School Information System

A Level Computing Project

Danail Todorov

[Analysis 4](#_Toc9259941)

[Purpose 4](#_Toc9259942)

[Client 4](#_Toc9259943)

[Functionality and objectives 5](#_Toc9259944)

[Connectivity 6](#_Toc9259945)

[Limitations 6](#_Toc9259946)

[Existing solutions 7](#_Toc9259947)

[Current solution 7](#_Toc9259948)

[Similar systems 7](#_Toc9259949)

[Technical solution 8](#_Toc9259950)

[Storing Data 8](#_Toc9259951)

[User Interaction – Graphical user interface 8](#_Toc9259952)

[Design 10](#_Toc9259953)

[Platform 10](#_Toc9259954)

[Database 10](#_Toc9259955)

[Students 11](#_Toc9259956)

[Teachers 11](#_Toc9259957)

[Subjects 12](#_Toc9259958)

[Periods 12](#_Toc9259959)

[Groups 13](#_Toc9259960)

[Student\_Group 13](#_Toc9259961)

[Lessons 14](#_Toc9259962)

[Grades 14](#_Toc9259963)

[Attendance 15](#_Toc9259964)

[Normalisation 15](#_Toc9259965)

[Cross-table parameterised SQL 16](#_Toc9259966)

[Example queries 16](#_Toc9259967)

[Setting-up the database 17](#_Toc9259968)

[Consistency of data 17](#_Toc9259969)

[Frontend GUI 18](#_Toc9259970)

[Log In screen 18](#_Toc9259971)

[Change Password 18](#_Toc9259972)

[Absences screen 19](#_Toc9259973)

[Grades screen 20](#_Toc9259974)

[Example screen using GuiZero 20](#_Toc9259975)

[Libraries 22](#_Toc9259976)

[GuiZero 22](#_Toc9259977)

[SQLite 23](#_Toc9259978)

[hashlib 23](#_Toc9259979)

[Time 23](#_Toc9259980)

[CSV 23](#_Toc9259981)

[Ease of use 23](#_Toc9259982)

[Technical Solution 24](#_Toc9259983)

[SIS.db 24](#_Toc9259984)

[SIS.py 25](#_Toc9259985)

[Testing 38](#_Toc9259986)

[Setting up the database 38](#_Toc9259987)

[Testing different queries 39](#_Toc9259988)

[User login 41](#_Toc9259989)

[Changing the password 42](#_Toc9259990)

[Querying data and displaying it 44](#_Toc9259991)

[Time tabling 44](#_Toc9259992)

[Absences 45](#_Toc9259993)

[Grades 45](#_Toc9259994)

[Appending data to the database 46](#_Toc9259995)

[Filtering groups 46](#_Toc9259996)

[Appending 50](#_Toc9259997)

[Exporting reports 51](#_Toc9259998)

[Evaluation 53](#_Toc9259999)

[Objectives 53](#_Toc9260000)

[User feedback 53](#_Toc9260001)

[Possible future extensions 54](#_Toc9260002)

[References 55](#_Toc9260003)

# Analysis

Purpose  
  
The “School information system”, in short SIS, is a database structure including a user-friendly graphical interface to be used by staff and students to access and edit information. It securely stores personal data as well as data about students’ subjects, grades and attendance during the academic year. It will be used by teachers to add information and by students to view a log of their activity.

Client  
  
The system is created for a new private sixth form school –” Oxford Best College” as they are expanding and need a more sophisticated way to store and access data about their students. Until now, this was all done using spreadsheets, however, due to the growing number of students in the past few years and the proposed expansion in a purposely made new site, the school need a well-design database structure to be used by the administration and also allow teacher and students to check on their day-to-day activity. It is a bespoke system tailored to the needs of the college which combines all the necessary functionality needed and provides an easy-to-use interface.  
  
The college offer a variety of subjects from which the students choose. A student chooses at least one or more subjects to study. Each subject is thought 4 times a week. Students are put into different groups (Classes) for each subject so that they could choose between all subjects without any overlapping. A group could be taught by one or more teachers. There are 6 periods during the day.  
  
After discussing the problem with the principal and the head teacher they expressed their needs.

## Functionality and objectives

1. A database structure to securely store all the needed data during the academic year. It should be easy to set up initially in the beginning of the year and be flexible and allow administration to easily add data or manage changes during the year.
   1. Personal data – The database should be able to store personal information for both students and teachers such as their first name and surname, age, etc. More parameters should be easy to add in latter stages when needed without the need to set-up the whole database again.
   2. Timetables – The database needs to facilitate all the needed functionality connected with time-tabling including:  
      - storing a record of the subjects one chooses  
      - storing a record of the different group and the students in each one  
      - storing when different groups have lessons (Day of the week and period of the day)  
      - storing a record of the teacher teaching each lesson
   3. Flexibility – The database should be easy to modify if changes occur during the year. It should be structured so that it administration is able to change the subjects a student takes, the students in a group or the teacher teaching group during a particular lesson.
   4. Absences – The database should be able to store information about the absences of each student including the date it was recorded on and the teacher who recorded it.
   5. Grades – The database should be able to store information about the grade of each student including the grade, the date it was recorded on and the teacher who recorded it.
2. A user-friendly graphical interface intended to be used by the teachers and students making it easier for them to interact with the database.
   1. Log In – Every user of the system uses their first name, surname and a password to log into their account. A default password is given to everyone at the beginning of the year and users are required to change it the first time they log in. Afterwards users use their own password to gain access to the system.
   2. Users – The system should differentiate between teachers and students allowing only teachers to enter new information into the database. Students are only allowed to view information.
   3. Timetable – Every user is presented with a personalised timetable, displaying a grid of the week with subjects:  
      - Teachers are displayed the lesson they need to carry out during the week.  
      - Students are displayed the lesson they need to attend during the week including the teacher carrying out each lesson.  
      Data about each user is queried from the database.
   4. Absences and grades – Every user is presented with a personalised list, displaying their absences and grades:  
      - Teachers are displayed with a record of all the absences and grades they have recorded.   
      - Students are displayed with a record of all the absences and grades associated with them.
   5. Inserting into database – Teachers are presented with options to enter absences and grades only about their students. The data is then recorded into the database. Options are pre-determent (i.e. there is no data entered manually in order to eliminate mistakes).
   6. Reports – The system allows users to export information about absences and grades as CSV file so that they could be edited in an external application allowing for further customisation.
3. Privacy – Every user is given a default password (“12345678”) at the beginning of the year and the system requires them to change in the first time they log in. The new password needs to be at least 8 characters long, and include at least 1 digit, 1 lowercase and 1 uppercase characters. All passwords are hashed (data is mapped using an algorithm and it is store in an unreadable way) and stored in the database. This allows for greater privacy as nobody even administrators could look up an individual’s password. In the event one forgets its password, an administrator could set it back to the default value and the user would be required to change it upon log in.  
     
   Furthermore, user should be presented only with their data. No data of other students or teachers should be visible to them. This is achieved with using passwords and designing the system so that there is no access to one’s information without the correct credentials.

Connectivity

The database is supposed to be set-up by an administrator(s) and then installed on a server. This allows for all the users on the network access to the database and allows them to send request to the server and receive back data. This makes it possible for the school IT support to introduce a firewall on the server therefore, only whitelisted devices (i.e. school computers) could interact with the data. This increases security.

## Limitations

1. Timetabling - The system is not able to construct a timetable based on the data it stores. This is a really complex logical problem which is not computationally solvable. Therefore, the creation of the timetable at the beginning of the academic year need to be done by hand. The database is only a structure to facilitate the information and query based on it. Changes or modifications to the timetable made later are easy to implemented in the database.
2. Online connection — The system is client-server based therefore, a connection is essential for it to function. This could create problems as if the IT is down clients would not be able to request data to the database. Furthermore, information for the time the system was down needs to be entered manually afterward.
3. File corruption – There is only one copy of the database stored on a local server. If corruption or fragmentation occurs all the data could be lost due to the problem. The school needs to ensure that data is backed up regularly in order to minimise the impact of such events.

## Existing solutions

### Current solution

Until now the college has been using “Excel” spreadsheets to organise all the data. There are sheets about the students, teachers, the timetabling, attendance and grades. However, it been becoming harder to maintain a clean record of all the data as it is entered by hand and the increasing number of students has made the process more complex overtime. My proposed solution is based on the same needs as it implements mostly the same tables, however, using a database makes it simpler and more efficient when dealing with larger number of students.

### Similar systems

There are similar systems existing. They work in modules for the different functions and could be scaled to include more or less functions.

1. WCBS’s 3Sys  
     
   3Sys [1] began its live as an account package, which was developed into school administration modules and now have become an integrated system. It has most of the needed functionality, however, input of information and the interface are complex to use.

**Figure 1: WCBS's 3Sys Logo**

1. Capita’s SIMS   
     
   Capita’s SIMS [2] was founded on school admin needs in the UK state sector where timetabling and monitoring attendance/truancy were key. There are very few reports features as it was not the initial intend of the system.

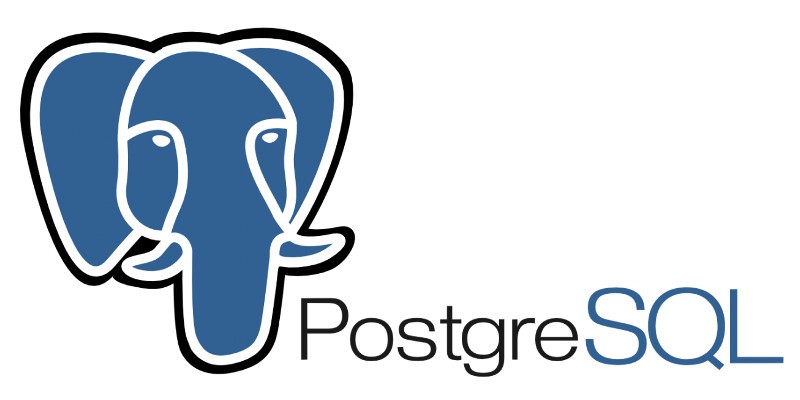
**Figure 2: Capita's SIMS Logo**

Both solutions as well as many of the rest are online based and require a constant connection to operate.” Oxford Best College” wants an on-site system as they have concerns over the reliability of their service provider. The college is looking for a bespoke system designed and tailored to their needs offering all the needed functionality in an easy-to-use package. Most other systems are bloated and hard to learn which could slow down staff in day-to-day usage. Having a system build for them from the ground up would allow for greater efficiency.

## Technical solution

### Storing Data

Databases are used to store and retrieved information. It makes it easier to append, analyse and present big chucks of data. Relational tables are commonly used to implement connections between data allowing for complex queries.

1. PostgreSQL  
     
   PostgreSQL [3] is the most popular solution used by many web-based applications. It is easily scalable and feature packed. Because of its wide use there are many tutorials and examples.

**Figure 3: PostgreSQL Logo**

1. SQLite   
     
   SQLite [4] is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is the most used database engine in the world. It is lighter than most other solutions, however, complex enough to be used to this project. As my library of choice, it is better documented in the design section.

**Figure 4: SQLite Logo**

User Interaction – Graphical user interface  
  
There are different libraries to provide a graphical functionality to a Python script. Most of the create a window which constantly loops itself (60 times per second) and displays widgets(objects) on top of it. Creating an GUI allows for better displaying of data and easier interactions between the user and the database as queries are done behind the scene.

1. Processing 3  
     
   Processing 3 [5] is a flexible software sketchbook and a language for learning how to code within the context of the visual arts. It is C-language based, however, Processing.py provides all the functionality using python. It offers interactive input and 2D or 3D output. The library is easy to learn and there is a handful of documentation and tutorial. However, the library does not include some of the needed widgets for this project such as a text boxes for input.

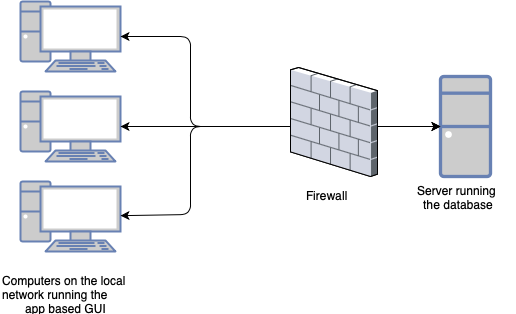
**Figure 5: Processing 3 Logo**

1. Tkinter and GuiZero  
     
   Tkinter [6] is the standard GUI library for Python. It provides a fast and easy way to create GUI applications. Tkinter provides various controls, commonly called widgets, such as buttons, labels and text boxes used for interacting with the user. For this project, GuiZero [7] would be used. It is a wrapper for the standard Tkinter library further simplifying the coding process while maintain all the functionality of Tkinter.

# Design

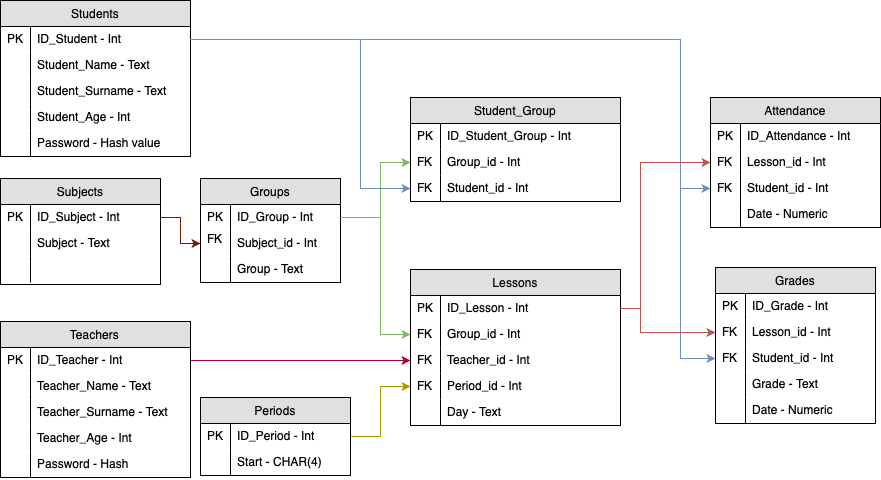
## Platform

The following design for the School Information System is based on a database structure intended to be installed on a local server and an app-based GUI to deal with request and queries which would be installed on the school computers connected to the network. This eliminates the need for a constant internet connection. The App would be written in Python so it could be installed and run on all major OS (Windows, MACOS, Linux).



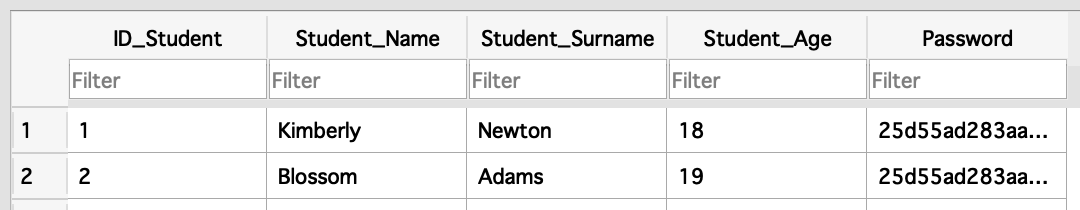
**Figure 6: Deploying the system**

Database

To store all the data including personal information and daily scheduling a form of relational database, most likely Sqlite3, Will be used.  
  
The database will follow the schema about to be described with the entity relationship diagram displayed below.  
  
  
  
This would provide the structure, i.e. tables, fields, relations and constrains. However, the database needs to set up by an administrator at the beginning of the academic year, entering the needed data. Using this structure makes it straight-forward what data needs to go where so that there are no mistakes made.

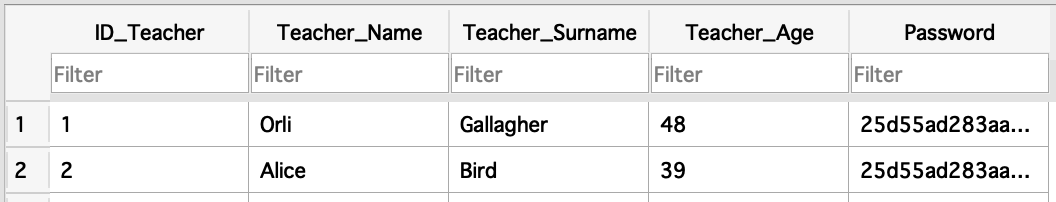
**Figure 7: Database relation diagram**

Students   
  
This table stored personal data about the students.  
This table could be altered later by the school if they choose to store more personal information on the students. More fields could be appended without the need to setup the database again.  
  
  
ID\_Student - Primary Key: A unique integer field is chosen to be the primary key rather than the name of the student. This allows for a unique value to be used as more than one student could share the same name in the school.  
  
Student\_Name: This field stores a variable length string of the first name of the student  
  
Student\_Surname: This field stores a variable length string of the surname of the student  
  
Student\_Age: This field stores an integer of the student’s age  
  
Password: This field stores a hashed value of the password the user. By default, the field stores the hashed value of (“12345678”). Users are made to change their password once they log in the system. The field is altered with the hashed value of the new password once changed. This allows for greater security as the hashed value could not be reversed to a string of the password making in nearly impossible for someone to read it.

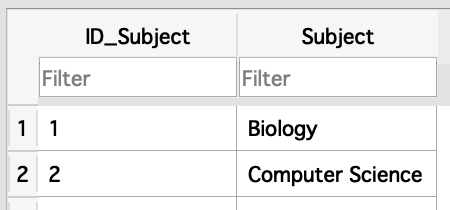


**Figure 8: Students table**

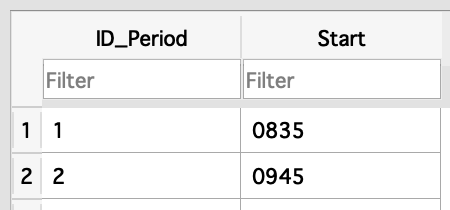
Teachers  
  
This table stores personal data about the teachers.  
This table could be altered later by the school if they choose to store more personal information on the teachers. More fields could be appended without the need to setup the database again.  
  
ID\_Teacher - Primary Key: A unique integer field is chosen to be the primary key rather than the name of the teacher. This allows for an unique value to be used as more than one teacher could share the same name in the school.  
  
