this feature allows a manager to delete a person’s account, this is achieved when someone is selected in the search system and the manager clicks delete.
* **Edit person (Manager):** this feature allows a manager to edit the information of a previously created account by selecting them in the search system and clicking edit.
* **Search for person (Manager):** this feature allows a manager to search for a person to edit, delete or simply view their data, this is done using a search system that automatically refines the search based on the inputs.
* **Report generator (Manager):** this screen will give the manager access to the functionality needed to generate detailed reports and charts on any given data set.
* **Message feed (Client, Trainer, Manager):** this feature will allow managers and trainers to send messages to the message board informing clients of various pieces of information.
* **Specify alert details (Client, Trainer, Manager):** this feature will allow a client, trainer and manager to specify how they wish to be alerted about new messages.
* **Send alerts (Client, Trainer, Manager):** this feature will send emails or SMS messages to people informing them of a new message.
* **Create message (Trainer, Manager):** this feature will allow a trainer or manager to create a message and send it to the message board, this is done through the use of a text box.
* **Show data for data set (Manager):** this feature allows for all the data of a specific data set to be displayed on screen
* **Create chart for data set (Manager):** this feature will allow a manager to display all the data of a data set in the form of an excel chart.
* **Create report for data set (Manager):** this feature will allow the manager to create a word document report with a graph and data.

**Explanation of data storage:**

In order to store all of the necessary data that my system will record I will be using a MySQL relational database. The database will be run on my laptop, this is not ideal as in order to access the database the laptop must be running. I am using a MySQL relational database because through the use of a relational database I can connect multiple tables together through the use of primary and foreign keys in order to create a data storage system that can be easily navigated and stores all of the data needed in an efficient fashion. I am also using a MySQL relational database because it includes a Python connector which can be easily set up, this allows me to connect the MySQL database to my project which is being coded in Python, this allows for data to be easily accessed, modified or created by the system. The Python connector for MySQL is easy to set up and when combined with the MySQL workbench It allows for the database to be easily created and modified without much hassle.

**Connecting python to the MySQL server:**

Connecting Python to the MySQL server is quite simple all you need to do is import the connector and define several variables. These variables are then used to access the server.

The host variable defines where on the network the user is being held, the user variable defines which user has the database, the password variable defines what password is used to access that user and the database variable defines which database should be accessed. All of these variables are then used by the MySQL connector in Python in order to access the database.

**Methods of access:**

There are several ways in which I could have accessed the data from the database, some of which include including: Serial, Sequential, Indexed Sequential, Direct access and Random access. For my project I have chosen to use direct access in order to get the data from the database below I shall explain each method of access at the list their pros and cons before explaining my decision to use direct access.

**serial access:**

Serial access is when the data is searched through chronologically until the desired data is found. if the desired data is not in the file, the system will still have to search through the entire file before realising that the data is not there.

**Pros:**

There are no positives to serial access in this situation.

**Cons:**

Serial access is far too slow to access data quickly from a large database.

**Sequential access:**

Sequential access involves the data being stored according to a primary key. This key could either be numerical or alphabetical in which case it would be stored in either ascending or descending order or in alphabetical order respectively. When searching for data the search can be cut off at the point where the search parameter exceeds a point in the list meaning the data is not in the list.

**Pros:**

This method is much faster than serial access

**Cons:**

This method involves the data being stored in a specific way meaning it can be more complicated for data to be input into the database. It is also slower compared to other methods.

**indexed sequential**

This method of data access involves all of the data being stored sequentially as well as having a data key. The system will first search through the key to find the start location of the data which it is looking for by comparing the primary key of the search parameters to that of the key. When it has found the sequential start of that part of data the system can begin the search at that point and similarly to sequential searches, cut off the search if the system passes a certain sequential point in the list.

**Pros:**

This method is much faster than sequential and serial.

**Cons:**

This method involves the data being stored sequentially which makes the input of data more complex and there are other faster methods of data access. This method also requires the creation of an index file which adds another layer of complexity and if the data cannot be sorted sequentially based on a primary key then this method of data access will not work.

**Random access:**

This method of access involves using a hashing algorithm that processes a primary key of a piece of data in order to create a storage location. When searching for the data the primary key that is being searched for is input into the hashing algorithm to generate the same storage location. In the event that when adding data two primary keys generate the same storage location, the system will leave a pointer to an overflow area at that storage location and store the data in that overflow area sequentially. When searching for data, if the data is not in the initial storage location the system follows the pointer to the overflow area and searches through the area sequentially.

**Pros:**

This method of data access is faster than sequential, serial and indexed sequential.

**Cons:**

This method is exceedingly complex as it requires the creation of a hashing algorithm that generates locations that do not collide often, generates locations that are relatively close together in the storage medium and does not take long to calculate. As well as this the not all primary keys lend themselves to being processed by a hashing algorithm.

**Direct access:**

This method involves finding the estimated amount of storage that the data that is being searched for would take up and then using this information to find the location in storage of the data. This is the method I will be using to access data in my database

**Pros:**

As fast or faster than all other methods whilst remaining quite simple.

**Cons:**

Can involve the use of fixed length rows and fields.

Out of all the methods listed above I have decided to use direct access as I believe that it is the most suitable for my system.

**Normalisation**

PRIMARY KEY

FOREIGN KEY\*

**Unormalised Form**

the unnormalized form represents a flat file database with no relations and only one table. This method of data storage is prone to data redundancy and makes it harder to ensure data integrity.

ID, Firstname, DOB, Phone, Email, Gender, Postcode, Address, Injuries, Illnesses, Height/m, Weight/kg, Smoke, Fitness, Qualifications, Experience, Username, Password, exercise, Trainer, Days, Start time, End time, Overview, Booking ID, Client ID, Class ID, Class Date, Alert type, Alert time, date, sender name, message

**Third Normal Form**

Third normal form is when the data has been separated into many separate tables that are connected via foreign keys. Third normal form is achieved when each column of a table has a single atomic value, all values of a column are of the same data type, all columns have unique names, there is a primary key for each table, there are no partial dependencies and there are no transitive dependencies. Third normal form drastically reduces data redundancy and increases data integrity.

Clients:

ID\*, Firstname, DOB, Phone, Email, Gender, Postcode, Address Injuries, Illnesses, Height/m, Weight/kg, Smoke, Fitness

trainers:

ID\*, Firstname, DOB, Phone, Email, Gender, Postcode, Address, Qualifications, Experience

managers:

ID\*, Firstname, DOB, Phone, Email, Gender, Postcode, Address, Qualifications, Experience

logins:

ID\*, Username, Password

Fitness Tracker:

ID\*, Exercise

Classes:

Class ID, Trainer, Days, Start time, End time, Overview

Bookings:

Booking ID, Client ID\*, Class ID\*, Class Date

Alerts:

ID\*, Alert type, Alert time

Alerts2:

ID\*, Alert type, Alert time

Attendances:

Client ID\*, class ID\*, date, name, phone

Messages:

Sender Id\*, Sender name, message

**Entity Relationship Diagram:**

Diagram

Description automatically generated

**Explanation of connections:**

* One client can have multiple exercises in the tracker table.
* One client can have only one alert specification in the Alerts or Alerts2 table.
* One client can have multiple attendances in the attendance table.
* One client can make multiple bookings in the bookings table.
* One class can have multiple associated attendances in the attendance table.
* One class can have multiple associated bookings in the bookings table.
* One login can be associated to only one client, manager or trainer.
* One trainer can make multiple classes in the class table.
* One manager or one trainer can have multiple messages in the messages table.

