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

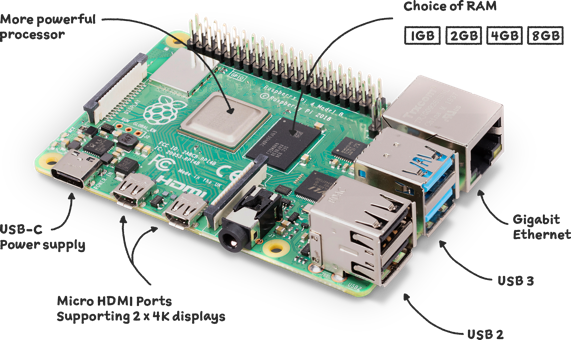
## Limitations

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 |
| Binary\_value | Binary | Shows if the user is signed in or not. 0 means not on site and 1 means signed in | As it is a standard binary value (can only be 1 or 0) means that if it is corrupt (not a 1 or 0) there would be an error |
| Time\_stamp | String | Stores the time the RFID card was tapped on the rad | The time is stored in a standardised form of  YYYY-MM-DD HH-MM-SS |
| **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

Name

Card ID

Is user signed in or out (1/0)

Data/time of last entry

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

Name: the user’s name

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

Is user signed in or out (1/0): This is just a binary column showing if the user is on site or not.

Card ID: this is the UID of the RFID card associated with that user.

## Test data

[to do]

## Pseudocode

[to do]

Development

1st iteration

### Reading the card, 1st attempt

I started by first researching, using forums and blog posts I found on the internet, how I would read the data from the RFID card. The first method I found (<https://forums.raspberrypi.com/viewtopic.php?t=12632>) I tried but did not work

[INSERT SCREENSHOT]

### Reading the card, 2nd attempt

So I then went back and found I a python module called mfrc522 (<https://pypi.org/project/mfrc522/>) in a blog post on how to read rfid cards using a pi (<https://pimylifeup.com/raspberry-pi-rfid-rc522/>) , initially this did not work as I didn’t enable the SPI (Serial peripheral interface)

[INSERT SCREENSHOT]

After enabling this sin the raspberry pi’s settings

[INSERT SCREENSHOT]

I was able to read the data from the card

[INSERT SCREENSHOT]

2nd Iteration

### Change card reading method

Text

Description automatically generated

Using (<https://github.com/nmcc1212/rfid2/blob/d627b2f2af4d4ee86963e30932556b7870d28e62/1st%20try.py>) for reading cards didn’t work, now using SimpleMFRC522

---

### Reader object

A screen shot of a computer

Description automatically generated with low confidence

A reader object was missing

---

### Error handling if RFID UID was not in the database

Text

Description automatically generated

Didn’t handle error if the RFID UID was not in the database, fixed using an if statement to detect if anything was returned by the querry

if result:

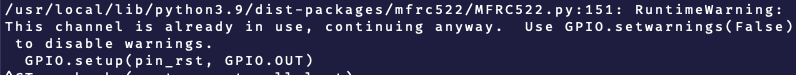
return result[0]

else:

return None

--

### GPIO clean up at exit



The GPIO pins showed as still in use after closing and relaunching the program fixed by adding the “atexit” library which runs a predefined script (GPIO.cleanup) when the program exits

--

### Create a check to see if the card is in the database



It looks as if the script works even though the card is not in the database, fixed by checking if None is returned, if it is printing an Error and then exiting



--

### Add card to database manually

Now I added the RFID card to the DB manually, using the following SQL command

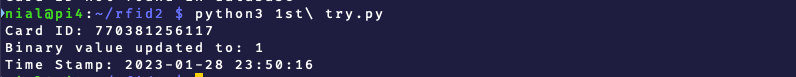
*INSERT INTO rfid2.card\_states (card\_id, binary\_value, timestamp) VALUES (770381256117, 0, "2023-01-01 00:00:00");*

Images of 2nd iteration (adding users manually )