Teacher\_Name: This field stores a variable length string of the first name of the teacher  
  
Teacher\_Surname: This field stores a variable length string of the surname of the teacher  
  
Teacher\_Age: This field stores an integer of the student’s age  
  
Password: This field stores a hashed value of the password the user. By default, the field stores the hashed value of (“12345678”). Users are made to change their password once they log in the system. The field is altered with the hashed value of the new password once changed. This allows for greater security as the hashed value could not be reversed to a string of the password making in nearly impossible for someone to read it.



**Figure 9: Teachers table**

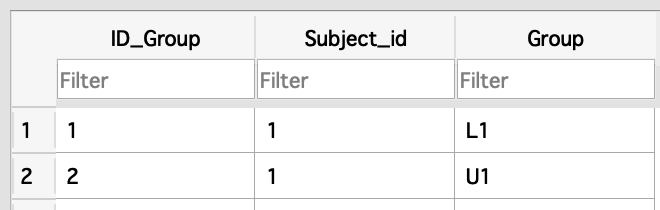
Subjects  
  
This table stores all the subjects offered by the school.  
  
ID\_Subject - Primary Key: A unique integer field is chosen to be the primary key rather than the name of the subject.  
  
Subject: This field stores a variable length string of the name of the subject

**Figure 10: Subjects table**

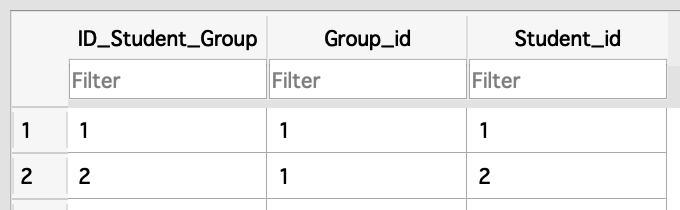
Periods  
  
This table stores the periods. The school have 6 periods during the day all of them an hour long.  
  
ID\_Period – Primary Key: A unique integer field is chosen to be the primary key  
  
Start – This field stores a fixed length string of 4 characters of the start time of every period in the format 0000-2359. (E.g. If a period starts at 08:35a.m. the value 0835 is stored as a string. If a period starts at 2:10p.m. the value 1410 is stored as a string.) Storing the values as a string instead of numerical values allows for easier displaying in the GUI as strings are more flexible to format.  


**Figure 11: Periods table**

Groups   
  
This table stores all the groups of students in the school. Students are put into different groups based on the subjects they study and the year the student is in. This allows for flexibility as students could be moved between groups if overlaps occur in their timetable due to the complex scheduling.  
  
ID\_Group – Primary Key: A unique integer field is chosen to be the primary key   
  
Subject\_id – Foreign Key: Reference the primary key of the Subjects table. One subject could be mapped to every group  
  
Group – This field stores a variable length string of the name of the group. It is up to the school to name the groups as they see appropriate.

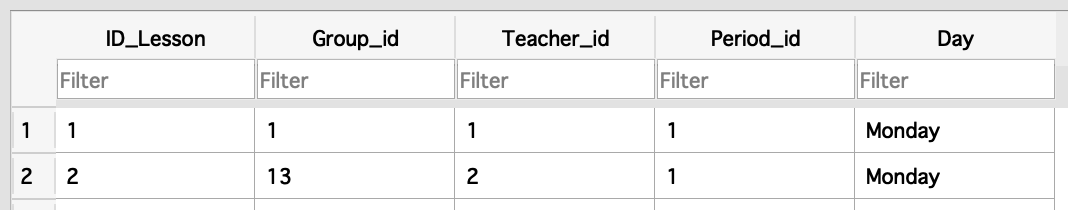
Student\_Group  
  
This is a composite table that stores record of which student is in which groups. Every record consists of a student and the group he is in. This is a many-to-many relation as one student could be in more than one group and one group consists of many students.  
  
ID\_Student\_Group – Primary Key: A unique integer field is chosen to be the primary key  
  
Group\_id – Foreign Key: Reference the primary key of the Groups table.  
  
Student\_id – Foreign Key: Reference the primary key of the Students table.

**Figure 12: Groups table**

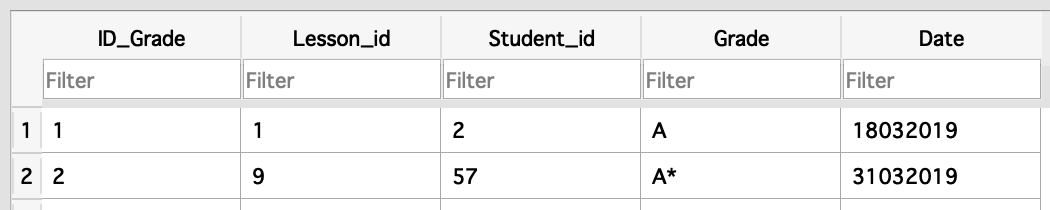


**Figure 13: Student\_Group table**

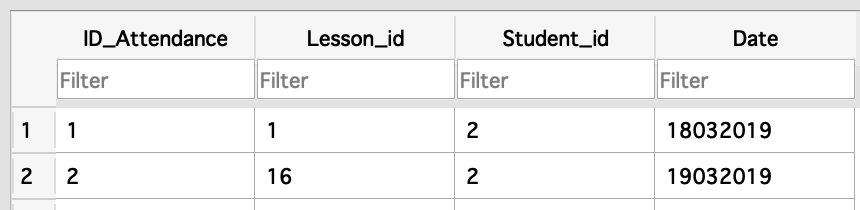
Lessons  
  
This table stores records of all the lessons during the week. It combines a group (of students), the teacher carrying out the lesson and when the lesson takes place. This is table is the base for the personalised timetabling feature of the GUI as it joins most other tables together.   
  
ID\_Lesson – Primary Key: A unique integer field is chosen to be the primary key  
  
Group\_id – Foreign Key: Reference the primary key of the Groups table  
  
Teacher\_id – Foreign Key: Reference the primary key of the Teachers table  
  
Period\_id – Foreign Key: Reference the primary key of the Periods table   
  
Day – This field stores a variable length string of the day of the week. (Acceptable values are: “Monday”, “Tuesday”, “Wednesday”, “Thursday”, “Friday”)



**Figure 13: Lessons table**

Grades  
  
This table stores records of all the grades given to students during the academic year.  
  
ID\_Grade – Primary Key: A unique integer field is chosen to be the primary key  
  
Lesson\_id – Foreign Key: Reference the primary key of the Lessons table  
  
Student\_id – Foreign Key: Reference the primary key of the Students   
  
Grade: This field stores a variable length string of the grade. (Acceptable values are: “A\*”, “A”, “B”, “C”, “D”, “E”, “U”)  
  
Date: This field stores a string value of the date the record has been inserted into the table in the format DDMMYYYY (E.g. If the date is 01.02.2003 the value 01022003 is stored as a string value). Storing the values as a string instead of numerical values allows for easier displaying in the GUI as strings are more flexible to format.  
  


**Figure 14: Grades table**

Attendance   
  
This table stores a record of all the absences of students during the academic year  
  
ID\_Attendance – Primary Key: A unique integer field is chosen to be the primary key  
  
Lesson\_id – Foreign Key: Referencing the primary key in the Lessons table  
  
Student\_id – Foreign Key: Referencing the primary key in the Students table  
  
Date: This field stores a string value of the date the record has been inserted into the table in the format DDMMYYYY (E.g. If the date is 01.02.2003 the value 01022003 is stored as a string value). Storing the values as a string instead of numerical values allows for easier displaying in the GUI as strings are more flexible to format.

**Figure 15: Attendance table**

### Normalisation

A degree of normalisation is used when designing the database. There are no repeating attributes or groups of attributes and there is no dependence between attributes. This allows for easy maintenance and modification of the database. By using normalisation and structuring the database in an efficient way this allows for faster queries as there is no overlap or unnecessary data repetition.   
  
This allows for adding more categories of personal data, easily changing students in different groups, changing the subjects one takes, modifying the start time of the different periods, etc. without much complication. Those are all essential changes which naturally occur during an academic year and therefore, the database is intentionally designed to make such modification easy and eliminate the need for re-entering all the data when a change occurs.

### Cross-table parameterised SQL

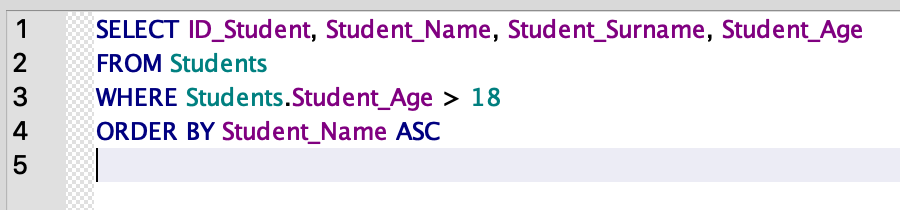
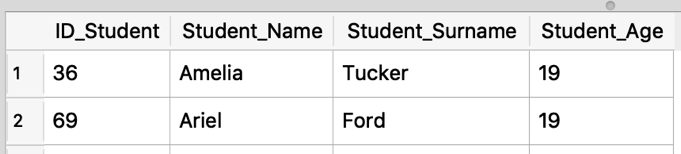
Tables in the database are linked together based on a common attribute so that data is connected. This allows for a query to access more than just one table but multiple ones and produce more sophisticated reports. By using cross-table parameterised SQL, a user is able to retrieve the exact data he is looking for based on a parameter he inputs. This makes it very convenient to find what one is long for.

Relations in this database include:

1. Subjects – Groups (one-to-many)
2. Groups – Student\_Group (many-to-many)
3. Students – Student\_Group (many-to-many)
4. Groups – Lessons (one-to-many)
5. Teachers – Lessons (one-to-many)
6. Periods – lessons (one-to-many)
7. Lessons-Attendance (one-to-many)
8. Students – Attendance (one-to-many)
9. Lessons – Attendance (one-to-many)
10. Students – Grades (one-to-many)
11. Lessons – Grades (one-to-many)

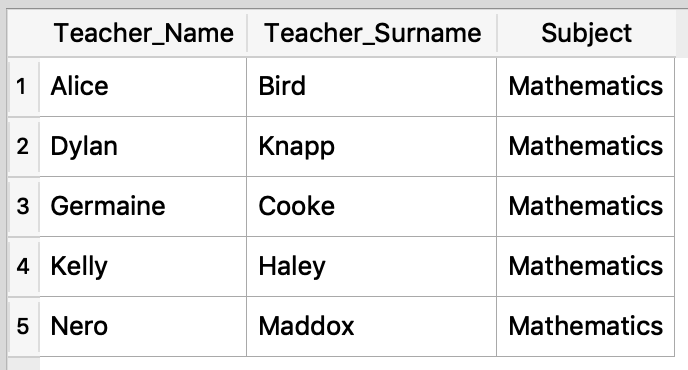
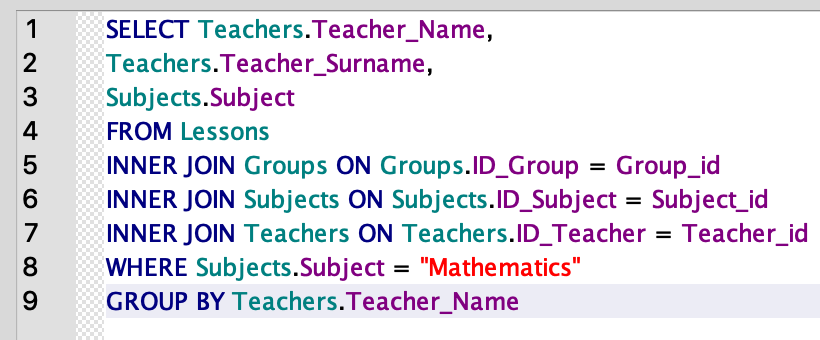
### Example queries

Example 1: Selecting students over 18 and ordering the records alphabetically by first name.

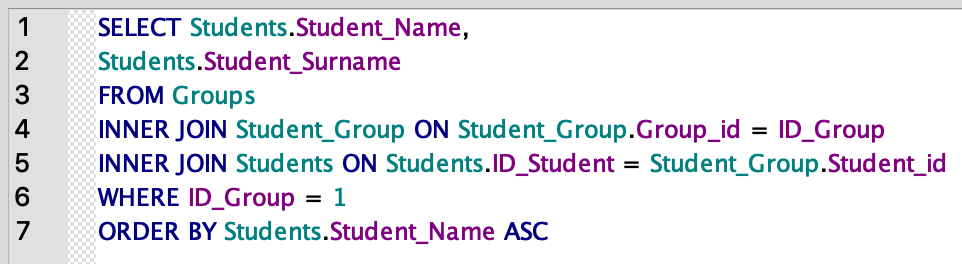
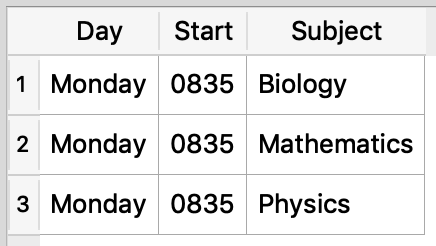


**Figure 16: Example query 1**

Example 2: Selecting all teachers carrying out mathematics lessons.  
  
   
  
  
Example 3: Selecting all lessons on Monday, Period 1(Starting at 08:35 a.m.).

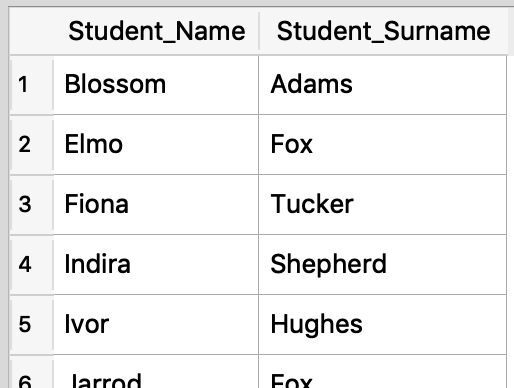
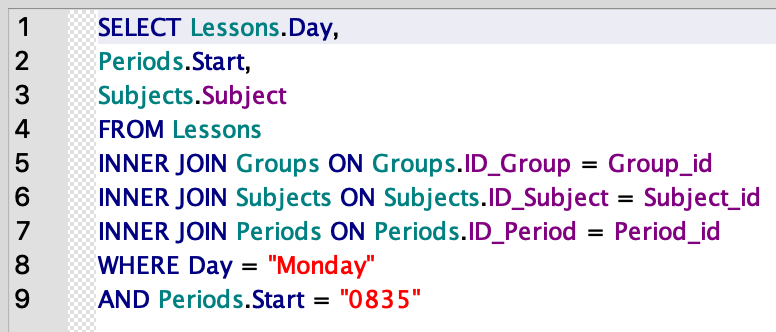


**Figure 17: Example query 2**



**Figure 18: Example query 3**

Example 4: Selecting all students in group “1”. (One of the biology groups).



**Figure 19: Example query 4**

### Setting-up the database

Inserting data at the beginning of the year could be done through a command terminal using SQL language. However, there are graphical solutions administrators could use, making it easier to do so. The one I would use when testing the system is “DB Browser for SQLite”.

### Consistency of data

To keep in the insertion of data consistent and prevent mistakes the user is presented either with checklists or dropdown menus to choose values from. This eliminates the need for validating if the inserted data is compatible as the values are predetermined. Furthermore, it makes it easier for teachers to insert information as everything is formatted in the backend.

## Frontend GUI

To allow users (Teachers, Students) for an easy interaction with the database a GUI would be written using Python 3.7 with the extension of the GuiZero library base on Tkinter. The GUI would consist of different view including a log in screen and screens displaying a timetable, absences and grades.

### Log In screen

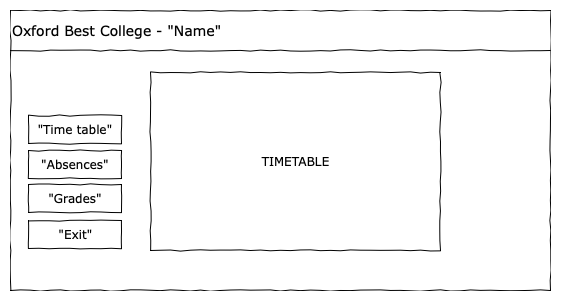
**Figure 20: GUI - Log-in diagram**

This view contains a “Username” and “Password” fields and a “Log in” and an “Exit” buttons. An error is displayed if there is no match for the user i.e. Wrong credentials.

### Change Password

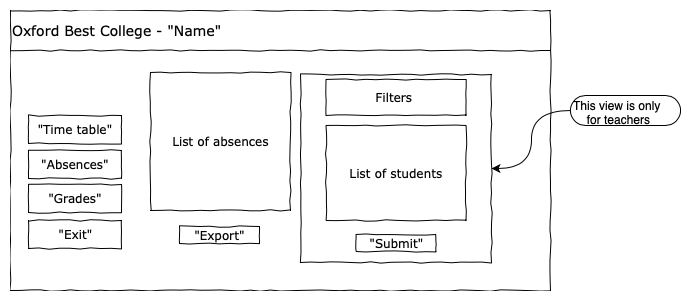
**Figure 21: GUI - Change pass diagram**

This view contains a “New Password” and “Confirm Passwords” fields and a “Submit” button. An error is displayed with the criteria for the new password if they are not met.   
  
Main screen   
  
This view is the main one the user is presented with when they log into the system with the correct credentials. It contains a menu bar to the left with options (“Timetable”, “Absences”, “Grades”, “Exit”) and a personalised timetable showing the lessons during the week.



**Figure 22: GUI - Main screen diagram**

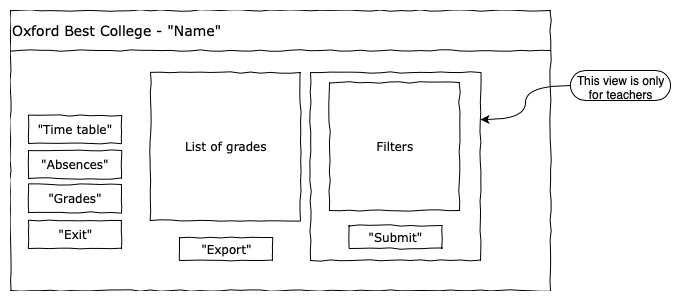
Absences screen  
  
This view contains the menu bar to the left and a list of the absences associated with the user. There is a button to export the absences data displayed as csv file. If the user is a teacher, they get a different view with options to choose between groups and add absences to the database.