|  |  |
| --- | --- |
| **Input** |  |
| **Output** |  |
| **Both** |  |

**Data Dictionaries:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **File Name:** | | Clients table | | **Table Name** | | clients | | **Primary Key Field:** | | ID | |
| **Method of access:** MySQL uses direct access through a python connector | | | | | | | | | | | |
| **Related to table:** nextid, tracker, logins | | | | | | **Foreign key:** N/A | | | | | |
| **Table description:** this table holds all personal info about clients | | | | | | | | | | | |
| **Estimated number of records:** 150 | | | | | | | | | | | |
| **Type of data structure:** database table containing records | | | | | | | | | | | |
| **Estimated size of records in bytes:** 250 | | | | | | | | | | | |
| **Estimated size of table:** 41.25KB | | | | | | | | | | | |
| **Field name** | **Required** | | **Indexed** | **Data type** | **Length**  **/bytes** | | **Input mask/validation rule** | **Default value** | **Description** | | **Typical data** |
| ID | Yes | | Yes | VarChar | 5 | | -presence check | Auto generated | Unique identifier for client | | C4 |
| Firstname | Yes | | No | VarChar | 20 | | -presence check  -type check | \_ | Client first name | | Callum |
| lastname | Yes | | No | VarChar | 20 | | -presence check  -type check | \_ | Client last name | | Reid |
| DOB | Yes | | No | VarChar | 10 | | -lookup check | Current date | Client date of birth | | 2020-06-11 |
| Phone | Yes | | No | VarChar | 11 | | -presence check  -type check  -length check | \_ | Client phone number | | 07873281987 |
| Email | Yes | | No | VarChar | 30 | | -presence check  -format check | \_ | Client email address | | callum@gmail.com |
| Gender | Yes | | No | VarChar | 6 | | -lookup check | Male | Client gender | | Male |
| Postcode | Yes | | No | VarChar | 7 | | -presence check  -length check | \_ | Client postal code | | Bt182gb |
| Address | Yes | | No | VarChar | 30 | | -presence check | \_ | Client house address | | 24b bridge road Helen’s bay |
| Injuries | No | | No | VarChar | 50 | | -presence check | None | Any underlying injuries | | Fractured femur |
| Illnesses | No | | No | VarChar | 50 | | -presence check | None | Any underlying illnesses | | Asthma |
| Height/cm | Yes | | No | VarChar | 3 | | -presence check  -type check | 0 | Client height | | 183 |
| Weight/kg | Yes | | No | VarChar | 3 | | -presence check  -type check | 0 | Client weight | | 80 |
| Smoke | Yes | | No | VarChar | 3 | | -lookup check | No | Does the client smoke or not | | No |
| fitness | Yes | | No | VarChar | 2 | | -lookup check | 1 | How fit is the client on a scale of 1 - 10 | | 9 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **File Name:** | | managers table | | **Table Name** | | managers | | **Primary Key Field:** | | ID | |
| **Method of access:** MySQL uses direct access through a python connector | | | | | | | | | | | |
| **Related to table:** nextid, logins | | | | | | **Foreign key:** N/A | | | | | |
| **Table description:** this table holds all personal info about managers | | | | | | | | | | | |
| **Estimated number of records:** 10 | | | | | | | | | | | |
| **Type of data structure:** database table containing records | | | | | | | | | | | |
| **Estimated size of records in bytes:** 219 | | | | | | | | | | | |
| **Estimated size of table:** 2.4KB | | | | | | | | | | | |
| **Field name** | **Required** | | **Indexed** | **Data type** | **Length**  **/bytes** | | **Input mask/validation rule** | **Default value** | **Description** | | **Typical data** |
| ID | Yes | | Yes | VarChar | 5 | | -presence check | Auto generated | Unique identifier for manager | | m4 |
| Firstname | Yes | | No | VarChar | 20 | | -presence check  -type check | \_ | manager first name | | Callum |
| lastname | Yes | | No | VarChar | 20 | | -presence check  -type check | \_ | manager last name | | Reid |
| DOB | Yes | | No | VarChar | 10 | | -lookup check | Current date | manager date of birth | | 2020-06-11 |
| Phone | Yes | | No | VarChar | 11 | | -presence check  -type check  -length check | \_ | manager phone number | | 07873281987 |
| Email | Yes | | No | VarChar | 30 | | -presence check  -format check | \_ | manager email address | | callum@gmail.com |
| Gender | Yes | | No | VarChar | 6 | | -lookup check | Male | manager gender | | Male |
| Postcode | Yes | | No | VarChar | 7 | | -presence check  -length check | \_ | manager postal code | | Bt182gb |
| Address | Yes | | No | VarChar | 30 | | -presence check | \_ | manager house address | | 24b bridge road Helen’s bay |
| qualifications | No | | No | VarChar | 50 | | -presence check | None | Any managerial qualifications | | Business and management degree |
| experience | No | | No | VarChar | 50 | | -presence check | None | Any previous managerial experience | | Worked for Microsoft as a manager |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **File Name:** | | trainers table | | **Table Name** | | trainers | | **Primary Key Field:** | | ID | |
| **Method of access:** MySQL uses direct access through a python connector | | | | | | | | | | | |
| **Related to table:** nextid, logins | | | | | | **Foreign key:** N/A | | | | | |
| **Table description:** this table holds all personal info about trainers | | | | | | | | | | | |
| **Estimated number of records:** 20 | | | | | | | | | | | |
| **Type of data structure:** database table containing records | | | | | | | | | | | |
| **Estimated size of records in bytes:** 219 | | | | | | | | | | | |
| **Estimated size of table:** 4.8KB | | | | | | | | | | | |
| **Field name** | **Required** | | **Indexed** | **Data type** | **Length**  **/bytes** | | **Input mask/validation rule** | **Default value** | **Description** | | **Typical data** |
| ID | Yes | | Yes | VarChar | 5 | | -presence check | Auto generated | Unique identifier for trainer | | t4 |
| Firstname | Yes | | No | VarChar | 20 | | -presence check  -type check | \_ | trainer first name | | Callum |
| lastname | Yes | | No | VarChar | 20 | | -presence check  -type check | \_ | trainer last name | | Reid |
| DOB | Yes | | No | VarChar | 10 | | -lookup check | Current date | trainer date of birth | | 2020-06-11 |
| Phone | Yes | | No | VarChar | 11 | | -presence check  -type check  -length check | \_ | trainer phone number | | 07873281987 |
| Email | Yes | | No | VarChar | 30 | | -presence check  -format check | \_ | trainer email address | | callum@gmail.com |
| Gender | Yes | | No | VarChar | 6 | | -lookup check | Male | trainer gender | | Male |
| Postcode | Yes | | No | VarChar | 7 | | -presence check  -length check | \_ | trainer postal code | | Bt182gb |
| Address | Yes | | No | VarChar | 30 | | -presence check | \_ | trainer house address | | 24b bridge road Helen’s bay |
| qualifications | No | | No | VarChar | 50 | | -presence check | None | Any training qualifications | | Physical education degree |
| experience | No | | No | VarChar | 50 | | -presence check | None | Any previous training experience | | Worked for PureGym as a trainer |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **File Name:** | | logins table | | **Table Name** | | logins | | **Primary Key Field:** | | ID | |
| **Method of access:** MySQL uses direct access through a python connector | | | | | | | | | | | |
| **Related to table:** nextid, clients, managers, trainers | | | | | | **Foreign key:** N/A | | | | | |
| **Table description:** this table holds all information required to log a user into the system | | | | | | | | | | | |
| **Estimated number of records:** 180 | | | | | | | | | | | |
| **Type of data structure:** database table containing records | | | | | | | | | | | |
| **Estimated size of records in bytes:** 45 | | | | | | | | | | | |
| **Estimated size of table:** 8.9KB | | | | | | | | | | | |
| **Field name** | **Required** | | **Indexed** | **Data type** | **Length**  **/bytes** | | **Input mask/validation rule** | **Default value** | **Description** | | **Typical data** |
| ID | Yes | | Yes | VarChar | 5 | | -presence check | Auto generated | Unique identifier for users | | t4 |
| username | Yes | | No | VarChar | 20 | | -presence check | \_ | Users unique identification name | | Creid765 |
| password | Yes | | No | VarChar | 20 | | -presence check  -length check | \_ | Users password for their account | | Password123 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **File Name:** | | nextid table | | **Table Name** | | nextid | | **Primary Key Field:** | | ID | |
| **Method of access:** MySQL uses direct access through a python connector | | | | | | | | | | | |
| **Related to table:** logins, clients, managers, trainers | | | | | | **Foreign key:** N/A | | | | | |
| **Table description:** this table holds all information required to generate a new ID number for a user | | | | | | | | | | | |
| **Estimated number of records:** 180 | | | | | | | | | | | |
| **Type of data structure:** database table containing records | | | | | | | | | | | |
| **Estimated size of records in bytes:** 5 | | | | | | | | | | | |
| **Estimated size of table:** 0.99KB | | | | | | | | | | | |
| **Field name** | **Required** | | **Indexed** | **Data type** | **Length**  **/bytes** | | **Input mask/validation rule** | **Default value** | **Description** | | **Typical data** |
| ID | Yes | | Yes | VarChar | 5 | | -presence check | Auto generated | The next ID to be used when generating ID’s for users | | 4 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **File Name:** | | tracker table | | **Table Name** | | tracker | | **Primary Key Field:** | | ID | |
| **Method of access:** MySQL uses direct access through a python connector | | | | | | | | | | | |
| **Related to table:** clients | | | | | | **Foreign key:** N/A | | | | | |
| **Table description:** this table holds all information required to track a client’s fitness progress | | | | | | | | | | | |
| **Estimated number of records:** 120 | | | | | | | | | | | |
| **Type of data structure:** database table containing records | | | | | | | | | | | |
| **Estimated size of records in bytes:** 505 | | | | | | | | | | | |
| **Estimated size of table:** 66.7KB | | | | | | | | | | | |
| **Field name** | **Required** | | **Indexed** | **Data type** | **Length**  **/bytes** | | **Input mask/validation rule** | **Default value** | **Description** | | **Typical data** |
| ID | Yes | | Yes | VarChar | 5 | | -presence check | Auto generated | Unique identifier for clients | | t4 |
| exercise | Yes | | No | VarChar | 500 | | -presence check | \_ | A two dimensional list that is retrieved, amended and processed by the system to display a graph and record client fitness | | c7, ['back squat', ('2020-06-11', '100', '20', '3', '700')] |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **File Name:** | | class table | | **Table Name** | | class | | **Primary Key Field:** | | ID | |
| **Method of access:** MySQL uses direct access through a python connector | | | | | | | | | | | |
| **Related to table:** bookings, alerts | | | | | | **Foreign key:** N/A | | | | | |
| **Table description:** this table holds all information about active classes created by a trainer | | | | | | | | | | | |
| **Estimated number of records:** 20 | | | | | | | | | | | |
| **Type of data structure:** database table containing records | | | | | | | | | | | |
| **Estimated size of records in bytes:** 5 | | | | | | | | | | | |
| **Estimated size of table:** 2.3KB | | | | | | | | | | | |
| **Field name** | **Required** | | **Indexed** | **Data type** | **Length**  **/bytes** | | **Input mask/validation rule** | **Default value** | **Description** | | **Typical data** |
| ID | Yes | | Yes | VarChar | 4 | | -presence check | Auto generated | Uniquely identifies class | | Cl1 |
| Trainer | Yes | | No | VarChar | 30 | | -lookup check | Auto selected | Identifies the trainer assigned to that class | | Joe scott |
| Days | Yes | | No | VarChar | 30 | | -lookup check  -presence check | \_ | Identifies what days the class repeats on | | Mon,wed,thu |
| Start time | Yes | | No | VarChar | 5 | | -presence check  -lookup check | \_ | Identifies what time the class starts | | 06:15 |
| End time | Yes | | No | VarChar | 5 | | -presence check  -lookup check | \_ | Identifies what time the class ends | | 07:15 |
| overview | Yes | | No | VarChar | 100 | | -presence check | \_ | Gives an overview of the class | | 45 minute AMRAP |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **File Name:** | | Bookings table | | **Table Name** | | bookings | | **Primary Key Field:** | | Booking ID | |
| **Method of access:** MySQL uses direct access through a python connector | | | | | | | | | | | |
| **Related to table:** classes, alerts | | | | | | **Foreign key:** client ID, class ID | | | | | |
| **Table description:** this table holds all information about active bookings created by clients | | | | | | | | | | | |
| **Estimated number of records:** 240 | | | | | | | | | | | |
| **Type of data structure:** database table containing records | | | | | | | | | | | |
| **Estimated size of records in bytes:** 22 | | | | | | | | | | | |
| **Estimated size of table:** 5.5KB | | | | | | | | | | | |
| **Field name** | **Required** | | **Indexed** | **Data type** | **Length**  **/bytes** | | **Input mask/validation rule** | **Default value** | **Description** | | **Typical data** |
| Booking ID | Yes | | Yes | VarChar | 4 | | -auto generated | \_ | Uniquely defines a booking | | 1 |
| Client ID | Yes | | No | VarChar | 5 | | -auto selected | \_ | Uniquely defines a client | | C4 |
| Class ID | Yes | | No | VarChar | 5 | | -auto selected | \_ | Uniquely defines a class | | Cl2 |
| Class date | Yes | | No | VarChar | 8 | | -auto selected | \_ | Defines the date which a class occurs at | | 10/07/20 |

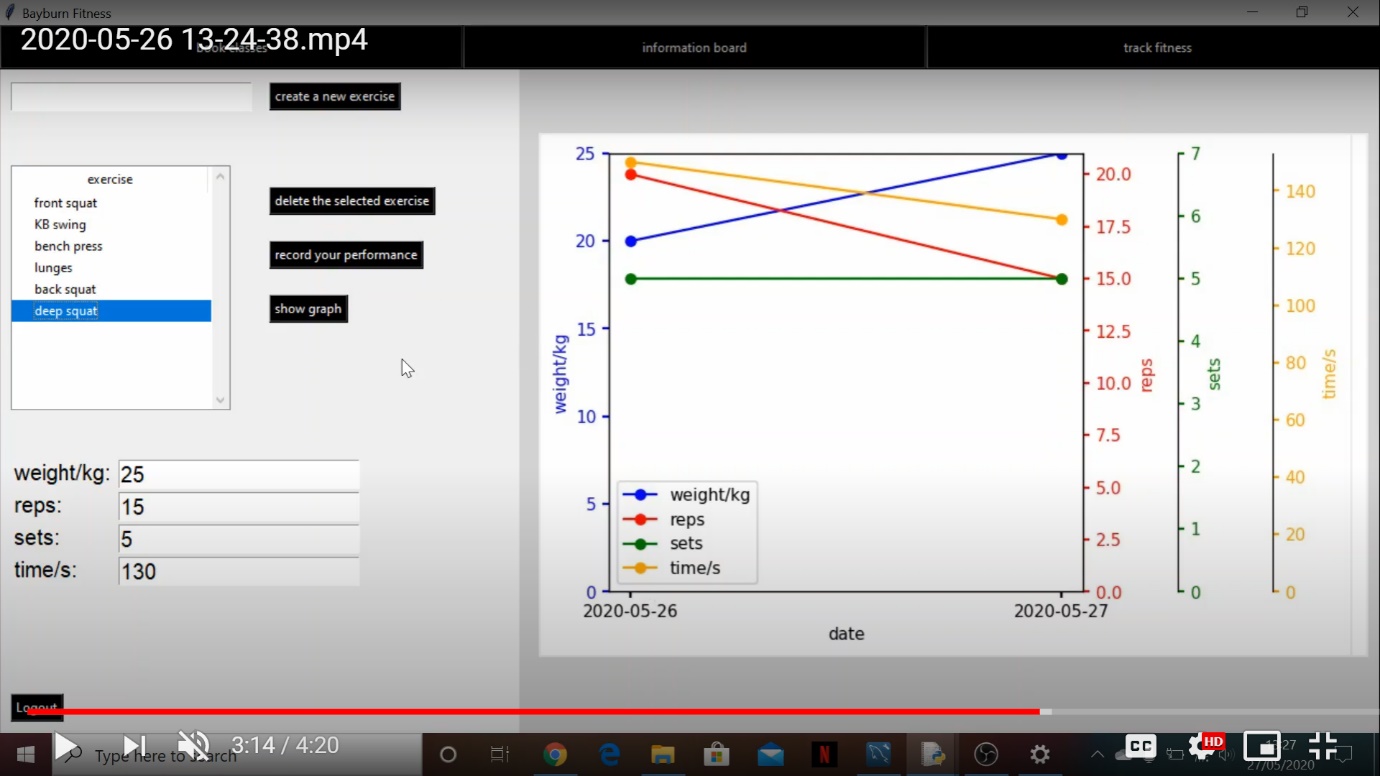
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **File Name:** | | Alerts table | | **Table Name** | | Alerts | | **Primary Key Field:** | | Client ID | |
| **Method of access:** MySQL uses direct access through a python connector | | | | | | | | | | | |
| **Related to table:** clients | | | | | | **Foreign key:** client ID | | | | | |
| **Table description:** this table holds all information about how the client will receive alerts | | | | | | | | | | | |
| **Estimated number of records:** 120 | | | | | | | | | | | |
| **Type of data structure:** database table containing records | | | | | | | | | | | |
| **Estimated size of records in bytes:** 8 | | | | | | | | | | | |
| **Estimated size of table:** 1.1KB | | | | | | | | | | | |
| **Field name** | **Required** | | **Indexed** | **Data type** | **Length**  **/bytes** | | **Input mask/validation rule** | **Default value** | **Description** | | **Typical data** |
| Client ID | Yes | | Yes | VarChar | 5 | | -auto selected | \_ | Defines the client for their alerts | | C4 |
| Alert type | Yes | | No | VarChar | 1 | | -lookup check | \_ | Defines the type of alert the client will receive | | 3 |
| Alert time | Yes | | No | VarChar | 2 | | -lookup check | \_ | Defines how long before the class in minutes the alert will be sent | | 35 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **File Name:** | | Attendances table | | **Table Name** | | Attendances | | **Primary Key Field:** | | Client ID, class ID | |
| **Method of access:** MySQL uses direct access through a python connector | | | | | | | | | | | |
| **Related to table:** clients, classes | | | | | | **Foreign key:** client ID, class ID | | | | | |
| **Table description:** this table holds the attendance records of all clients | | | | | | | | | | | |
| **Estimated number of records:** N/A | | | | | | | | | | | |
| **Type of data structure:** database table containing records | | | | | | | | | | | |
| **Estimated size of records in bytes:** 8 | | | | | | | | | | | |
| **Estimated size of table:** N/A | | | | | | | | | | | |
| **Field name** | **Required** | | **Indexed** | **Data type** | **Length**  **/bytes** | | **Input mask/validation rule** | **Default value** | **Description** | | **Typical data** |
| Client ID | Yes | | Yes | VarChar | 5 | | -auto selected | \_ | Defines the client for their alerts | | C4 |
| class ID | Yes | | No | VarChar | 5 | | -lookup check | \_ | Defines the class that has been attended | | 3 |
| date | Yes | | No | VarChar | 20 | | -lookup check | \_ | Defines the date of attendance | | 24/10/2020 |
| name | Yes | | No | VarChar | 100 | | -lookup check | \_ | Defines the name of client | | Callum Reid |
| phone | Yes | | No | VarChar | 11 | | -lookup check | \_ | Defines the phone number of client | | 07865123886 |
| **File Name:** | | Alerts2 table | | **Table Name** | | Alerts2 | | **Primary Key Field:** | | Client ID | |
| **Method of access:** MySQL uses direct access through a python connector | | | | | | | | | | | |
| **Related to table:** clients | | | | | | **Foreign key:** client ID | | | | | |
| **Table description:** this table holds the alert preferences of clients for the messaging system | | | | | | | | | | | |
| **Estimated number of records:** N/A | | | | | | | | | | | |
| **Type of data structure:** database table containing records | | | | | | | | | | | |
| **Estimated size of records in bytes:** 3 | | | | | | | | | | | |
| **Estimated size of table:** 360 | | | | | | | | | | | |
| **Field name** | **Required** | | **Indexed** | **Data type** | **Length**  **/bytes** | | **Input mask/validation rule** | **Default value** | **Description** | | **Typical data** |
| Client ID | Yes | | Yes | VarChar | 5 | | -auto selected | \_ | Defines the client for their alerts | | C4 |
| Alert Type | Yes | | No | VarChar | 5 | | -lookup check | \_ | Defines what alert should be sent | | 3 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **File Name:** | | Messages table | | **Table Name** | | Messages | | **Primary Key Field:** | | Sender ID | |
| **Method of access:** MySQL uses direct access through a python connector | | | | | | | | | | | |
| **Related to table:** trainers, managers | | | | | | **Foreign key:** Sender ID | | | | | |
| **Table description:** this table holds the messages sent by all trainers and managers | | | | | | | | | | | |
| **Estimated number of records:** N/A | | | | | | | | | | | |
| **Type of data structure:** database table containing records | | | | | | | | | | | |
| **Estimated size of records in bytes:** 100 | | | | | | | | | | | |
| **Estimated size of table:** N/A | | | | | | | | | | | |
| **Field name** | **Required** | | **Indexed** | **Data type** | **Length**  **/bytes** | | **Input mask/validation rule** | **Default value** | **Description** | | **Typical data** |
| Sender ID | Yes | | Yes | VarChar | 5 | | -auto selected | \_ | Defines the client for their alerts | | C4 |
| Sender name | Yes | | No | VarChar | 5 | | -lookup check | \_ | Defines the class that has been attended | | 3 |
| message | Yes | | No | VarChar | 20 | | -lookup check | \_ | Defines the date of attendance | | 24/10/2020 |

