School Information System

A LEVEL COMPUTING PROJECT

Danail Todorov

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.

    1.1. 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.

    1.2. 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

    1.3. 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.

    1.4. 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.

    1.5. 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.

    2.1. 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.

    2.2. 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.

    2.3. 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.

    2.4. 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.

2.5. 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).

2.6. 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

Figure 1: WCBS's 3Sys Logo

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.

2. Capita’s SIMS

Figure 2: Capita's SIMS Logo

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.

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

Figure 3: PostgreSQL Logo

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.

2. SQLite

Figure 4: SQLite Logo

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.

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

Figure 5: Processing 3 Logo

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.

2. 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).

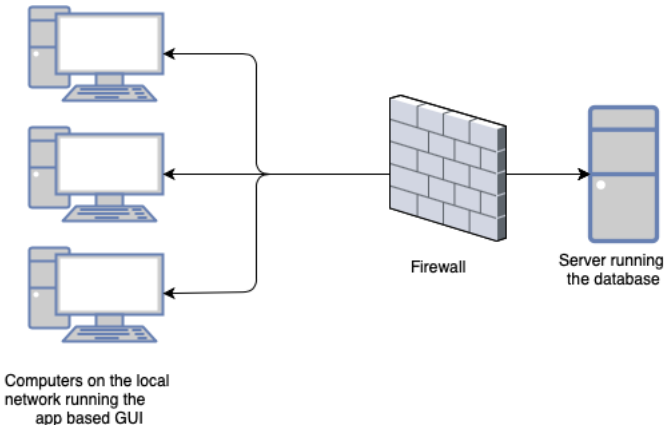


Figure 6: Deploying the system

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.

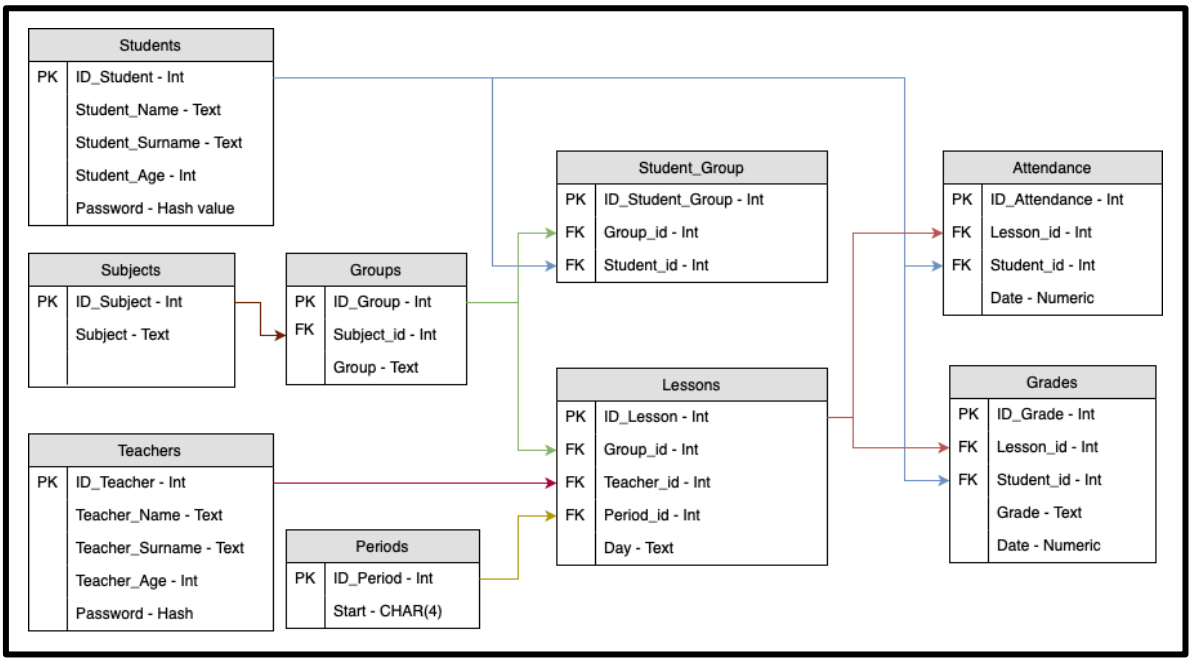


Figure 7: Database relation diagram

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.

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.

	ID_Student	Student_Name	Student_Surname	Student_Age	Password
	Filter	Filter	Filter	Filter	Filter
1	1	Kimberly	Newton	18	25d55ad283aa...
2	2	Blossom	Adams	19	25d55ad283aa...

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.

	ID_Teacher	Teacher_Name	Teacher_Surname	Teacher_Age	Password
	Filter	Filter	Filter	Filter	Filter
1	1	Orli	Gallagher	48	25d55ad283aa...
2	2	Alice	Bird	39	25d55ad283aa...

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

	ID_Subject	Subject
	Filter	Filter
1	1	Biology
2	2	Computer Science

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.

	ID_Period	Start
	Filter	Filter
1	1	0835
2	2	0945

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.

	ID_Group	Subject_id	Group
	Filter	Filter	Filter
1	1	1	L1
2	2	1	U1

Figure 12: Groups table

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.

	ID_Student_Group	Group_id	Student_id
	Filter	Filter	Filter
1	1	1	1
2	2	1	2

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”)

	ID_Lesson	Group_id	Teacher_id	Period_id	Day
	Filter	Filter	Filter	Filter	Filter
1	1	1	1	1	Monday
2	2	13	2	1	Monday

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.

	ID_Grade	Lesson_id	Student_id	Grade	Date
	Filter	Filter	Filter	Filter	Filter
1	1	1	2	A	18032019
2	2	9	57	A*	31032019

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.

	ID_Attendance	Lesson_id	Student_id	Date
	Filter	Filter	Filter	Filter
1	1	1	2	18032019
2	2	16	2	19032019

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.

```
1  SELECT ID_Student, Student_Name, Student_Surname, Student_Age
2  FROM Students
3  WHERE Students.Student_Age > 18
4  ORDER BY Student_Name ASC
5
```

	ID_Student	Student_Name	Student_Surname	Student_Age
1	36	Amelia	Tucker	19
2	69	Ariel	Ford	19

Figure 16: Example query 1

Example 2: Selecting all teachers carrying out mathematics lessons.

```
SELECT Teachers.Teacher_Name,
Teachers.Teacher_Surname,
Subjects.Subject
FROM Lessons
INNER JOIN Groups ON Groups.ID_Group = Group_id
INNER JOIN Subjects ON Subjects.ID_Subject = Subject_id
INNER JOIN Teachers ON Teachers.ID_Teacher = Teacher_id
WHERE Subjects.Subject = "Mathematics"
GROUP BY Teachers.Teacher_Name
```

	Teacher_Name	Teacher_Surname	Subject
1	Alice	Bird	Mathematics
2	Dylan	Knapp	Mathematics
3	Germaine	Cooke	Mathematics
4	Kelly	Haley	Mathematics
5	Nero	Maddox	Mathematics

Figure 17: Example query 2

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

```
SELECT Students.Student_Name,
Students.Student_Surname
FROM Groups
INNER JOIN Student_Group ON Student_Group.Group_id = ID_Group
INNER JOIN Students ON Students.ID_Student = Student_Group.Student_id
WHERE ID_Group = 1
ORDER BY Students.Student_Name ASC
```

	Day	Start	Subject
1	Monday	0835	Biology
2	Monday	0835	Mathematics
3	Monday	0835	Physics

Figure 18: Example query 3

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

```
SELECT Lessons.Day,
Periods.Start,
Subjects.Subject
FROM Lessons
INNER JOIN Groups ON Groups.ID_Group = Group_id
INNER JOIN Subjects ON Subjects.ID_Subject = Subject_id
INNER JOIN Periods ON Periods.ID_Period = Period_id
WHERE Day = "Monday"
AND Periods.Start = "0835"
```

	Student_Name	Student_Surname
1	Blossom	Adams
2	Elmo	Fox
3	Fiona	Tucker
4	Indira	Shepherd
5	Ivor	Hughes
6	Jarrod	Fox

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

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.

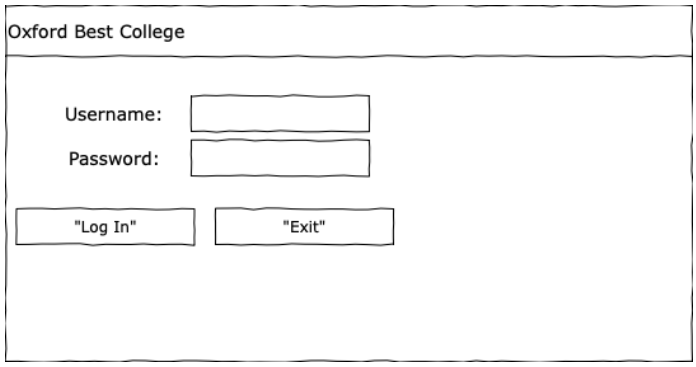


Figure 20: GUI - Log-in diagram

Change Password

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.

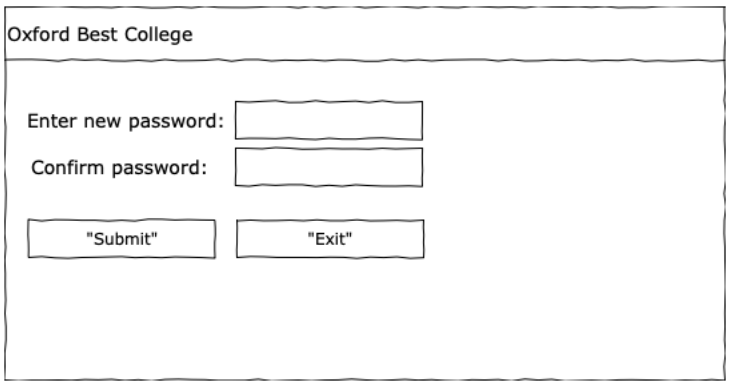


Figure 21: GUI - Change pass diagram

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.

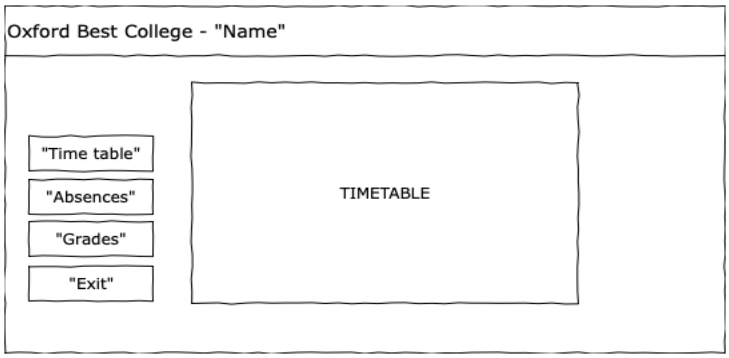


Figure 22: GUI - Main screen diagram

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.

