RFID Sign-in and Sign-out system

# **Analysis**

## The problem

The school's existing method which involves a student signing their name on a sheet of paper, there are many things wrong with the current approach such as the fact that it can easily be manipulated by signing someone else's name, and it is out of the way when leaving the school.

My approach would use the students' existing RFID cards to identify them in the system, this is effortless and quicker as will not take more than 1 or 2 seconds to read the ID card and sign the student in.

This benefits the school as it will allow them, in the case of a fire to quickly print out a list of who is on site, for a quick and accurate register. Another benefit is that if a student is truanting then the school is still accountable and responsible for them while they are on the site, so it will allow staff to monitor who is on site but not in lesson allowing them to send a member of staff to look for the student. *This also increases student safety such as it also allows for tracking on-site guests in the school who may have malicious intentions*

It would link into SIMS using their API, this is because my school currently uses SIMS as a central system for all student data including attendance, SIMS allows all teachers to securely access and review attendance data and generate reports.

All student data is stored securely in an SQL database running on the device, this is required to make is GDPR and DPA compliant as student data is classed as sensitive in the Data Protection Act meaning it must be kept protected and secure

## Features solvable by computational method

### Subprograms/Modularity

I would use subprograms/modularity these allow for parts of the code to be segmented, tested, and run separately. the 3 main components of my program are

The reader, this is the part of the program that will interact with the card

The SQL database connection, this is the part of the program to interact with the database, using the username and password provided

The data handling such as the SIMS API (now a website). This will be used to read the data in the database and show it to the user, such as a teacher

### Thinking ahead

I will use thinking ahead, this is when parts of the program, before they are needed, are loaded into memory this is so that when they are needed, they are faster to access. This would allow my program to run faster as the computer doesn’t have to fetch it from storage when needed

I would use abstraction to remove unnecessary data such as []

## Essential features

### SQL database

* Encryption for GDPR/DPA compliance as student data is classed as sensitive data
* SQL is designed for fast data management

### Python

* I have pre-existing experience using it
* Can interface with the RFID reader and SQL database at the same time to create users and log sign in/out activity

## Research

### Inventry

**Overview**

A local school uses a system similar to my proposed approach, they use a solution by a company called inventry. The inventry system has a feature that when a fire button is pressed then the system shows who is signed in and on site so that staff know who is on site is the case of an emergency. The drawback of this system is it requires pupils to be issues with a different id badge to use the system, my system will use the existing id cards.

**Parts to apply to my system**

I would like to copy the emergency report feature that the inventry solution has, this would likely make use of a webserver running on the raspberry pi

Graphical user interface

Description automatically generated

### Entrysign

**Overview**

Another solution is entrysign, they use a touchscreen device placed in the main entrance where the student types in their name. The disadvantage to this system would be that it can easily be bypassed in the same way the paper solution can, such as typing someone else name. But an advantage would be that if a student forgot their ID card, they could still sign in.

**Parts to apply to my system**

I would also like to replicate this touchscreen for my solution but this would be too expensive for the current project



## Stakeholders

### SLT (Headteacher, Head of 6th form)

An easy way to access data regarding students' attendance over a period of time, to be able to filter by student or attendance

### IT staff

A system that is easy to maintain and manage, this means the code is easy to understand for future updates or if the school want to add new features

### Students

Students would be the main user of the system interacting with it on a daily basis, this means it must be reliable and easy to use

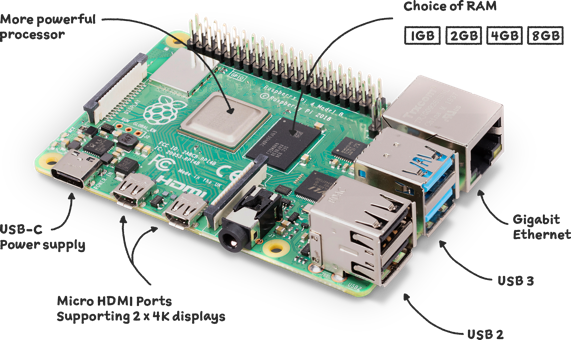
## Limitations

Requires card initialisation, this is a one-time process where each RFID card has to be added into the database and assigned a name

## Hardware requirements

### Raspberry Pi

I have used this as they are small, portable, cheap and have the GPIO pins needed to connect to the RFID reader