**Outputs**

**Line Graphs:**



Line graphs will be one form of output generated by the system. The main area where line graphs will be generated will be in the fitness tracker. Here line graphs will be generated to track the fitness of a client as well as make predictions and targets for their fitness in the future. Line graphs will be used to compare weight, reps, sets and time when doing a workout. They will also be used to create future predictions and targets by averaging the gradient of results and following it to a certain date.

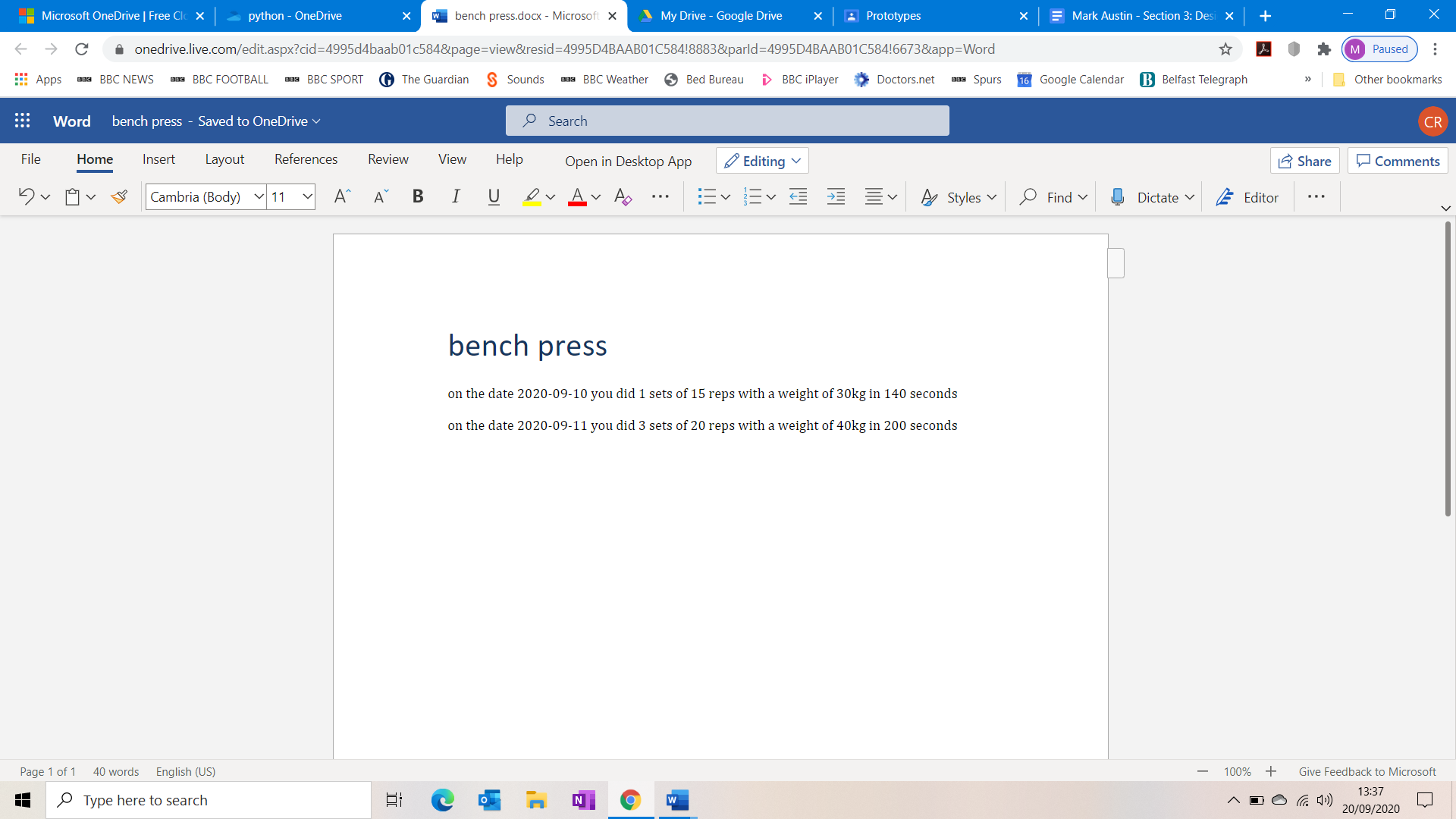
**Bar Charts:**

Bar charts will be anther form of output for my system and they will be used for several different purposes. They will mainly be generated by the manager to display overall information abut bookings and clients, this information can consist of: monthly comparisons between the amount of bookings and the amount of attendances (as shown above), monthly comparisons of the amount of bookings for each trainer, monthly sign ups of new clients, overall bookings for each class, etc. They will also be used to create future predictions by averaging the gradient of results and following it to a certain date, this will allow a manager to more accurately assess what classes and trainers are performing best and which need to be improved.

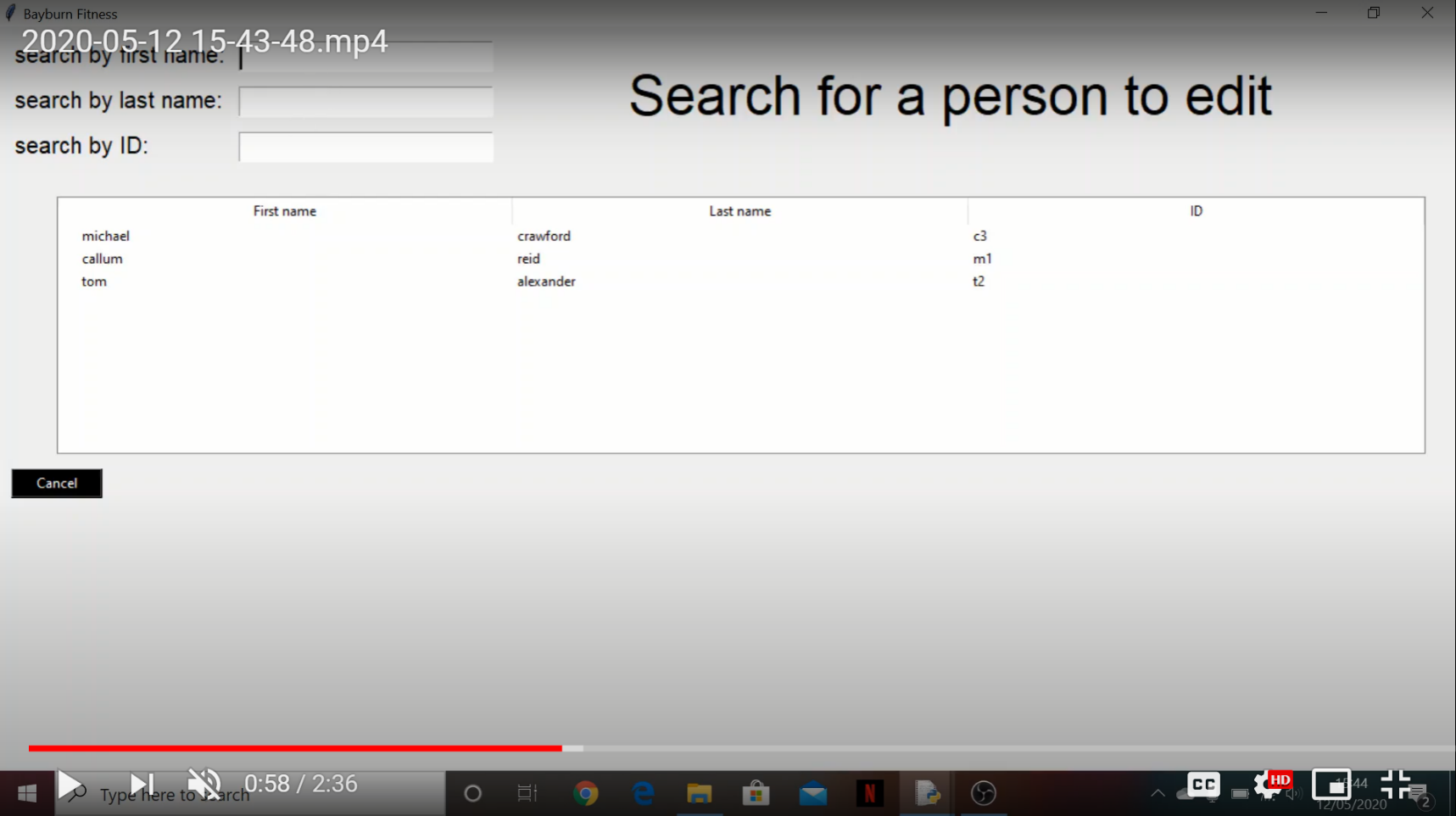
**Pie Charts:**

Pie charts will also be generated by the system and much like bar charts, they will be mostly generated by the manager to give useful data. The data that will be output by the pie charts will primarily be about clients. Pie charts will display data such as the ratio of male to female clients (as shown above), the ratio of fitness levels in clients, the ratio of age groups in clients, etc. this data will be used by managers to inform trainers on what classes would be best suited to the majority of clients. For example, if a large majority of clients are young and fit, then it would be beneficial to have some more high intensity classes.