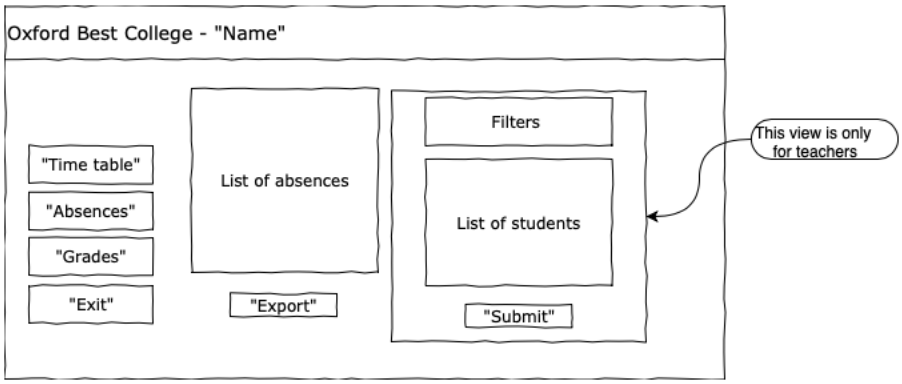


Figure 12: GUI – Absences screen diagram

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.

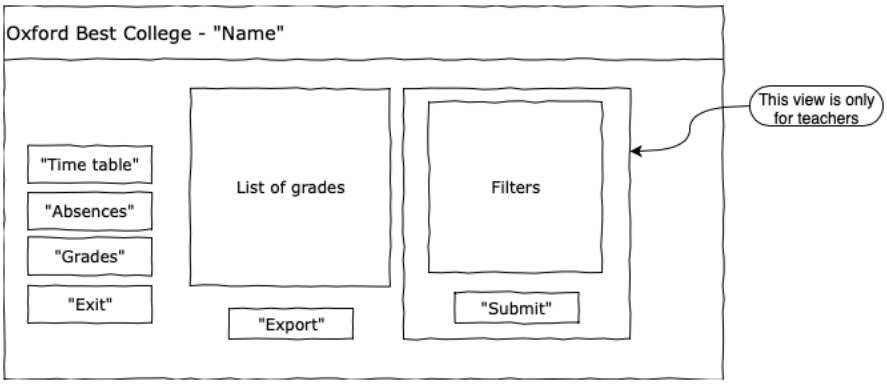


Figure 13: GUI – Grades screen diagram

Example screen using GuiZero

Example 1: “Hallo world”

Code:

```
from guizero import *

SYSTEM = App(title = "Example 1", height = 30, width = 200)
text = Text(SYSTEM, text= "Hallo World")

SYSTEM.display()
```

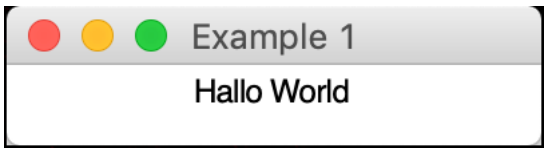


Figure 14: Example GUI 1

Example 2: “Log In”

Code:

```
from guizero import *

SYSTEM = App(title = "Example 2", height = 300, width = 300)
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, 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()

SYSTEM.display()
```

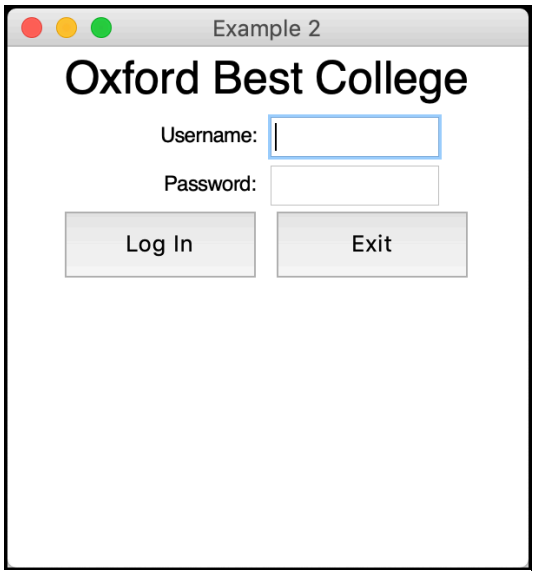


Figure 15: Example GUI 2

Example 3: Timetable

Code:

```
1  from guizero import *
2
3  SYSTEM = App(title = "Example 3", height = 200, width = 400, layout =
   "grid")
4  Grid = []
5
6  Days = ["Monday","Tuesday","Wednesday","Thursday","Friday"]
7  for Day in range (5):
8      Grid.append( Text(SYSTEM, text = Days[Day], grid = [Day+1,0]) )
9
10 Periods = ["08:35","09:45","11:10","13:00","14:10","15:15"]
11 for Period in range (6):
12     Grid.append( Text(SYSTEM, text = Periods[Period], grid =
   [0,Period+1]) )
13
14 SYSTEM.display()
```

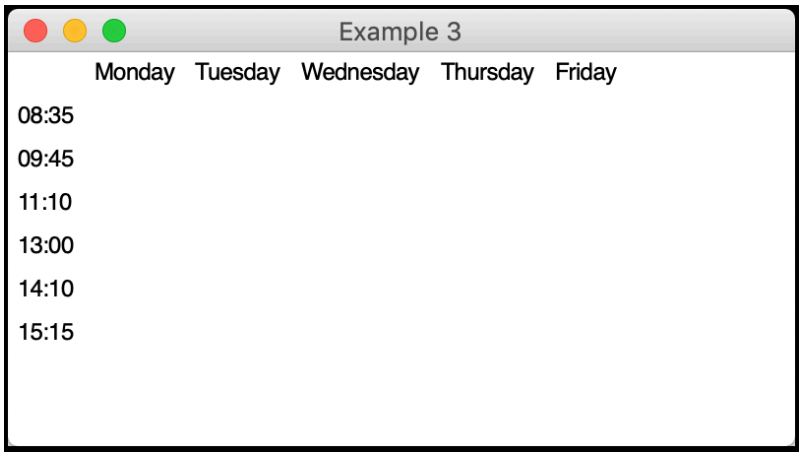


Figure 16: Example GUI 3

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:

```
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:

```
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
```

```
198
199     s_absences_list.destroy()
200     s_absences_list = Box(s_absences, height = 300, width = 350, grid =
    [0,0,1,4], align = "left")
201
202     if user_type == 1:
203         query = '''select  Date,
204           Subjects.Subject
205           from Attendance
206           inner join Lessons on Lessons.ID_Lesson = Lesson_id
207           inner join Groups on Groups.ID_Group = Group_id
208           inner Join Subjects on Subjects.ID_Subject = Subject_id
209           Where Attendance.Student_id =  ?
210           '''
211     elif user_type == 2:
212         filter_days_absences()
213         query = '''select  Date,
214           Students.Student_Name,
215           Students.Student_Surname
216           from Attendance
217           inner join Students on Students.ID_student = Student_id
218           inner join Lessons on Lessons.ID_Lesson = Lesson_id
219           Where Lessons.Teacher_id =  ?
220           '''
221
222     cursor.execute(query, str(user_id))
223     data = cursor.fetchall()
224     text_absences = ""
225     for absence in data:
226         absence = list(absence)
227         absence[0] = str(absence[0])
228         absence[0] =
    absence[0][:2]+"."+absence[0][2:4]+"."+absence[0][4:]
229         if len(absence) == 2:
230             for item in absence:
231                 text_absences += str(item) + "  -  "
232             text_absences = text_absences[:len(text_absences)-4]
233         else:
234             text_absences += absence[0] + "  -  " + absence[1] + " " +
    absence[2]
235         text_absences += "\n"
236     text_absences = text_absences[:len(text_absences)-1]
237     l_absences = TextBox(s_absences_list, text = text_absences, grid =
    [0,0], align = "top", width = 100, height = 12, scrollbar = True,
    multiline = True)
238     l_absences.disable()
239     text_total = ["Total absences:", len(data)]
240     l_absences_total = Text(s_absences_list, text = text_total[0]+"
    "+str(text_total[1]), grid  =[0,1])
241     b_absences_reports = PushButton(s_absences_list, absance_report,
    text = "Export report", grid = [0,2])
```

```
242 def filter_days_absences():
243     global user_id, c_abs_days, s_absences_filters
244     s_absences_filters.destroy()
245     s_absences_filters = Box(s_absences,  grid = [1,0,1,1], align =
    "top")
246     query = ''' Select Day
247             from Lessons
248             where Teacher_id = ?
249             '''
250     cursor.execute(query, str(user_id))
251     list_days = [x for x in cursor.fetchall()]
252     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")
253 def filter_periods_absences(selected_value):
254     global user_id, c_abs_periods
255     query = ''' select Start
256             from Lessons
257             inner join Periods on Periods.ID_Period = Lessons.Period_id
258             Where Teacher_id = ?
259             and Day = ?'''
260     cursor.execute(query,(str(user_id),selected_value))
261     list_periods = [str(x[0]) for x in cursor.fetchall()]
262     list_periods =  sorted(set([x[:2]+":"+x[2:4] for x in
    list_periods]))
263     c_abs_periods = Combo(s_absences_filters, options=list_periods,
    command=list_students_append, grid=[0,1], width=12, align="left")
264 def list_students_append(selected_value):
265     global user_id, list_students, s_absences_append
266     s_absences_append.destroy()
267     s_absences_append = Box(s_absences, layout = "grid", align =
    "left", grid = [1,1,1,3])
268     day = c_abs_days.value
269     periods = c_abs_periods.value
270     query = '''select Students.Student_Name,
271             Students.Student_Surname,
272             Students.ID_Student
273             from Lessons
274             inner join Student_Group on Student_Group.Group_id =
    Lessons.Group_id
275             inner join Students on Students.ID_Student =
    Student_Group.Student_id
276             inner join Periods on Periods.ID_Period = Lessons.Period_id
277             where Day = ?
278             and Periods.Start = ?
279             and Teacher_id = ?'''
280     cursor.execute(query,
    (str(day),str(periods[:2]+periods[3:]),str(user_id) ))
281     data = cursor.fetchall()
282     list_students = []
283     for student in range(len(data)):
```

```
284         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"))
285     button_submit_absences = PushButton(s_absences_append,
286 add_absences, text = "Submit absences", grid = [0,(student+1)])
287 def get_absent():
288     global list_students, list_absences
289     list_absences=[]
290     for student in list_students:
291         if student.value == 1:
292             list_absences.append(student.get_text())
293 def add_absences():
294     global list_absences, user_id, s_absences_append
295     query = '''select ID_lesson
296                 FROM
297                 Lessons
298                 inner join Periods on Periods.ID_Period = Period_id
299                 Where Day = ?
300                 and Periods.Start = ?
301                 and Teacher_id = ?'''
302     cursor.execute(query, (str(c_abs_days.value),
    str(c_abs_periods.value[:2]+c_abs_periods.value[3:]), str(user_id)))
303     lesson_id = cursor.fetchone()[0]
304     query = '''INSERT INTO Attendance
305             (Lesson_id, Student_id, Date)
306             values(?,?,?)'''
307     for student in list_absences:
308         student = student.split("-")
309         cursor.execute(query, (lesson_id,
    student[0][0],time.strftime("%d%m%Y", time.localtime())))
310         database.commit()
311     screen_absences()
312     s_absences_append.destroy()
313     s_absences_append = Box(s_absences, grid = [0,1])
314 def absance_report():
315
316     global text_absences, text_total
317
318     text_absences_2 = text_absences.split("\n")
319     with open('Report_Attendance.csv', 'w') as csvfile:
320         filewriter = csv.writer(csvfile, delimiter=',', quotechar =
    '|', quoting = csv.QUOTE_MINIMAL)
321         filewriter.writerow(["Date","Name"])
322         for line in text_absences_2:
323             line = line.strip()
324             line = line.split(" - ")
325             filewriter.writerow(line)
326         filewriter.writerow(text_total)
327
328
329 def screen_grades():
```