### MFRC522 RFID card reader

this reads the card and sends the data back to the Raspberry Pi



### Micro SD card

this is connected into the raspberry pi and would store the Linux based operating system, the code and the SQL database

## Success criteria

|  |  |  |  |
| --- | --- | --- | --- |
| **Number** | **Criteria** | **Explanation** | **How to evidence** |
| 1 | Card is read by RFID reader | This ensures the program is able to read from RFID cards | Screenshot of the program reading the UID from a card |
| 2 | Database is read/written to from python | This ensures the program is able to interact with the database | Screenshot of database before and after card is read |
| 3 | User is created in database through a python script |  | Screenshot before and after user has been added, as well as a screenshot of the code output |
| 4 | Attendance time is stored in database | This takes the current time and adds it to the database | Screenshot the database showing the time value |
| 5 | Data available for teachers to view | This will be done through a website | Screenshot the web interface |
| 6 | A binary value is changed to show 0 if the user is not in school and 1 if the user is in school |  | Screenshot the database before and after showing the value change when a card is tapped |
| 7 | Easy to understand code with comments |  | Screenshot the code |
| 8 | Code makes use of functions |  | Screenshot the code |
| 10 | Text output to explain what values were updated |  | Screenshot the python output |
| 11 | A web interface to show what users are in school |  | Screenshot the web interface |
| 12 | Ability to filter the web interface by values such as ID number or the binary value |  | Screenshot the filter options on the web interface |

## Interview

Questions:

1. How could you improve the current system?
2. Have you used a system like this before
3. Does the proposed system meet your requirements?
4. Was it easy to use?
5. How would you like to access the data from the program?
6. Do you have anything else to add?

Question one and two evaluates their history with the current system and with other RFID based systems

Question 3 is a broad question relating to their criteria for the system

Question 4 establishes if there were any issues with an RFID based system

Question 5 is a specific question regarding how the data gathered by the program would be displayed

The final question concludes by asking if they have anything else they want to say that was not covered by the questions

Design

Flowcharts

Diagram

Description automatically generated

Diagram

Description automatically generated

## Variable list

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Type | Use | Validation |
| **Database variables** | | | |
| Host | string | the ip of the device hosting the database, in this case the database is run on the same machine so it is localhost. | This can be validated by checking the string is a valid IP address (e.g xxx.xxx.xxx.xxx)  Additionally, it will be validated by the database connection library that attempts the connection to the database |
| Username | String | username used to access the database | this is validated by the database connection library that attempts the connection to the database |
| Password | String | password used to sign in to access the database |
| Database\_name | String | name of the database that is being accessed |
| **RFID card variables** | | | |
| ID | Integer | the unique id of the RFID card, read using RFID reader. Stored in the database |  |
| Text | String | The raw text stored on the RFID card, read using the RFID reader |  |
| Name | String | The name of the user. inputted when initialising a user, stored in the database along with ID |  |

## Choices

I have decided to use a database as they are efficient for storage of large data sets which would be needed if many students used the device. It also allows for a centralised database server to be used with multiple scanners placed in different locations which creates an area for expansion if needed further down the line. Databases can also be sorted and searched quickly so in the event of an emergency teachers and other members of staff can easily access data regarding who is on site

I have decided to use a raspberry pi, as it runs Linux which is a specialised operating system used in server environments, as a result of this it is very stable and unlikely to crash, reducing maintenance, also, as it is used often in secure server environments, the Linux kernel is constantly updated with new security patches and features. This increases the security and reliability of the device.

Another reason I chose the raspberry pi is because it has built in Wi-Fi meaning it can be controlled, maintained, accessed, and updated remotely without needed to physically connect to the device.

Lastly, I chose the raspberry pi as it has built in GPIO (general purpose input output) pins which allow for practically anything from LEDs to speakers to be connecting but in this case, it allows easy connection for the RFID scanner that is used to scan badges

My original plan was to use the SIMS program so the data would be easily viewable through the existing system, but I have discovered this would not be possible as their API documentation is unavailable without being a SIMS integrator which would be out of the scope of this project and the budget of this project, and without having a SIMS subscription key. So now my plan is to create my own web interface hosted on the raspberry pi

## How the database is formatted

For the attendance table, we are holding three pieces of data for each recorded RFID tap.

id –an integer that is used to keep track of the current row and increases automatically

user\_id – This is an integer, used to tie attendance with a user in our users table that has the same id.

clock\_in – a SQL timestamp. This timestamp is used to track when the user taps their RFID card onto the RFID reader.