**Word Document:**

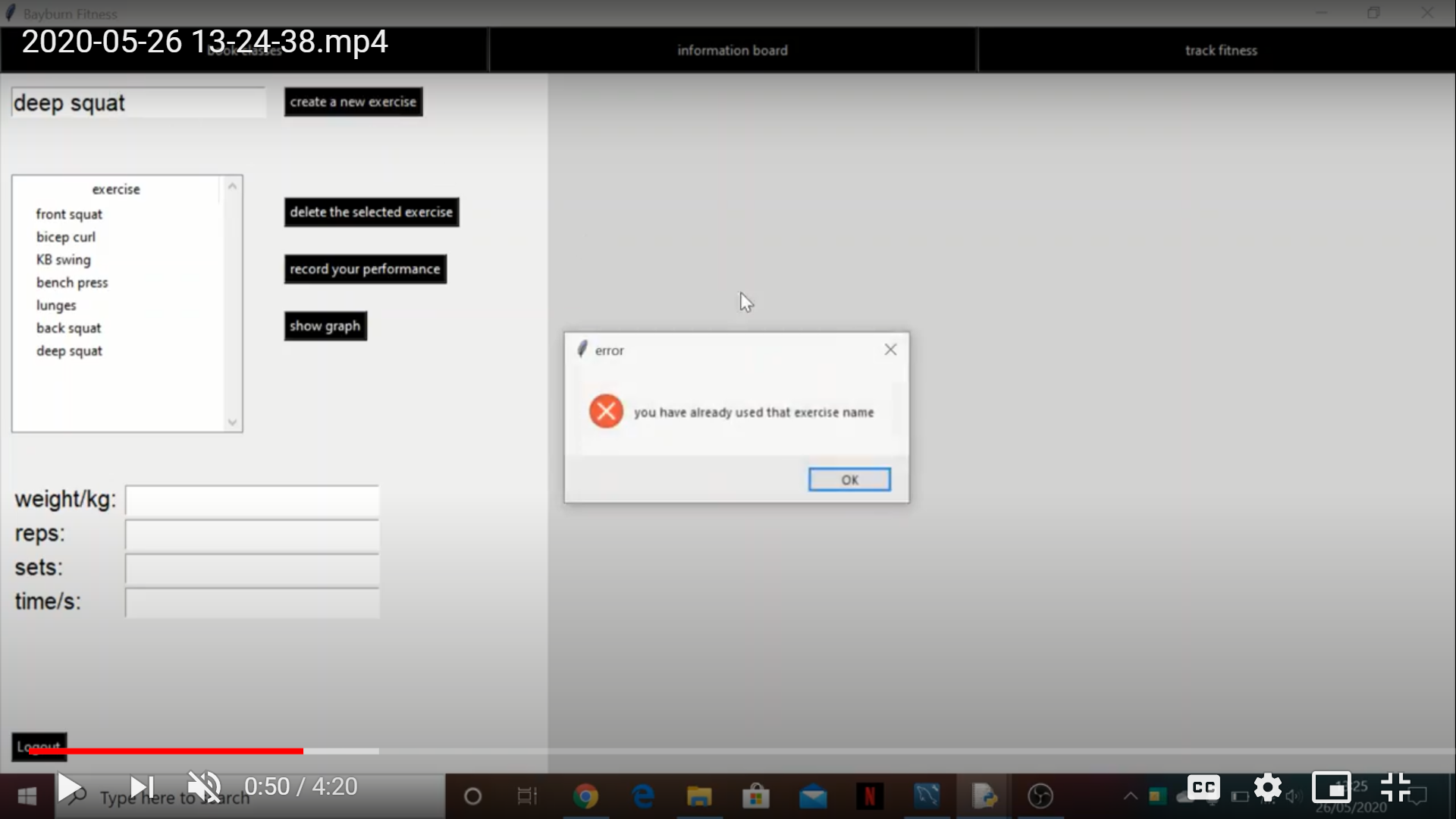


There are several situations where a word document will be generated. They will be generated by clients to give a permanent record of past fitness for a specific exercise (as shown above). They will also be generated by managers to keep records of all clients and their subsequent personal information, bookings and all of their subsequent details, and payments for bookings along with all transfer details for each payment. The word documents will take any data that would be displayed by a graph or chart and record it in an easy to read text format and save it so that it can be stored more permanently. Each word document will be saved with a different name and will also have a date tagged onto it in order to ensure that when searching through the data at a later date, it is all well organised and can be searched through easily.

**Tree view:**

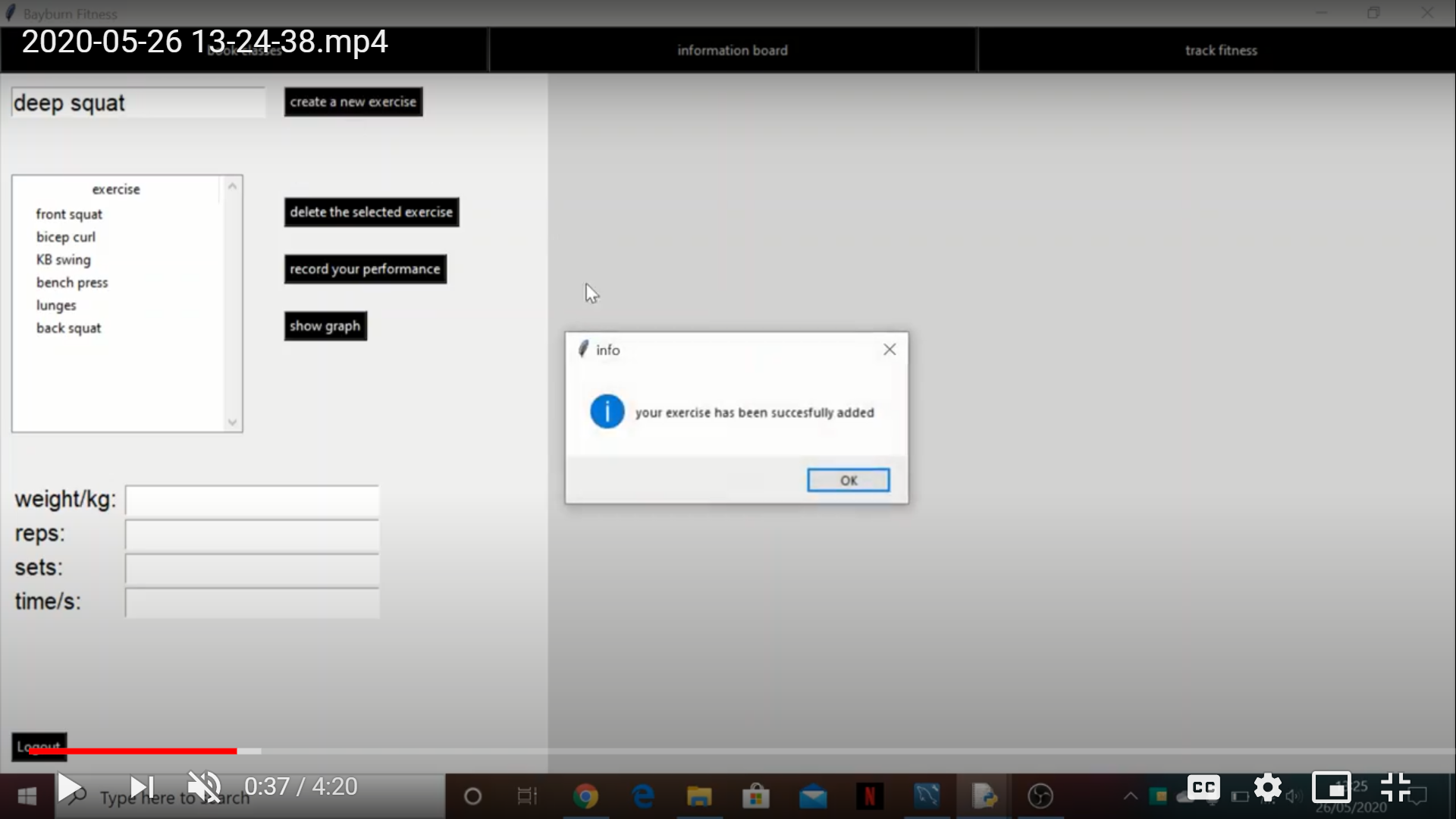
Tree views will be used on many screens in the system as they are great for displaying large amounts of data in a flat file format. They will display data such as client names and ID (as shown above), all possible classes hat can be booked by a clients. All classes that have been booked by a specific client, all exercises that have been recorded by a specific clients, all bookings made by any client, messages on the public message board, attendance of clients for a specific class, etc. Tree views will be used to display most of the data on a screen when accessing the MY SQL database as the are exceptionally good at displaying data in a easy to read format.

**Error Messages:**

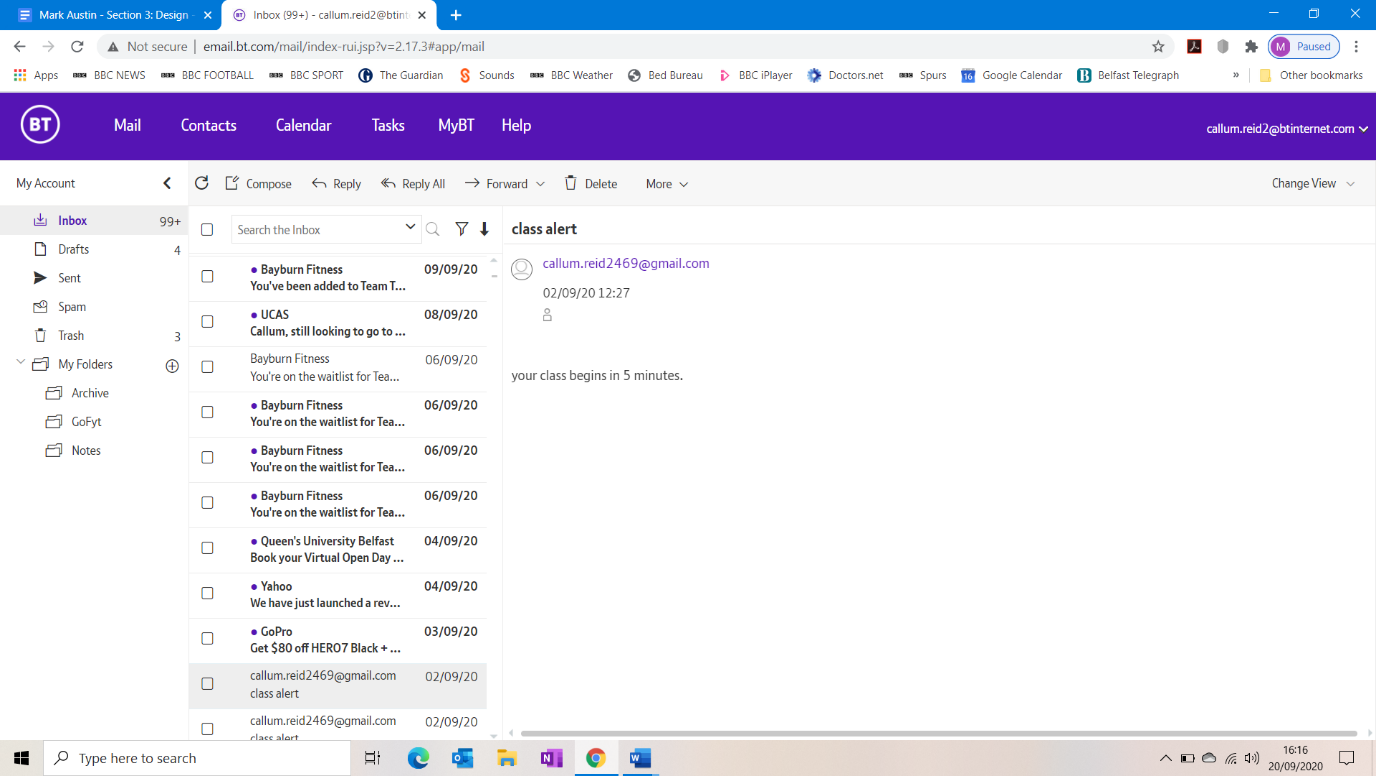


Error messages will be generated by the system in situations where data is incorrectly inputted by the user or data that is entered by the user does not meet a set of criteria. The error messages will inform the user of what has gone wrong and explain how to fix the problem by listing any data entry criteria that must be met. These messages should help increase the intuitiveness of the system and allow the user to operate the system without any prior knowledge or training.