```
330     s_timetable.hide()
331     s_absences.hide()
332     s_grades.show()
333     display_grades()
334 def display_grades():
335     global user_type, user_id, s_grades_list, text_grades
336     s_grades_list.destroy()
337     s_grades_list = Box(s_grades,  height = 300, width = 350, grid =
    [0,0,1,4], align = "left")
338 
339     if user_type == 1:
340         query = '''select  Date,
341           Subjects.Subject,
342           Grade
343 
344           from Grades
345           inner join Lessons on Lessons.ID_Lesson = Lesson_id
346           inner join Groups on Groups.ID_Group = Group_id
347           inner Join Subjects on Subjects.ID_Subject = Subject_id
348           Where Grades.Student_id =  ?'''
349     elif user_type == 2:
350         query = '''select  Date,
351           Students.Student_Name,
352           Students.Student_Surname,
353           Grade
354           from Grades
355           inner join Students on Students.ID_student = Student_id
356           inner join Lessons on Lessons.ID_Lesson = Lesson_id
357           Where Lessons.Teacher_id =  ?'''
358         filter_days_grades()
359 
360     cursor.execute(query, str(user_id))
361     data = cursor.fetchall()
362     text_grades = ""
363     for grade in data:
364         grade = list(grade)
365         grade[0] = str(grade[0])
366         grade[0] = grade[0][:2]+"."+grade[0][2:4]+"."+grade[0][4:]
367 
368         if len(grade) == 3:
369             for item in grade:
370                 text_grades += str(item) + "  -  "
371             text_grades = text_grades[:len(text_grades)-4]
372         else:
373             text_grades += grade[0] + "  -  " + grade[1] + " " +
    grade[2] + "  -  "+ grade[3]
374         text_grades += "\n"
375     text_grades = text_grades[:len(text_grades)-1]
376     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)
```

```
465
466
467 # ----- GUI -----
468 SYSTEM = App(title = "School Information System", height = 350, width =
    1000, layout = "grid")
469
470 # ----- Log In -----
471 s_login = Box(SYSTEM, layout = "grid", grid = [0,0], align = "top")
472 l_logo = Text(s_login, text = "Oxford Best College", size = 26, grid =
    [0,0,2,1])
473 l_user = Text(s_login, text = "Username:", grid = [0,2], align =
    "right")
474 l_pass = Text(s_login, text = "Password:", grid = [0,3], align =
    "right")
475 b_login = PushButton(s_login, logIn, text = "Log In", width = 10, grid
    = [0,4])
476 b_exit = PushButton(s_login, exit, text = "Exit", width = 10, grid =
    [1,4])
477 input_user = TextBox(s_login, grid = [1,2], align="left")
478 input_pass = TextBox(s_login, grid = [1,3], align="left")
479 input_user.focus()
480
481 # ----- Change default password -----
482 s_change_pass = Box(SYSTEM, layout = "grid", grid = [0,0], align =
    "top")
483 l_logo = Text(s_change_pass, text = "Oxford Best College", size = 26,
    grid = [0,0,2,1])
484 l_new_pass = Text(s_change_pass, text = "Enter new Pasword:", grid =
    [0,1], align = "right")
485 l_confirm_pass = Text(s_change_pass, text = "Confirm Password:", grid =
    [0,2], align = "right")
486 input_new_pass = TextBox(s_change_pass, grid = [1,1], align = "left")
487 input_confirm_pass = TextBox(s_change_pass, grid = [1,2], align =
    "left")
488 button_submit = PushButton(s_change_pass, change_pass, text = "Submit",
    width = 10, grid = [0,3])
489 button_exit = PushButton(s_change_pass, exit, text = "Exit", width =
    10, grid = [1,3])
490 input_new_pass.focus()
491 s_change_pass.hide()
492
493 # ----- Menu bar -----
494 s_menu = Box(SYSTEM, height = "fill", layout = "grid", align = "left",
495 grid = [0,1])
496 button_timetable = PushButton(s_menu, screen_timetable, text =
    "Timetable", width = 10, height = 2, grid = [0,2,2,1])
497 button_absences = PushButton(s_menu, screen_absences, text =
    "Absences", width = 10, height = 2, grid = [0,3,2,1])
498 button_grades = PushButton(s_menu, screen_grades, text = "Grades",
    width = 10, height = 2, grid = [0,4,2,1])
499
```

```
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.

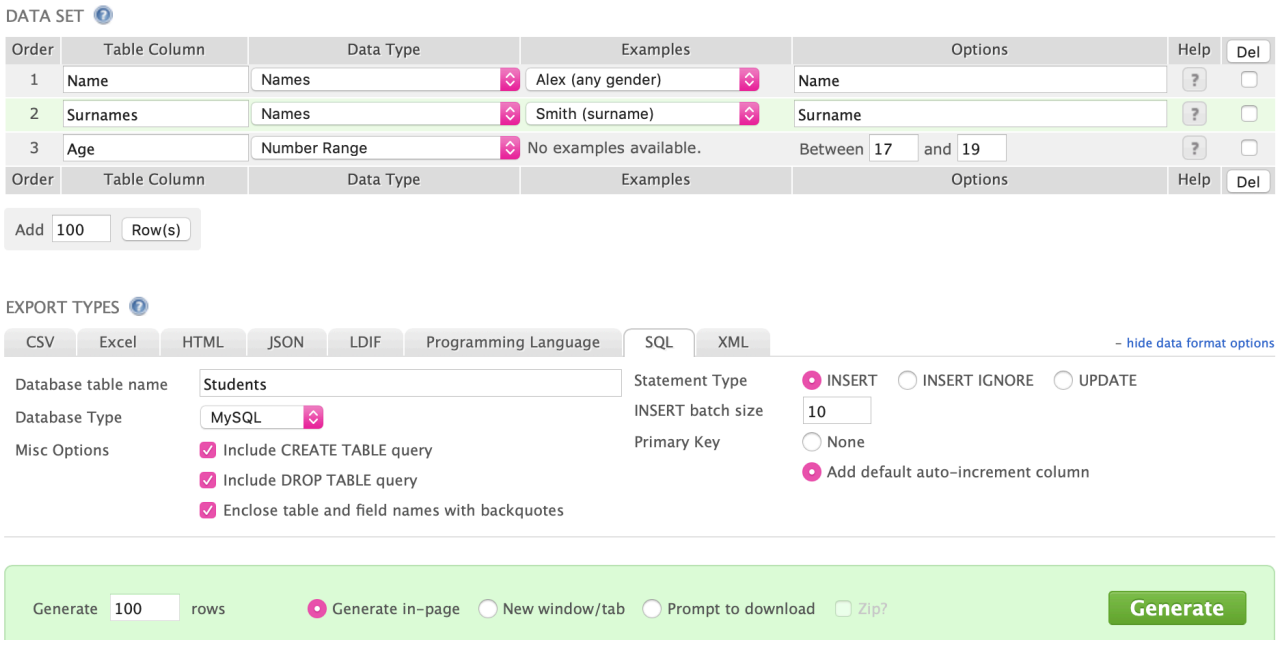
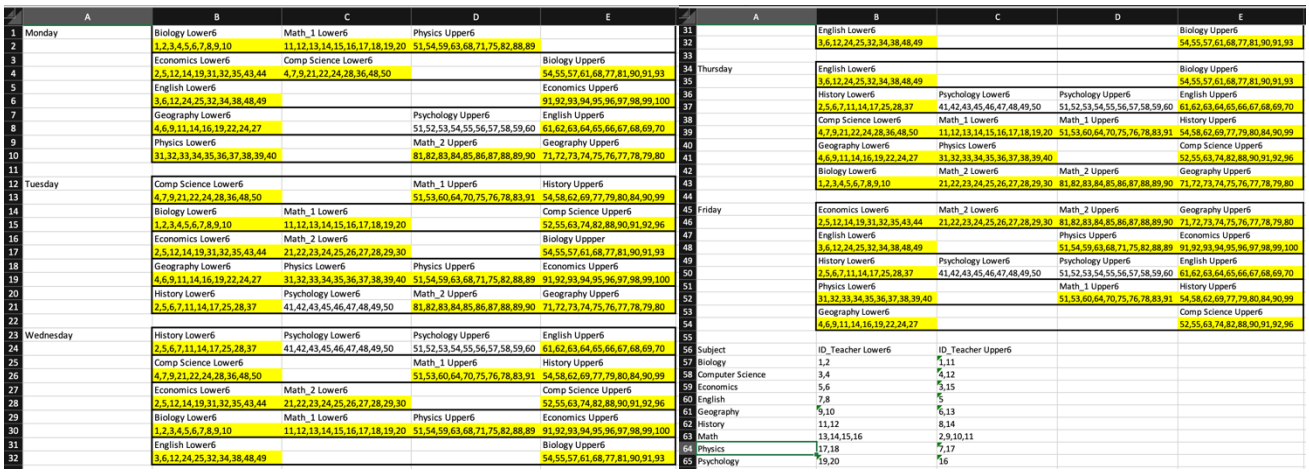


Figure 17: www.generatedata.com

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.