For the users table, we are holding four pieces of data for each user that we add.

id – This is an integer that is used to keep track of the current user and increases automatically.

rfid\_uid –the UID that of the RFID card

name –the name of the user

created –when the user was created

## User interface structure

### Users page

View attendance

RFID UID number

Name

#

This is a table view with the following columns populated using values from the database

#: this is the users unique primary key used to identify them within the database(e.g. 1,2,3,4 etc.)

Name: the users name

RFID UID number: this is the unique RFID number stored on the RFID card used to identify the user when the rfid card is scanned

### Attendance page

View users

Is user signed in or out (y/n)

Data/time of last entry

Name

This is a table view with the following columns populated using values from the database

Name: the users name

Data/time of last entry: this shows when the user last tapped their card against the reader

Is user signed in or out (y/n): this is just a yes or no column showing if the user is on site or not

## Test data

## Pseudocode

Development

## Stage 1, SQL Server

The SQL server need to be setup on the raspberry pi by using the MySQL server. To setup the database the following commands were run on the SQL server. The comments (shown in grey) will explain what each command is doing

Errors I encountered:

* Instead of having an external file to keep track of the current ID and increment it by one, or to query the database to find the highest ID currently in there and increment it by one. I discovered there was an easier way already built into SQL called “AUTO\_INCREMENT” this can be seen in line 15 and 29.
* I also had an issue where each time the loop was run (100s of times per second) there would be a new, empty SQL record created. To fix this I add the “NOT NULL” to each

Graphical user interface, text, application, email

Description automatically generated

Stage 2, Test read program

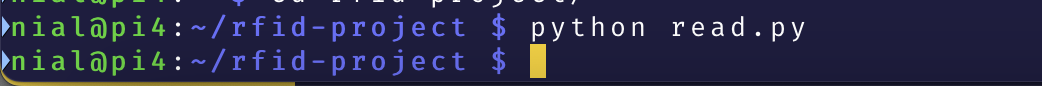
First, to test if reading the cards using the RFID cards would work I created a simple program that would read the card and print the UID of the card in an infinite loop. This was just a proof of concept and was not actually used in the final system.

**Errors I encountered:**

* At first, I created the loop using a while statement and a counter, but I discovered that since the loop ran hundreds of times per second this would not be sustainable.

After further research I became aware of a python function called “try” this allows for an infinite loop to be run easily without the needed for a finite counter. This can be seen at line 8. Try also allows for better error handling as it will run the “finally” statement before exiting

The screenshot below shows the program running and then almost instantly closing wherever the counter is reached

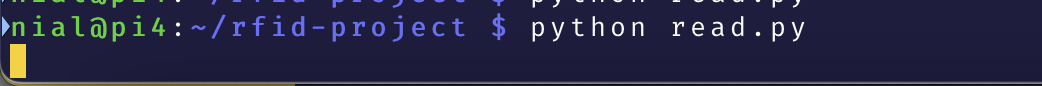


The screenshot below shows the previous version of the code with the old while loop (Line 9)

Graphical user interface, text, application

Description automatically generated

The screenshot below shows the fixed behaviour where the program will wait endlessly



Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generatedStage 5, Sign in script

The python program starts by importing the necessary modules:

* the RPi.GPIO library for controlling the GPIO pins
* the SimpleMFRC522 module for reading RFID tags,
* the mysql.connector library for connecting to the MySQL database.

Next, the program sets up a connection to the MySQL database using the mysql.connector.connect function. The host, user, password, and database are passed to the function are used to connect to the database. A cursor object is also created to execute queries.

The program then enters an infinite loop. It infinitely prompts the user to place a card on the RFID reader. When a card is detected, the program queries the database to see if the card is already registered by looking for a matching rfid\_uid value in the users table.

Stage 4, Card initialisation

As a new iteration of the system, instead of manually adding the name and RFID UID of a user into the database manually through the use of SQL INSERT commands, I decided that I would create a python script to allow this to be a much more user friendly process.

This program will add new users and cards into the database. It uses the Raspberry Pi’s GPIO pins to interface with the RFID reader.

The python program starts by importing the necessary modules:

* the RPi.GPIO library for controlling the GPIO pins
* the SimpleMFRC522 module for reading RFID tags,
* the mysql.connector library for connecting to the MySQL database.