**Figure 12: GUI – Absences screen diagram**

Grades screen  
  
This view contains the menu bar to the left and a list of the grades associated with the user. There is a button to export the grades data displayed as a csv file. If the user is a teacher, they get a different view with options to choose between groups and add grades to the database.

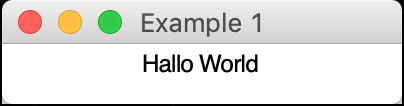
**Figure 13: GUI – Grades screen diagram**



Example screen using GuiZero  
  
Example 1: “Hallo world”

Code:



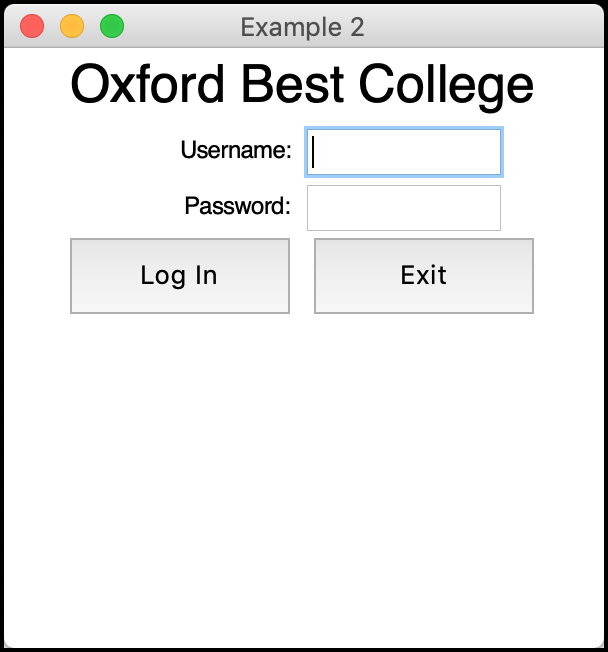


**Figure 14: Example GUI 1**

Example 2: “Log In”

Code:



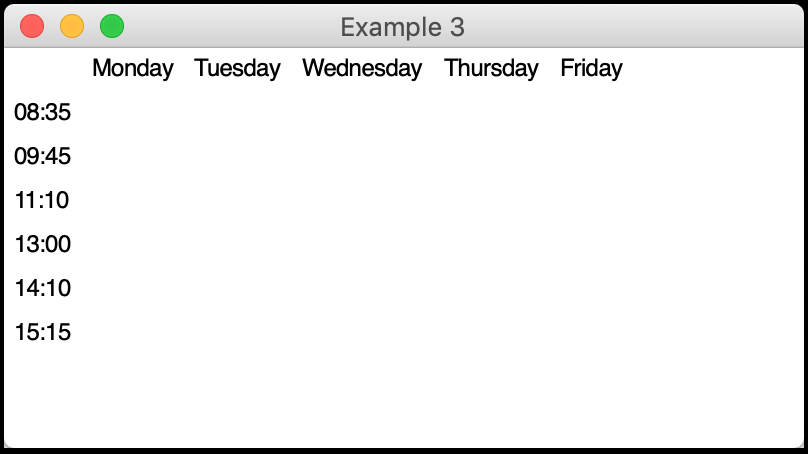


**Figure 15: Example GUI 2**

Example 3: Timetable

Code:





**Figure 16: Example GUI 3**

## Libraries

### GuiZero

This is a Python3 library for creating GUIs which is a wrapper for the standard Tkinter library (included by default when Python is installed).

The aim is to simplify the process of designing an interface making it accessible for new developer. The library is based on different widgets (text boxes, buttons, sliders, etc.) Yet it is flexible enough to be used for projects up to A-Level standard and provides the needed functionality foe this project. Furthermore, there is comprehensive and accessible documentation with examples online   
  
Limitations: Buttons are used to call functions to execute different task. However, there is way to pass parameter through them as this creates an infinite loop. This is a limitation of the library and the only work around is the use of global variables. This allows functions to use variables from the outside as there is no other way to pass them into the function.

SQLite  
  
This is a Python3 library with an embedded SQL database engine. This allows it communicate with databases giving developers access through a Python script. This is the base for interacting between the database and the GUI. It is a light and compact solution yet meets all the requirements for this project.

hashlib  
  
This is a build-in Python3 library for encrypting data using a variety of hashing algorithm. It allows for securely storing passwords in the database. It eliminates the need for a proprietary hashing algorithm to be developed especially for this project.

This algorithm maps the password to a unique value (a hash) and the process in one-directional – i.e. one could not get back to the original password using the hash value. The hash is not readable (e.g. “25d55ad283aa400af464c76d713c07ad”) therefore, even if an unauthorised person gain access to the database they would not be able to retrieve any passwords. This makes the system secure and keeps the privacy of the users.

Time  
  
This is a build-in Python3 library for dealing with dates and times. It would be used to get the current date of the local computer used to be used when storing absences and grade.   
  
Limitations: Even though, this library makes it easier for teachers to append data to the database as they are not required to enter the current date, if the computer clock is not set correctly, a wrong value would be stored. This should not be a big problem as most modern computers clocks are automatically synchronised.

CSV  
  
This is a build-in Python3 library for reading and writing CSV files which are the preferred format for dealing with databases and spreadsheets. This library is used to export the report of the grades and absences so that they could be printed. It provides a file with all the data which could easily be read by a different program and customised further to the needs of the college.

## Ease of use

The system is designed to be intuitive. The interface is simplified with only the relevant buttons and options presented at a time. The user is presented only with their information so that privacy is kept.

Teachers are presented with dropdown menus or checklists which eliminates the need for validating if the inserted data is compatible as the values are predetermined. This should eliminate any errors due to wrong data entered into the database. Designing the system like this would greatly simplify the testing stage as there would be no need for testing boundary or erroneous values.

# Technical Solution

## SIS.db

Code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100 | BEGIN TRANSACTION;  CREATE TABLE IF NOT EXISTS `Teachers` (  `ID\_Teacher` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  `Teacher\_Name` TEXT NOT NULL,  `Teacher\_Surname` TEXT NOT NULL,  `Teacher\_Age` INTEGER NOT NULL,  `Password` TEXT NOT NULL  );  INSERT INTO `Teachers` VALUES (1,'Orli','Gallagher',48,'25d55ad283aa400af464c76d713c07ad');  INSERT INTO `Teachers` VALUES (2,'Alice','Bird',39,'25d55ad283aa400af464c76d713c07ad');  ...  CREATE TABLE IF NOT EXISTS `Subjects` (  `ID\_Subject` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  `Subject` TEXT NOT NULL  );  INSERT INTO `Subjects` VALUES (1,'Biology');  INSERT INTO `Subjects` VALUES (2,'Computer Science');  ...  CREATE TABLE IF NOT EXISTS `Students` (  `ID\_Student` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  `Student\_Name` TEXT NOT NULL,  `Student\_Surname` TEXT NOT NULL,  `Student\_Age` INTEGER NOT NULL,  `Password` TEXT NOT NULL DEFAULT "12345678"  );  INSERT INTO `Students` VALUES (1,'Kimberly','Newton',18,'25d55ad283aa400af464c76d713c07ad');  INSERT INTO `Students` VALUES (2,'Blossom','Adams',19,'b6f74a2411057b37778854d1fdadc642');  ...  CREATE TABLE IF NOT EXISTS `Student\_Group` (  `ID\_Student\_Group` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  `Group\_id` INTEGER NOT NULL,  `Student\_id` INTEGER NOT NULL,  FOREIGN KEY(`Student\_id`) REFERENCES `Students`(`ID\_Student`),  FOREIGN KEY(`Group\_id`) REFERENCES `Groups`(`ID\_Group`)  );  INSERT INTO `Student\_Group` VALUES (1,1,1);  INSERT INTO `Student\_Group` VALUES (2,1,2);  ...  CREATE TABLE IF NOT EXISTS `Periods` (  `ID\_Period` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  `Start` CHAR ( 4 ) NOT NULL  );  INSERT INTO `Periods` VALUES (1,'0835');  INSERT INTO `Periods` VALUES (2,'0945');  ...  CREATE TABLE IF NOT EXISTS `Lessons` (  `ID\_Lesson` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  `Group\_id` INTEGER NOT NULL,  `Teacher\_id` INTEGER NOT NULL,  `Period\_id` INTEGER NOT NULL,  `Day` TEXT NOT NULL CHECK(Day IN ( "Monday" , "Tuesday" , "Wednesday" , "Thursday" , "Friday" )),  FOREIGN KEY(`Group\_id`) REFERENCES `Groups`(`ID\_Group`),  FOREIGN KEY(`Teacher\_id`) REFERENCES `Teachers`(`ID\_Teacher`),  FOREIGN KEY(`Period\_id`) REFERENCES `Periods`(`ID\_Period`)  );  INSERT INTO `Lessons` VALUES (1,1,1,1,'Monday');  INSERT INTO `Lessons` VALUES (2,13,2,1,'Monday');  ...  CREATE TABLE IF NOT EXISTS `Groups` (  `ID\_Group` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  `Subject\_id` INTEGER NOT NULL,  `Group` TEXT NOT NULL,  FOREIGN KEY(`Subject\_id`) REFERENCES `Subjects`(`ID\_Subject`)  );  INSERT INTO `Groups` VALUES (1,1,'L1');  INSERT INTO `Groups` VALUES (2,1,'U1');  ...  CREATE TABLE IF NOT EXISTS `Grades` (  `ID\_Grade` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  `Lesson\_id` INTEGER NOT NULL,  `Student\_id` INTEGER NOT NULL,  `Grade` TEXT NOT NULL CHECK(Grade IN ( "A\*" , "A" , "B" , "C" , "D" , "E" , "U" )),  `Date` TEXT NOT NULL,  FOREIGN KEY(`Lesson\_id`) REFERENCES `Lessons`(`ID\_Lesson`),  FOREIGN KEY(`Student\_id`) REFERENCES `Students`(`ID\_Student`)  );  INSERT INTO `Grades` VALUES (1,1,2,'A',18032019);  CREATE TABLE IF NOT EXISTS `Attendance` (  `ID\_Attendance` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  `Lesson\_id` INTEGER NOT NULL,  `Student\_id` INTEGER NOT NULL,  `Reason` TEXT NOT NULL,  `Date` TEXT NOT NULL,  FOREIGN KEY(`Student\_id`) REFERENCES `Students`(`ID\_Student`),  FOREIGN KEY(`Lesson\_id`) REFERENCES `Lessons`(`ID\_Lesson`)  );  INSERT INTO `Attendance` VALUES (1,1,2,'Sick',18032019);  INSERT INTO `Attendance` VALUES (2,16,2,'Sick',19032019);  COMMIT; |