	A	B	C	D	E
1	Monday	Biology Lower6	Math_1 Lower6	Physics Upper6	
2		1,2,3,4,5,6,7,8,9,10	11,12,13,14,15,16,17,18,19,20	51,54,59,63,68,71,75,82,88,89	
3		Economics Lower6	Comp Science Lower6		Biology Upper6
4		2,5,12,14,19,31,32,35,43,44	4,7,9,21,22,24,28,36,48,50		54,55,57,61,68,77,81,90,91,93
5		English Lower6			Economics Upper6
6		3,6,12,24,25,32,34,38,48,49			91,92,93,94,95,96,97,98,99,100
7		Geography Lower6		Psychology Upper6	English Upper6
8		4,6,9,11,14,16,19,22,24,27		51,52,53,54,55,56,57,58,59,60	61,62,63,64,65,66,67,68,69,70
9		Physics Lower6		Math_2 Upper6	Geography Upper6
10		31,32,33,34,35,36,37,38,39,40		81,82,83,84,85,86,87,88,89,90	71,72,73,74,75,76,77,78,79,80
11					
12	Tuesday	Comp Science Lower6		Math_1 Upper6	History Upper6
13		4,7,9,21,22,24,28,36,48,50		51,53,60,64,70,75,76,78,83,91	54,58,62,69,77,79,80,84,90,99
14		Biology Lower6	Math_1 Lower6		Comp Science Upper6
15		1,2,3,4,5,6,7,8,9,10	11,12,13,14,15,16,17,18,19,20		52,55,63,74,82,88,90,91,92,96
16		Economics Lower6	Math_2 Lower6		Biology Uppper
17		2,5,12,14,19,31,32,35,43,44	21,22,23,24,25,26,27,28,29,30		54,55,57,61,68,77,81,90,91,93
18		Geography Lower6	Physics Lower6	Physics Upper6	Economics Upper6
19		4,6,9,11,14,16,19,22,24,27	31,32,33,34,35,36,37,38,39,40	51,54,59,63,68,71,75,82,88,89	91,92,93,94,95,96,97,98,99,100
20		History Lower6	Psychology Lower6	Math_2 Upper6	Geography Upper6
21		2,5,6,7,11,14,17,25,28,37	41,42,43,45,46,47,48,49,50	81,82,83,84,85,86,87,88,89,90	71,72,73,74,75,76,77,78,79,80
22					
23	Wednesday	History Lower6	Psychology Lower6	Psychology Upper6	English Upper6
24		2,5,6,7,11,14,17,25,28,37	41,42,43,45,46,47,48,49,50	51,52,53,54,55,56,57,58,59,60	61,62,63,64,65,66,67,68,69,70
25		Comp Science Lower6		Math_1 Upper6	History Upper6
26		4,7,9,21,22,24,28,36,48,50		51,53,60,64,70,75,76,78,83,91	54,58,62,69,77,79,80,84,90,99
27		Economics Lower6	Math_2 Lower6		Comp Science Upper6
28		2,5,12,14,19,31,32,35,43,44	21,22,23,24,25,26,27,28,29,30		52,55,63,74,82,88,90,91,92,96
29		Biology Lower6	Math_1 Lower6	Physics Upper6	Economics Upper6
30		1,2,3,4,5,6,7,8,9,10	11,12,13,14,15,16,17,18,19,20	51,54,59,63,68,71,75,82,88,89	91,92,93,94,95,96,97,98,99,100
31		English Lower6			Biology Upper6
32		3,6,12,24,25,32,34,38,48,49			54,55,57,61,68,77,81,90,91,93

	A	B	C	D	E
31		English Lower6			Biology Upper6
32		3,6,12,24,25,32,34,38,48,49			54,55,57,61,68,77,81,90,91,93
33					
34	Thursday	English Lower6			Biology Upper6
35		3,6,12,24,25,32,34,38,48,49			54,55,57,61,68,77,81,90,91,93
36		History Lower6	Psychology Lower6	Psychology Upper6	English Upper6
37		2,5,6,7,11,14,17,25,28,37	41,42,43,45,46,47,48,49,50	51,52,53,54,55,56,57,58,59,60	61,62,63,64,65,66,67,68,69,70
38		Comp Science Lower6	Math_1 Lower6	Math_1 Upper6	History Upper6
39		4,7,9,21,22,24,28,36,48,50	11,12,13,14,15,16,17,18,19,20	51,53,60,64,70,75,76,78,83,91	54,58,62,69,77,79,80,84,90,99
40		Geography Lower6	Physics Lower6		Comp Science Upper6
41		4,6,9,11,14,16,19,22,24,27	31,32,33,34,35,36,37,38,39,40		52,55,63,74,82,88,90,91,92,96
42		Biology Lower6	Math_2 Lower6	Math_2 Upper6	Geography Upper6
43		1,2,3,4,5,6,7,8,9,10	21,22,23,24,25,26,27,28,29,30	81,82,83,84,85,86,87,88,89,90	71,72,73,74,75,76,77,78,79,80
44					
45	Friday	Economics Lower6	Math_2 Lower6	Math_2 Upper6	Geography Upper6
46		2,5,12,14,19,31,32,35,43,44	21,22,23,24,25,26,27,28,29,30	81,82,83,84,85,86,87,88,89,90	71,72,73,74,75,76,77,78,79,80
47		English Lower6		Physics Upper6	Economics Upper6
48		3,6,12,24,25,32,34,38,48,49		51,54,59,63,68,71,75,82,88,89	91,92,93,94,95,96,97,98,99,100
49		History Lower6	Psychology Lower6	Psychology Upper6	English Upper6
50		2,5,6,7,11,14,17,25,28,37	41,42,43,45,46,47,48,49,50	51,52,53,54,55,56,57,58,59,60	61,62,63,64,65,66,67,68,69,70
51		Physics Lower6		Math_1 Upper6	History Upper6
52		31,32,33,34,35,36,37,38,39,40		51,53,60,64,70,75,76,78,83,91	54,58,62,69,77,79,80,84,90,99
53		Geography Lower6			Comp Science Upper6
54		4,6,9,11,14,16,19,22,24,27			52,55,63,74,82,88,90,91,92,96
55					
56	Subject	ID_Teacher Lower6	ID_Teacher Upper6		
57	Biology	1,2	1,11		
58	Computer Science	3,4	4,12		
59	Economics	5,6	3,15		
60	English	7,8	5		
61	Geography	9,10	6,13		
62	History	11,12	8,14		
63	Math	13,14,15,16	2,9,10,11		
64	Physics	17,18	7,17		
65	Psychology	19,20	16		

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.)

```
1   SELECT  Subjects.Subject,
2           Teachers.Teacher_Name,
3           Teachers.Teacher_Surname
4           FROM Lessons
5           INNER JOIN Groups ON Groups.ID_Group=Lessons.Group_id
6           INNER JOIN Student_Group ON Student_Group.Group_id=Groups.ID_Group
7           INNER JOIN Students ON Students.ID_Student = Student_Group.Student_id
8           INNER JOIN Teachers ON Teachers.ID_Teacher=Lessons.Group_id
9           INNER JOIN Periods ON Periods.ID_Period=Lessons.Period_id
10          INNER JOIN Subjects ON Subjects.ID_Subject=Groups.Subject_id
11          Where Students.ID_Student= 2
12          AND Lessons.Day= "Monday"
13          AND Periods.Start= "0835"
```

	Subject	Teacher_Name	Teacher_Surname
1	Biology	Orli	Gallagher

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)

```
SELECT ID_Student,
       Password
       FROM Students
       WHERE Student_Name = "Blossom"
       AND Student_Surname = "Adams"
```

	ID_Student	Password
1	2	ebec84f5c3636ccd90c3d58f589b0c3e

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))

```
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 =  2
```

	Date	Subject
1	18032019	Biology
2	19032019	Biology
3	31032019	Biology
4	31032019	Biology

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.)

```
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 = "Monday"
       and Periods.Start = "0835"
       and Teacher_id = 1
```

	Student_Name	Student_Surname	ID_Student
1	Kimberly	Newton	1
2	Blossom	Adams	2
3	Elmo	Fox	3
4	Jarrod	Fox	4
5	Rinah	Blair	5
6	Fiona	Tucker	6
7	Nell	Whitley	7

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)

```
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 =  1
```

	Date	Student_Name	Student_Surname	Grade
1	18032019	Blossom	Adams	A
2	31032019	Burke	Jacobs	A*
3	31032019	Blossom	Adams	E
4	31032019	Kimberly	Newton	B
5	31032019	Blossom	Adams	C
6	31032019	Elmo	Fox	A*
7	31032019	Fiona	Tucker	D

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

Test 6: Query to inserting grades in the database.

```
1  INSERT INTO Grades
2          (Lesson_id, Student_id, Grade, Date)
3          values(1,1,"C","07052019")
```

7	31032019	Fiona	Tucker	D
8	7052019	Kimberly	Newton	C

Figure 24: Test Query 6 - Inserting grades

User login

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

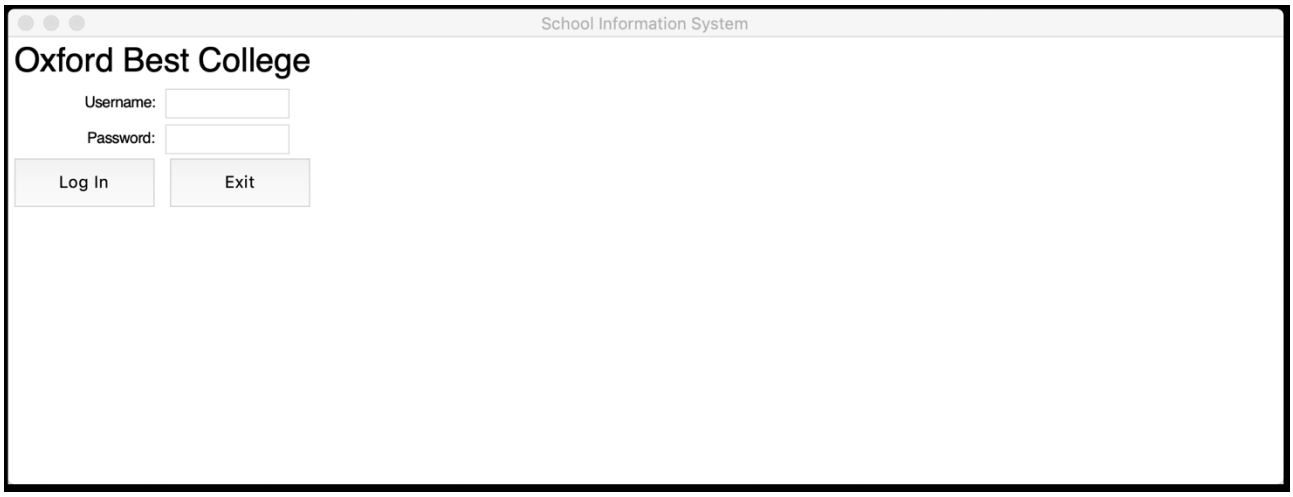


Figure 25: login screen

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.