Next, the program sets up a connection to the MySQL database using the mysql.connector.connect function. The host, user, password, and database are passed to the function are used to connect to the database. A cursor object is also created to execute queries.

The program then enters an infinite loop. It infinitely prompts the user to place a card on the RFID reader. When a card is detected, the program queries the database to see if the card is already registered by looking for a matching rfid\_uid value in the users table.

If the card is already registered, the script prompts the user to overwrite the existing user, and if the user chooses to do so, the script updates the existing user's name in the database. If the card is not already registered, the script prompts the user to enter a name for the new user and then inserts a new user into the database. The variable

Finally, after the user is saved, the script outputs that the user has been saved and the loop starts again.

**Errors I encountered:**

* The code executed without any errors, but after checking the database, no data was added by the script

[add screenshot of code without error]

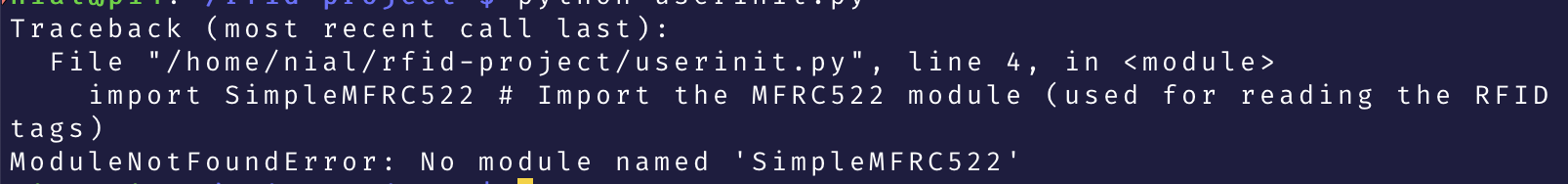
After further researching I discovered that executing the SQL INSERT command (line 44) does not actually add any information to the database, it only “stages” it so then it also needed to be “committed” to the database, this was solved using “db.commit()” after the INSERT command as shown on line 45.

Graphical user interface, text, application

Description automatically generated

* Module SimpleMFRC522 not found

Below is a screenshot of the error



Below is a portion of the code containing the error (Line 4)

Text

Description automatically generated

After researching the issue, I found that SimpleMFRC522 was the name of the submodule not the module itself. So I fixed the by adding “*from mfrc522*” which is the name of the main module before the “*import SimpleMFRC522*” (this can be seen on Line 4 in the screenshot belowText

Description automatically generated

Graphical user interface, text, application

Description automatically generated

## Stage 4, New Web based interface

After further research, it would be far beyond the budget for this project to be able to buy a licence for the SIMS API, instead I am using a website and adding these success criteria

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Explanation** | **How to test** |
| Web interface showing users names | This would show the user who is in the database |  |
| Web interface showing UID numbers | This would be used for troubleshooting, if the UID of the card matches what is in the database |  |
| Idealy a Web interface login system | This would secure the web interface from anyone not authorised to view it |  |
| Homepage | A simple homepage with 2 buttons, one for the users table and one for the attendance table |  |
| Web interface showing timestamp of when card was tapped | This would be used by staff to see who is in school and when they registered |  |
| Web interface showing if user is signed in or out | This would show if a user tapped their card on the reader on a given day |  |

To allow the web interface to interact with the database, the Medoo framework (<https://medoo.in>) will be used, this is a framework that interacts with the SQL server, it is free and open source

The below screenshot contains the configuration needed for the Medoo framework to connect to the database

Graphical user interface, text, application

Description automatically generated

After this the homepage for the web based interface needs to be made, at this point I made the decision to use the Bootstrap CSS framework, this includes premade CSS and JavaScript for navigation bars, themes and buttons.

Errors I encountered:

Different commenting styles between PHP and HTML

Conclusion

Additional features:

Multiple scanners around the school could be used for multiple exits, or one per classroom to replace a register in class that takes up valuable learning time. They could all feed back to a centralised SQL server instead of running one on each scanner. This could also be done on a server such as Amazon AWS, Google’s GCP, or Microsoft Azure to prevent downtime using cloud database. This also meaning in an emergency, the machine on site that holds the database is destroyed or disconnected, a teacher would still be able to see who is on site in case of a