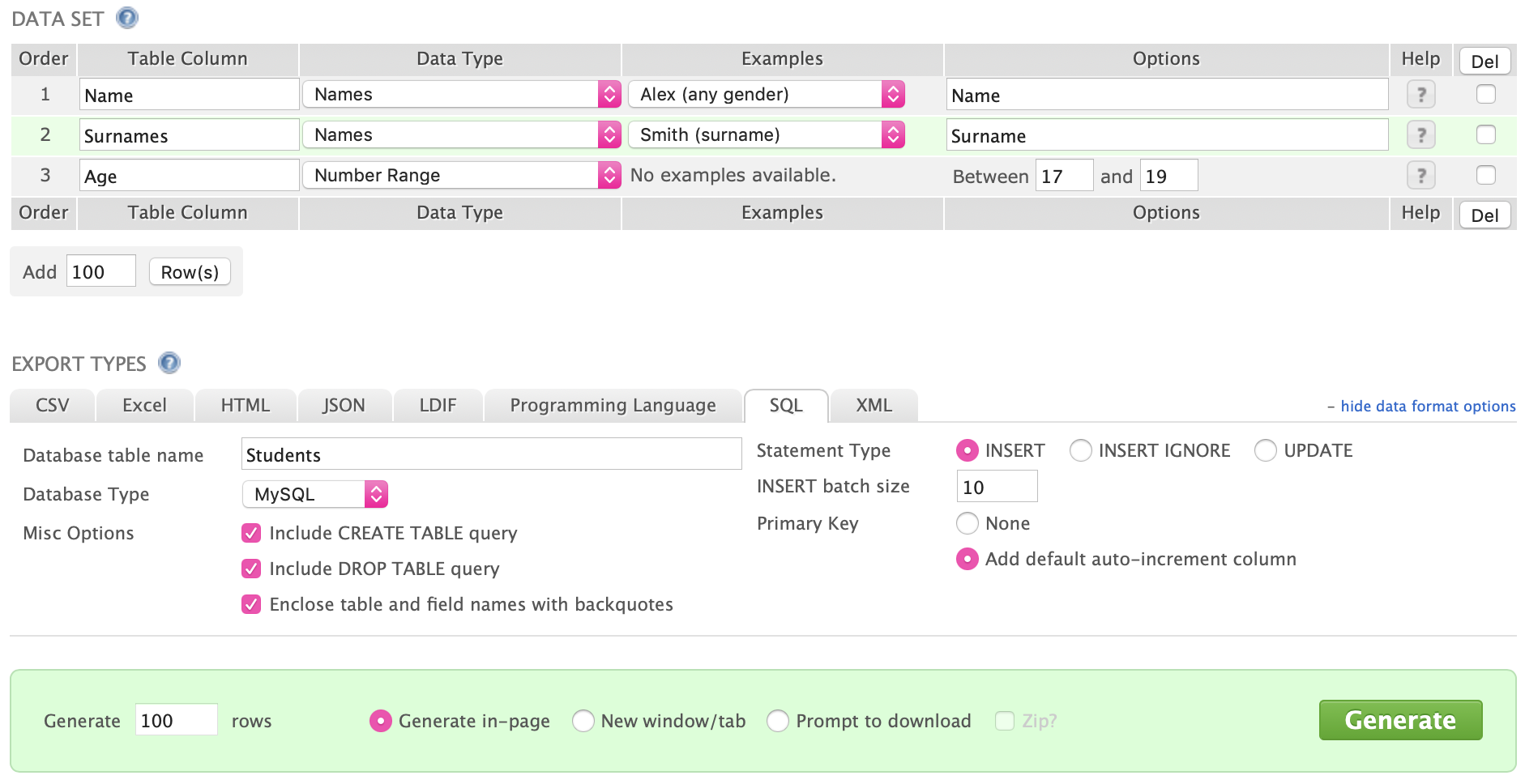
## SIS.py

Code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495  496  497  498  499  500  501  502  503  504  505  506  507  508  509  510  511  512  513  514  515  516  517  518  519 | from guizero import \*  import sqlite3  import hashlib  import csv  import time  # ----- Importing the database -----  database = sqlite3.connect('SIS.db', timeout = 10)  cursor = database.cursor()  user\_type = 0 #0-Default, 1-Student, 2-Teacher  user\_id = 0  user\_name = ""  def screen\_timetable():  s\_timetable.show()  s\_absences.hide()  s\_grades.hide()  def timetable\_fill(user\_type, user\_id):  cursor.execute ('''SELECT start  From Periods''')  Periods = cursor.fetchall()  Periods = [x[0] for x in Periods]  Days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]  Grid = []  if user\_type == 1:  query = '''SELECT Subjects.Subject,  Teachers.Teacher\_Name,  Teachers.Teacher\_Surname  FROM Lessons  INNER JOIN Groups ON Groups.ID\_Group=Lessons.Group\_id  INNER JOIN Student\_Group ON Student\_Group.Group\_id=Groups.ID\_Group  INNER JOIN Students ON Students.ID\_Student = Student\_Group.Student\_id  INNER JOIN Teachers ON Teachers.ID\_Teacher=Lessons.Group\_id  INNER JOIN Periods ON Periods.ID\_Period=Lessons.Period\_id  INNER JOIN Subjects ON Subjects.ID\_Subject=Groups.Subject\_id  Where Students.ID\_Student= ?  AND Lessons.Day= ?  AND Periods.Start= ?  ;'''  elif user\_type == 2:  query = '''SELECT Subjects.Subject  FROM Lessons  INNER join Groups on Groups.ID\_Group = Lessons.Group\_id  INNER join Subjects on Subjects.ID\_Subject = Groups.Subject\_id  INNER join Periods on Periods.ID\_Period = Lessons.Period\_id  WHERE Teacher\_id = ?  AND Lessons.Day= ?  AND Periods.Start= ?'''  for Day in range (5):  for Period in range (6):  cursor.execute(query,(user\_id, Days[Day], Periods[Period]))  data = cursor.fetchone()  if data == None:  Grid.append("")  elif len(data)>1:  Grid.append(data[0]+"\n"+data[1]+" "+data[2])  else:  Grid.append(data)  return Grid  def timetable\_build(user\_type, user\_id):  Grid = []  Days = ["Monday","Tuesday","Wednesday","Thursday","Friday"]  for Day in range (5):  Grid.append( Text(s\_timetable, text = Days[Day], grid = [Day+1,0]) )  Periods = ["08:35","09:45","11:10","13:00","14:10","15:15"]  for Period in range (6):  Grid.append( Text(s\_timetable, text = Periods[Period], grid = [0,Period+1]) )  for Day in range (5):  for Period in range (6):  Grid.append( Text(s\_timetable, text = "", grid = [Day+1,Period+1]) )  lessons = timetable\_fill(user\_type, user\_id)  for lesson in range(30):  Grid[lesson+11].value = lessons[lesson]  def validate\_pass():  password = input\_confirm\_pass.value  if len(password) >= 8:  if sum([int(x.islower()) for x in password]) >= 1:  if sum([int(x.isupper()) for x in password]) >= 1:  if sum([int(x.islower()) for x in password]) >= 1:  return True  else:  info(title = "", text = '''Your new password is not valid !  Password need to be:  - 8 characters long  - Have at least 1 digit  - Have at least 1 uppercase AND 1 lowercase character''')  return False  def screen\_change\_pass():  s\_login.destroy()  s\_change\_pass.show()  def change\_pass():  global user\_type, user\_id, user\_name  query = ''' UPDATE Students  SET Password = ?  WHERE ID\_Student = ?'''  query1 = '''UPDATE Teachers  SET Password = ?  WHERE ID\_Teacher = ?'''  if input\_new\_pass.value == input\_confirm\_pass.value:  if validate\_pass() == True:  password = hashlib.md5(input\_confirm\_pass.value.encode())  if user\_type == 1:  cursor.execute(query, (password.hexdigest(), user\_id))  elif user\_type == 2:  cursor.execute(query1, (password.hexdigest(), user\_id))  database.commit()  user\_name = input\_user.value  s\_change\_pass.destroy()  screen\_change\_main()  else:  info(title = "", text = "Passwords not the same!")  def logIn():  global user\_type, user\_id, user\_name  pass\_need\_change = False  password = hashlib.md5(input\_pass.value.encode())  if len(input\_user.value.split()) == 2:  user = input\_user.value.split()  else:  user = ["", ""]  cursor.execute('''SELECT ID\_Student,  Password  FROM Students  WHERE Student\_Name=?  AND Student\_Surname=?''', (user[0],user[1]))  data = cursor.fetchone()  if data != None:  if password.hexdigest() == data[1]:  user\_id = data[0]  user\_type = 1  user\_name = input\_user.value  if input\_pass.value == "12345678":  pass\_need\_change = True  if user\_type == 0:  cursor.execute('''SELECT ID\_Teacher,  Password  FROM Teachers  WHERE Teacher\_Name=?  AND Teacher\_Surname=?''', (user[0],user[1]))  data = cursor.fetchone()  if data != None:  if password.hexdigest() == data[1]:  user\_id = data[0]  user\_type = 2  user\_name = input\_user.value  if input\_pass.value == "12345678":  pass\_need\_change = True  if pass\_need\_change:  screen\_change\_pass()  if user\_type == 0:  info(title = "", text = "Wrong Username or Password !")  elif user\_type == 1 or user\_type == 2:  s\_login.destroy()  screen\_change\_main()  def screen\_change\_main():  s\_menu.show()  timetable\_build(user\_type, user\_id)  s\_timetable.show()  s\_logo = Box(SYSTEM, height = "fill", grid = [0,0,10,1])  l\_logo = Text(s\_logo, text = "Oxford Best College - "+ user\_name, size=20)  def exit():  database.close()  SYSTEM.destroy()  def screen\_absences():  s\_timetable.hide()  s\_absences.show()  s\_grades.hide()  display\_absences()  def display\_absences():  global user\_type, user\_name, s\_absences\_list, text\_absences, text\_total  s\_absences\_list.destroy()  s\_absences\_list = Box(s\_absences, height = 300, width = 350, grid = [0,0,1,4], align = "left")  if user\_type == 1:  query = '''select Date,  Subjects.Subject  from Attendance  inner join Lessons on Lessons.ID\_Lesson = Lesson\_id  inner join Groups on Groups.ID\_Group = Group\_id  inner Join Subjects on Subjects.ID\_Subject = Subject\_id  Where Attendance.Student\_id = ?  '''  elif user\_type == 2:  filter\_days\_absences()  query = '''select Date,  Students.Student\_Name,  Students.Student\_Surname  from Attendance  inner join Students on Students.ID\_student = Student\_id  inner join Lessons on Lessons.ID\_Lesson = Lesson\_id  Where Lessons.Teacher\_id = ?  '''  cursor.execute(query, str(user\_id))  data = cursor.fetchall()  text\_absences = ""  for absence in data:  absence = list(absence)  absence[0] = str(absence[0])  absence[0] = absence[0][:2]+"."+absence[0][2:4]+"."+absence[0][4:]  if len(absence) == 2:  for item in absence:  text\_absences += str(item) + " - "  text\_absences = text\_absences[:len(text\_absences)-4]  else:  text\_absences += absence[0] + " - " + absence[1] + " " + absence[2]  text\_absences += "\n"  text\_absences = text\_absences[:len(text\_absences)-1]  l\_absences = TextBox(s\_absences\_list, text = text\_absences, grid = [0,0], align = "top", width = 100, height = 12, scrollbar = True, multiline = True)  l\_absences.disable()  text\_total = ["Total absences:", len(data)]  l\_absences\_total = Text(s\_absences\_list, text = text\_total[0]+" "+str(text\_total[1]), grid =[0,1])  b\_absences\_reports = PushButton(s\_absences\_list, absance\_report, text = "Export report", grid = [0,2])  def filter\_days\_absences():  global user\_id, c\_abs\_days, s\_absences\_filters  s\_absences\_filters.destroy()  s\_absences\_filters = Box(s\_absences, grid = [1,0,1,1], align = "top")  query = ''' Select Day  from Lessons  where Teacher\_id = ?  '''  cursor.execute(query, str(user\_id))  list\_days = [x for x in cursor.fetchall()]  list\_days = set([str(x[0]) for x in list\_days])  c\_abs\_days = Combo(s\_absences\_filters, options=list\_days, command=filter\_periods\_absences, grid=[0,0], width=12, align="top")  def filter\_periods\_absences(selected\_value):  global user\_id, c\_abs\_periods  query = ''' select Start  from Lessons  inner join Periods on Periods.ID\_Period = Lessons.Period\_id  Where Teacher\_id = ?  and Day = ?'''  cursor.execute(query,(str(user\_id),selected\_value))  list\_periods = [str(x[0]) for x in cursor.fetchall()]  list\_periods = sorted(set([x[:2]+":"+x[2:4] for x in list\_periods]))  c\_abs\_periods = Combo(s\_absences\_filters, options=list\_periods, command=list\_students\_append, grid=[0,1], width=12, align="left")  def list\_students\_append(selected\_value):  global user\_id, list\_students, s\_absences\_append  s\_absences\_append.destroy()  s\_absences\_append = Box(s\_absences, layout = "grid", align = "left", grid = [1,1,1,3])  day = c\_abs\_days.value  periods = c\_abs\_periods.value  query = '''select Students.Student\_Name,  Students.Student\_Surname,  Students.ID\_Student  from Lessons  inner join Student\_Group on Student\_Group.Group\_id = Lessons.Group\_id  inner join Students on Students.ID\_Student = Student\_Group.Student\_id  inner join Periods on Periods.ID\_Period = Lessons.Period\_id  where Day = ?  and Periods.Start = ?  and Teacher\_id = ?'''  cursor.execute(query, (str(day),str(periods[:2]+periods[3:]),str(user\_id) ))  data = cursor.fetchall()  list\_students = []  for student in range(len(data)):  list\_students.append( CheckBox(s\_absences\_append, command = get\_absent, text = str(str(data[student][2])+" - "+data[student][0]+" "+data[student][1]), grid = [0,student], align = "left"))  button\_submit\_absences = PushButton(s\_absences\_append, add\_absences, text = "Submit absences", grid = [0,(student+1)])  def get\_absent():  global list\_students, list\_absences  list\_absences=[]  for student in list\_students:  if student.value == 1:  list\_absences.append(student.get\_text())  def add\_absences():  global list\_absences, user\_id, s\_absences\_append  query = '''select ID\_lesson  FROM  Lessons  inner join Periods on Periods.ID\_Period = Period\_id  Where Day = ?  and Periods.Start = ?  and Teacher\_id = ?'''  cursor.execute(query, (str(c\_abs\_days.value), str(c\_abs\_periods.value[:2]+c\_abs\_periods.value[3:]), str(user\_id)))  lesson\_id = cursor.fetchone()[0]  query = '''INSERT INTO Attendance  (Lesson\_id, Student\_id, Date)  values(?,?,?)'''  for student in list\_absences:  student = student.split("-")  cursor.execute(query, (lesson\_id, student[0][0],time.strftime("%d%m%Y", time.localtime())))  database.commit()  screen\_absences()  s\_absences\_append.destroy()  s\_absences\_append = Box(s\_absences, grid = [0,1])  def absance\_report():  global text\_absences, text\_total  text\_absences\_2 = text\_absences.split("\n")  with open('Report\_Attendance.csv', 'w') as csvfile:  filewriter = csv.writer(csvfile, delimiter=',', quotechar = '|', quoting = csv.QUOTE\_MINIMAL)  filewriter.writerow(["Date","Name"])  for line in text\_absences\_2:  line = line.strip()  line = line.split(" - ")  filewriter.writerow(line)  filewriter.writerow(text\_total)  def screen\_grades():  s\_timetable.hide()  s\_absences.hide()  s\_grades.show()  display\_grades()  def display\_grades():  global user\_type, user\_id, s\_grades\_list, text\_grades  s\_grades\_list.destroy()  s\_grades\_list = Box(s\_grades, height = 300, width = 350, grid = [0,0,1,4], align = "left")  if user\_type == 1:  query = '''select Date,  Subjects.Subject,  Grade  from Grades  inner join Lessons on Lessons.ID\_Lesson = Lesson\_id  inner join Groups on Groups.ID\_Group = Group\_id  inner Join Subjects on Subjects.ID\_Subject = Subject\_id  Where Grades.Student\_id = ?'''  elif user\_type == 2:  query = '''select Date,  Students.Student\_Name,  Students.Student\_Surname,  Grade  from Grades  inner join Students on Students.ID\_student = Student\_id  inner join Lessons on Lessons.ID\_Lesson = Lesson\_id  Where Lessons.Teacher\_id = ?'''  filter\_days\_grades()  cursor.execute(query, str(user\_id))  data = cursor.fetchall()  text\_grades = ""  for grade in data:  grade = list(grade)  grade[0] = str(grade[0])  grade[0] = grade[0][:2]+"."+grade[0][2:4]+"."+grade[0][4:]  if len(grade) == 3:  for item in grade:  text\_grades += str(item) + " - "  text\_grades = text\_grades[:len(text\_grades)-4]  else:  text\_grades += grade[0] + " - " + grade[1] + " " + grade[2] + " - "+ grade[3]  text\_grades += "\n"  text\_grades = text\_grades[:len(text\_grades)-1]  l\_grades = TextBox(s\_grades\_list, text = text\_grades, grid = [0,0], align = "top", width = 100, height = 13, scrollbar = True, multiline = True)  l\_grades.disable()  b\_grades\_reports = PushButton(s\_grades\_list, grades\_report, text="Export report", grid = [0,1])  def filter\_days\_grades():  global user\_id, s\_grades\_filters, c\_days  s\_grades\_filters.destroy()  s\_grades\_filters = Box(s\_grades, grid = [1,0,1,1], align = "top" )  query = ''' Select Day  from Lessons  where Teacher\_id = ?  '''  cursor.execute(query, str(user\_id))  list\_days = [x for x in cursor.fetchall()]  list\_days = set([str(x[0]) for x in list\_days])  c\_days = Combo(s\_grades\_filters, options=list\_days, command=filter\_periods\_grades, grid=[0,0], width=12, align="top")  def filter\_periods\_grades(selected\_value):  global user\_id, c\_periods  query = ''' select Start  from Lessons  inner join Periods on Periods.ID\_Period = Lessons.Period\_id  Where Teacher\_id = ?  and Day = ?'''  cursor.execute(query,(str(user\_id),selected\_value))  list\_periods = [str(x[0]) for x in cursor.fetchall()]  list\_periods = sorted(set([x[:2]+":"+x[2:4] for x in list\_periods]))  c\_periods = Combo(s\_grades\_filters, options = list\_periods, command = filter\_students\_grades, grid = [0,1], width = 12, align = "top")  def filter\_students\_grades(selected\_value):  global user\_id, c\_students  day = c\_days.value  periods = c\_periods.value  query = '''select Students.Student\_Name,  Students.Student\_Surname,  Students.ID\_Student  from Lessons  inner join Student\_Group on Student\_Group.Group\_id = Lessons.Group\_id  inner join Students on Students.ID\_Student = Student\_Group.Student\_id  inner join Periods on Periods.ID\_Period = Lessons.Period\_id  where Day = ?  and Periods.Start = ?  and Teacher\_id = ?'''  cursor.execute(query, (str(day),str(periods[:2]+periods[3:]),str(user\_id) ))  list\_students = [str(x[0]+" "+x[1]) for x in cursor.fetchall()]  c\_students = Combo(s\_grades\_filters, options = list\_students, command = filter\_grades\_grades, grid = [0,2], width = 12, align = "top")  def filter\_grades\_grades():  global c\_grades  list\_grades = ["A\*", "A", "B", "C", "D", "E", "U"]  c\_grades = Combo(s\_grades\_filters, options = list\_grades, command = get\_grade, grid = [0,3], width = 12, align = "top")  def get\_grade():  button\_submit\_grades = PushButton(s\_grades\_filters, add\_grades, text = "Submit grade", grid = [0,4], width = 12, align = "top")  def add\_grades():  global s\_grades\_append  query = '''SELECT ID\_lesson  FROM  Lessons  INNER JOIN Periods ON Periods.ID\_Period = Period\_id  Where Day = ?  AND Periods.Start = ?  AND Teacher\_id = ?'''  cursor.execute(query, (str(c\_days.value), str(c\_periods.value[:2]+c\_periods.value[3:]), str(user\_id)))  lesson\_id = cursor.fetchone()[0]  query = '''SELECT ID\_Student  FROM Students  WHERE Student\_Name = ?  AND Student\_Surname = ?  '''  student\_name = str(c\_students.value).split()  cursor.execute(query, (student\_name))  student\_id = cursor.fetchone()[0]  query = '''INSERT INTO Grades  (Lesson\_id, Student\_id, Grade, Date)  values(?,?,?,?)'''  student = str(c\_students.value)  cursor.execute(query, (lesson\_id, student\_id, c\_grades.value,time.strftime("%d%m%Y", time.localtime())))  database.commit()  screen\_grades()  s\_grades\_append.destroy()  s\_grades\_append = Box(s\_grades, layout = "auto", grid = [0,1])  def grades\_report():  text\_grades\_2 = text\_grades.split("\n")  with open('Report\_Grades.csv', 'w') as csvfile:  filewriter = csv.writer(csvfile, delimiter=',', quotechar = '|', quoting = csv.QUOTE\_MINIMAL)  filewriter.writerow(["Date","Name","Grade"])  for line in text\_grades\_2:  line = line.strip()  line = line.split(" - ")  filewriter.writerow(line)  # ----- GUI -----  SYSTEM = App(title = "School Information System", height = 350, width = 1000, layout = "grid")  # ----- Log In -----  s\_login = Box(SYSTEM, layout = "grid", grid = [0,0], align = "top")  l\_logo = Text(s\_login, text = "Oxford Best College", size = 26, grid = [0,0,2,1])  l\_user = Text(s\_login, text = "Username:", grid = [0,2], align = "right")  l\_pass = Text(s\_login, text = "Password:", grid = [0,3], align = "right")  b\_login = PushButton(s\_login, logIn, text = "Log In", width = 10, grid = [0,4])  b\_exit = PushButton(s\_login, exit, text = "Exit", width = 10, grid = [1,4])  input\_user = TextBox(s\_login, grid = [1,2], align="left")  input\_pass = TextBox(s\_login, grid = [1,3], align="left")  input\_user.focus()  # ----- Change default password -----  s\_change\_pass = Box(SYSTEM, layout = "grid", grid = [0,0], align = "top")  l\_logo = Text(s\_change\_pass, text = "Oxford Best College", size = 26, grid = [0,0,2,1])  l\_new\_pass = Text(s\_change\_pass, text = "Enter new Pasword:", grid = [0,1], align = "right")  l\_confirm\_pass = Text(s\_change\_pass, text = "Confirm Password:", grid = [0,2], align = "right")  input\_new\_pass = TextBox(s\_change\_pass, grid = [1,1], align = "left")  input\_confirm\_pass = TextBox(s\_change\_pass, grid = [1,2], align = "left")  button\_submit = PushButton(s\_change\_pass, change\_pass, text = "Submit", width = 10, grid = [0,3])  button\_exit = PushButton(s\_change\_pass, exit, text = "Exit", width = 10, grid = [1,3])  input\_new\_pass.focus()  s\_change\_pass.hide()  # ----- Menu bar -----  s\_menu = Box(SYSTEM, height = "fill", layout = "grid", align = "left", grid = [0,1])  button\_timetable = PushButton(s\_menu, screen\_timetable, text = "Timetable", width = 10, height = 2, grid = [0,2,2,1])  button\_absences = PushButton(s\_menu, screen\_absences, text = "Absences", width = 10, height = 2, grid = [0,3,2,1])  button\_grades = PushButton(s\_menu, screen\_grades, text = "Grades", width = 10, height = 2, grid = [0,4,2,1])  button\_exit = PushButton(s\_menu, exit, text = "Exit", width = 10, height = 2, grid = [0,5,2,1])  s\_menu.hide()  # ----- Timetable -----  s\_timetable = Box(SYSTEM, height = "fill", layout = "grid", align = "left", grid = [2,1])  s\_timetable.hide()  # ----- Absences -----  s\_absences = Box(SYSTEM, height = "fill", layout = "grid", align = "left", grid = [2,1])  s\_absences\_list = Box(s\_absences, grid = [0,0])  s\_absences\_filters = Box(s\_absences, grid = [1,0])  s\_absences\_append = Box(s\_absences, grid = [0,1])  s\_absences.hide()  # ----- Grades -----  s\_grades = Box(SYSTEM, height = "fill", layout = "grid", align = "left", grid = [2,1])  s\_grades\_list = Box(s\_grades, grid = [0,0])  s\_grades\_filters = Box(s\_grades, grid = [1,0])  s\_grades\_append = Box(s\_grades, layout = "auto", grid = [0,1])  s\_grades.hide()  SYSTEM.display() |

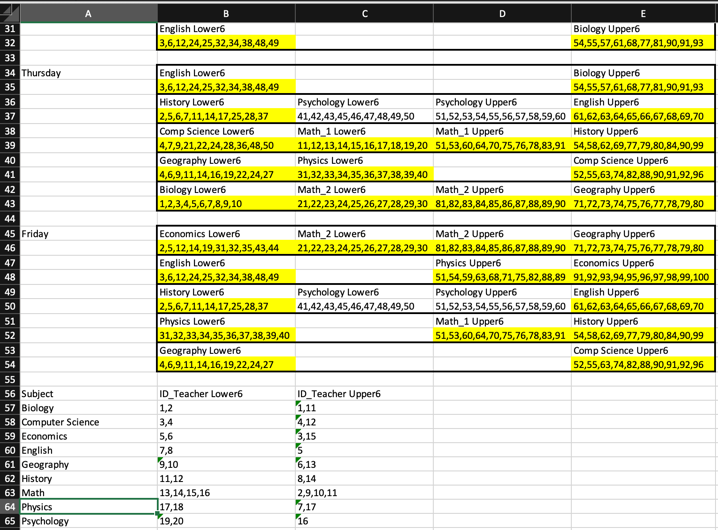
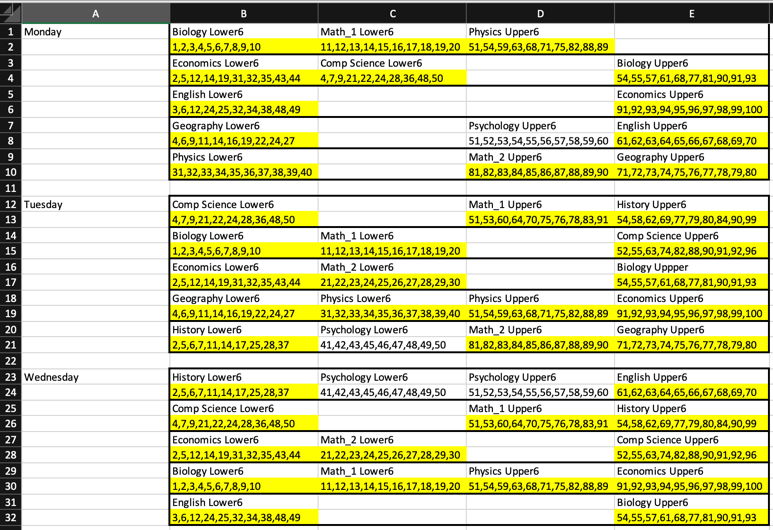
# Testing

|  |  |
| --- | --- |
| Test | Description |
| Database | Testing whether the database functionate correctly. Testing executed by insert data and testing different queries. |
| GUI | Testing whether all the elements of the GUI display correctly. Testing executed by running the different screen views. |
| GUI querying data | Testing whether the GUI displays personalised information based on the current user. Test executed by logging in as different users (both students and teachers). |
| GUI buttons and options | Testing whether the GUI functionates as expected. Testing executed by trying all different buttons and options. |
| Inserting data into database | Testing whether the GUI could inset data into the database. Testing executed by inserting new data and updating the GUI. |
| Exporting data | Testing whether the system could export data. Testing executed by exporting different datasets. |

Setting up the database  
  
In order to test the functionality of the system, I needed to set up the database as it would be done by administrators in the beginning of the academic year.  
  