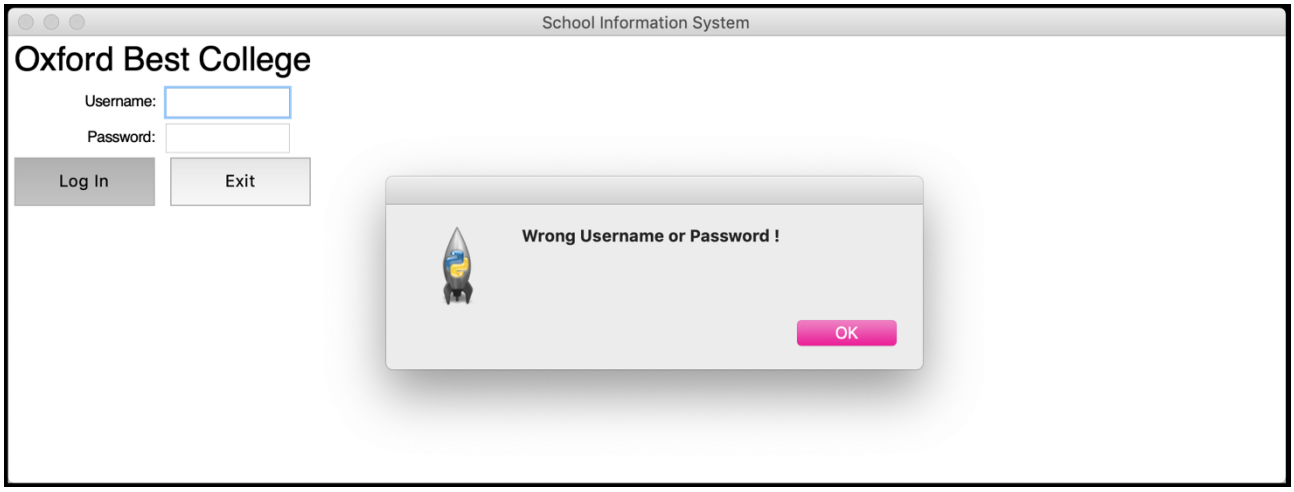


Figure 26: Wrong credentials

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

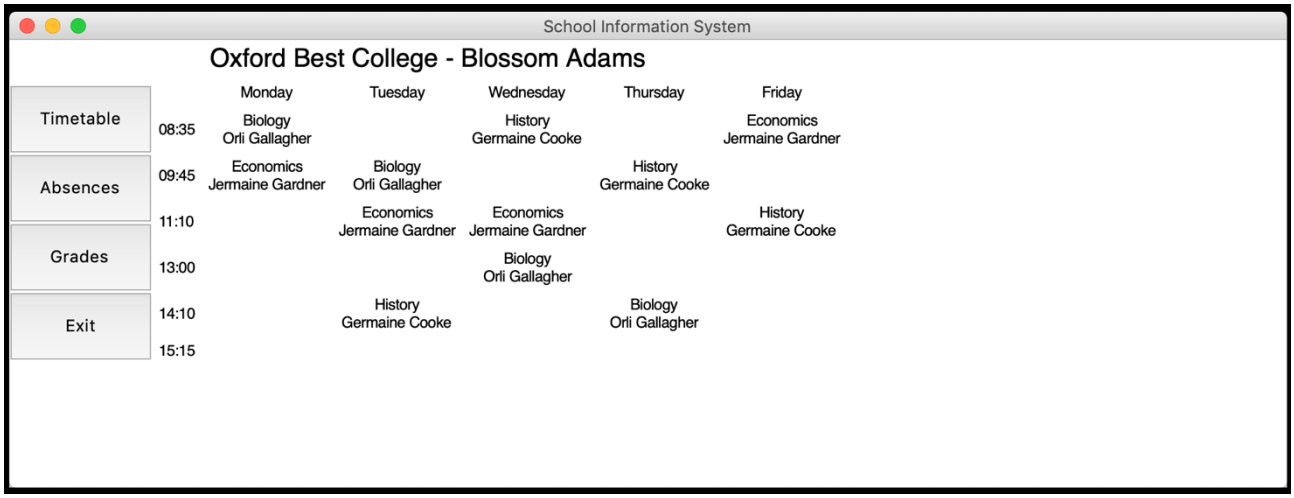


Figure 27: main/timetable screen

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.

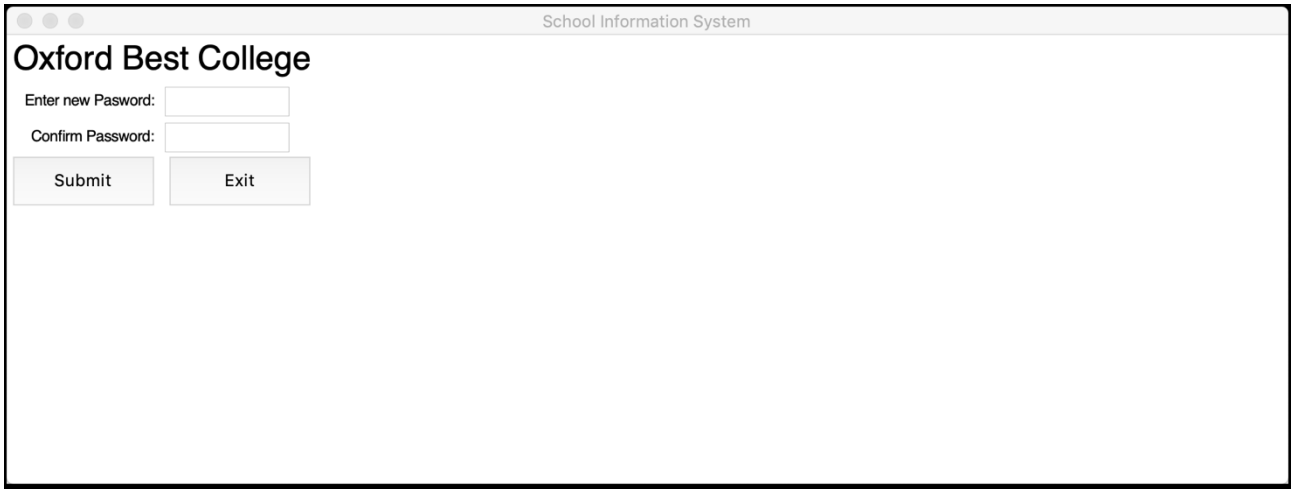


Figure 28: Change password screen

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

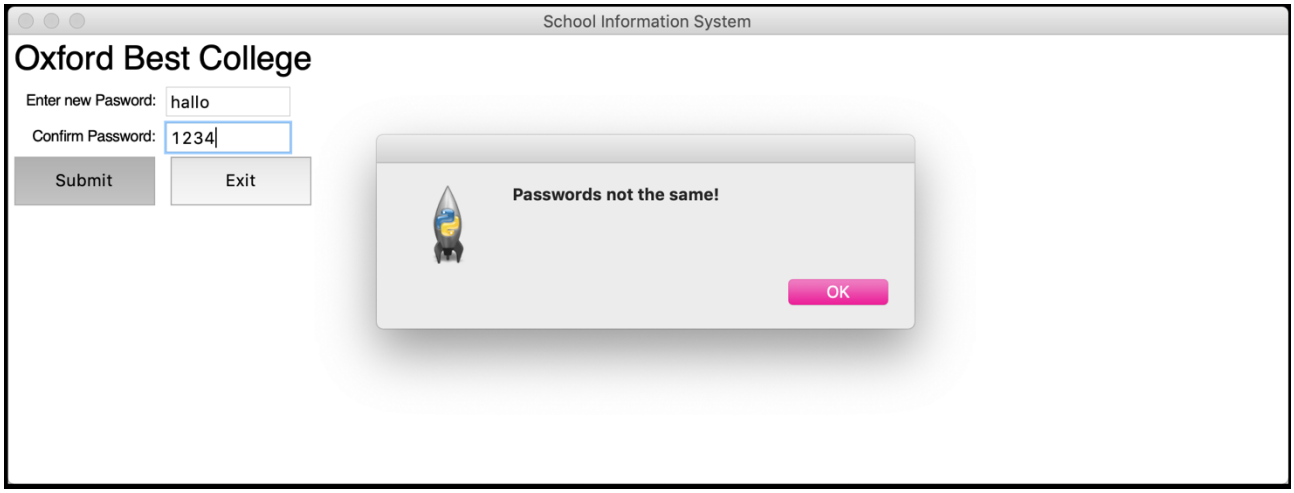


Figure 29: "Password not the same!" error

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

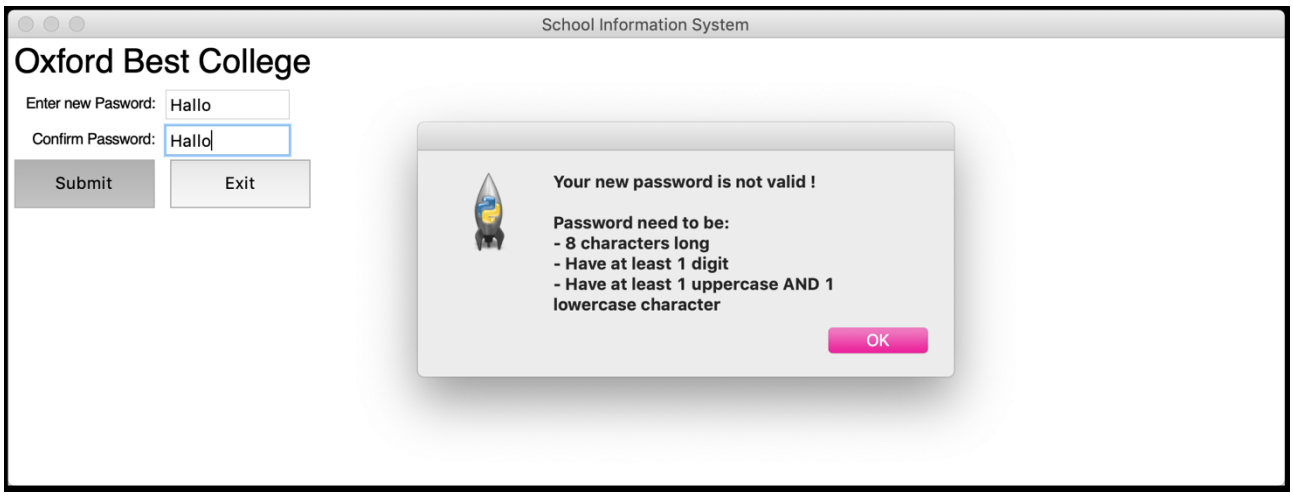


Figure 30: Not a valid password error

Querying data and displaying it

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.

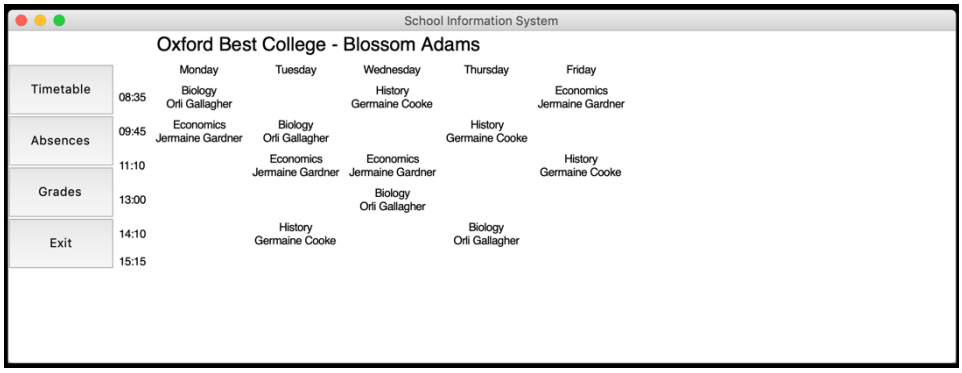


Figure 31: Timetable screen - Student

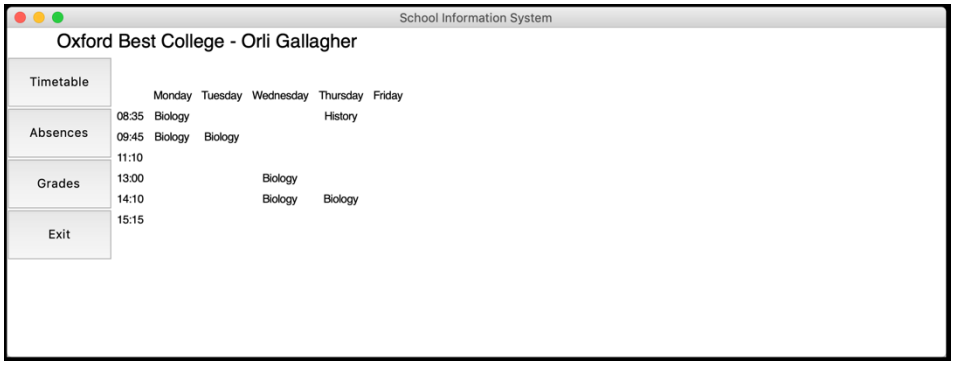


Figure 32: Timetable screen - Teacher

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.