A screenshot of a computer

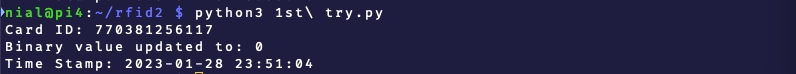
Description automatically generated with low confidence

The card has just been added



The card has been tappedText

Description automatically generated with medium confidence

This is now reflected in the database

Again, the card has been tappedGraphical user interface, text

Description automatically generated with medium confidence

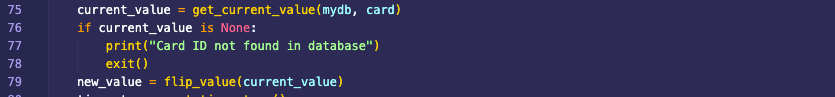
Again, reflected in the database

3rd Iteration

### Create a function for adding new cards to the database

Text

Description automatically generated with medium confidence



Instead of exit() instead, we now call the add\_new\_card() function

### Add support for Names

To add another column to the database for the name of the user I used to SQL code below

ALTER TABLE card\_states

ADD COLUMN name VARCHAR(255);

Add another %s to the add\_new\_cards() function, this allows for another input, the new name variableText

Description automatically generated

### Name missing in SQL INSERT

Text

Description automatically generatedDidn't add names to the SQL insert command, this meant that the script would provide 4 variables (including the name) but the SQL INSERT would only accept 3 as name was not defined, so



Now becomes



### New get name function



The script did not get the name from the database, meaning when the script tried to print the name it didn't know it. This only occurred for when the add\_new\_card() function was not used, as if it was the name would have been provided and stored when add\_new\_card was run

I created a new function get\_name() to get the name from the databaseText

Description automatically generated

Then I changed the IF statement that is used to determine if the user is in the database or not. Now if the user is already in the database the new get\_name() function is called, if the user is not in the database, the add\_new\_card() function is called, which asks for the name already

Graphical user interface, text, application, chat or text message

Description automatically generated

### Final code for iteration 3

The database is empty

Now the card is scanned

Text

Description automatically generated

This can now be seen in the database

Text

Description automatically generated

Now the same card is scanned again

Text

Description automatically generated

4th Iteration

To create the web interface I am using Flask, a web framework written in python. (<https://en.wikipedia.org/wiki/Flask_(web_framework)>)

### Use subfolder template for the HTML files

Text

Description automatically generated

The template that holds the HTML file containing the structure for the website "index.html" could not be found, this is because index.html was placed in the same folder as the python script whereas it should have been in the folder called templates

--

### Change host address

The site can only be accessed from the pi (known as localhost) but not over the network, fixed by adding a bind address of 0.0.0.0, this means the website was bound to all interfaces (in this case the ethernet interface) instead of just localhost, (<https://github.com/flutter/flutter/issues/32629#issuecomment-492176223>)

A screenshot of a computer

Description automatically generated with medium confidence

### Improving the looks

At this point the site was fully functional, as shown below, but I wanted to improve the look of the site

Text, letter

Description automatically generated

To do this I added bootstrap, jQuerry and Popper, these are CSS/JS libraries containing elements such as buttons, navbars and tables

First I created a table to layout the data in a more visually appealing way, this can be seen below

Conclusion

Success Criteria

|  |  |
| --- | --- |
| **Criteria** | **Met?** |
| Card is read by RFID reader | Yes |
| Database is read/written to from python | Yes |
| User is created in database through a python script | Yes |
| Attendance time is stored in database | Yes |
| Data available for teachers to view | Yes |
| A binary value is changed to show 0 if the user is not in school and 1 if the user is in school | Yes |
| Easy to understand code with comments | Yes |
| Code makes use of functions | Yes |
| Text output to explain what values were updated | Yes |
| A web interface to show what users are in (binary value = 1) and what users are signed out (binary value = 0) | Yes |
| Ability to filter the web interface by values such as ID number or the binary value | No |

Evidence

### Card is read by RFID reader.

[INSERT SCREENSHOT]

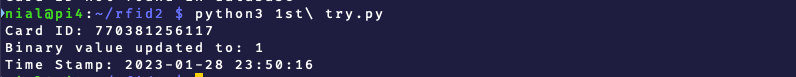
### Database is read and written to from python.