First, I inserted 100 students and 20 teachers. This is a small sample compared to the one the college would be using; however, it is sufficient enough for testing purposes. I used a website [8] to generate the data. It allows you to randomly create data – the first and surnames and the age in this instance.  
  
  
  
Then I inserted 9 different subjects, the 6 periods of the day, and different groups to put the students in.  
  
Afterwards, I created a timetable for the week including the different groups the students would be put in, the teachers carrying out the lessons and assigned all the students into groups. This would be the complete scheduling of the school. I did this manually by using an Excel spreadsheet. I inserted all the data manually into the database as it is supposed at the beginning of the academic year.

**Figure 17: www.generatedata.com**

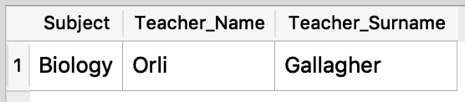
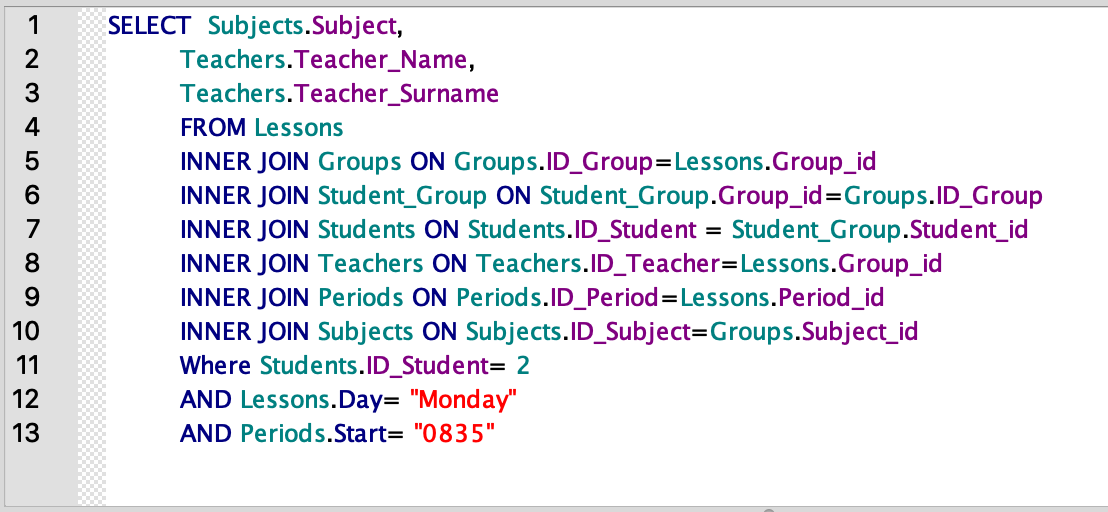
**Figure 18: Timetable spreadsheet**



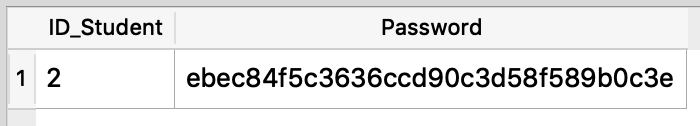
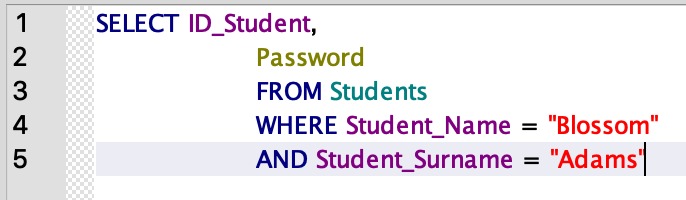
Testing different queries

Test 1: Query to select the subject and the teacher for a parameterised student and period. (Used for displaying the timetable.)

**Figure 19: Test Query 1 - Subject and teacher**



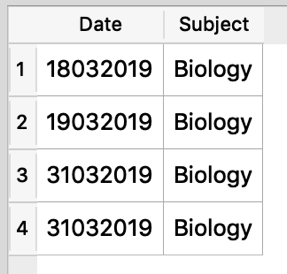
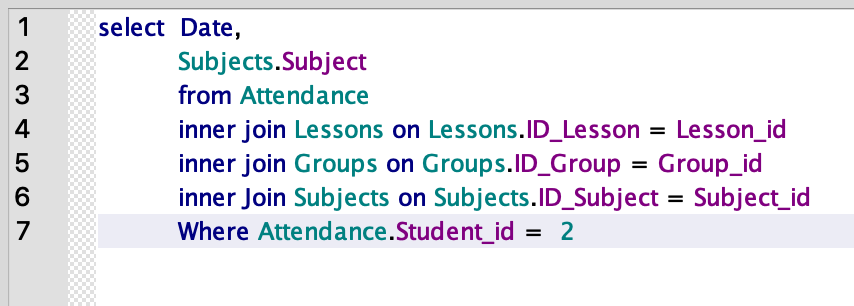
Test 2: Query to select the password for a parameterised name and surname. (Used in the login process)



**Figure 20: Test Query 2 - Password**

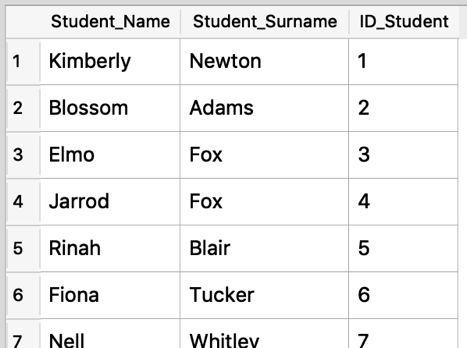
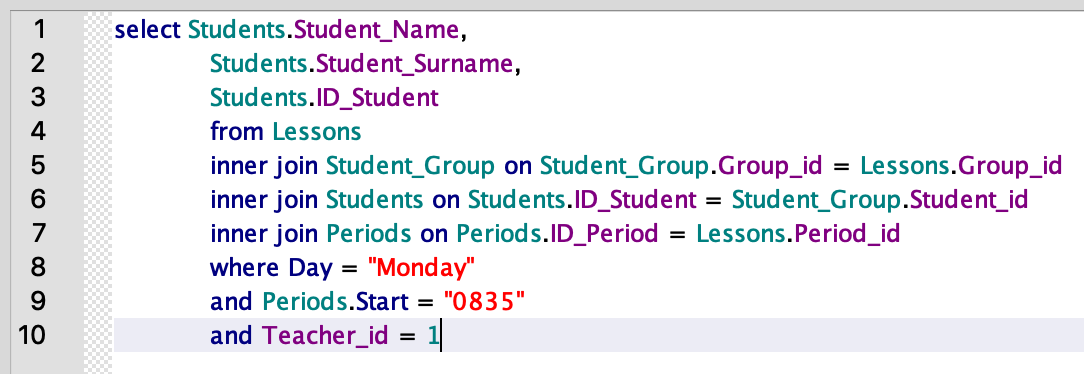
Test 3: Query to select the absences for a parameterised Student\_id (Used to display the absences of a student(s))

**Figure 21: Test Query 3 - Absences**



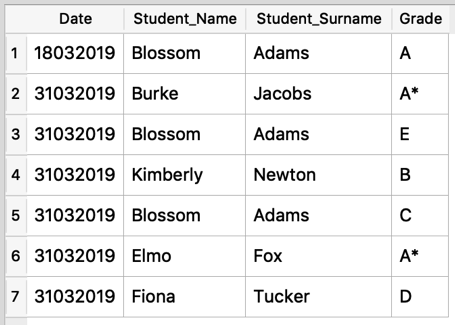
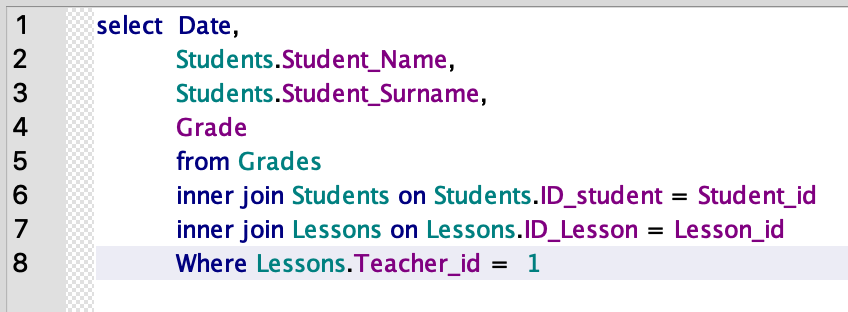
Test 4: Query to select all the names of all students in a lesson for a parameterised lesson. (Used for filtering students when teacher append data.)

**Figure 22: Test Query 4 - Students in a lesson**



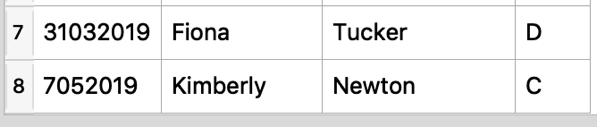
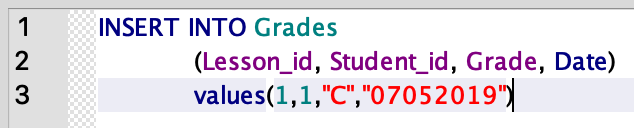
Test 5: Query to select all the grades a teacher has recorded for a parameterised teacher. (Used to display that data)

**Figure 23: Test Query 5 - Grades recorded by a teacher**

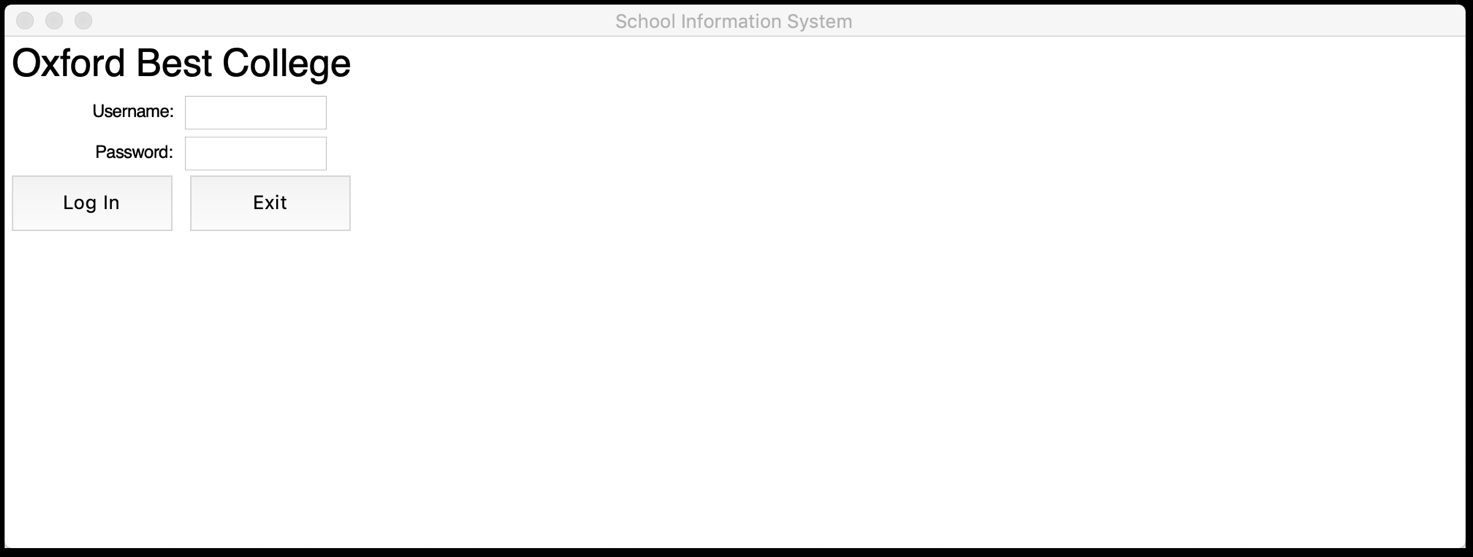


Test 6: Query to inserting grades in the database.

**Figure 24: Test Query 6 - Inserting grades**

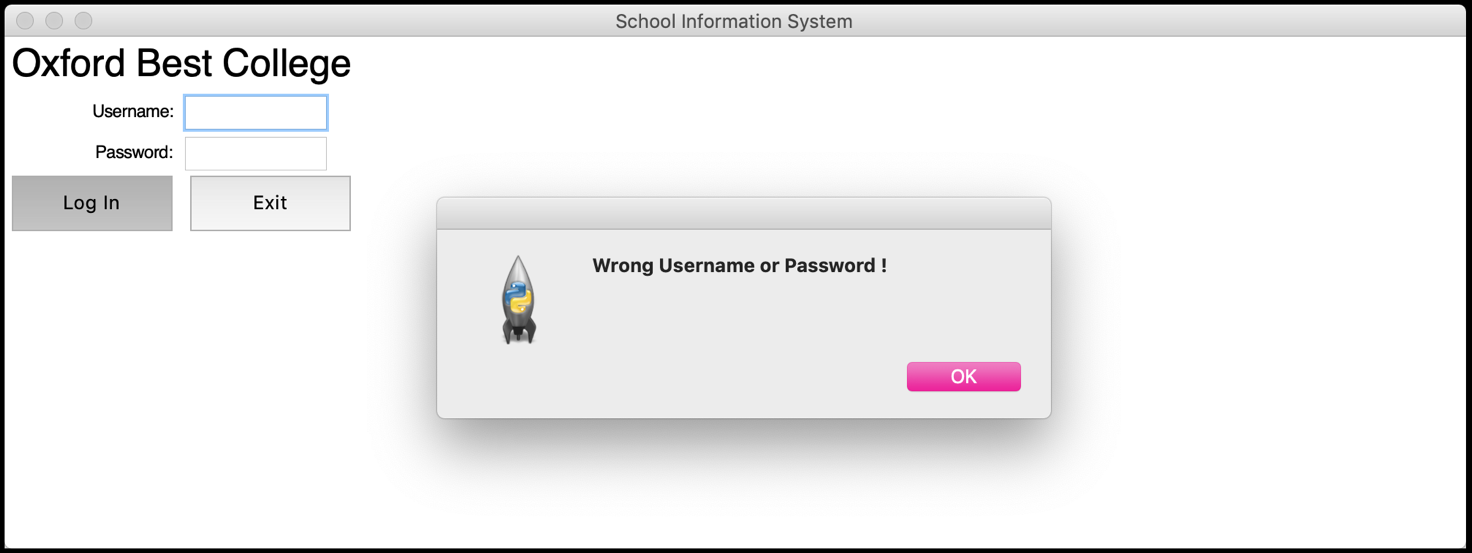


User login  
  
When the GUI is opened the user is presented with a login screen.

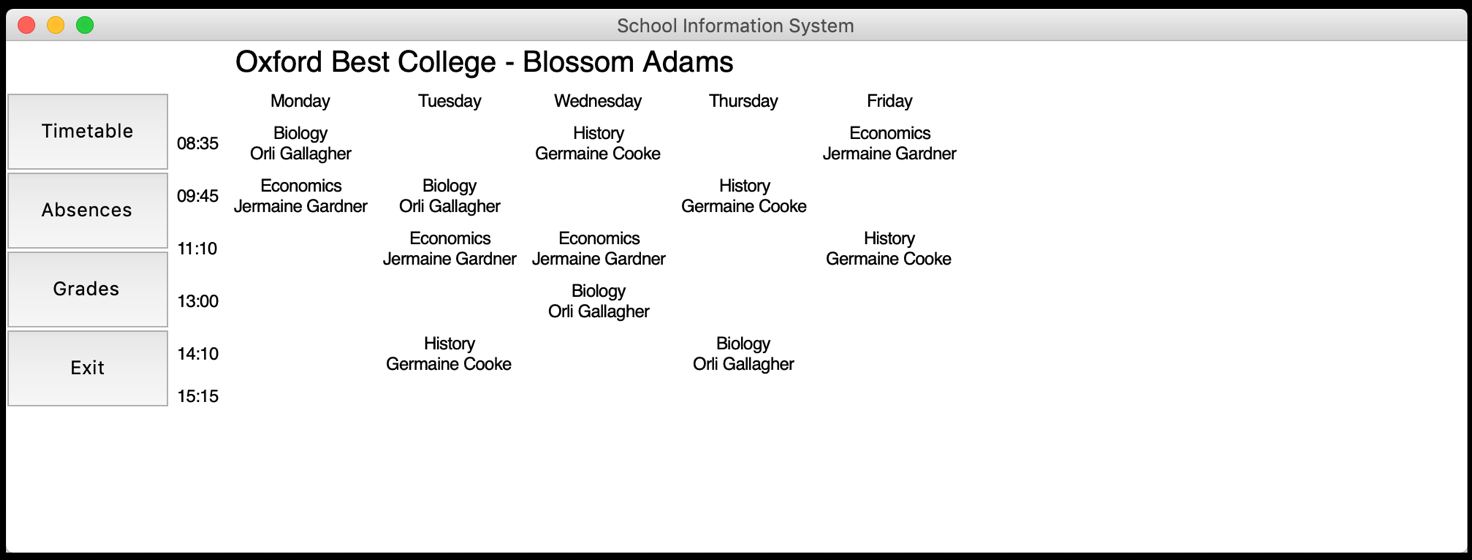


**Figure 25: login screen**

The system checks if the entered credentials are correct. Currently all passwords are set to “12345678”. If the credentials are wrong and the system could not find a match the user is presented an error.