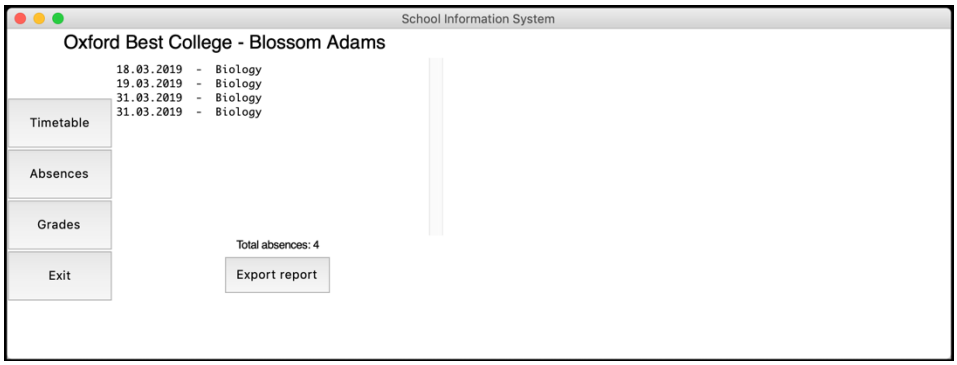


Figure 33: Absences screen - Student

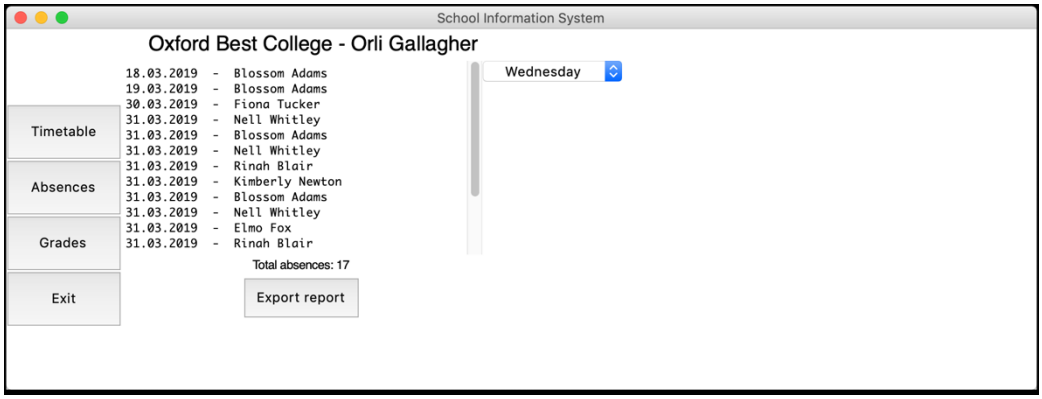


Figure 34: Absences screen - Teacher

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.

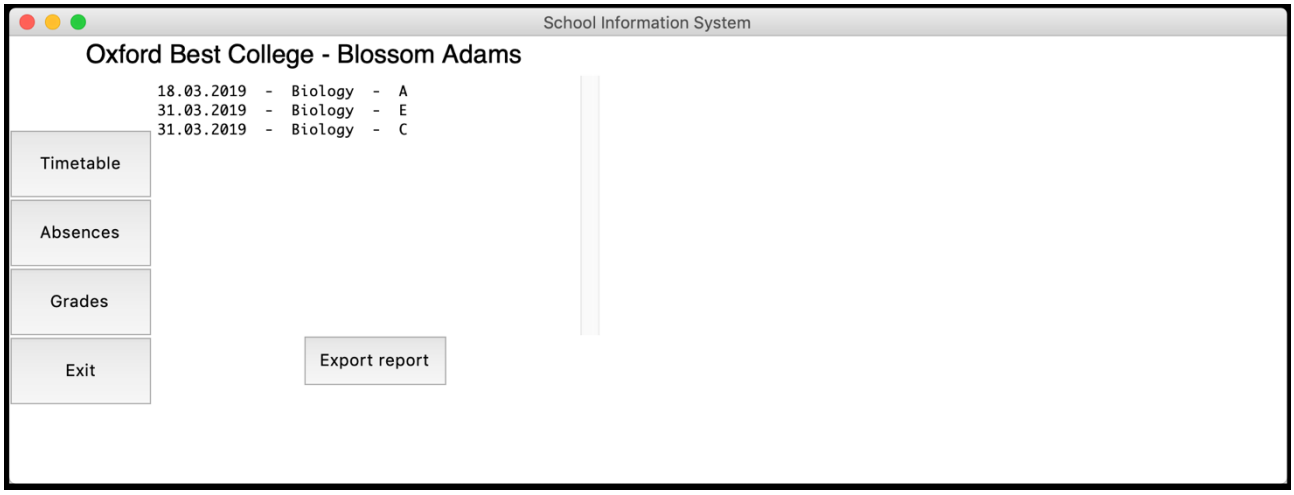


Figure 35: Grades - Student

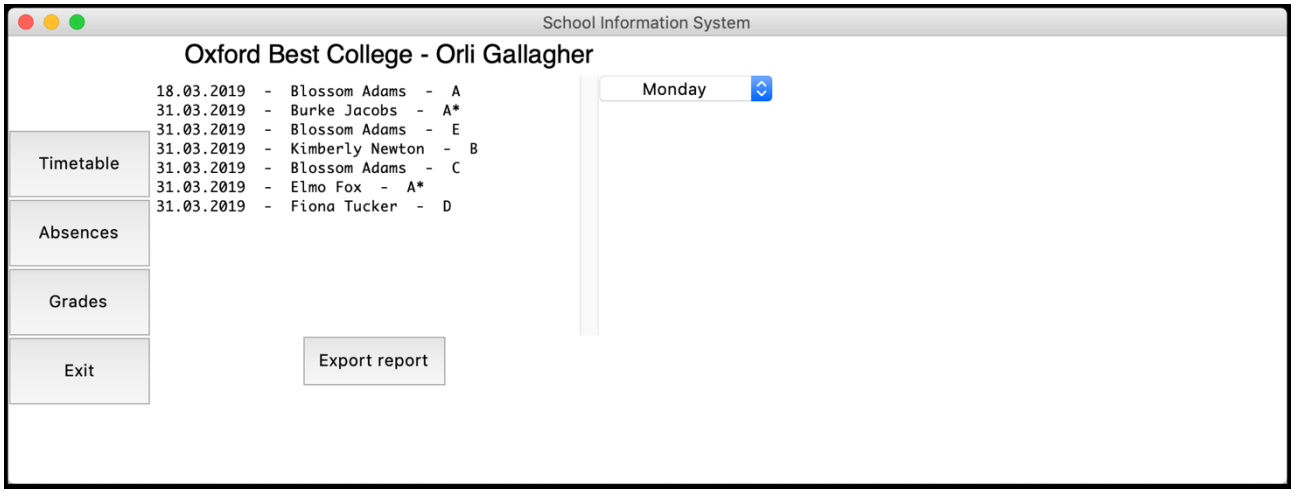


Figure 36: Grades - Teacher

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.

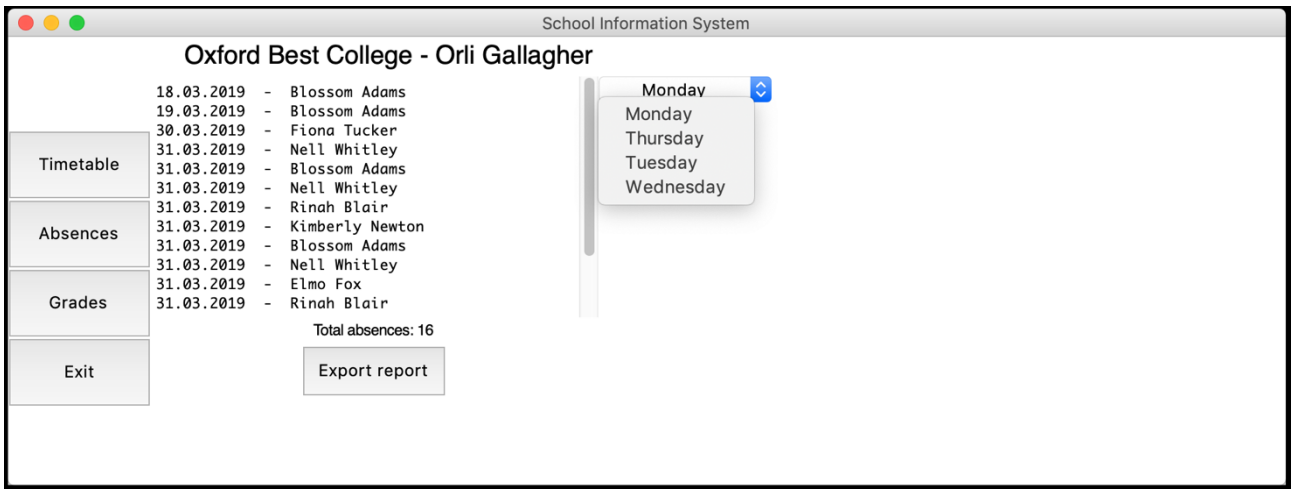


Figure 37: Weekdays filter

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.

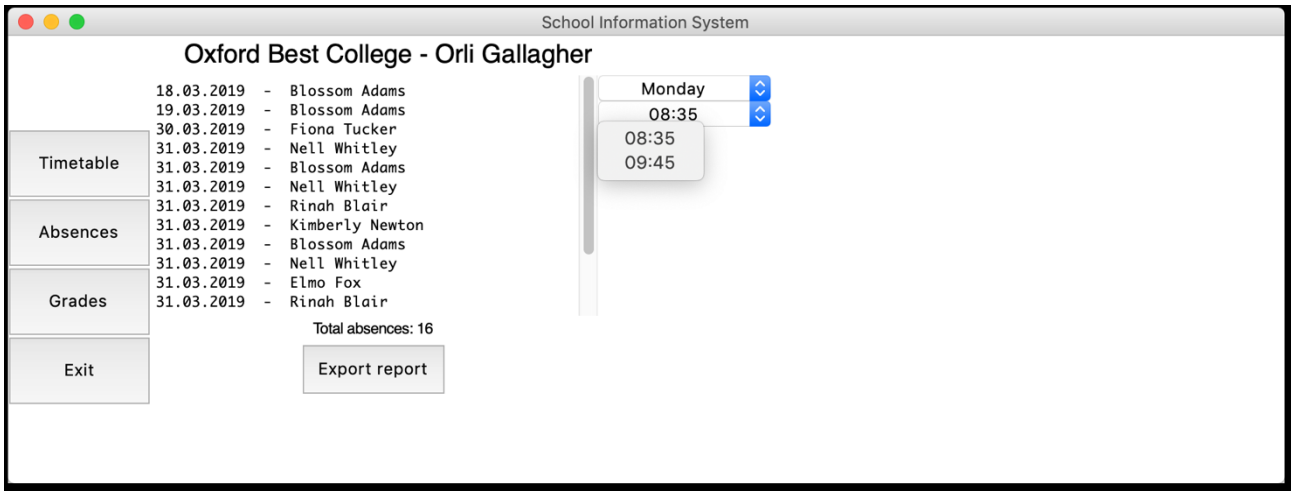


Figure 38: Periods filter

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.