The screenshot below shows a record in the database

Text

Description automatically generated with medium confidence

And this screenshot shows this record being read by the python script



Here you can see the same timestamp, Card ID and binary value as was seen in the database in the above screenshot

### User is created in database through a python script.

Text

Description automatically generated

In the first screenshot, the user is created in the python script. The second image shows this record stored in the MySQL database

Text

Description automatically generated

### Attendance time is stored in database.

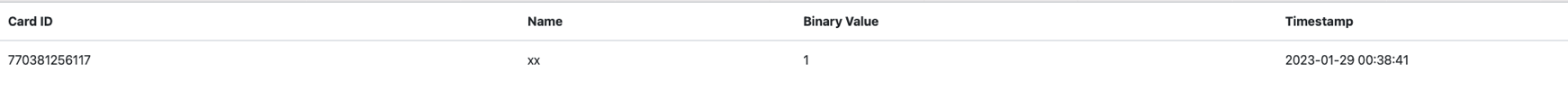
Highlighted in red shows the time storage in the database

Text

Description automatically generated

### Data available for teachers to view.

The screenshot below is of the website which shows data for 1 card, including card number, name, time and binary value



### A binary value is changed to show 0 if the user is not in school and 1 if the user is in school.

Text

Description automatically generated

Text

Description automatically generated

In the 2 above screenshots the user is signed out (as shown by the binary value = 0, shown in red), then the card is scanned, and the binary value is changed to 1 (as shown in red) this shows that the user is signed in

Text

Description automatically generated

### Easy to understand code with comments.

The screenshot below shows the comments (shown in green) for each function and for each import statement.

### Code makes use of functions.

The screenshot below shows all the functions in the code (shown in red)

Graphical user interface, text

Description automatically generated

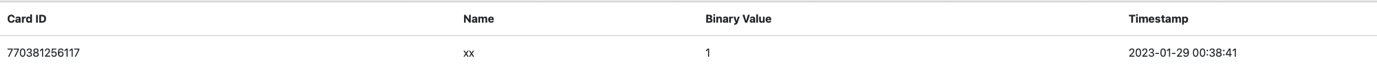
### Text output to explain what values were updated.

Here you can see that it explains the binary value was changed to 1

Text

Description automatically generated

### A web interface to show what users are in (binary value = 1) and what users are signed out (binary value = 0)



Limitations

At the moment, the biggest limitation is the lack of filters in the web interface, this would prevent teachers from being able to quickly refine the table such as showing only users that signed in today, or showing only users that are signed in (binary value of 1)

Issues that could arise from this include inaccurate reporting, this could be caused when a teacher has to look through the table and manually select, for example, students that had signed in (binary value = 1), as the teacher would have to go through the data manually they may miss or overlook some students when the list becomes too long, as there would be over 200 students in the table

### How to avoid this

With more time it would be possible to implement a filter system for the web interface, this would make data analysis, such as student attendance much easier to do

Additional features

Multiple scanners could be placed 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

Maintenance

As the code is open-source, meaning the source code can be viewed and edited, stakeholders could easily add features to the program if needed, this is helped by the fact that the program makes use of functions, the code can easily be understood and changed by anyone with knowledge of Python (also SQL and HTML knowledge may be needed). Functions also mean the program can be easily expanded without affecting the core way the program functions.

Also, as the code makes use of comments for each function, this would allow for a better understanding of the way the code works for someone who has not seen the coe before.