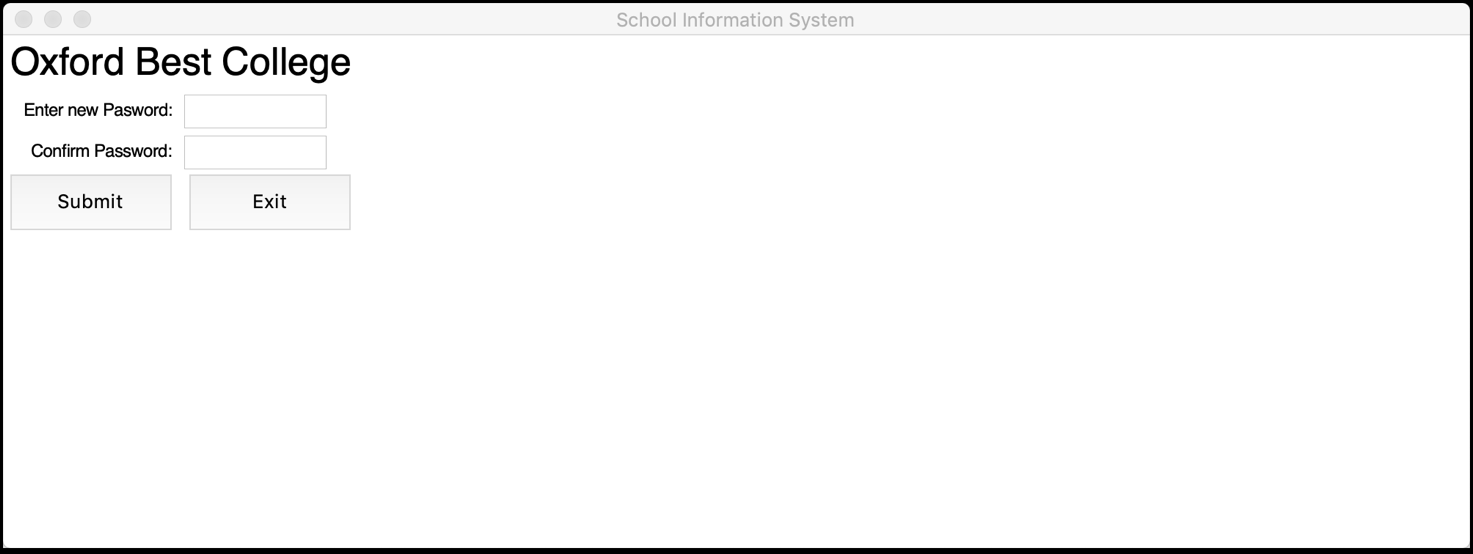


**Figure 26: Wrong credentials**

  
  
If details are correct the user is presented with the main view.

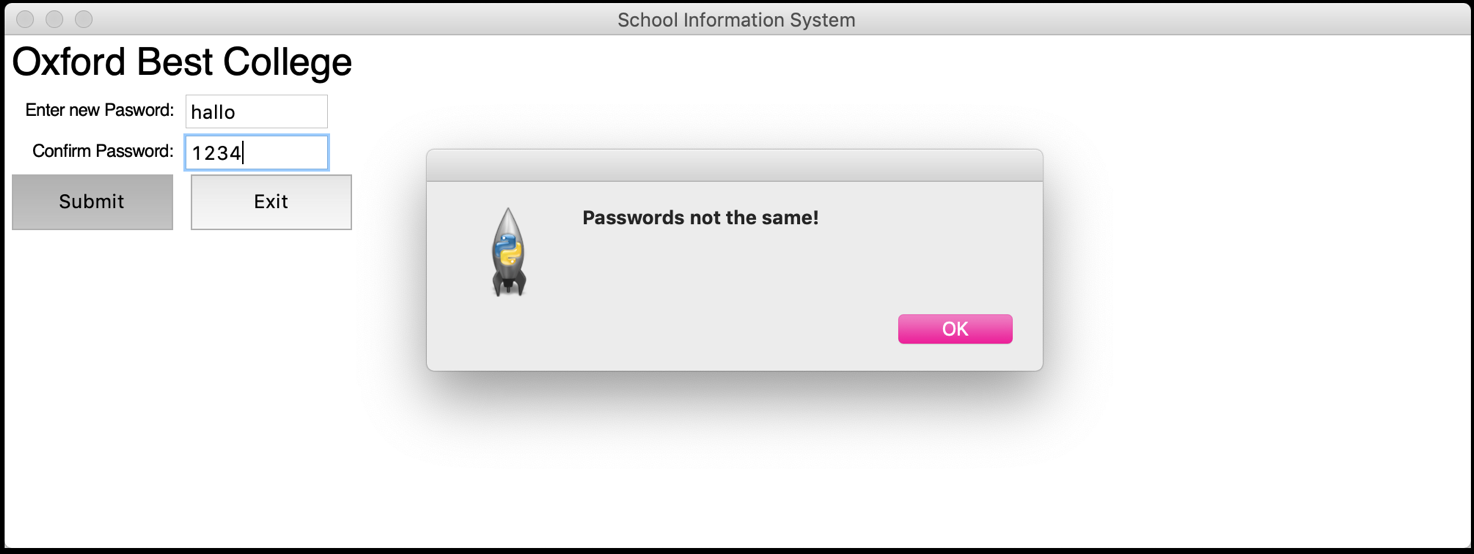
**Figure 27: main/timetable screen**

Changing the password  
  
All passwords are set to “12345678” by default. The first a user is logged in they are prompted to change their password. They are presented with a different view.



**Figure 28: Change password screen**

The system checks if the password is entered correctly in both fields and an error is displayed otherwise.



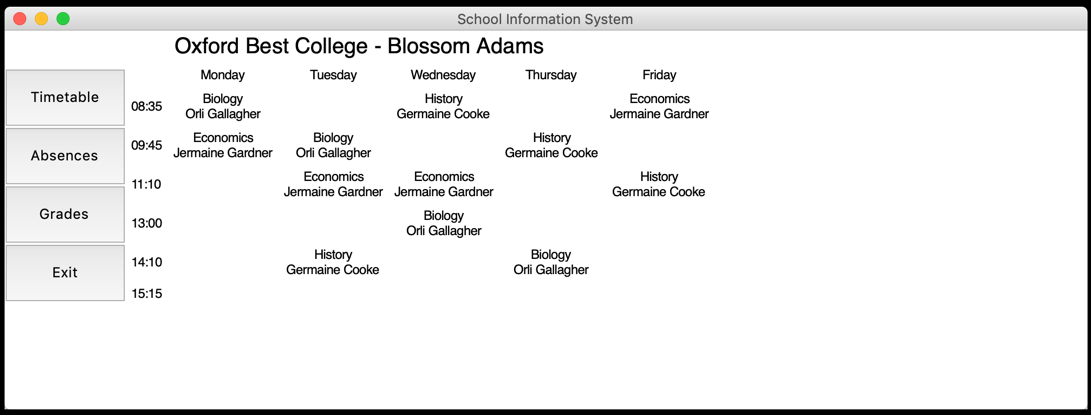
**Figure 29: "Password not the same!" error**

The system checks if the password meets the required criteria and an error is displayed otherwise. Password need to be:  
- 8 characters long  
- Have at least 1 digit  
- Have at least 1 uppercase AND 1 lowercase character

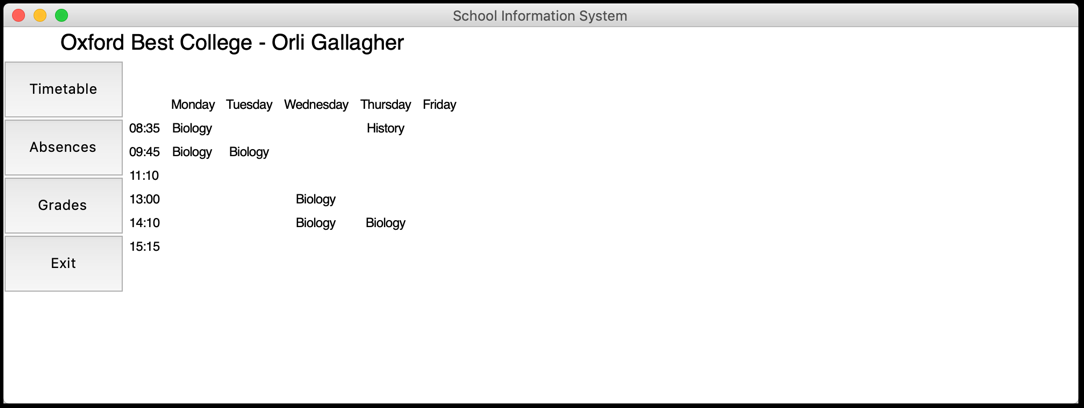
## Querying data and displaying it

**Figure 30: Not a valid password error**

Time tabling   
  
Once the user is logged in, they are presented with the main view. It contains a menu, their name on the top and personalised timetable. Users are able to access this screen by clicking the “timetable” button in the menu. If the user is a student the timetable also displays the name of the teacher carrying out the lesson.

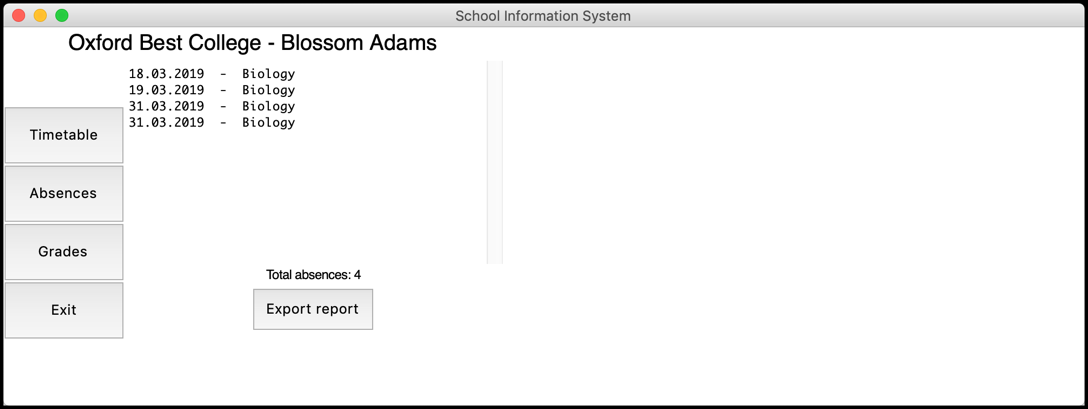


**Figure 31: Timetable screen - Student**

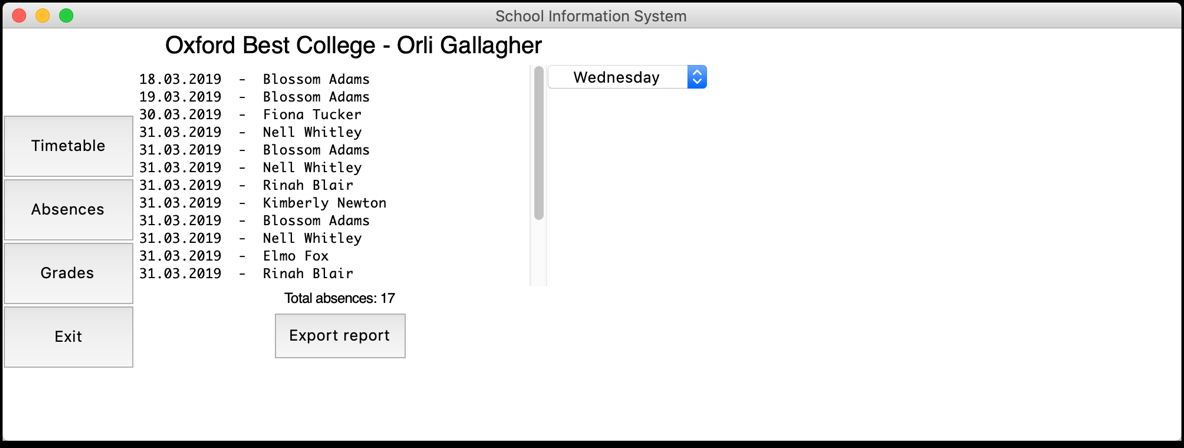


**Figure 32: Timetable screen - Teacher**

Absences   
  
The absences view presents users with a list of all their absences including a date and a subject. Users are able to access this screen by clicking the “Absences” button in the menu.

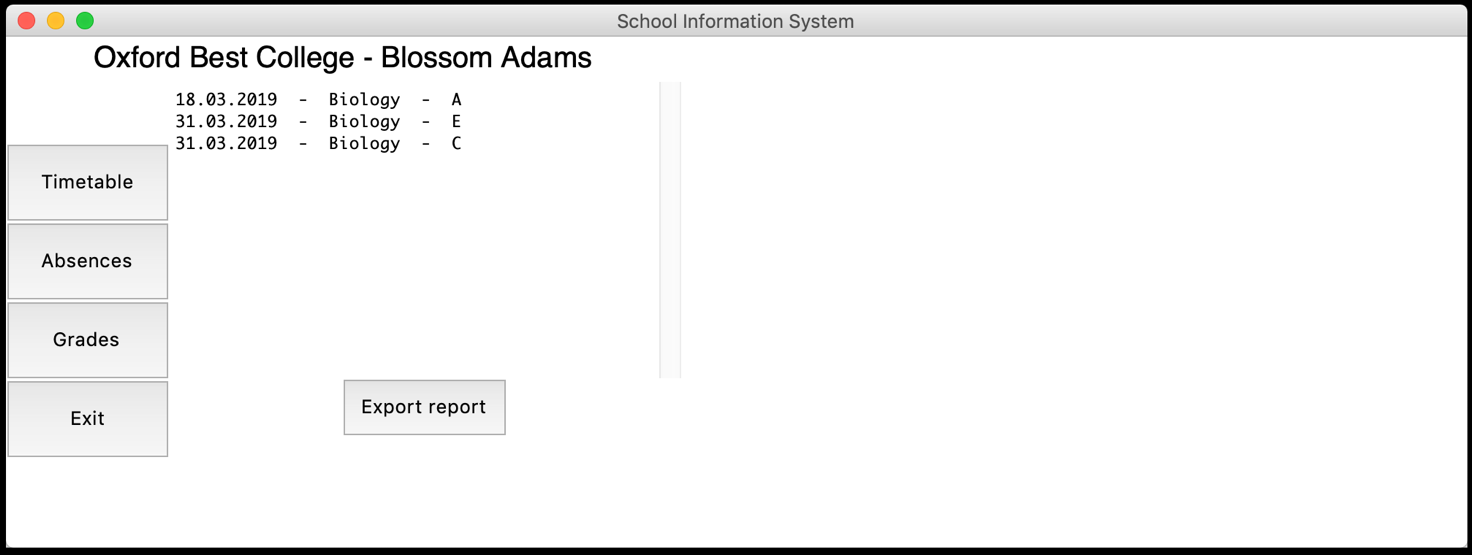


**Figure 33: Absences screen - Student**

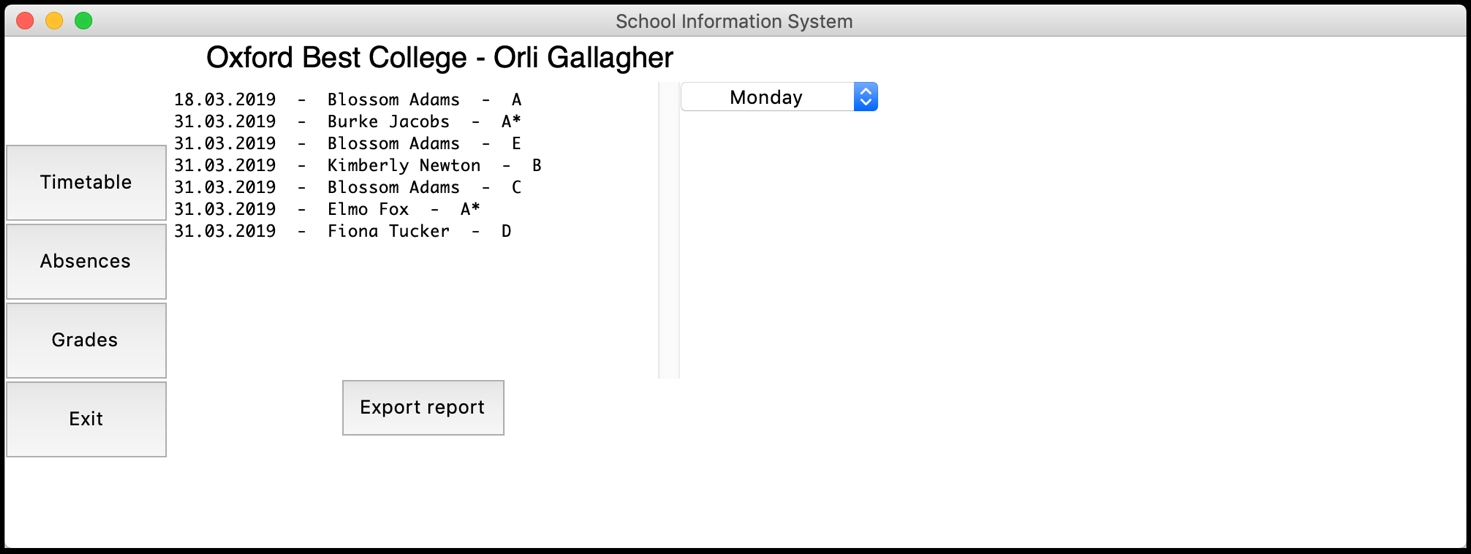


**Figure 34: Absences screen - Teacher**

Grades   
  
The grades view presents users with a list of all their grades including a date, a subject, and a grade. Users are able to access this screen by clicking the “Grades” button in the menu.