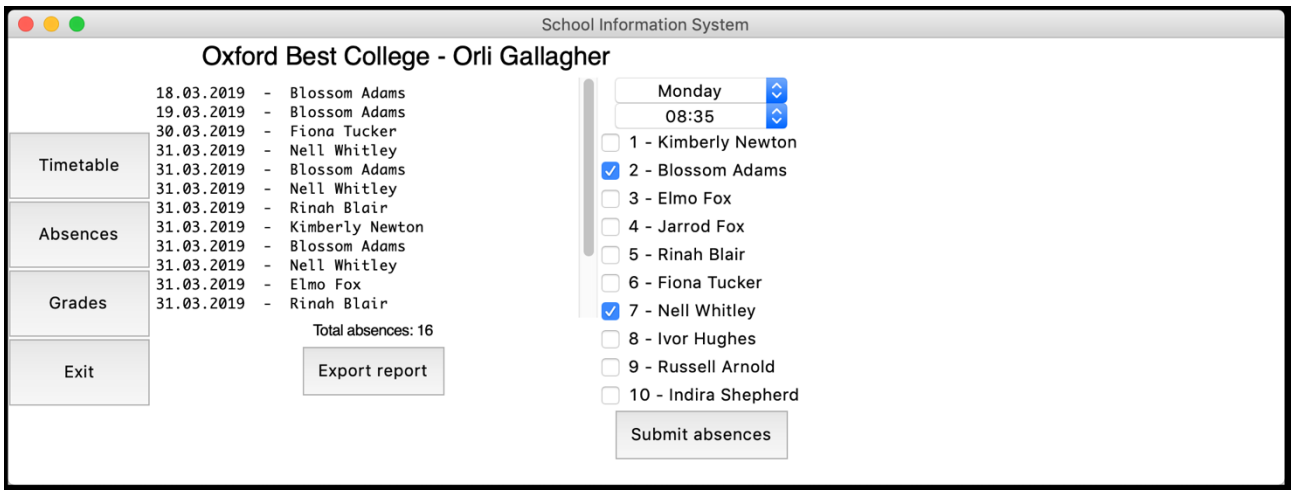


Figure 39: Checklist filter

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.

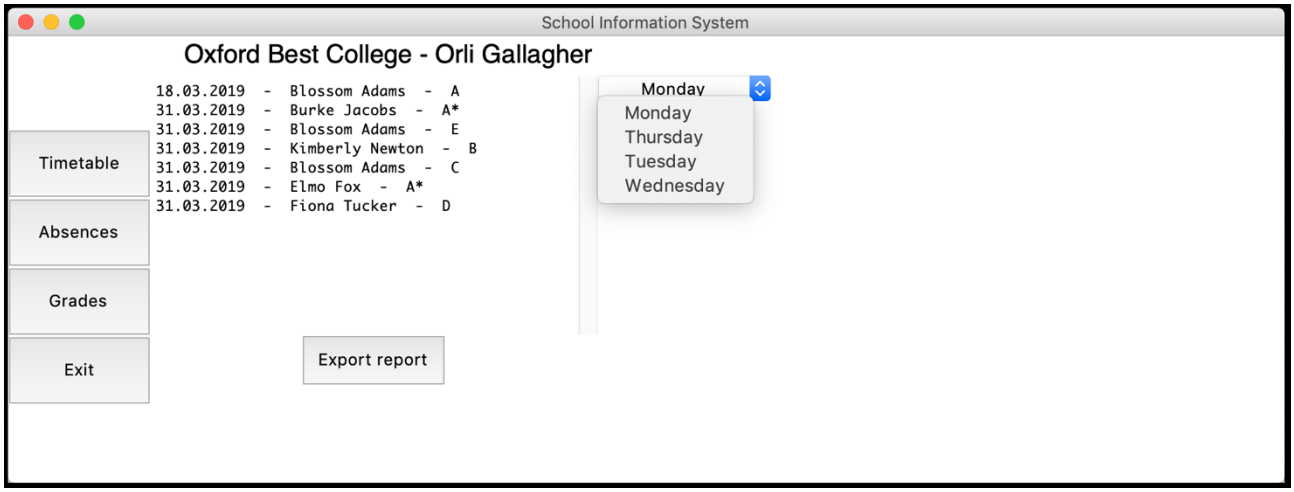


Figure 40: Weekdays filter

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.

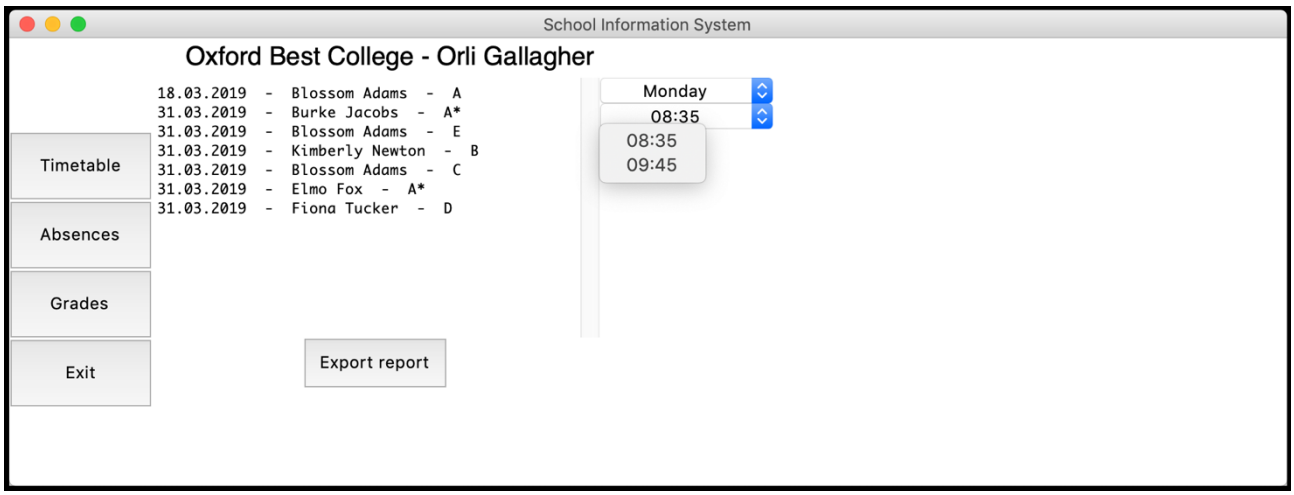


Figure 41: Periods filter

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.