Final code

Main.py

import mysql.connector *#used for reading and writing to the database*

import RPi.GPIO as GPIO *#used for controlling the GPIO pins*

from datetime import datetime *#used for getting the current date and time*

from mfrc522 import SimpleMFRC522 *#used for reading the RFID card*

import atexit *#used for cleaning up GPIO on exit*

atexit.register(GPIO.cleanup) *#clean up GPIO on exit*

*# Connect to MySQL database*

def connect\_to\_database():

mydb = mysql.connector.connect(

host="localhost",

user="rfid2",

password="ad",

database="rfid2"

)

return mydb

*# Set up RFID reader and read card ID*

def read\_card\_id():

reader = SimpleMFRC522()

card = ""

while True:

card, text = reader.read()

break

return card

*# Retrieve current binary value from database*

def get\_current\_value(mydb, card):

mycursor = mydb.cursor()

sql = "SELECT binary\_value FROM card\_states WHERE card\_id = %s"

val = (card,)

mycursor.execute(sql, val)

result = mycursor.fetchone()

if result:

return result[0]

else:

return None

*# Retrieve name from database*

def get\_name(mydb, card):

mycursor = mydb.cursor()

sql = "SELECT name FROM card\_states WHERE card\_id = %s"

val = (card,)

mycursor.execute(sql, val)

result = mycursor.fetchone()

if result:

return result[0]

else:

return None

*# Flip binary value*

def flip\_value(value):

if value == 0:

new\_value = 1

else:

new\_value = 0

return new\_value

*# Get the current date and time*

def get\_time\_stamp():

now = datetime.now()

time\_stamp = now.strftime("%Y-%m-%d %H:%M:%S")

return time\_stamp

*# Update database with new value and timestamp*

def update\_database(mydb, card, new\_value, time\_stamp):

mycursor = mydb.cursor()

sql = "UPDATE card\_states SET binary\_value = %s, timestamp = %s WHERE card\_id = %s"

val = (new\_value, time\_stamp, card)

mycursor.execute(sql, val)

mydb.commit()

*# Add new card to database*

def add\_new\_card(mydb, card):

mycursor = mydb.cursor()

name = input("Enter name: ")

sql = "INSERT INTO card\_states (card\_id, binary\_value, timestamp, name) VALUES (%s, %s, %s, %s)"

val = (card, 0, get\_time\_stamp(), name)

mycursor.execute(sql, val)

mydb.commit()

return name

*# Main function*

def main():

mydb = connect\_to\_database()

card = read\_card\_id()

current\_value = get\_current\_value(mydb, card)

if current\_value is None:

name = add\_new\_card(mydb, card)

else:

name = get\_name(mydb, card)

new\_value = flip\_value(current\_value)

time\_stamp = get\_time\_stamp()

update\_database(mydb, card, new\_value, time\_stamp)

print("Card ID:", card)

print("Binary value updated to:", new\_value)

print("Time Stamp:", time\_stamp)

print("Name:", name)

main()

4th iteration.py (the web interface)

from flask import Flask, render\_template

import mysql.connector

app = Flask(\_\_name\_\_)

*# Connect to MySQL database*

def connect\_to\_database():

mydb = mysql.connector.connect(

host="localhost",

user="rfid2",

password="ad",

database="rfid2"

)

return mydb

*# A function to Create RFID reader object and read card ID*

def get\_all\_cards():

mydb = connect\_to\_database()

mycursor = mydb.cursor()

sql = "SELECT card\_id, binary\_value, timestamp, name FROM card\_states"

mycursor.execute(sql)

result = mycursor.fetchall()

cards = []

for card in result: *#create a*

cards.append({

'card\_id': card[0],

'binary\_value': card[1],

'timestamp': card[2],

'name': card[3]

})

return cards

@app.route('/')

def index():

cards = get\_all\_cards() *#get all cards from database*

return render\_template('index.html', cards=cards) *#pass cards to the web interface*

app.run(debug=True, host='0.0.0.0') *#run the web interface, binding to all interfaces*

templates/index.html