**Information messages:**

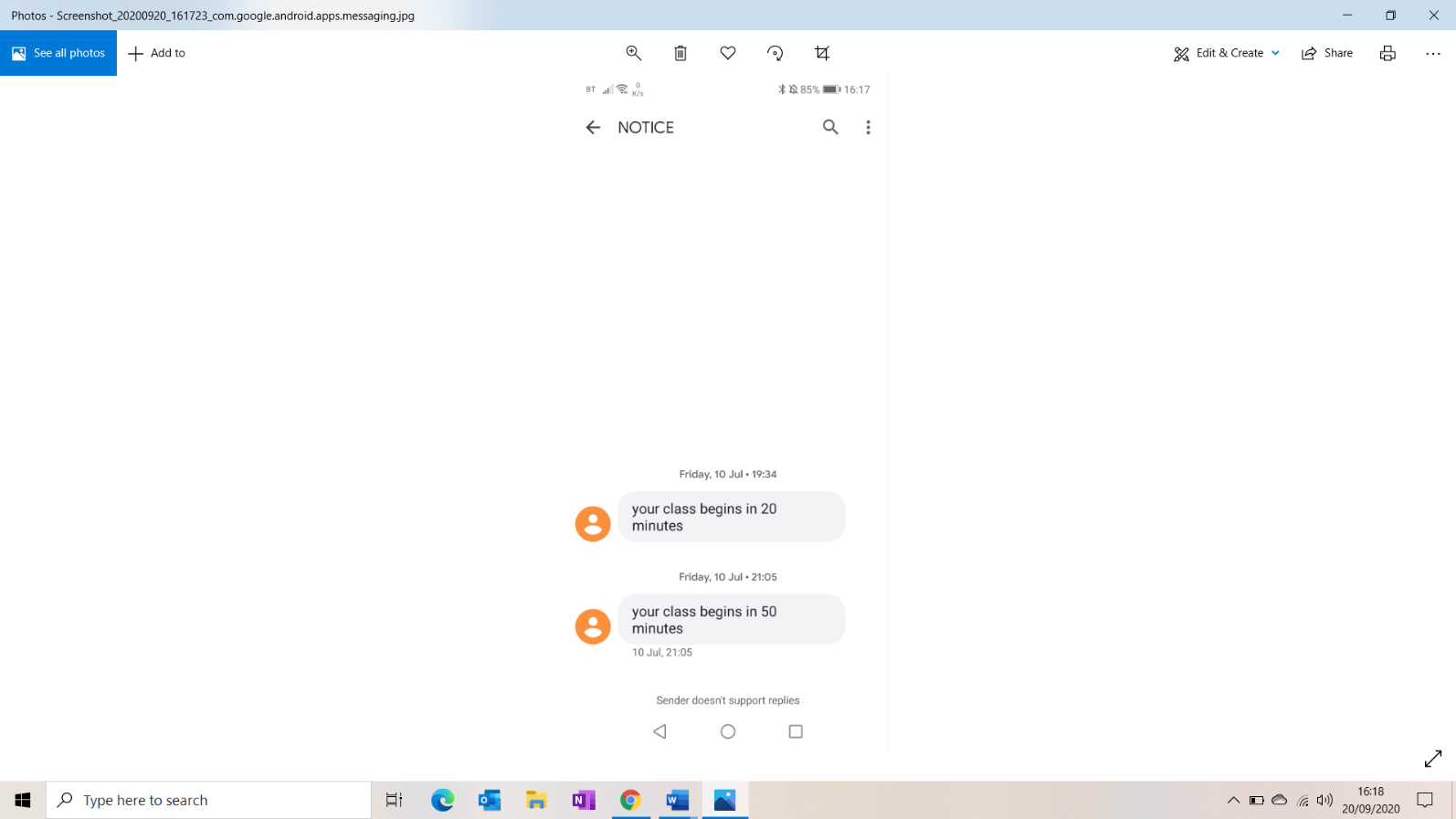


Information messages will be generated when the system performs an operation in the background and it would be beneficial to make the user aware of it. These messages will be generated when data has been successfully added to the database, data has been successfully updated and when data has been successfully deleted from the database. These messages will also be generated when a use clicks an information button on each screen which will tell the user about the screen and inform them of any entry requirements for data points.

**Email:**

Emails will be generated and sent to clients to inform them of when a class is starting. When booking a class a client will select whether they want to be informed of the class by email, text, both or neither. They will also be asked how long before the class they wish to be informed. If the client selects to be informed by email 5 minutes before a class then the email above will be sent 5 minutes before their booking time. Clients will also be emailed when their account is created or deleted. The length of time before a class when the client is notified by either text or email can be changed by the client at any time.

**Text:**



Texts will be generated and sent to clients to inform them of when a class is starting. When booking a class a client will select whether they want to be informed of the class by email, text, both or neither. They will also be asked how long before the class they wish to be informed. If the client selects to be informed by text 20 minutes before a class then the text above will be sent 20 minutes before their booking time. The length of time before a class when the client is notified by either text or email can be changed by the client at any time.

**Additional inputted and outputted data:**

Below I will highlight all the data that is outputted from the system, when it is outputted from the system and why it is outputted from the system. I will also be highlighting data that is calculated from other pieces of data and outputted, this data may not be initially stored in the database but it is eventually calculated from data that is, and then outputted.

**Outputted data:**

* Class reminder emails will be sent to clients 30 minutes before a class they have booked into. These emails are to try and reduce the amount of people who book into classes and forget to attend.
* Class reminder texts will be sent to clients 30 minutes before a class they have booked into. These texts are to try and reduce the amount of people who book into classes and forget to attend.
* When an announcement is made by a manager or a trainer on the information board an email will be sent to all the clients, containing the information of the announcement.
* When an announcement is made by a manager or a trainer on the information board text will be sent to all the clients, containing the information of the announcement.
* When a client selects an exercise on the fitness tracker, they can generate a graph which shows their progress on that exercise.
* A manager will be able to generate a list of all the people who attended classes, this list will show which class the person attended, at what time they attended, with which trainer they attended and whether or not they have paid for their class or not.
* Managers will be able to generate several reports based on classes which will show which classes have the most people booked into, which classes have the most people attending, which trainer has the most clients attending their classes, etc.
* Managers will have the ability to search for a client or trainer and access all their personal information. This information will then be outputted onto the screen.
* Trainers will be able to view all of the people who have booked into their classes, this information will be outputted to one of the screens that the trainer will have access to.

**Calculated data to be outputted:**

* Clients will be allowed to see statistical averages of there performances on the fitness tracker these averages and estimations will be outputted beside the graph on the fitness tracker screen.
* Managers will be allowed to see statistical averages of class bookings and attendances which will be outputted in the form of reports and to the manager reports screen.

**Validation checks**

In this section I will outline the pseudocode for the various forms of validation checks that I will be using for data entry throughout the development of my system.

**Range check:**

This check ensures that a value is within a certain range of numerical values. This check could be used to validate ages.

START

INPUT value

IF value < lowerbound OR value > upperbound:

OUTPUT “error message”

DECLINE value

ELSE:

ACCEPT value

END IF

END

**Presence check:**

This check ensures that data has actually been input by the user. This check could be used to validate almost all variables that are not pre-selected.

START

IF value == “”:

OUTPUT “error message”

DECLINE value

ELSE:

ACCEPT value

END IF

END

**Lookup check:**

this check ensures that the inputted data matches an option from a list of pre-specified options. This check could be used for gender.

There is no pseudocode for this validation check as when inputting data which must be selected from a list, I will use dropdown menus which eliminate the possibility of entering anything other than an option on the list.

**Format check:**

A format check is a check that ensures a value follows a specific structure. The following is a format check for an email. This check could also be used for thigs such as postcodes however the pseudocode would have to be modified.

START

INPUT email

Accept = FALSE

has@symbol = FALSE

FOR I IN email:

IF I == ‘@’:

has@symbol = TRUE

END IF

END FOR

IF has@symbol == FALSE:

OUTPUT “error message”

DECLINE email

ELIF email[LEN(email)-5:LEN(email)-1] != “.com”

OUTPUT “error message”

DECLINE email

ELSE:

ACCEPT email

END IF

END

**Length check:**

This validation check ensures that the value entered is of a certain length or between a range of lengths. This check could be used for phone numbers.

START

INPUT value

IF LEN(value) < lowerbound OR LEN(value) > upperbound:

OUTPUT “error message”

DECLINE value

ELSE:

ACCEPT value

END IF

END

**Type check:**

This validation ensures that the value entered is of a certain data type. This could be used for phone numbers. It could also be used for names however the pseudocode would have to be modified.

START

INPUT value

IF value.isnumeric == FALSE:

OUTPUT “error message”

DECLINE value

ELSE:

ACCEPT value

END IF

END

**Uniqueness check:**

This check ensures that the inputted data does not match any data that is a list. The following is a uniqueness check for a username. This check could also be used for IDs however my system will automatically generate these in an incremental order.

START

INPUT value

list = SQL[SELECT username FROM clients]

accept = TRUE

FOR i IN list:

If value == i:

accept = FALSE

END IF

END FOR

IF accept == FALSE:

OUTPUT “error message”

DECLINE value

ELSE:

ACCEPT value

END IF

END

**Pseudocode functions:**

In this section I will show pseudocode for all major functions in my system. I will leave out small functions used for page navigation and screen resetting/updating/clearing etc.

**Connecting to database:**

START

Mydb = mysql.connector.connect(

Host = ‘host’

User = ‘user’

Password = ‘password’

Database = ‘database’)

END

**Submit username and password for login:**

START

INPUT username

INPUT password

logins = MYSQL[SELECT \* FROM logins]

Found = FALSE

FOR i IN logins:

IF username == i[1] AND password == i[2]

Found = TRUE

ID = i[0]

END IF

END FOR

IF found == FALSE:

OUTPUT “error message”

ELIF ID[0] == ‘m’:

RaiseFrame(manager)

ELIF ID[0] == ‘t’:

RaiseFrame(trainer)

ELIF ID[0] == ‘c’:

RaiseFrame(client)

END IF

END

**Submit a user to be added:**

START

INPUT role

IF role == manager:

INPUT username

INPUT password

INPUT Firstname

INPUT lastname

INPUT DOB

INPUT phone

INPUT email

INPUT gender

INPUT postcode

INPUT address

INPUT qualifications

INPUT experience

ListOfIDs = MYSQL[SELECT ID FROM IDs]

NextID = (‘M’ + (ListOfIDs[-1] + 1))

MYSQL[INSERT INTO IDs NextID]

MYSQL[INSERT INTO managers Firstname

lastname

DOB

phone

email

gender

postcode

address

qualifications

experience

NextID

]

MYSQL[INSERT INTO logins nextID

Username

Password

]

ELIF role == trainer:

INPUT username

INPUT password

INPUT Firstname

INPUT lastname

INPUT DOB

INPUT phone

INPUT email

INPUT gender

INPUT postcode

INPUT address

INPUT qualifications

INPUT experience

ListOfIDs = MYSQL[SELECT ID FROM IDs]

NextID = (‘M’ + (ListOfIDs[-1] + 1))

MYSQL[INSERT INTO IDs NextID]

MYSQL[INSERT INTO trainers Firstname

lastname

DOB

phone

email

gender

postcode

address

qualifications

experience

NextID

]

MYSQL[INSERT INTO logins nextID

Username

Password

]

ELIF role == client:

INPUT username

INPUT password

INPUT Firstname

INPUT lastname

INPUT DOB

INPUT phone

INPUT email

INPUT gender

INPUT postcode

INPUT address

INPUT fitness

INPUT smokes

INPUT height

INPUT weight

INPUT illnesses

INPUT injuries

ListOfIDs = MYSQL[SELECT ID FROM IDs]

NextID = (‘M’ + (ListOfIDs[-1] + 1))

MYSQL[INSERT INTO IDs NextID]

MYSQL[INSERT INTO trainers Firstname

lastname

DOB

phone

email

gender

postcode

address

fitness

smokes

height

weight

illnesses

injuries

NextID

]

MYSQL[INSERT INTO logins nextID

Username

Password

]

END

**Refine the search in the search window for users that you want to edit(function is run every time someone enters something into one of the search boxes):**

START

INPUT Firstname

INPUT Lastname

INPUT ID

records = treeview.get\_children()

FOR elements IN records:

treeview.delete(elements)

END FOR

Managers = MYSQL[SELECT Firstname, Lastname, ID FROM managers]

Trainers = MYSQL[SELECT Firstname, Lastname, ID FROM trainers]

Clients = MYSQL[SELECT Firstname, Lastname, ID FROM clients]

List = []

FOR I IN (managers,trainers,clients):