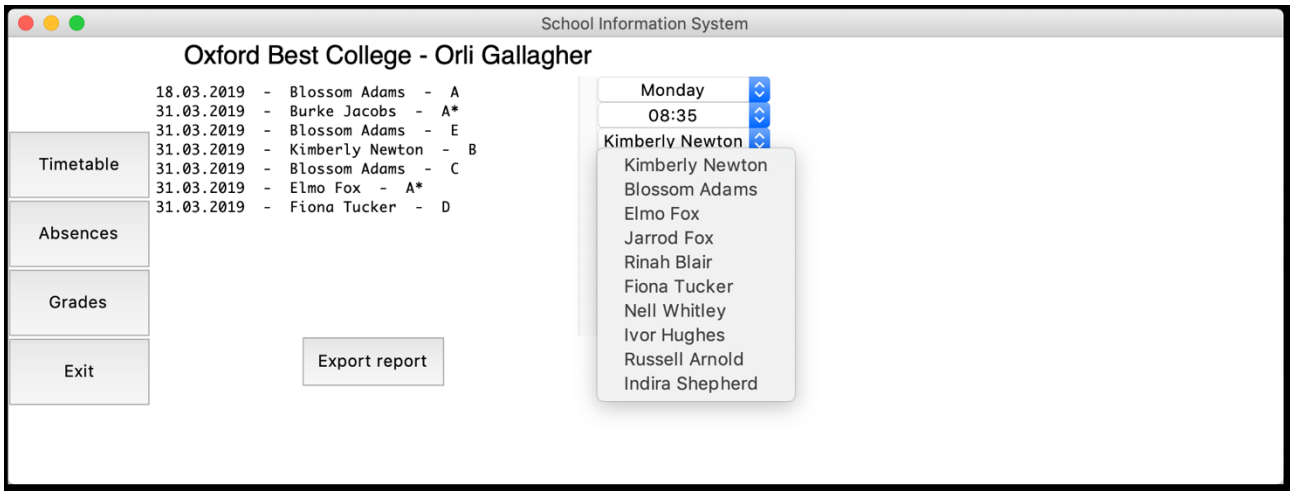


Figure 42: Students filter

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

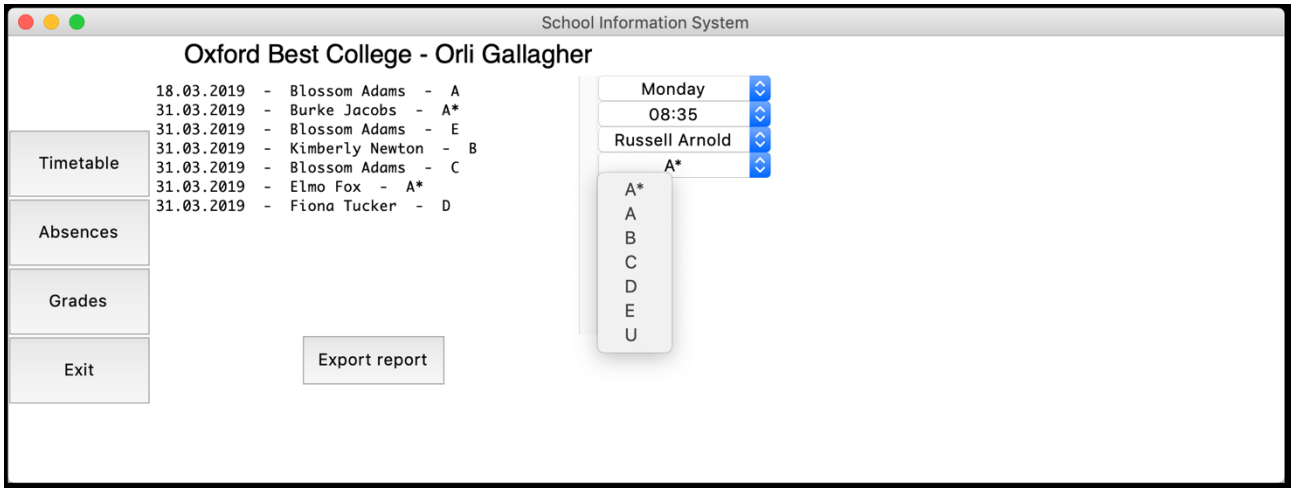


Figure 43: Grades filter

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

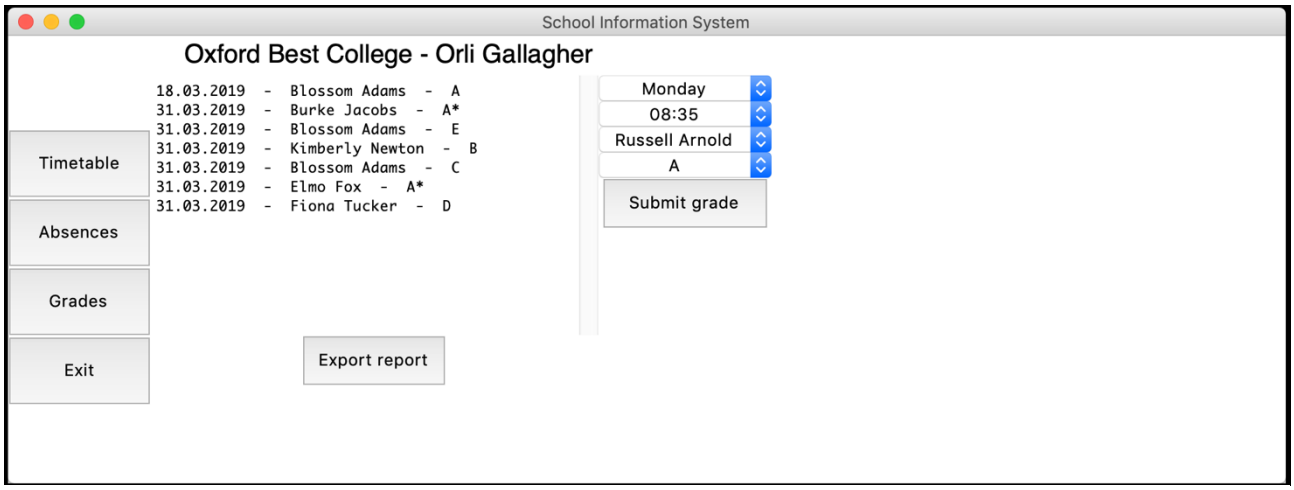


Figure 44: "Submit" button displayed

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.

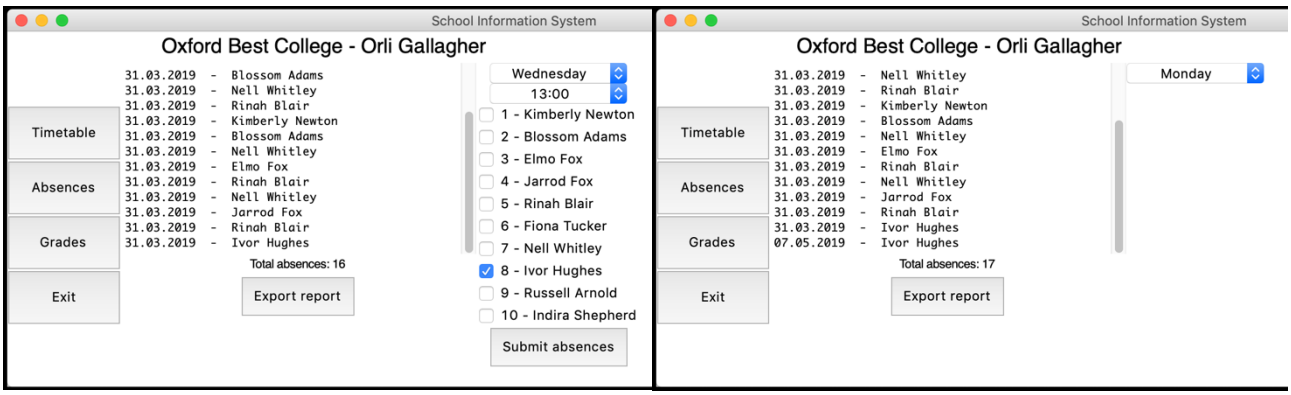


Figure 45: Inserting an absence into the database

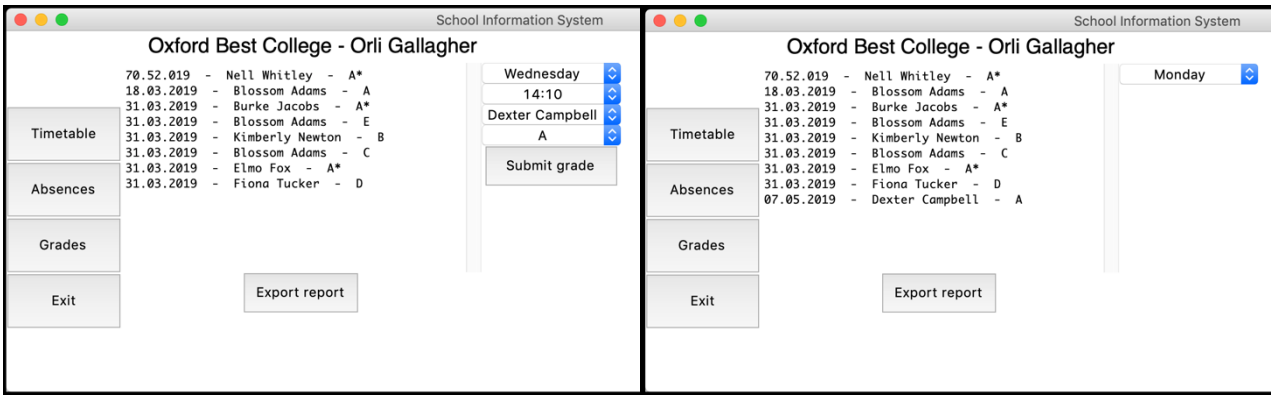
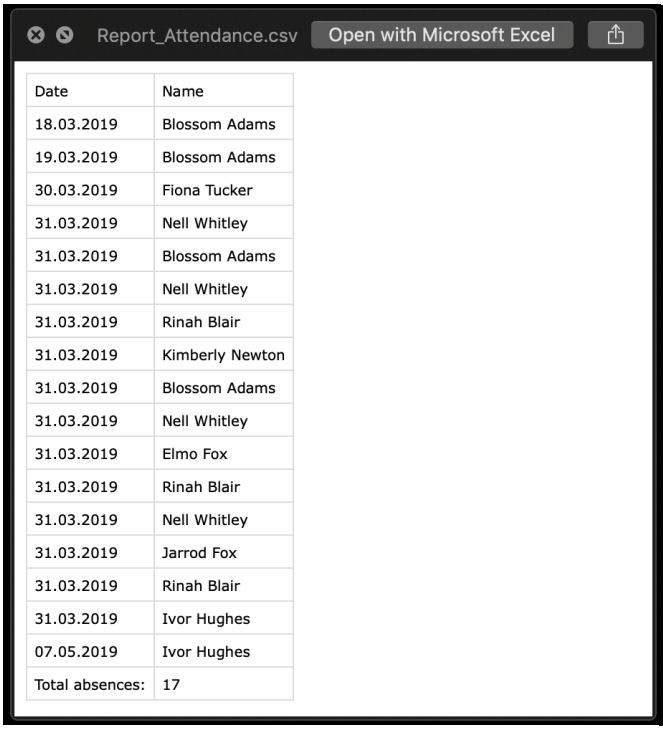


Figure 46: Inserting a grade into the database

Exporting reports

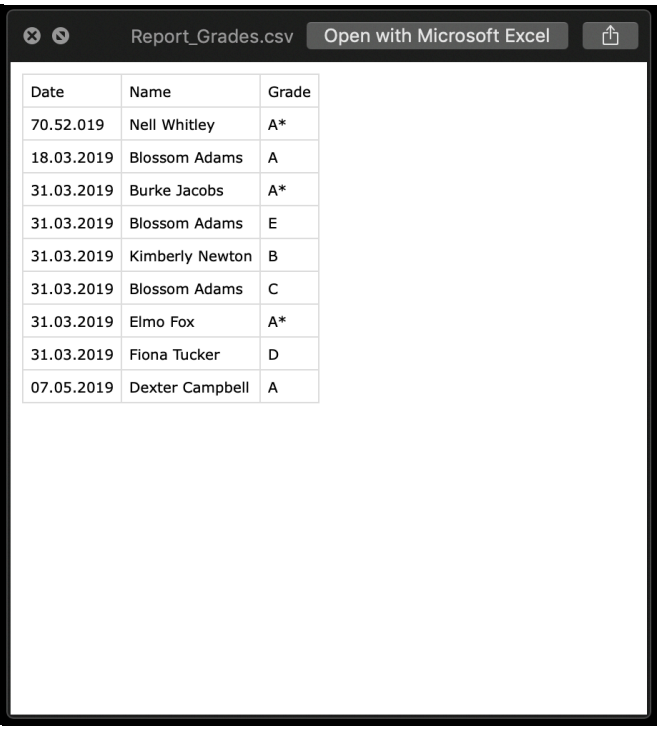
In the absences and grades screens all user are presented with an “Export” button. When clicked the displayed data is exported as a csv file. The file is stored in the same folder where the program is stored.



Report_Attendance.csv Open with Microsoft Excel

Date	Name
18.03.2019	Blossom Adams
19.03.2019	Blossom Adams
30.03.2019	Fiona Tucker
31.03.2019	Nell Whitley
31.03.2019	Blossom Adams
31.03.2019	Nell Whitley
31.03.2019	Rinah Blair
31.03.2019	Kimberly Newton
31.03.2019	Blossom Adams
31.03.2019	Nell Whitley
31.03.2019	Elmo Fox
31.03.2019	Rinah Blair
31.03.2019	Nell Whitley
31.03.2019	Jarrod Fox
31.03.2019	Rinah Blair
31.03.2019	Ivor Hughes
07.05.2019	Ivor Hughes
Total absences:	17

Figure 57: Absence report



Report_Grades.csv Open with Microsoft Excel

Date	Name	Grade
70.52.019	Nell Whitley	A*
18.03.2019	Blossom Adams	A
31.03.2019	Burke Jacobs	A*
31.03.2019	Blossom Adams	E
31.03.2019	Kimberly Newton	B
31.03.2019	Blossom Adams	C
31.03.2019	Elmo Fox	A*
31.03.2019	Fiona Tucker	D
07.05.2019	Dexter Campbell	A

Figure 58: Grades 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
4. https://www.sqlite.org/index.html
5. https://processing.org
6. https://docs.python.org/3/library/tk.html
7. https://lawsie.github.io/guizero/about/
8. https://www.generatedata.com