**Figure 35: Grades - Student**

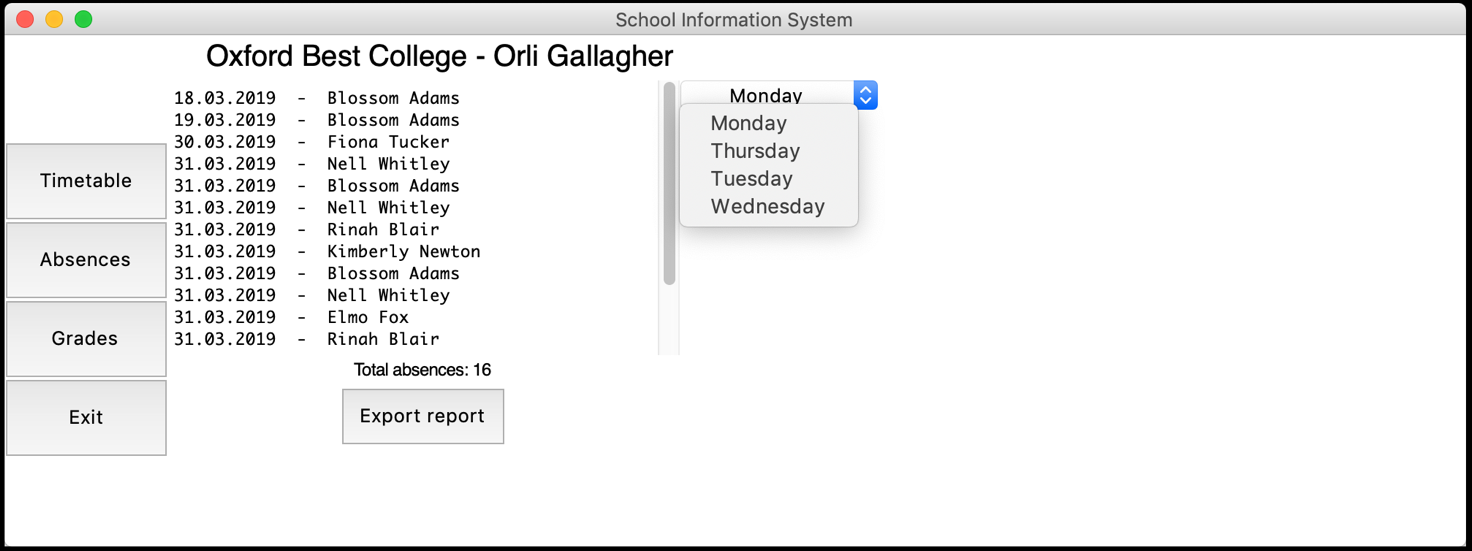


**Figure 36: Grades - Teacher**

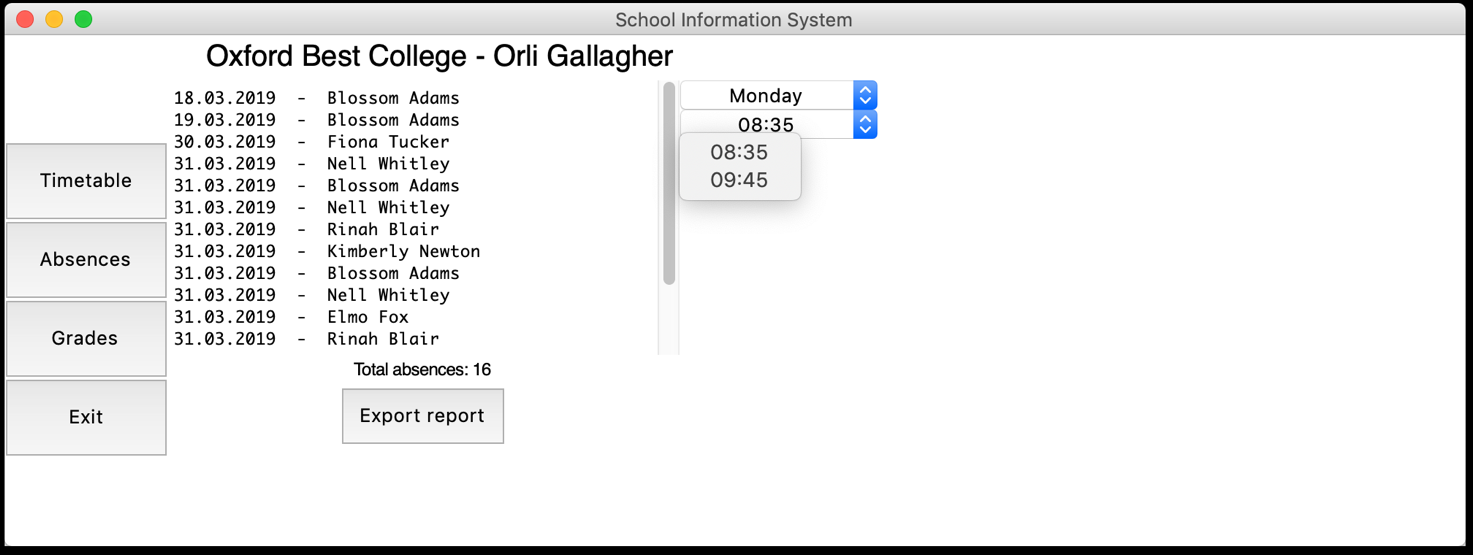
Appending data to the database  
  
If the user is a teacher, they are able to append data into the database. They are presented with different views in the absences and grades screens in order to insert data.  
they are presented with a set of filters to make inserting data more convenient.

### Filtering groups

Absences   
  
The first filter is a dropdown menu from which the user chooses a date of the week. Teachers are presented with only the days they have lesson on.

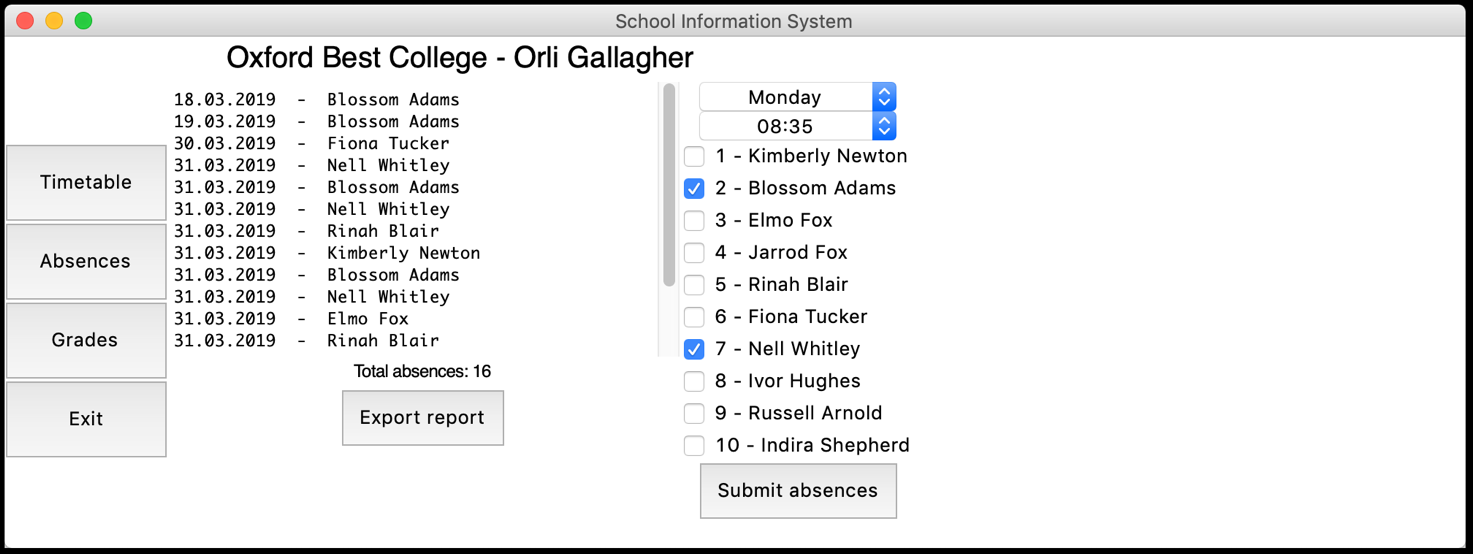
  
  
A second filter is displayed which is dropdown menu from which the user chooses the period of the day. Teacher are presented with only the periods they have on the chosen day.

**Figure 37: Weekdays filter**



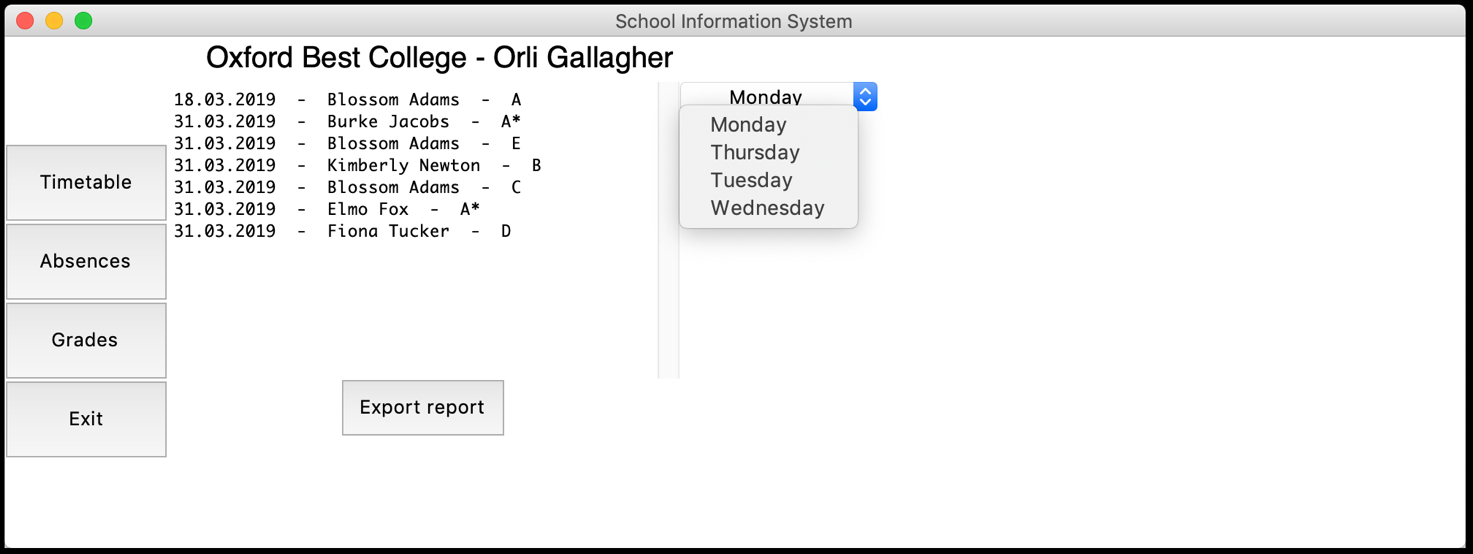
**Figure 38: Periods filter**

The user is then displayed a checklist with all the students in the group of the selected lesson. They can select which students are absent.

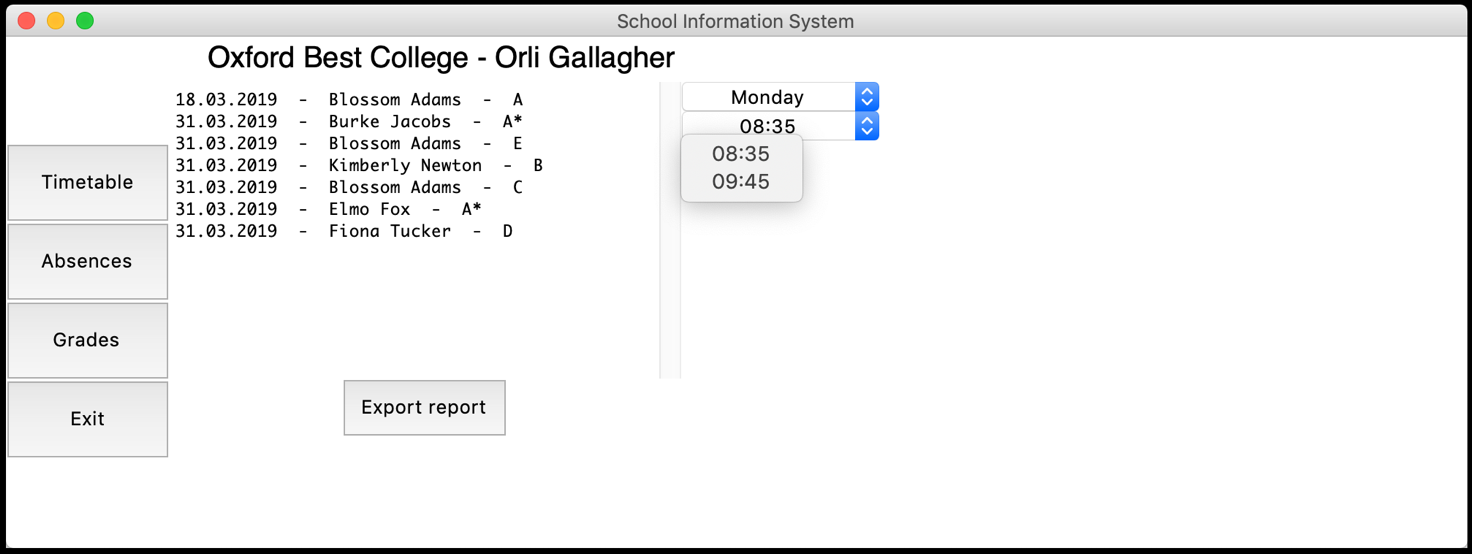


**Figure 39: Checklist filter**

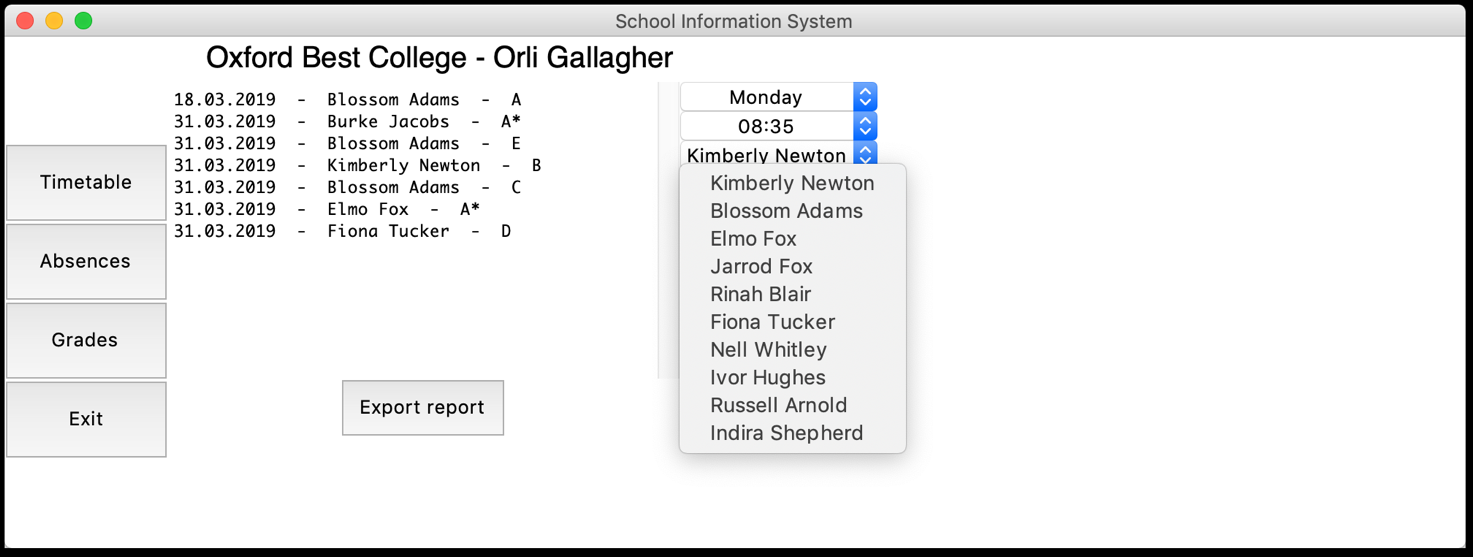
Grades   
  
The first filter is a dropdown menu from which the user chooses a date of the week. Teachers are presented with only the days they have lesson on.

  
  
A second filter is displayed which is a dropdown menu from which the user chooses the period of the day. Teacher are presented with only the periods they have on the chosen day.