List.append(i)

New\_list = []

Selectors = []

if Firstname != "":

selectors.append(Firstname)

else:

selectors.append("@")

END IF

if Lastname != "":

selectors.append(Lastname)

else:

selectors.append("@")

END IF

if ID != "":

selectors.append(ID)

else:

selectors.append("@")

END IF

FOR i IN list:

match = True

IF selectors[0] != "@":

IF selectors[0] != i[0][0:len(selectors[0])]:

match = False

END IF

END IF

IF selectors[1] != "@":

IF selectors[1] != i[1][0:len(selectors[1])]:

match = False

END IF

END IF

IF selectors[2] != "@":

IF selectors[2] != i[2][0:len(selectors[2])]:

match = False

END IF

END IF

IF match == True:

new\_list.append(i)

END IF

END FOR

FOR row IN new\_list:

treeview.insert('','end',text = row[0],values=(row[1],row[2]))

END FOR

END

**Select a person to edit from the search window and fill their data into the ‘add user’ screen:**

START

INPUT ID

Managers = MYSQL[SELECT \* FROM managers]

Trainers = MYSQL[SELECT \* FROM trainers]

Clients = MYSQL[SELECT \* FROM clients]

List = []

FOR i IN (managers,trainers,clients):

List.append(i)

END FOR

FOR i IN list:

IF i[0] == ID:

Person = i

END IF

END FOR

Logins = MYSQL[SELECT \* FROM logins]

FOR I in logins:

IF I[0] == ID:

Username = i[1]

Password = i[2]

END IF

END FOR  
IF ID[0] == ‘m’ or ID[]0 == ‘t’:

username.set(username)

password.set(password)

firstname.set(person[1])

lastname.set(person[2])

date.set(person[3])

phone.set(person[4])

email.set(person[5])

gender.set(person[6])

postcode.set(person[7])

address.set(person[8])

qualifications.set(person[9])

experience.set(person[10])

ELSE:

username.set(username)

password.set(password)

firstname.set(person[1])

lastname.set(person[2])

date.set(person[3])

phone.set(person[4])

email.set(person[5])

gender.set(person[6])

postcode.set(person[7])

address.set(person[8])

fitness.set(person[9])

smokes.set(person[10])

height.set(person[11])

weight.set(person[12])

illnesses.set(person[13])

injuries.set(person[14])

END IF

END

**Edit a user who has been selected:**

START

INPUT selectedID

MYSQL[DELETE FROM logins WHERE ID = ‘selectedID’]

MYSQL[DELETE FROM clients WHERE ID = ‘selectedID’]

MYSQL[DELETE FROM managers WHERE ID = ‘selectedID’]

MYSQL[DELETE FROM trainers WHERE ID = ‘selectedID’]

INPUT role

IF role == manager:

INPUT username

INPUT password

INPUT Firstname

INPUT lastname

INPUT DOB

INPUT phone

INPUT email

INPUT gender

INPUT postcode

INPUT address

INPUT qualifications

INPUT experience

NextID = selectedID

MYSQL[INSERT INTO managers Firstname

lastname

DOB

phone

email

gender

postcode

address

qualifications

experience

NextID

]

MYSQL[INSERT INTO logins nextID

Username

Password

]

ELIF role == trainer:

INPUT username

INPUT password

INPUT Firstname

INPUT lastname

INPUT DOB

INPUT phone

INPUT email

INPUT gender

INPUT postcode

INPUT address

INPUT qualifications

INPUT experience

NextID = selectedID

MYSQL[INSERT INTO trainers Firstname

lastname

DOB

phone

email

gender

postcode

address

qualifications

experience

NextID

]

MYSQL[INSERT INTO logins nextID

Username

Password

]

ELIF role == client:

INPUT username

INPUT password

INPUT Firstname

INPUT lastname

INPUT DOB

INPUT phone

INPUT email

INPUT gender

INPUT postcode

INPUT address

INPUT fitness

INPUT smokes

INPUT height

INPUT weight

INPUT illnesses

INPUT injuries

NextID = selectedID

MYSQL[INSERT INTO trainers Firstname

lastname

DOB

phone

email

gender

postcode

address

fitness

smokes

height

weight

illnesses

injuries

NextID

]

MYSQL[INSERT INTO logins nextID

Username

Password

]

END

**Delete a person who has been selected:**

START

INPUT selectedID

MYSQL[DELETE FROM logins WHERE ID = ‘selectedID’]

MYSQL[DELETE FROM bookings WHERE client\_ID= ‘selectedID’]

MYSQL[DELETE FROM tracker WHERE ID= ‘selectedID’]

MYSQL[DELETE FROM alerts WHERE client\_ID= ‘selectedID’]

MYSQL[DELETE FROM clients WHERE ID = ‘selectedID’]

MYSQL[DELETE FROM managers WHERE ID = ‘selectedID’]

MYSQL[DELETE FROM trainers WHERE ID = ‘selectedID’]

END

**Submit a new exercise to the fitness tracker:**

START

GET currentID

INPUT exercise

MYSQL[INSERT INTO tracker (currentID,exercise)]

END

**Delete exercise:**

START

INPUT exercise

MYSQL[DELETE FROM tracker WHERE exercise = exercise]

END

**Add fitness data to an exercise:**

START

INPUT exercise\_name

INPUT weight

INPUT reps

INPUT sets

INPUT time

INPUT currentID

Exercises = MYSQL[SELECT FROM tracker WHERE ID = currentID]

FOR i IN exercise:

If i[0] == exercise\_name:

Exercise = i

END IF

END FOR

MYSQL[DELETE FROM tracker WHERE ID = currentID AND exercise = exercise]

Date = datetime.today

Exercise\_ID = exercise[0]

Exercise\_data = exercise[1]

New = (date,weight,reps,sets,time)

Exercise\_data.append(new)

MYSQL[INSERT INTO tracker (exercise\_ID,Exercise\_data)]

END

**Plot a graph of a specific exercise:**

START

INPUT currentID

INPUT exercise\_name

Exercises = MYSQL[SELECT FROM tracker WHERE ID = currentID]

FOR i IN exercise:

If i[0] == exercise\_name:

Exercise = i

END IF

END FOR

Data = exercise[1:len(exercise)]

Date = []

Weight = []

Reps = []

Sets = []

Time = []

FOR I IN data:

Date.append(data[1])

weight.append(data[2])

reps.append(data[3])

sets.append(data[4])

time.append(data[5])

END FOR

PLOT GRAPH USING MATPLOTLIB

Xaxis = DATE

Yaxis1 = weight

Yaxis1 = reps

Yaxis1 = sets

Yaxis1 = time

SAVE PLOT as “line plot.jpg”

Show “line plot.jpg”

END

**Update treeview of classes:**

START

INPUT my\_date

my\_date2 = datetime.strptime(my\_date, '%d/%m/%y')

day = (my\_date2.strftime("%a")).lower()

classes = MYSQL [SELECT \* FROM classes]

records3 = treeview3.get\_children()

FOR elements IN records3:

treeview3.delete(elements)

END FOR

FOR i IN classes:

match = False

days = i[2].split(',')

FOR p IN days:

IF p == day:

match = True

END IF

END FOR

IF match == True:

treeview3.insert( (i[0],i[1],i[3],i[4],my\_date,i[5])

END FOR

END

**Book a class:**

START

INPUT class\_ID

INPUT client\_ID

INPUT class\_date

Bookings = MYSQL[SELECT \* FROM bookings]

IF bookings == []:

nextID = 1

ELSE:

nextID = int(bookings[-1][0])+1

bookingID = nextid

MYSQL[INSERT INTO bookings(booking\_ID,client\_ID,class\_ID,class\_date) VALUES (bookingID,client\_ID,class\_ID,class\_Date)]

END

**Delete a selected booking:**

START

INPUT ID

MYSQL[DELETE FROM bookings WHERE booking\_ID = ID ]

END

**Refresh bookings treeview and table:**

START

INPUT currentID

records4 = treeview4.get\_children()

FOR elements IN records4:

treeview4.delete(elements)

END FOR

Bookings = MYSQL[SELECT \* FROM bookings]

FOR i IN range(0,len(bookings)-1):

MYSQL[DELETE FROM bookings WHERE booking\_ID <> bookings[i][0]

AND client\_ID = bookings[i][1] AND class\_ID = bookings[i][2] AND

class\_date = bookings[i][3];]

Bookings = MYSQL[SELECT \* FROM bookings]

END FOR

Bookings = MYSQL[SELECT \* FROM bookings]

FOR i IN bookings:

D1 = date.today()

D2 = i[3]

IF D1 > D2:

delID = i[0]

MYSQL[DELETE FROM bookings WHERE booking\_ID = delID;]

END IF

END FOR

Bookings = MYSQL[SELECT \* FROM bookings]

Bookings2 = []

FOR i IN bookings:

IF i[1] == currentID:

bookings2.append(i)

END IF

END FOR

classes = MYSQL[SELECT \* FROM classes]

FOR i IN bookings:

class\_date = i[3]

newID = i[0]

FOR p IN classes:

IF i[2] == p[0]:

start\_time = p[3]

end\_time = p[4]

trainer = p[1]

overview = p[5]

treeview4.insert(trainer,start\_time,end\_time,class\_date,overview)

END IF

END FOR

END FOR

**Refresh on screen alert selectors:**

alerts = MYSQL[SELECT \* FROM alerts]

FOR i IN alerts:

IF i[0] == ID:

alert\_time.set(int(i[2]))

message\_type.set(int(i[1]))

END IF

END FOR

END

**Update alerts table:**

INPUT alert\_time

INPUT message\_type

INPUT current\_ID

If current\_ID[0] == ‘c’

MYSQL[DELETE FROM alerts WHERE client\_ID = current\_ID;]

MYSQL[INSERT INTO alerts(client\_ID,alert\_type,alert\_time) VALUES

(current\_ID,message\_type,alert\_time);]

END IF

END

**Send alerts:**

START

bookings = MYSQL[SELECT \* FROM bookings]

clients = MYSQL[SELECT \* FROM clients]

alerts = MYSQL[SELECT \* FROM alerts]

classes = MYSQL[SELECT \* FROM classes]

alert\_list = []

FOR i IN bookings:

FOR p IN alerts:

IF i[1] == p[0]:

class\_date = i[3]

alert\_type = p[1]

alert\_time = p[2]

FOR u IN classes:

IF u[0] == i[2]:

class\_time = u[3]

END IF

END FOR

FOR u IN clients:

IF u[0] == i[1]:

email = u[5]

phone = u[4]

END IF

END FOR

END IF

new = (alert\_type,alert\_time,class\_time,class\_date,email,phone)

alert\_list.append(new)

END FOR

END FOR

alert\_list2 = []

FOR i IN alert\_list:

IF i[0] == '4' or i[0] == '0' or i[1] == '0':

PASS

ELSE:

alert\_list2.append(i)

END IF

END FOR

alert\_list = alert\_list2

alert\_list2 = []

D1 = date.today()

T1 = time.now()

FOR i IN alert\_list:

D2 = i[3]

T2 = i[2] – i[1]

IF D1 != D2:

PASS

ELSE:

IF T1 != T2:

PASS

ELSE:

alert\_list2.append(i)

END IF

END IF

END FOR

alert\_list = alert\_list2

FOR i IN alert\_list:

IF i[0] == '2':

send\_email(i[4],i[1])

ELIF i[0] == '1':

send\_text(i[4],i[1])

ELIF i[0] == '3':

send\_email(i[4],i[1])

send\_text(i[4],i[1])

END IF

END FOR

root.after(60000,send\_alert)

END

**send an email:**

START

INPUT receiver

INPUT mins

port = 465

smtp\_server = "smtp.gmail.com"

sender\_email = "sender email"

receiver\_email = str(receiver)

password = "password"

message = """\

Subject: class alert

your class begins in {} minutes.""".format(str(mins))

context = ssl.create\_default\_context()

with smtplib.SMTP\_SSL(smtp\_server, port, context=context) as server:

server.login(sender\_email, password)

server.sendmail(sender\_email, receiver\_email, message)

END

**Send text:**

START

INPUT receiver

INPUT mins

client = boto3.client("sns",aws\_access\_key\_id="MY ACCES KEY",aws\_secret\_access\_key="MY SECRET KEY",region\_name="MY REGION")

client.publish(PhoneNumber=str(receiver),Message="your class begins in {} minutes".format(mins))

END

**Generate a report on an exercise:**

START

TRY:

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM tracker")

tracker = mycursor.fetchall()

FOR i in tracker:

IF i[0] == ID and eval(i[1])[0] == selected\_exercise:

exercise = i[1]

END IF

END FOR

exercise = eval(exercise)

data = exercise[1:len(exercise)]

IF data == []:

SHOW ERROR

ELSE:

document = Document()

document.add\_heading(selected\_exercise, 0)

FOR i IN data:

new\_term = "on the date {} you did {} sets of {} reps with a

weight of {}kg in {}

seconds".format(i[0],i[3],i[2],i[1],i[4])

p = document.add\_paragraph(new\_term)

END FOR

document.save('{}.docx'.format(selected\_exercise))

END IF

EXCEPT:

SHOW ERROR

END

**Create a class:**

INPUT monday

INPUT tuesday

INPUT wednesday

INPUT thursday

INPUT friday

INPUT saturday

INPUT sunday

INPUT time1

INPUT time2

INPUT class\_duration

INPUT overview

days = []

if mon == 1:

days.append("mon")

if tue == 1:

days.append("tue")

if wed == 1:

days.append("wed")

if thu == 1:

days.append("thu")

if fri == 1:

days.append("fri")

if sat == 1:

days.append("sat")

if sun == 1:

days.append("sun")

verified = True

if days == []:

verified = False

if class\_durationx == "":

verified = False

if len(overview) == 1:

verified = False

try:

duration = int(class\_durationx)

if verified == True:

start\_time = (str(time1x) + ":" + str(time2x))

end\_time\_mins = (((int(time1x) \* 60) + int(time2x)) + duration)

temp = str(end\_time\_mins % 60)

if len(temp) == 1:

temp = "0" + temp

end\_time = str(end\_time\_mins // 60) + ":" + temp

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM trainers")

trainers = mycursor.fetchall()

for i in trainers:

if i[0] == ID:

trainer = (str(i[1]) + " " + str(i[2]))

temp = start\_time.split(":")

if len(temp[1]) == 1:

start\_time = temp[0] + ":" + "0" + temp[1]

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM classes")

classes = mycursor.fetchall()

if classes == []:

nextID = 1

else:

nextID = int(classes[-1][0])+1

classID = nextID

days = str(days)

days = days.replace("'","")

days = days.replace("[","")

days = days.replace("]","")

days = days.replace(" ","")

mycursor = mydb.cursor()

MYSQL = """INSERT INTO classes(class\_ID,trainer,days,start\_time,end\_time,overview) VALUES (classID,trainer,days,start\_time,end\_time,overview);"""

update\_treeview8()

messagebox.showinfo("info", "your class has been created")

else:

messagebox.showerror("error", "you must fill in all fields")

except:

messagebox.showerror("error", "class duration must be a number")

**select class:**

INPUT cID

MsgBox = tk.messagebox.askquestion ('Delete','Are you sure you want to delete this class',icon = 'warning')

if MsgBox == 'yes':

mycursor = mydb.cursor()

MYSQL = """DELETE FROM classes WHERE class\_ID = cID AND class\_ID = cID"""

mycursor = mydb.cursor()

MYSQL = """DELETE FROM bookings WHERE class\_ID = cID AND class\_ID = cID"""

update\_treeview8()

**update treeview8:**

records8 = treeview8.get\_children()

for elements in records8:

treeview8.delete(elements)

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM trainers")

trainers = mycursor.fetchall()

for i in trainers:

if i[0] == ID:

name = str(i[1] + " " + i[2])

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM classes")

classes = mycursor.fetchall()

for i in classes:

if i[1] == name:

treeview8.insert('','end',text = i[0],values=(i[2],i[3],i[4],i[5]))

**update treeview9:**

trainers = MYSQL = "SELECT \* FROM trainers"

for i in trainers:

if i[0] == ID:

Tname = str(i[1]) + " " + str(i[2])

records9 = treeview9.get\_children()

for elements in records9:

treeview9.delete(elements)

classes = MYSQL = "SELECT \* FROM classes"

dates = []

d = datetime.today() + timedelta(days=30)

for i in range(0,31):

dates.append(d)

d = d - timedelta(days=1)

dates2 = []

for i in dates:

date = str(i)[0:10]

day = (datetime.strptime(date.replace('-',' '), '%Y %m %d').strftime('%a')).lower()

new = (date,day)

dates2.append(new)

dates2.reverse()

for i in dates2:

for p in classes:

if str(p[1]) != Tname:

pass

found = False

for q in p[2].split(','):

if q == i[1]:

found = True

if found == True:

treeview9.insert('','end',text = i[0],values=(i[0],p[0],p[3],p[4],p[5]))

**check bookings:**

records10 = treeview10.get\_children()

for elements in records10:

treeview10.delete(elements)

item = treeview9.selection()

date = (treeview9.item(item)['text'])

date = date.split('-')

date.reverse()

date = (date[0],date[1],date[2][:2])

date = (date[0]+'/'+date[1]+'/'+date[2])

classID = (treeview9.item(item)['values'][1])

bookings = MYSQL = “SELECT \* FROM bookings"

clients = MYSQL = “SELECT \* FROM clients"

for i in bookings:

if str(i[2]) == str(classID) and str(i[3]) == str(date):

clientID = i[1]

for i in clients:

if i[0] == clientID:

name = (i[1] + " " + i[2])

phone = i[4]

treeview10.insert('','end',text = i[0],values=(clientID,classID,date,name,phone))

**mark as present:**

d = str(datetime.today())[0:10]

date = d.split('-')

date.reverse()

date = (date[0],date[1],date[2][:2])

datet = (date[0]+'/'+date[1]+'/'+date[2])

item = treeview10.selection()

clientID = (treeview10.item(item)['values'][0])

classID = (treeview10.item(item)['values'][1])

date = (treeview10.item(item)['values'][2])

name = (treeview10.item(item)['values'][3])

phone = (treeview10.item(item)['values'][4])

if date == datet:

MsgBox = tk.messagebox.askquestion ('record','Do you wish to record this person as present',icon = 'info')

if MsgBox == 'yes':

mycursor = mydb.cursor()

MYSQL = """INSERT INTO attendances(clientID,classID,date\_,name\_,phone) VALUES (clientID,classID,date,name,phone);"""

messagebox.showinfo("info", "this attendance has been rcorded")

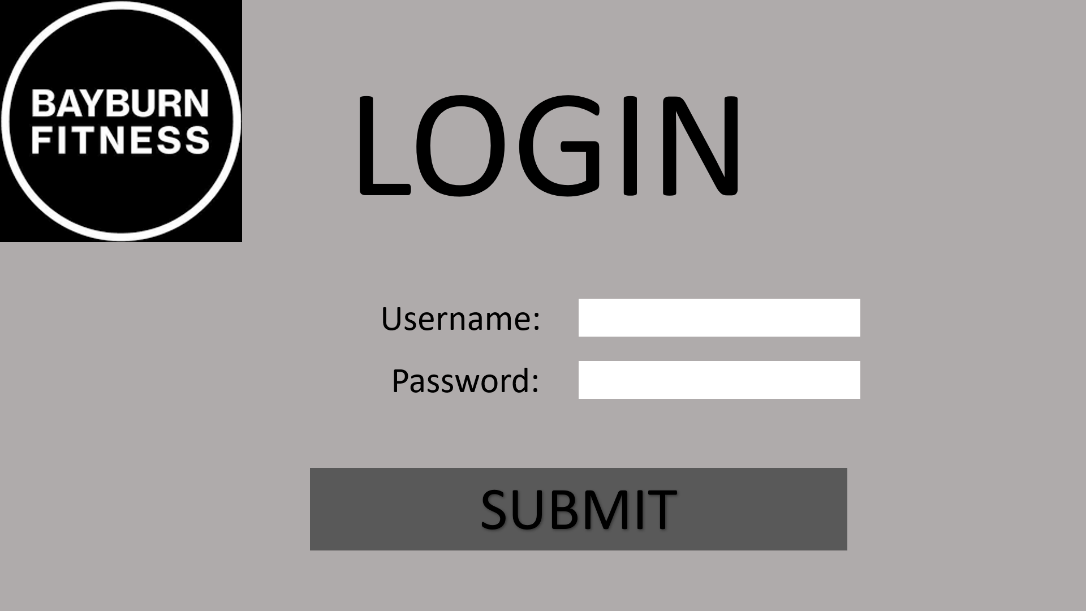
else:

ERROR

These functions display all the necessary code to complete my project.

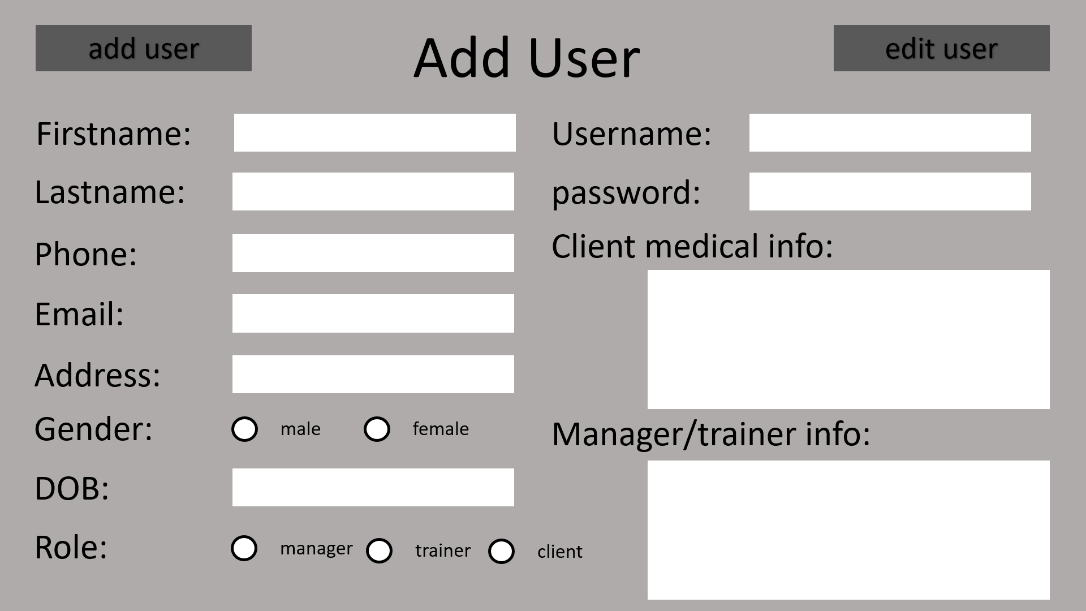
**User interfaces:**

**Login:** This user interface will allow a user to input their username and password and then submit them. If they match to an ID on the system, then the user will be logged on to the initial screen for the role assigned to that ID.



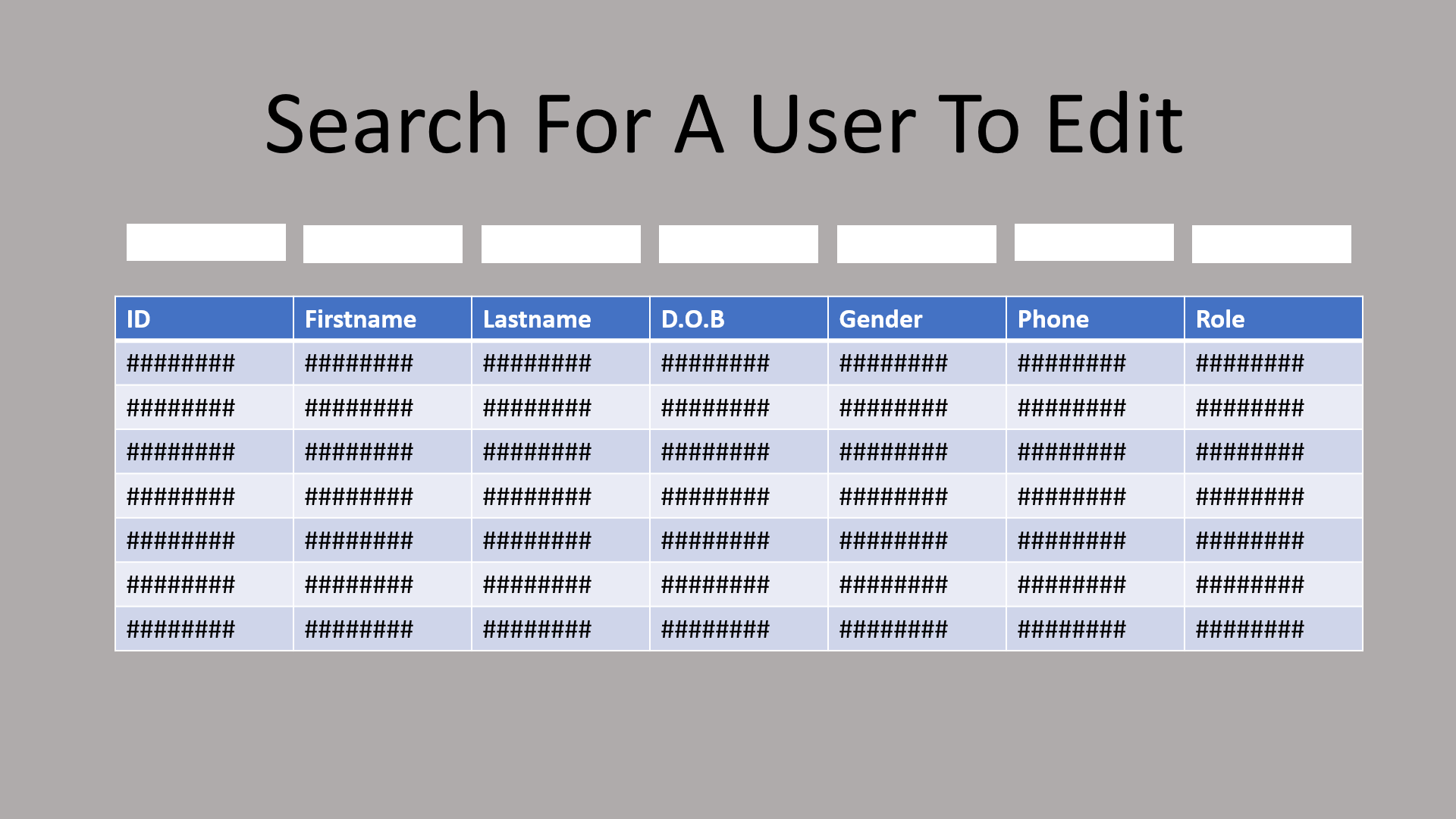
**Add user:**

This user interface is the first screen that a manager is greeted with when logging in. it allows the user to add edit or delete users from the system. The add user button submits all info in the fields to the system to be verified and processed whilst the edit user button brings the user to the search screen to find a user. Once a user is found they can be selected and their data will be filled into the form, from here they can be edited or deleted.



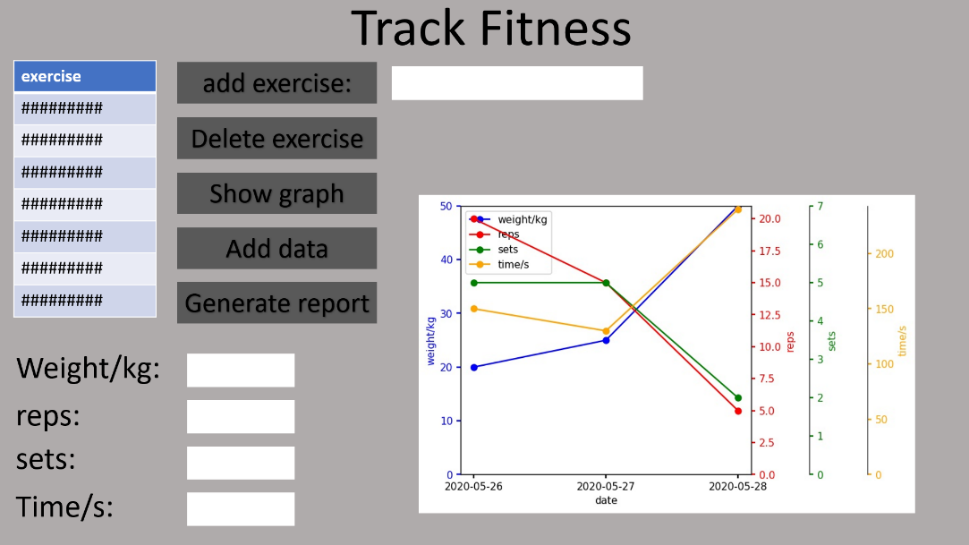
**search for user:**

this is the screen that you are directed to when you press the edit button on the add user screen. The 7 boxes are search boxes that automatically update the treeview as you type into them. They allow you to narrow down the search to find the user you want to edit. Once found you can select them and either edit or delete their information.



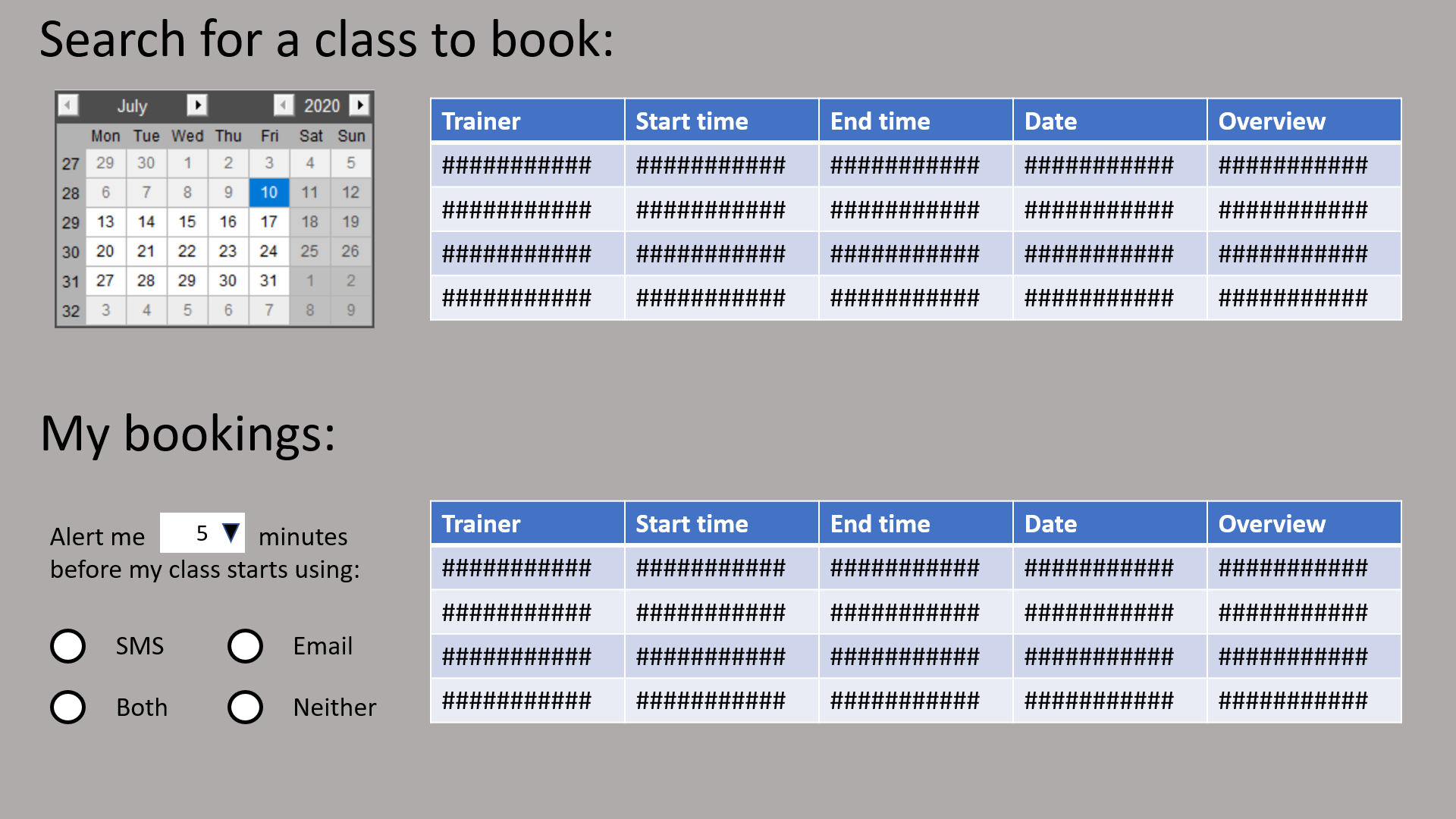
**Track fitness:**

This screen allows a client to track their fitness by adding and deleting exercises which can then have fitness data added to them. The fitness data consists of weight, reps, sets and time taken. When data is added it is added to the current date. When the client presses the show graph button a graph is generated showing the dates on the x axis and their fitness data on the y axis. Exercises can be added and deleted by the user. The client will also be able to generate a report based on their performance for a specific exercise in the form of a word document.

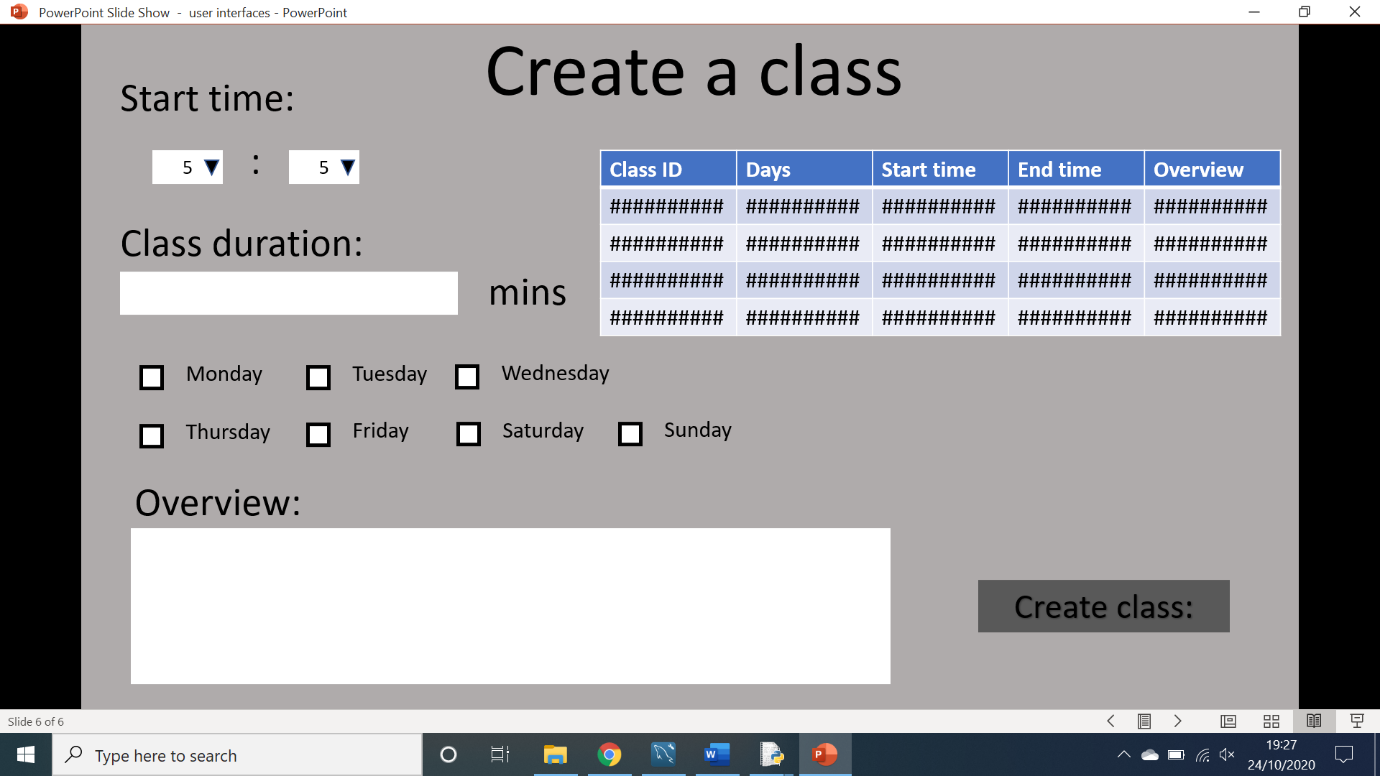


**Book classes:**

This screen allows the client to select a date on the calendar and when they do this, all classes that are happening on that date will be displayed on the adjacent treeview along with all relevant information. When the client selects one of these classes the system will book that class for the client and all of the client’s bookings will appear in the bottom treeview. The client will also be able to select how long before a class they are alerted and how they are to be alerted.



**Create classes:**

This screen allows a trainer to create a class for clients to book into. All of the classes that the trainer has made will be displayed on the treeview and whenever a new class is created, that treeview is updated. To delete a class the trainer will simply click on the class they wish to delete on the treeview, and they will be prompted on if they want to delete it or not. All fields are validated before entry to ensure the trainer does not input any incorrect data.

**View bookings:**

On this screen, the trainer can see all of their classes for the next 30 days and see which clients have booked into them. The top treeview is populated with all of the trainer’s classes that are happening in the next 30 days. In order to view who is attending the classes, the trainer must simply click on one of them and the clients who have booked into these classes will be populated in the bottom treeview. In order to record someone as present, the trainer must click on their name on the bottom treeview. This will prompt the user to confirm that they wish to record them as present and if so it will record them as present for this class, this can however only be done on the day of the class. If you try to record someone as present for a class that is not today, an error message will be generated.