**Figure 40: Weekdays filter**

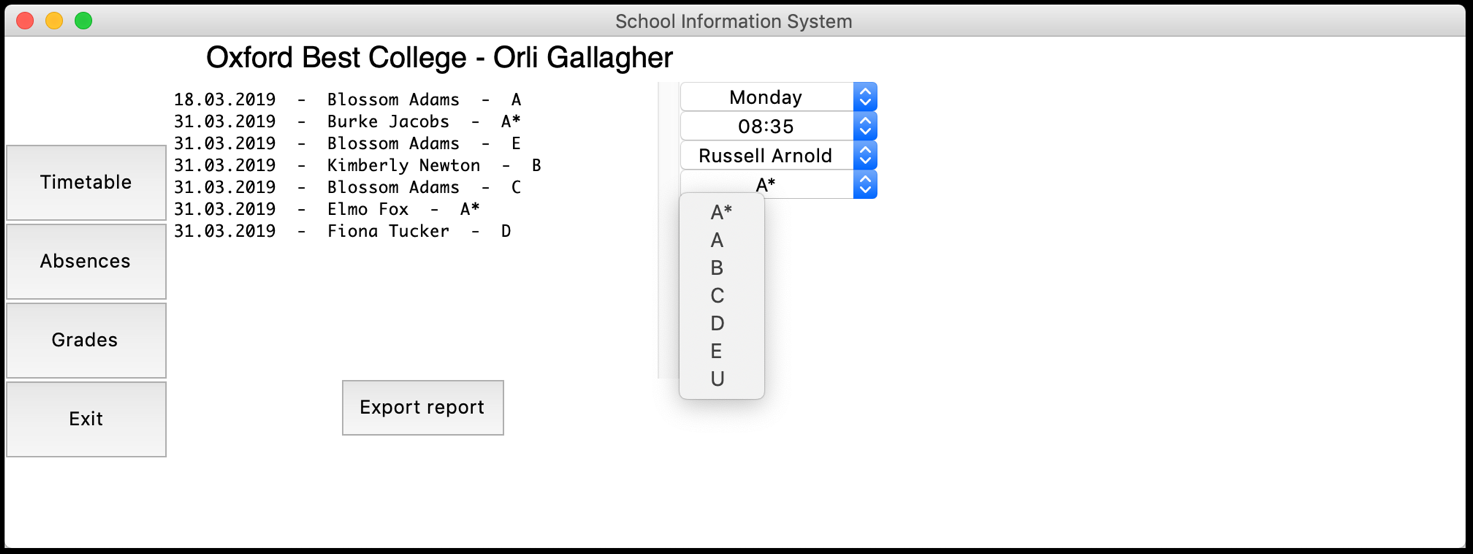
  
  
A third filter is displayed which is a dropdown menu from which the user chooses the students in the selected lesson. Teachers are presented with only the students in the lesson corresponding to the day and period.

**Figure 41: Periods filter**



**Figure 42: Students filter**

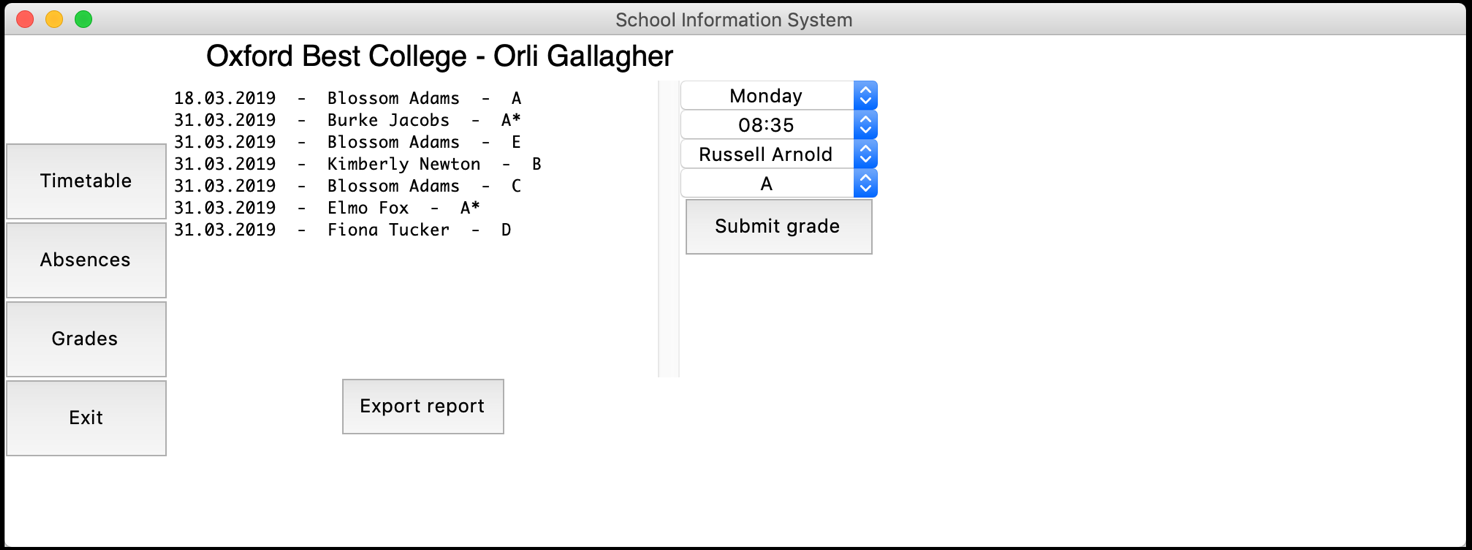
A fourth filter is displayed which is a dropdown menu from which the user selects the grade.



**Figure 43: Grades filter**

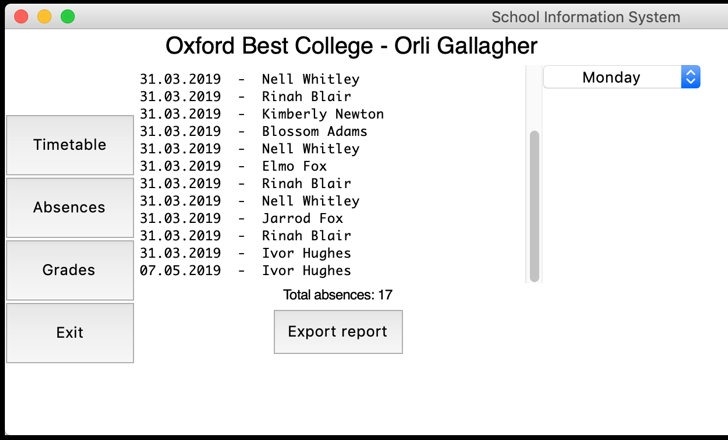
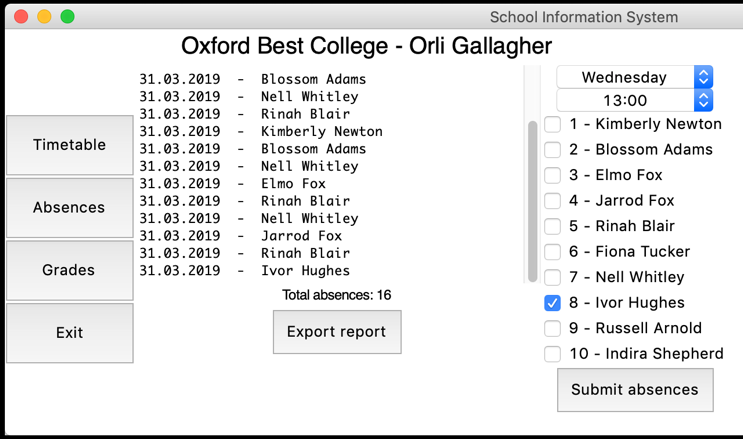
When all filters are selected a “Submit grade” button is displayed.

**Figure 44: "Submit" button displayed**

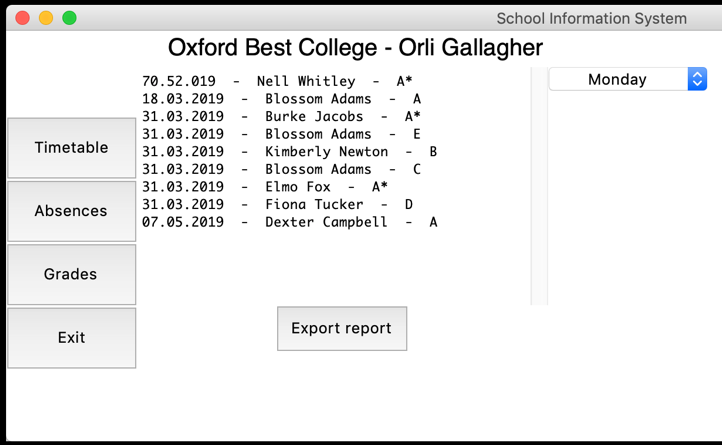
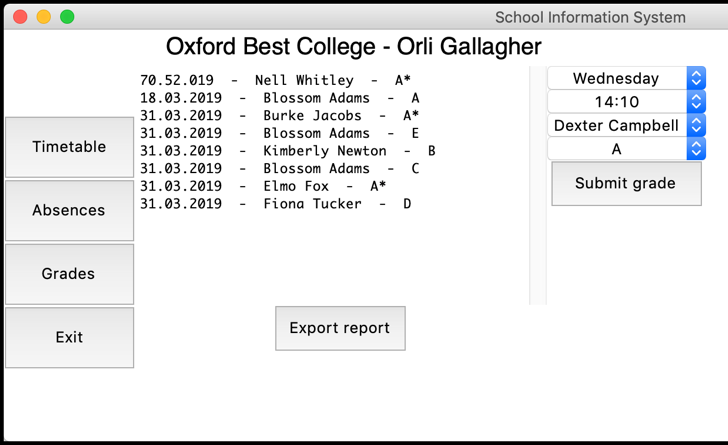


Appending  
  
When the user clicks the “Submit” button the data is inserted into the database. We can see that as the list on the left is updated with the new data and the filter are reset to default.

**Figure 45: Inserting an absence into the database**

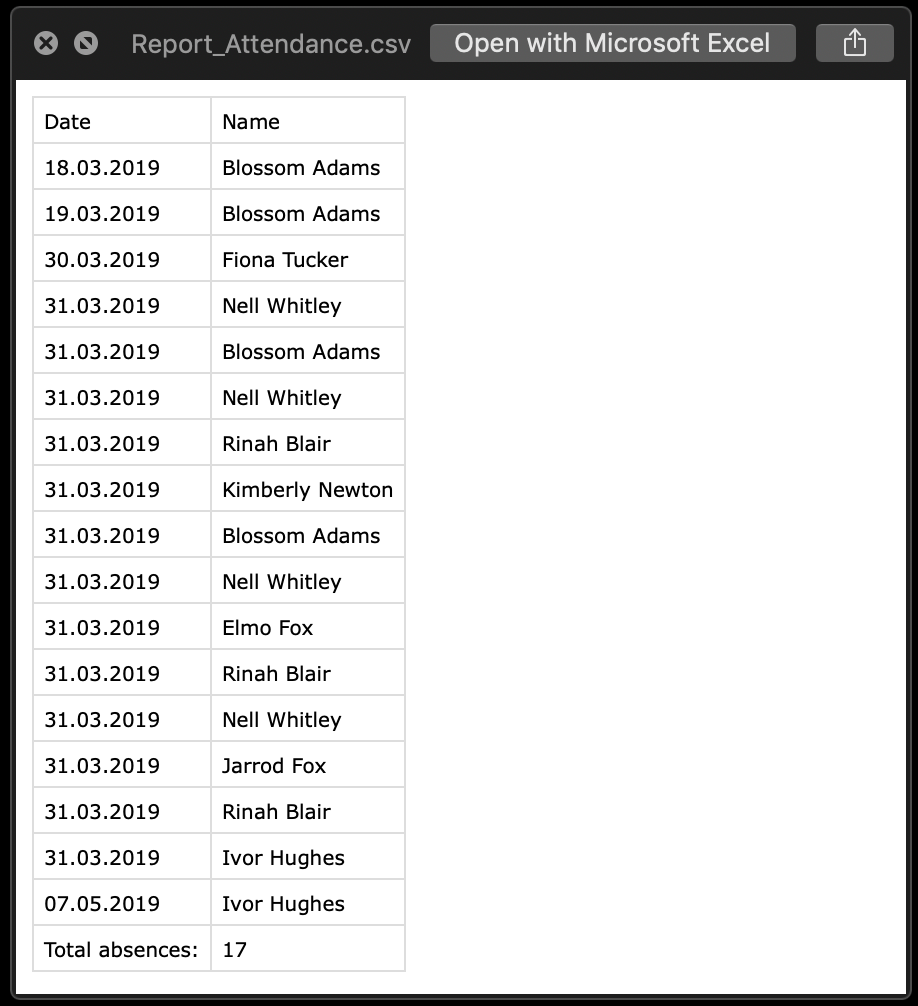
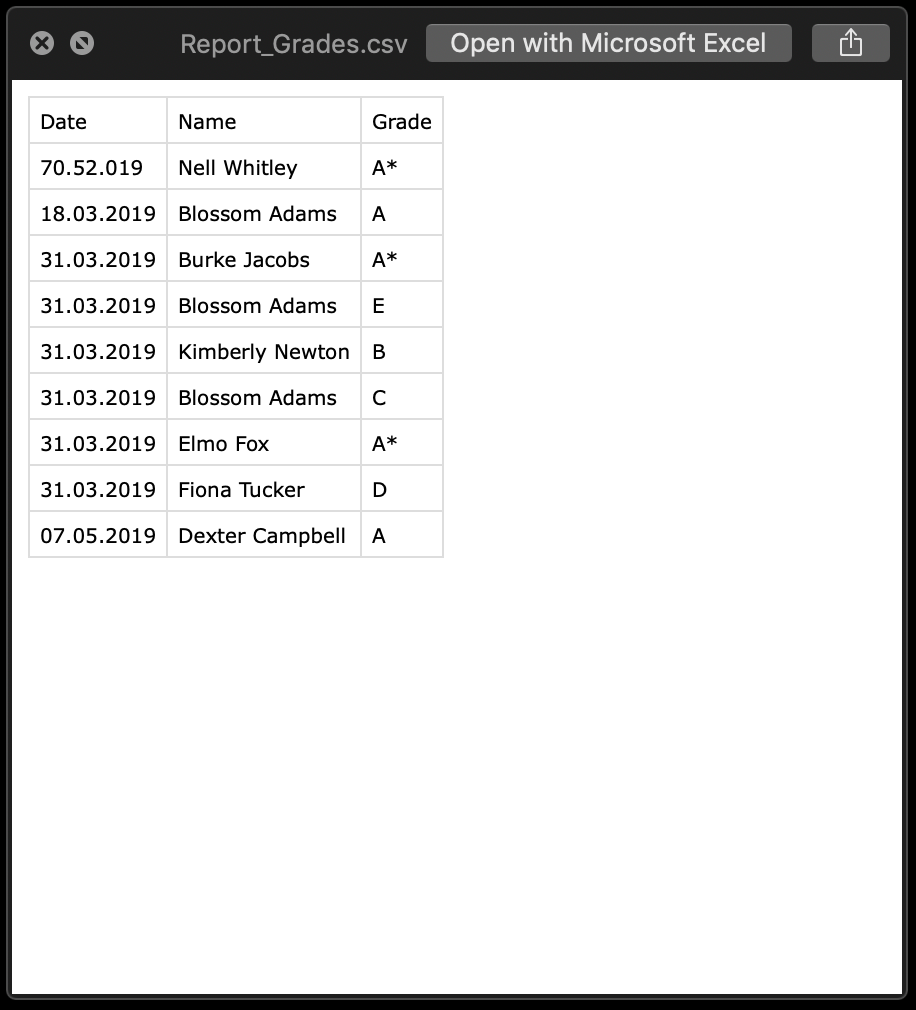


**Figure 46: Inserting a grade into the database**



Exporting reports

In the absences and grades screens all user are presented with an “Export” button. When clicked the displayed data is exported as a csv file. The file is stored in the same folder where the program is stored.



**Figure 58: Grades report**

**Figure 57: Absence report**

# Evaluation

Objectives  
  
The table summarises all the objectives described in [Functionality and objectives]

|  |  |  |
| --- | --- | --- |
| Objective | Met | Comment |
|  | | |
| Database: | Yes | A database structure is created as described including all the functionality. |
| Storing personal information | Yes | The database is able to store personal information about teachers and students in the college. |
| Dealing with timetabling | Yes | The database facilitates all the needed data to store and display a timetable. |
| Flexibility | Yes | Administrator are able to add additional field for personal information. Furthermore, they could easily change groups, subjects and teachers by updating the existing records.  Modifications could be made without the need to set-up the whole database again. |
| Attendance | Yes | The database is able to store information about attendance. |
| Grades | Yes | The database is able to store information about grades. |
|  | | |
| GUI: | Yes | A GUI is created as described including all the functionality. It is simple and intuitive. |
| Log in | Yes | A log in screen is displayed. A change-passwords screen is displayed the first time a user logs into the system. Users are presented with errors when necessary.  Log in functionality based on names and passwords works correctly. |
| Users | Yes | The GUI differentiates between user and displays personalised data. |
| Timetables | Yes | The GUI displays a personalised timetable for the current user. |
| Absences and grades | Yes | The GUI displays personalised lists of the absences and grades for the current user. |
| Inserting into database | Yes | A teacher is able to use the GUI to insert data into the database. |
| Reports | Yes | The system can create and export csv file of the displayed data. |
|  |  |  |
| Privacy | Yes | Passwords are stored securely as hashed values. Users see only data about themselves and are not able to access others data. |

As shown, I have met all the set objectives of this project.

## User feedback

After finishing the project, I presented the result to my client –” Oxford Best College” – and walked them through the system. I showed them all the functionality and features and explained how it is supposed to be set up and used. Their feedback was as follow:  
  
They found the database easy and convenient to set up. The provided solution was way more sophisticated than the current use of spreadsheets and the college was very happy with the scalability as they are planning on expanding. The database would be able to accommodate all their needs.   
Furthermore, they found the GUI to user-friendly which is a big objective as they would not need to train users how to interact with the system. They pointed out that the interface is not very pretty, however, it is intuitive and gets the job done.   
They said the GUI provides all the needed functionally and they were more than pleased with the overall result.

Possible future extensions  
  
 - Improving the design of the GUI – The current design is basic and could be improved. However, this requires the use of different library or even a different platform to build the system on. For example, more detailed graphics could be implemented to make the GUI appear better.  
  
- Including an automated scheduling algorithm for the timetabling. Currently it is a very difficult process which needs to be done by hand which is very time consuming. An algorithm could be implemented to do so, however, this is a hardly-computational, very complex feature and a problem of itself.  
  
- Improving the reports – Currently the user is presented only with raw data about absences and grades. This data could be used to create averages about the attendance of the student, groups or the whole academic year. Furthermore, report about the average performance based on the grades could be implemented.

# References

1. https://www.wcbs.co.uk/solutions-for-your-school/academic
2. <https://www.capita-sims.co.uk>
3. <https://www.postgresql.org>

1. <https://www.sqlite.org/index.html>

1. [https://processing.org](https://processing.org/)

1. <https://docs.python.org/3/library/tk.html>

1. <https://lawsie.github.io/guizero/about/>

1. [https://www.generatedata.com](https://www.generatedata.com/)